

OPTIMAL PARAMETER CHOICE FOR  
BLOOM FILTER-BASED  
PRIVACY-PRESERVING RECORD  
LINKAGE

Von der Fakultät für Gesellschaftswissenschaften der Universität Duisburg-Essen  
zur Erlangung des akademischen Grades

Dr. phil.

genehmigte Dissertation

von

Christian Borgs

aus

Oberhausen

1. Gutachter: Prof. Dr. Rainer Schnell
2. Gutachter: PD Dr. Günther Heller

Tag der Disputation: 31.07.2019

# Acknowledgements

Many people have been very important in completing this work. I am not a fan of a great many words, so this will be brief, but well deserved.

First, I would like to thank my advisor, Rainer Schnell, who has continued to support me and my thesis through the whole time. I am thankful for the ideas, the understanding and the motivation that has pushed me over the finish line at last.

I thank my committee members for their hard work and swiftness, especially my second reviewer, Günther Heller, for their support and encouragement.

Everyone in my research department deserves my thanks, especially my research group for their input, comments and support. Outside of my University, Marcel Noack has helped me tremendously with his comments and input.

I have to thank all my friends, near and far, old and new for sticking with me while I disappeared a lot, especially towards the end. Without you all, I couldn't have retained my sanity.

Finally, I thank my family and loved ones for believing in me at all times. Thank you, everyone.

## *Ideas and previous publications*

Identifying the need for an optimal parameter choice algorithm and the idea for the topic of this thesis is due to Rainer Schnell. The same is true for using the response surface method (RSM) for optimal parameter choices. Most variables for the models were discussed with Rainer Schnell as well – most of them were his ideas.

I created most examples and some images as part of my work for Rainer Schnell. One example is Figure 3.2 and the resulting Dice coefficient calculation, which was taken from Schnell/Borgs (2018c). Explaining Bloom filters and PPRL was part of many publications I was part of. For this reason, the explanations here will be very close to some previously published ones. However, except where denoted otherwise, all work presented in this thesis is my own original work.

# Abstract

Record Linkage is, for most scientific disciplines, an increasingly popular set of methods to gather or enrich research data for analysis. Since in most countries, perfectly unique personal identifier numbers (PIDs) are not available, data linkage is restricted to attributes like names and birth dates to discriminate between records and their corresponding real-life entities. However, these identifiers are often legally required to be encrypted. This gave way to the field of *Privacy-preserving Record Linkage (PPRL)*. Recently, Bloom filters have gained much attention in PPRL research. Hindering their widespread use is the fact that choosing the right parameters for private linkage operations will, at the moment, require in-depth expert knowledge about the data, since the quality of Privacy-preserving Record Linkage (PPRL) using Bloom filters is highly dependent on the encryption parameter choices. Since there is currently no literature about the optimal choice for these parameters, this thesis aims for an optimal choice automation method for best linkage quality using model estimates based on simulations of the entire parameter space. After giving an in-depth overview of the state of the art in PPRL, the approach is described in depth. The resulting models are then tested using simulated and real-world data sets. Using a naive approach based on current recommendations is tested against the encryption parameters resulting from the model estimates. The results are compared in-depth for each data set. It can be shown that the optimal parameter choices consistently outperform current best-practice parameter settings, sometimes drastically. The thesis concludes with an outlook on open research questions and closes with updated recommendations for Bloom filter (BF)-based Privacy-preserving Record Linkage (PPRL).

# Table of Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Table of Contents</b>	<b>iii</b>
<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>xii</b>
<b>List of Source Code</b>	<b>xiv</b>
<b>Terms and abbreviations</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Thesis structure . . . . .	6
1.2 The term ‘Record Linkage’ . . . . .	7
1.3 Techniques often confused with Record Linkage . . . . .	8
1.4 ‘Big Data’ and Record Linkage . . . . .	9
1.5 The Record linkage process . . . . .	10
1.5.1 Data preprocessing . . . . .	10
1.5.2 Choice of identifiers . . . . .	12
1.5.3 Deduplication . . . . .	13
1.5.4 Blocking . . . . .	13
1.5.5 Matching algorithms . . . . .	14
1.5.5.1 Exact matching . . . . .	14
1.5.5.2 Errors in data . . . . .	15
1.5.5.3 Rule-based matching . . . . .	16
1.5.5.4 Probabilistic matching . . . . .	16
1.5.6 Classification of linkage pairs . . . . .	24
1.5.6.1 Cost-based classifications . . . . .	25
1.5.6.2 Machine learning for classifications: Decision trees . . . . .	26

1.5.6.3	Machine learning for classifications: Support vector machines . . . . .	26
1.5.6.4	Other machine learning classification methods: Neural networks and deep learning . . . . .	28
1.5.7	Evaluation . . . . .	29
<b>2</b>	<b>Privacy-preserving Record Linkage (PPRL)</b>	<b>32</b>
2.1	Definition of PPRL . . . . .	32
2.2	Prelude: Hash functions and HMACs . . . . .	33
2.3	Overview of PPRL methods . . . . .	36
2.3.1	Matching by encrypted PID . . . . .	37
2.3.2	Trusted third-party designs . . . . .	37
2.3.3	Secure multi-party protocols . . . . .	39
2.3.4	Phonetic codes . . . . .	39
2.3.5	String hashing . . . . .	40
2.3.5.1	Encrypted PIDs . . . . .	41
2.3.5.2	Basic ALC . . . . .	41
2.3.5.3	Swiss ALC . . . . .	41
2.3.5.4	Encrypted Statistical Linkage Key (ESL) . . . . .	42
2.3.6	Bloom filters . . . . .	43
2.3.7	Parameter choice for Bloom filters . . . . .	44
2.4	Privacy-preserving blocking and scalability . . . . .	45
2.5	PPRL applications . . . . .	47
<b>3</b>	<b>Bloom filter-based methods</b>	<b>48</b>
3.1	Constructing Bloom filters . . . . .	48
3.1.1	Steps when encrypting records using Bloom filters . . . . .	52
3.1.2	Standardisation and preprocessing . . . . .	53
3.1.3	Padding and $q$ -gram partitioning . . . . .	53
3.1.4	Mapping elements to bit vectors . . . . .	54
3.1.4.1	Double hashing . . . . .	54
3.1.4.2	Random hashing . . . . .	54
3.2	Bloom filter-based encryptions and security . . . . .	55
3.2.1	Constraint satisfaction solver-based attack . . . . .	55
3.2.2	Substitution cipher-based frequency attack . . . . .	56
3.2.3	Frequency-based and pattern mining-based attack . . . . .	57
3.3	Hardening methods . . . . .	57
3.3.1	Random hashing . . . . .	58
3.3.2	Random bits . . . . .	58
3.3.3	Salting . . . . .	59
3.3.4	BLIP/RAPPOR . . . . .	59

3.3.5	Rehashing . . . . .	61
3.3.6	Record-Level Bloom filters . . . . .	62
3.3.7	Balanced Bloom filters . . . . .	62
3.3.8	Vector folding . . . . .	63
3.3.9	Markov chain Bloom filters (MCBFs) . . . . .	64
3.3.10	Cellular automata . . . . .	64
3.3.11	One-time pad-secured Bloom filters . . . . .	65
3.3.12	MinHash Bloom filters (MHBF) . . . . .	66
3.3.13	Summary . . . . .	67
3.4	Linking encryptions based on Bloom filters . . . . .	67
3.4.1	Using Bloom filters with probabilistic Record Linkage . . . . .	67
3.4.2	Multibit trees . . . . .	68
3.5	Properties of Bloom filters . . . . .	70
3.5.1	Variance of linkage quality with repeated Bloom filter generation . . . . .	71
3.5.2	Parameter and identifier choices . . . . .	74
3.5.3	Bits set and optimal parameters of Bloom filters . . . . .	76
3.5.3.1	Testing optimality using synthetic data . . . . .	78
3.5.3.2	Test setup . . . . .	78
3.5.3.3	Results . . . . .	79
3.5.3.4	Testing using non-synthetic data . . . . .	80
3.5.4	Implications . . . . .	82
<b>4</b>	<b>Optimal identifier and parameter choice</b>	<b>84</b>
4.1	Overview of the chapter . . . . .	84
4.2	Optimal identifier choice . . . . .	85
4.3	Optimal parameter choice . . . . .	87
4.3.1	Test setup . . . . .	87
4.3.2	The PPRL package for R . . . . .	90
4.3.3	Datasets . . . . .	91
4.3.3.1	German telephone CD (TelCD) . . . . .	92
4.3.3.2	Real-life mortality register data (Mortality & commercial/hospital data) . . . . .	94
4.3.3.3	FEBRL-generated data (FEBRL) . . . . .	94
4.3.3.4	FEBRL-generated Western Australian data (FEBRL WA) . . . . .	96
4.3.3.5	North Carolina Voter data (NC Voter) . . . . .	96
4.4	Models for optimal parameter choice . . . . .	97
4.4.1	Variable selection . . . . .	97

4.4.2	Models . . . . .	101
4.4.2.1	Simple linear model (LM) . . . . .	104
4.4.2.2	Bayesian model selection based on BIC (BIC) . . . . .	108
4.4.2.3	Response surface model (RSM) . . . . .	110
4.4.2.4	Random Forest-based approach (RF) . . . . .	113
4.5	Resulting estimated and best-practice parameter choices . . . . .	118
4.5.1	Using recommendations from the literature as best-practice suggestions . . . . .	118
4.5.2	Estimating an optimal k using the 50% rule . . . . .	119
4.5.3	Optimal parameters chosen by the models . . . . .	119
<b>5</b>	<b>Results of the optimal parameter choice</b>	<b>121</b>
5.1	Overview of the chapter . . . . .	121
5.2	Recap: Training and test data . . . . .	121
5.2.1	Training data . . . . .	122
5.2.2	Test data . . . . .	122
5.3	Multibit tree settings . . . . .	122
5.4	Model diagnostics . . . . .	123
5.4.1	Assumption violations . . . . .	124
5.4.2	Cross-validation and root mean squared error (RMSE) . . . . .	130
5.5	Results . . . . .	131
5.5.1	Recall and precision . . . . .	133
5.5.2	Mean of recall and precision . . . . .	135
5.5.3	Central results . . . . .	137
5.6	Summary of the results . . . . .	139
<b>6</b>	<b>Conclusion and outlook</b>	<b>140</b>
6.1	Key findings . . . . .	140
6.1.1	Identifier choice . . . . .	141
6.1.2	Parameter choice . . . . .	141
6.1.3	Limitations . . . . .	141
6.2	Open research questions . . . . .	142
6.3	Outlook and conclusion . . . . .	143
	<b>References</b>	<b>145</b>
	<b>Appendices</b>	<b>167</b>
	<b>A Source code overview</b>	<b>168</b>
	<b>B Source code used for (data) preparation</b>	<b>171</b>

<b>C</b>	<b>Source code of the model implementation and evaluation</b>	<b>196</b>
<b>D</b>	<b>Source code used for all result plots</b>	<b>248</b>
<b>E</b>	<b>Stand-alone Source code</b>	<b>279</b>
<b>F</b>	<b>Other Code</b>	<b>296</b>
F.1	Google Scholar results for ‘Big Data’ . . . . .	296
F.2	Database results for ‘Record Linkage’ in Medline and Sociological Abstracts . . . . .	297
F.3	Full R Version Info . . . . .	300

# List of Figures

1.1	Number of results for a database search for “Record Linkage” in two large databases by year. . . . .	5
1.2	Number of results for a Google Scholar search for “Big Data” by year. . . . .	9
1.3	The Record Linkage process, simplified and using different terminology, based on Christen (2012). . . . .	10
1.4	Relationships between true matching states and classifications.	24
1.5	Example of a small decision tree that could be built for the example link files shown in Table 1.4. . . . .	27
1.6	Example of a SVM with the examples from Table 1.4. . . . .	27
1.7	Example deep learning application for Record Linkage. . . . .	29
2.1	Part of a hashing machine, as patented by Schmidt/Klawitter (1927). . . . .	34
2.2	Illustration of a single round of the 16 rounds of the MD5 algorithm. Figure taken from Abualsaud et al. (2008: 293). . . . .	35
2.3	Aspects of Privacy-preserving Record Linkage. Image from Vatsalan et al. (2013: 953). . . . .	37
2.4	Design of linking hospital data using a trusted third party. Figure taken from Schnell (2016a). . . . .	38
2.5	Design of linking external data (here: cohort data) to a private mortality register using a trusted third party. Figure from Schnell/Borgs (2018b). . . . .	38
2.6	Excerpt of the original patent (Russel 1918: 1) which later became Soundex, including the phonetic code replacements for letters. . . . .	40
2.7	Hashing the unique PID 41234 using SHA-1 as an HMAC. . . . .	41
2.8	Basic ALC construction using the entry “James Sullivan, M, 20.1.1976” and SHA-1 as an HMAC. . . . .	41

2.9	Swiss ALC construction using the entry “James Sullivan, M, 20.1.1976”, Soundex and SHA-1 as an HMAC. . . . .	42
2.10	ESL construction using the entry “James Sullivan, M, 20.1.1976” and SHA-1 as an HMAC. . . . .	42
2.11	Bloom filter construction using the entry “James Sullivan, M, 20.1.1976”. The length of the filter is $l = 40$ bits. . . . .	43
3.1	Example of Bloom filter construction . . . . .	49
3.2	Another example of Bloom filter construction . . . . .	50
3.3	Hashing the unique PID 41234 using SHA-1 with the year of birth as a salt. . . . .	59
3.4	Linkage quality of BLIP/RAPPOR depending on the number of hash functions $k$ and the bit-flipping probability $f$ . Figure taken from Schnell/Borgs (2016b). . . . .	60
3.5	Rehashing using a sliding window of size $w = 8$ . . . . .	61
3.6	Four-time XOR folding of a bit vector of length $l = 16$ . Figure taken from Schnell/Borgs (2016c). . . . .	63
3.7	Subset of the transition graph for bigrams following TH. . . . .	64
3.8	Replacement rules for Rule 90. Figure taken from Schnell/Borgs (2018a). . . . .	65
3.9	Example Multibit tree construction. . . . .	68
3.10	Multibit trees tested against union bit trees, both with and without Symdex preprocessing. . . . .	70
3.11	Distribution of the mean prec./rec. when using $n = 30$ different passwords to hash the same data into Bloom filters using three different $k$ . The blue lines are the respective MPR means for each $k$ . . . . .	72
3.12	Q-Q plots of the mean prec./rec. when using $n = 30$ different passwords to hash the same data into Bloom filters using three different $k$ . . . . .	73
3.13	Different identifier sets used to encrypt CLKs can yield very different results in terms of linkage quality. . . . .	74
3.14	FEBRL-generated data and German telephone CD data encrypted with the same parameters yield different linkage quality based on their respective characteristics. . . . .	76
3.15	Number of hash functions used ( $k$ ) and the resulting percentage of bits set to 1 in a Bloom filter of length $l = 256$ . . . . .	78
3.16	Using synthetic data, percentage of bits set to 1 in a Bloom filter plotted against precision. The blue line is a loess smoother. . . . .	79

3.17	Using synthetic data, percentage of bits set to 1 in a Bloom filter plotted against recall. The blue line is a loess smoother. . . . .	80
3.18	Using synthetic data, percentage of bits set to 1 in a Bloom filter plotted against mean prec./rec.. The blue line is a loess smoother.	81
3.19	Using synthetic data, number of hash functions $k$ plotted against mean prec./rec.. The blue line is a loess smoother. . . . .	82
3.20	Percentage of bits set to 1 in a Bloom filter of length $l = 256$ and resulting mean prec./rec. for several datasets using Multibit trees with a Tanimoto threshold $T = 0.85$ . . . . .	83
3.21	Using synthetic data, number of hash functions $k$ plotted against mean prec./rec. for several datasets using Multibit trees with a Tanimoto threshold $T = 0.85$ . The blue line is a loess smoother.	83
4.1	Memory used (in MB) for linking files of different file sizes for $l = 250$ , $l = 500$ and $l = 1000$ . . . . .	88
4.2	Time taken (in minutes) when linking files of different file sizes for $l = 250$ , $l = 500$ and $l = 1000$ . . . . .	89
4.3	Bigram counts of last names of German (left) and Polish nationals (right) of a large administrative database. Note the skewed distribution with few very frequent bigrams. . . . .	99
4.4	Tanimoto treshold and time taken to link (in minutes) 1 million by 1 million records using Multibit trees in R. . . . .	103
4.5	Tanimoto thresholds and time taken to link (in minutes) records with different file sizes using Multibit trees in R. . . . .	103
4.6	Diagnostic plots for the simple linear model (LM). . . . .	106
4.7	Example response surface (a) and contour plot (b) for the second-order model given in Equation 4.16. Figure from Myers et al. (2016: 5). . . . .	111
4.8	Response surface plot and contour plot of the final RSM-based prediction of the training datasets. Note that all the values are standardised to be between 0 and 1, as this is required to fit the model appropriately (Myers et al. 2016). . . . .	113
4.9	Example of a single regression tree with the estimates for the mean prec./rec. that could be built from the the example training data shown in Table 4.13. . . . .	114
4.10	The Random Forest approach for a single input ( $x$ ), with one output ( $k$ ) and $B$ random decision trees. Figure by Verikas et al. (2016: 601). . . . .	115

4.11	Approximating a logarithmic transformation of uniform random numbers using a linear regression and Random Forest-based regression. . . . .	116
4.12	Variable importance for the Random Forest model. . . . .	117
5.1	Leaf limits and time taken to link $n = 100,000$ CLKs by Tanimoto threshold (0.8 to 1.0). . . . .	123
5.2	Surface plots for the FEBRL data and the NC voter data. Both show the Tanimoto threshold and the number of $k$ hash functions along with the resulting mean of precision and recall. . . . .	124
5.3	Diagnostic plots for the simple linear model (LM). . . . .	125
5.4	Diagnostic plots for the linear model based on variable selection by the stepAIC function using the BIC as the criterion (BIC). . . . .	126
5.5	Diagnostic plots for the response surface model (RSM). . . . .	127
5.6	Diagnostic plots of the Random Forest-based regression (RF) for each dataset and for all the settings of $q$ . . . . .	128
5.7	Predicted mean prec./rec. and actual mean prec./rec. for different $k$ with $q = 1$ for the NC Voter data using the BIC-selected linear model. Correlation coefficient is $r = 0.983$ . The dotted line represents $r = 1$ . . . . .	129
5.8	Predicted mean prec./rec. and actual mean prec./rec. for the NC Voter data using Random Forest-based regression for the four choices of $q$ . . . . .	130
5.9	Tanimoto threshold, parameter choice-method and resulting recall for the training data. . . . .	133
5.10	Tanimoto threshold, parameter choice method and resulting recall for the test data. . . . .	134
5.11	Tanimoto threshold, parameter choice method and resulting precision for the training data. . . . .	134
5.12	Tanimoto threshold, parameter choice method and resulting precision for the test data. . . . .	135
5.13	Result plot for the optimisation in terms of mean prec./rec. using the training data. As the models were trained with these datasets, better linkage quality was expected. . . . .	136
5.14	Result plot for the optimisation in terms of mean prec./rec. using the test data. . . . .	136
5.15	Mean of precision and recall for linking for several datasets with encryption parameters based on best-practice suggestions and different methods of choosing optimal parameters. All the results are shown for a Tanimoto threshold of 0.9. . . . .	138

# List of Tables

1.1	Example files with $n = 2$ records for linking. Without preprocessing, only one record would match. . . . .	11
1.2	Types of errors in data fields. Fraction for each error type of all the errors from the experiments by Damerau (1964) and Peterson (1986). . . . .	15
1.3	Example link file and resulting agreement patterns. . . . .	20
1.4	Example link file and resulting agreement states for DOB/sex and Jaro-Winkler similarities for names with a true match state (M is a match; NM a non-match) as an input into a decision tree. . . . .	26
3.1	Preprocessing used in the PPRL package and throughout this thesis. . . . .	53
3.2	Mean precision/recall and confidence intervals for three $k$ hash functions for linking Bloom filters ( $l = 500$ ) using the same data encrypted using different passwords $n = 30$ times. . . . .	71
3.3	Identifiers used for each parameter set presented in Figure 3.13. Table from Brown et al. (2017a). . . . .	75
4.1	Optimal identifier choice for all datasets used. . . . .	86
4.2	Some possible identifier combinations and their resulting agreement weights for the FEBRL WA data. . . . .	86
4.3	Variables with fixed values used for the encryption and testing of the training data. . . . .	88
4.4	Variables and their parameter space used for training data generation. . . . .	89
4.5	Replacement rules implemented in PPRL. . . . .	91

---

4.6	All the datasets used for either training the models (Training) or evaluating the models (Testing) against best-practice parameter settings, with their populations' main language and their data type (real-world or simulated). . . . .	92
4.7	Overview of error types and probabilities for each error type for the corrupted German telephone CD file. . . . .	93
4.8	Overview of error types and probabilities for each error type for the FEBRL-generated files. Note that there are more (unused) fields in the FEBRL generator, which is why the sum of the selection probabilities is not 100%. . . . .	95
4.9	Independent variables hypothesised to influence the predicted linkage quality (mean prec./rec.). . . . .	98
4.10	Properties of all datasets used for modelling . . . . .	101
4.11	Regression tables for the simple linear model (LM) and the BIC-selected linear model (BIC). Confidence intervals are shown in braces. . . . .	109
4.12	Regression table for the response surface method. . . . .	112
4.13	Example training data file that can be input into a regression tree. . . . .	114
4.14	Selected publications and parameters tested or suggested. . . . .	118
4.15	Optimal parameter choice estimations for all datasets as given by each model for a Bloom filter length of $l = 500$ bits. . . . .	120
5.1	All the training datasets used to train the models with their main language population and their data type (real-world or simulated). . . . .	122
5.2	All the test datasets used for evaluating the model against commonly suggested settings with their main language population and their data type (real-world or simulated). . . . .	122
5.3	Means, cross-validation-based coefficients of variation (CV), standard errors (SE) and RMSEs for the point estimates (MPR) for each model and dataset. . . . .	132
5.4	Mean resulting mean of precision and recall (MPR) by parameter choice method over all the datasets, as well as bootstrapped confidence intervals for the means. . . . .	137

# List of Source Code

A.1	Source code for the full replication, including local repositories.	169
B.1	Source code to generate the sampled FEBRL WA files. . . . .	171
B.2	Source code for the pre-evaluation of BF properties and all resulting plots. . . . .	172
B.3	Source code for the meta data. . . . .	187
C.1	Main code for evaluating the training data. Requires meta data source code to be run first. . . . .	196
C.2	Evaluating evaluation data set FEBRL WA for testing OPC. . . .	205
C.3	Evaluating evaluation data set NC Voter for testing OPC. . . .	214
C.4	Evaluating evaluation data set Mortality/Commercial for testing OPC. . . . .	222
C.5	Full evaluation routine for all models against best-practice solutions. Main evaluation routine. Required for plotting results. . .	231
D.1	Source code for the bigram frequency plot. . . . .	248
D.2	Source code for time/RAM plot by tree type and $l$ . . . . .	249
D.3	Source code for all result plots. . . . .	257
E.1	Standalone Source code for optimal identifier choice estimation.	279
E.2	Standalone Source code for optimal parameter choice estimation.	285
F.1	Source code for Google Scholar results for Big Data. . . . .	296
F.2	Source code for Record Linkage in Medline and Sociological Abstracts. . . . .	297
F.3	R version and package info for Windows. . . . .	300
F.4	R version and package info for Ubuntu. . . . .	302

# Terms and abbreviations

$T$	Tanimoto threshold $T$
$k$	Amount of hash functions used to hash elements into a Bloom filter
$l$	Bit length of a Bloom filter
$q$	Length of the subsets to split the input into
Accuracy	$ACC = \frac{TP+TN}{TP+TN+FP+FN}$
Mean prec./rec.	Mean of precision and recall
Precision	Precision = $\frac{TP}{TP+FP}$ , also known as Positive Predictive Value (PPV)
Recall	Recall = $\frac{TP}{TP+FN}$ , also known as Sensitivity or True Positive rate
Specificity	SPC = $\frac{TN}{TN+FP}$ , also known as True Negative rate
BF	Bloom filter
CLK	Cryptographic Longterm-key
EM	EM algorithm
EU	European Union
F-measure	F-Score, F-Measure, F1-Scores. Harmonic mean of Precision and Recall
FN	False negative
FP	False positive
MBT	Multibit tree
OPC	Optimal parameter choice
PID	Personal Identification Number

PM	Possible match
PPRL	Privacy-preserving Record Linkage
PRL	Probabilistic Record Linkage
RF	Random Forests
RL	Record Linkage
RS	Response surface
RSM	Response surface model
TP	True positive
US	United States of America

# Introduction

Quantitative social science has, for a long time, been primarily concerned with the use of survey-based or administrative data to find answers to empirical research questions. For researchers conducting such data gathering, (the field of *survey research*) Groves (2011) identifies three “eras” of research as follows: 1930–1960 as the “Era of Invention”, 1960–1990 as the “Era of Expansion” and the current “era”, 1990 onward. Since the current “era” began, survey response rates have been slowly declining (Groves 2011). To remedy this, new methods are needed to enrich data and compensate for the decline in response rates. Linking survey data to data from other sources and merging information from different data of several origins is sometimes seen as an answer to the declining survey response rates. Robert Groves, former director of the US Census Bureau, was quoted saying:

“Well, I guess I don’t have much hope for surveys. You guys are down to a nine percent response rate or something like that? I just don’t see anything that we have in our toolkit to overcome the massive social forces that are producing that behavior. [...] So I think the future is really messy. It’s piecing together a variety of data that are relevant to the phenomena we’re interested in. I think Partha and Hermann’s world of much more sophisticated statistical modeling and blending data together is the future of surveys” (Robert Groves, as cited in Habermann et al. (2017: 131)).

Additionally, the possibilities for research using novel data sources, such as electronic communication and social media, require new sets of methods (Chang et al. 2014). An example of new techniques spawning a new area of research is the field of *computational social sciences*, which gave rise to modern social

---

science techniques (Giles 2012). The term “big data” (which will be discussed in depth in Section 1.4) is closely related to this research area, usually implying that the amount and the complexity of the data used in this research is more extensive than in the “conventional” social sciences. Some innovations in the field are large-scale social network analyses and a range of methods to work with, process and enrich massive data (Foster et al. 2017).

“Enriching” data usually means combining data sources in this context. In the best case, this is a simple merging operation using uniquely identifying information, such as questionnaire IDs or social security numbers. For many applications, such information is unavailable for a variety of reasons, including legal limits on the use of such information for linking data, the unstructured nature of many datasets used for linking, the complete absence of any identifying data because of differing requirements for the data collection, or merely the presence of erroneous information in the identifying record fields. For each of these situations, special techniques from the field of “Record Linkage”<sup>1</sup> will have to be applied.

The resulting linked data sets are a valuable tool for many research fields, not only for sociology or the social sciences but also for adjacent research fields: For example, they are used to enrich the data on criminal cases, research and anti-terrorism investigations (Woodhams/Bennell 2015). For the social sciences, there are several areas where data linkage is already being conducted routinely. A widespread solution is to link survey data with administrative data (Bender et al. 2010). In Germany, for example, the research division of the German Federal Labour Institute (IAB<sup>2</sup>) linked data from an employee study to the employer data from their administrative registers to create the SOEP-LEE dataset (Weinhardt et al. 2017). This allows for a broader set of research questions to be investigated. According to Schnell (2016b), further applications for social scientists and survey researchers include building sampling frames, over- and undercount estimations, linking surveys with administrative data to validate responses and retrospective panel construction.

The most extensive linkage application in Germany thus far was the German Census of 2011 (Reisch 2012; Statistisches Bundesamt 2015; Münnich et al. 2011), where local population registers were linked to efficiently estimate the German population count (approximately 82 million) without surveying the

---

<sup>1</sup>For an overview of some history of linking data in general, see Fienberg (2013).

<sup>2</sup>See <https://iab.de> (Last accessed 05.04.2019).

entire populace. Another example is the grant-funded German Record Linkage Center (GRLC<sup>3</sup>), which has linked several studies with administrative data for many different research purposes (Antoni/Schnell 2017).

In other countries, broader applications of linking data can be found. An example in the field of political science would be a study in the United States of America by Hamilton/Stampone (2013), where the location of the survey respondents' residences and the interview date were linked to the local weather records. Thus, Hamilton/Stampone (2013) found evidence that politically independent voters based their acceptance of climate change as a fact on the relative difference between the temperature at the date of the interview and the 30-year running mean temperature for this date. Cooler days than the average led to less acceptance of climate change and vice versa. For all other voting groups, the acceptance of climate change was explained purely by party affiliation (Democrats mostly accepted climate change irrespective of the weather, while only 1 in 5 Republicans did so, irrespective of the weather).

Australia is another strong example: According to the *International Population Data Linkage Network* (IPDLN<sup>4</sup>), which is a worldwide organisation for connecting institutions conducting linkage operations with population data, Australia leads the world in terms of the number of data linkage centres. The IPDLN also includes national initiatives such as the Canadian "Longitudinal Health and Administrative Data Initiative".<sup>5</sup> An overview of such data linkage initiatives can be found in Doiron et al. (2013).

To give an example of research that is possible by linking data, Soloff et al. (2007) provided an overview of an application of linking children's data from a longitudinal study ("Growing up in Australia") to healthcare and educational records. This has enabled social scientists to use medical (such as illnesses) and educational control variables (such as the highest educational achievements or repeated classes), as well as studying the effect of social factors on health. As in Australia, in the Scandinavian countries<sup>6</sup>, linking and maintaining national registers for (particularly health) science is the norm and is widely regarded as useful:

---

<sup>3</sup>See <http://record-linkage.de> (Last accessed 23.04.2019).

<sup>4</sup>See <https://www.ipdln.org/> (Last accessed: 07.04.2019).

<sup>5</sup>Information about the initiative is archived under <https://www150.statcan.gc.ca/n1/pub/82-622-x/2010004/intro-eng.htm#archived> (Last accessed: 01.04.2019).

<sup>6</sup>Usually including Norway, Sweden, Denmark and Finland; sometimes, Iceland.

“[...] matching of registers through [Personal Identification Number] and matching of national health registers without the explicit approval of the individual patient is to the benefit for both the individual patient and for society” (Ludvigsson et al. 2009: 659).

In a stark contrast, in Germany, not a single national database is available for data linkage for research. This has led to fragmented databases scattered across jurisdictions and topics. In 2017, the German Federal Office for Statistics identified 214 separate national registers (Statistisches Bundesamt 2017: 4). Not one of these can be linked across topical fields using a unique ID. Of course, linking some databases is – in theory – possible, if some unique identifiers are contained in both the registries. A social security number will be contained in some databases but will be missing in all the databases unrelated to social security or income-related records.<sup>7</sup> This is unlike the Scandinavian registers, which usually include a unique national personal identification number (PID). Such a unique PID has been forbidden by the German Supreme Court since the ruling known as the “Volkszählungsurteil” (Bundesverfassungsgericht 1983). This jurisdiction is believed to still be heavily influenced by the totalitarian past of the country (Martini et al. 2017). Currently (as of 2019), a unique PID is still forbidden in Germany.

If not for legal problems, the theoretical range of applications for linking records is manifold: Christen (2012: 11–20) listed censuses, the health sector, national security, crime and fraud detection and prevention, business mailing/client lists, bibliographic databases, online shopping, social sciences and genealogy as active fields of research and private-sector applications of data linkage.

Despite the potential appeal to social science researchers, the primary research field using data linkage remains the health sector. Epidemiologists can use linked data to assess the influence of socio-economic factors on health indicators. Grundy/Tomassini (2005) used data linkage to add socio-demographic indicators to a longitudinal study in the UK to evaluate mortality and child birth patterns. Without the linked data, they could not have controlled the effects of economic or social factors.

This<sup>8</sup> is why linkage applications are predominantly found in the health sciences. Comparing the results for the search term “Record Linkage” since the

---

<sup>7</sup>Its use will also not be allowed for linking data outside of a federal agency–internal linkage.

<sup>8</sup>And because it is likely easier to justify a public interest in medical research when dealing with data protection officers or governmental bodies.

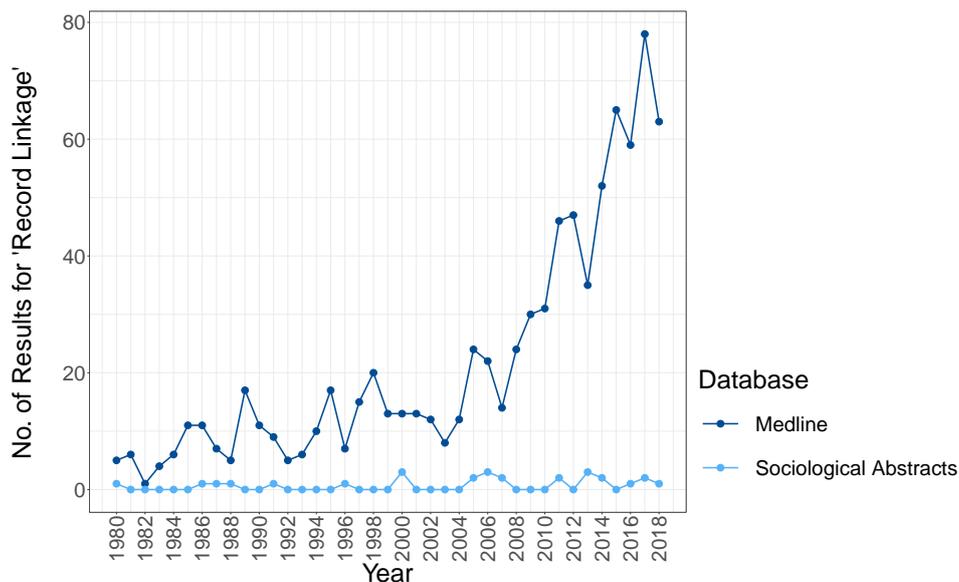


Figure 1.1: Number of results for a database search for “Record Linkage” in two large databases by year. Figure reproduced with new numbers, originally created by Rainer Schnell and used for several presentations, e.g. Schnell (2017b).

1980s in a large sociological database with the same query in a medical science database shows a striking difference (see Figure 1.1<sup>9</sup>). While the number of publications using Record Linkage is rising steadily in the Medline database, Sociological Abstracts has listed only a handful of publications every few years for the last 28 years.

In particular, for the health sciences, linking information helps to treat people better, through cancer research and improvements in understanding individual risk factors leading to undesirable outcomes (e.g. obesity, smoking, heart attacks or type-II diabetes.<sup>10</sup>).

There is also an active commercial interest in linked data. Data-driven services have generated substantial revenues in the recent years. In contrast, data has been described as the “new oil”<sup>11</sup> (Marr 2018). For example, data relating to preferences of individuals are used to drive targeted advertising, generating as much as 110.9 billion dollars in the 2017 revenue<sup>12</sup> for one of the biggest ad-related data holders on the planet (as of 2019), Google. Google operates

<sup>9</sup>For a description of the queries and the source code, see Appendix F.2.

<sup>10</sup>For further examples, see Christen (2012).

<sup>11</sup>The term is believed to be coined by Clive Humby. The analogy itself is weak; for some of the criticism, see Marr (2018).

<sup>12</sup>For Google’s mother company Alphabet. Source: [https://abc.xyz/investor/pdf/2017Q4\\_alphabet\\_earnings\\_release.pdf](https://abc.xyz/investor/pdf/2017Q4_alphabet_earnings_release.pdf) (Last accessed: 20.04.2019).

by linking several information sources on individuals to compile databases relevant for selling targeted advertising.

Outside of commercial data aggregation, linking information usually requires the legal permission of the data holding parties, or even a central register available for research. For countries without a unique PID usable for linking registries, other personal information has to be used for linking, such as names or birth dates. As these personal identifiers are usually considered to be sensitive private information, linking them without a form of encryption (more on encryptions in general will be discussed in Section 2.2) is often not allowed (as, for example, in Europe (Council of the European Union 2016)). Record Linkage using encrypted identifiers is called *Privacy-preserving Record Linkage* (PPRL) and is a very active research field (for an in-depth overview, refer to Chapter 2). One relatively new and prominent PPRL technique used for encrypting and linking private information is the use of Bloom filters (Schnell et al. 2009; for details, see Chapter 3). A major drawback of all the PPRL methods is the impossibility of checking the quality of the results after linkage, as all the identifiers used for linking will be encrypted to preserve privacy. Thus, choosing the right encryption parameters is of utmost importance to ensure a successful linking operation. This will be the major focus of this work.

## 1.1 Thesis structure

This chapter will provide an overview of the definition of Record Linkage in general, the other terms often confused with it and the Record Linkage process.

Chapter 2 will provide an overview of the broad topic of the thesis – Privacy-preserving Record Linkage (PPRL): its definition and legal scope, an overview of current methods used in practice and private blocking. As Bloom filter-based methods are the focus of this contribution, they are explained in detail in Chapter 3. All the current attacks, hardening methods and further improvements are discussed in detail.

With respect to linking Bloom filter-based methods, the resulting linkage quality is highly dependent on the selection of the encryption parameters and of the identifiers used. Recent work by Brown et al. (2017b) using Bloom filters with several identifier choices used for the same linkage showed substantial differences in the linkage quality. Thus far, to the best of my knowledge, no

study on the optimal choice of these parameters has been published.

This is no flaw of the encryption method itself, but showcases the paramount importance of proper identifier and parameter choice for Bloom filter-based Record Linkage. Reasonable choices regarding identifiers used and parameters chosen for the Bloom filters are necessary requirements before linking can achieve high-quality results. This thesis aims at providing insight into this fundamental problem and offers approaches to solve it. An automation method will be provided to find the optimal parameter and identifier choice resulting in the best possible linkage quality using predictive models based on training data covering all the possible parameters. This will be tackled in Chapter 4. After describing the test setup and the details of the implementations as well as the results of the implementation for the optimal identifier choice, Chapter 5 will provide an overview of the simulation results, model estimates and their results using optimal parameter choices against current best-practice recommendations.

Finally, Chapter 6 will discuss these findings, drawbacks, open questions and further research needed.

All the code needed to reproduce all the relevant plots, results and evaluations can be found in the Appendix.

## 1.2 The term ‘Record Linkage’

In this thesis, *Record Linkage* will refer to finding the data representations of the same real-world entities in at least two databases. The goal is to link two files in a way that finds the true overlap of all the entities represented in these databases. Acronyms for Record Linkage are plentiful: According to Schnell (2016b), all combinations of the words “record, name, entity [and] identity” (Schnell 2016b: 662) with the words “resolution, detection, linkage, deduplication, matching [and] identification” (Schnell 2016b: 662) are in use. In the computer sciences, the terms “data matching” (Christen 2012; Scannapieco et al. 2007) and “entity resolution” (Christen 2012; Kolb et al. 2012) are commonplace, while in health and social sciences (particularly in survey methodology), “Record Linkage” is common (Stausberg et al. 2017; Tokle/Bender 2017; Schnell 2019).

## 1.3 Techniques often confused with Record Linkage

There are several other terms which are regularly confused with Record Linkage but refer to different techniques. The following set of definitions is loosely based on Schnell (2017a):

<i>Statistical matching</i>	Aims to find statistically similar pairs of records (Rässler 2002). Statistical matching is used in practice when the “true” overlap of two data sources is either not available or is deemed to be very small. It requires both data sources to contain discriminatory variables (such as income and employment variables on purchasing patterns or age). In contrast to Record Linkage, there is often no attempt to find “real” pairs but to generate “similar” pairs, which will then contain many (imputed) values with variables from both datasets.
<i>Data Fusion</i>	Often erroneously confused with statistical matching. It usually denotes using multiple sensors for the same measurements. Here, the goal is to merge the (sensor) data of a single measurement in a way that gives a robust estimator (Hall/Llinas 1997).
<i>Appending</i>	Adding additional cases to existing data. No Record Linkage.
<i>Schema matching</i>	Usually a form of appending. The goal is to merge different data sources with different methods of data representation or storage (differences in date formats, encodings, column names, etc.). It is particularly widely used in the field of (German) health informatics. For an exhaustive overview, see Bellahsene et al. (2011).
<i>Merging/joining</i>	Combining different types of information about the same entities using a unique key, such as a PID. The key can be unique in both datasets A and B (1:1-merge) or have duplicates in A (m:1-merge), B (1:m-merge) or both datasets (m:m-merge). This is usually the step taken after Record Linkage or when Record Linkage techniques are not needed.
<i>Table lookups</i>	Same as merging. This usually involves extracting records from a (relational) database which agrees on a unique key. Therefore, it is a special case of implicit merging, although syntactically different.
<i>Duplicate detection</i>	Finding duplicate entries/lines/entities/IDs in a single dataset. A special case of Record Linkage, where a file is linked with itself.
<i>Propensity score matching</i>	In the absence of randomised control and treatment groups, the output of a (usually logistic) regression model based on respondent characteristics is used to group patients into artificial control or treatment groups (Rosenbaum/Rubin 1985). No Record Linkage.

## 1.4 'Big Data' and Record Linkage

As is evident from the amount of research published, “big data” has become an all-encompassing term that has gained popularity over the last few years. Figure 1.2 shows the number of hits for the term over the last few years since 2000.<sup>13</sup> While the term was used sparingly in the 2000s, its popularity started to increase after 2012.

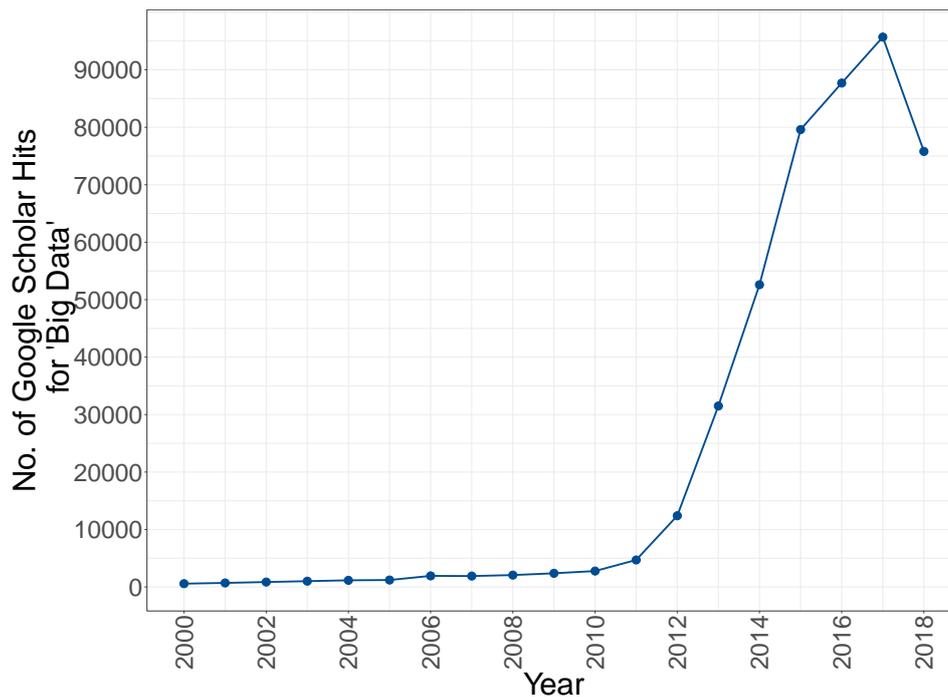


Figure 1.2: Number of results for a Google Scholar search for “Big Data” by year.

There is no formal definition of the term, but it usually refers to very large, sometimes unstructured data not primarily intended for research. Most of the time, “big data” is useless for most broader research topics, unless it is linked to other data sources (Schnell 2018). Big data sources require Record Linkage to enable any potential research, furthering the importance of data linkage. However, most “big data” sources lack the information needed to link them.

An example would be the data recorded from cell phone masts. These data cannot be linked to personal information unless all unique media access control (MAC) addresses of the persons to link them to are known. MAC addresses, also known as hardware addresses, are unique identifiers for every type of hardware

<sup>13</sup>For further details and plot code, see Appendix F.1.

involved in network operations, such as network cards, routers or cell phones. Blocks of MAC addresses are given to clients by the IEEE Registration Authority, where a list of addresses assigned to companies can be viewed.<sup>14</sup> Every piece of hardware communicating in networks must have a unique MAC address. Without knowing all the MAC addresses of all the devices of a person, this very large database is not suitable for linkage or any research requiring control variables that are, for example, socio-economic in nature.

However, for some rare applications, linking such datasets is indeed possible. One such example is a project where road sensor data in the Netherlands were linked to transportation survey data using the license plates captured by roadside cameras (Klingwort et al. 2018). The resulting road sensor data were then used to weight the survey estimates. Such applications, although rare, will possibly be more important in the future of many research fields (Habermann et al. 2017).

## 1.5 The Record linkage process

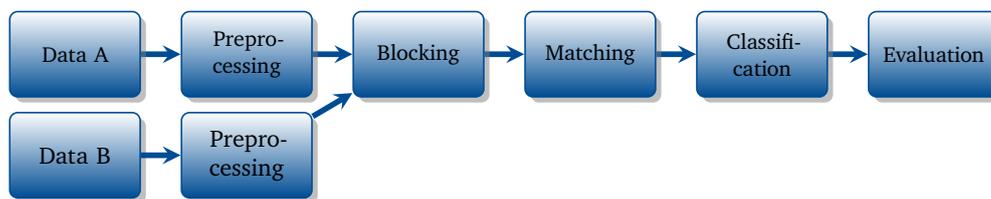


Figure 1.3: The Record Linkage process, simplified and using different terminology, based on Christen (2012).

An in-depth overview of the Record Linkage process can be found in Christen (2012). Figure 1.3 shows the (simplified) workflow for most Record Linkage projects linking two databases (A and B). The following section will discuss each of these steps in more detail.

### 1.5.1 Data preprocessing

Real-world data from separate data holders will usually use different ways to store the same information. For example, the way to save a date of birth will

<sup>14</sup>See <https://regauth. standards. ieee. org/ standards- ra- web/ pub/ view. html> (Last accessed: 01.04.2019).

Table 1.1: Example files with  $n = 2$  records for linking. Without preprocessing, only one record would match.

File A			
first	last	dob	sex
Peter	Jörns	01/20/1987	male
Sarah	Peters	01/01/1946	female

File B			
VN	LN	DOB	GENDER
PETER	JOERNS	20.1.1987	m
SAHRA	PETERS	1.1.1946	f

be different between countries, as the placing and separation of year, month and days varies across countries. Another problem arises when months are not stored using their numeric values, but stored using their names (or even abbreviations) in the desired language. Years of birth are sometimes stored using just the last two digits, omitting the century. For this reason alone, the so-called *preprocessing* of data before linkage is an invaluable part of every Record Linkage application.

Cleaning the data of inconsistencies, foreign character sets, erroneous characters through encoding errors, implausible values and standardising variables (e.g. the way addresses or dates are represented in the data) is a part of this step. Table 1.1 gives an example of two files to be linked exhibiting different representations of the same real-world entities, i.e. persons. Several problems present themselves: An obvious problem is differing column names, which can be remedied by merely coherently renaming them. Date and sex representations differ, requiring a standardisation of structure and formats (Christen 2012). File A has a record with special (non-ASCII) characters in it (“ö” instead of “oe”), requiring harmonisation, and finally, a *potential* error in the name Sahra (it could be “Sarah”, but it could be a correct (Hebrew) variant of the name) presents itself.

According to Christen (2012), there are four steps necessary in every preprocessing operation:

1. Remove unwanted characters and words.
2. Expand abbreviations and correct misspellings.

3. Segment attributes into well-defined and consistent output attributes.
4. Verify the correctness of attribute values.

Steps 1 and 2 will be covered in Section 3.1.2, while errors in data in general will be covered in Section 1.5.5.2. Removing unwanted characters usually consists of a set of replacement rules for common errors. Further examples of removing unwanted characters include additional string delimiters (e.g. “), removing tabs and removing unnecessary control operators (e.g. extra blank lines). Alternative approaches have been suggested: both Damerau (1964) and Pollock/Zamora (1984) proposed an algorithm generating a simplified key, which is then used to correct the misspelt words. Christen (2008) proposed the use of a Hidden Markov-model approach to correct misspellings.

Standardising date formats is part of step 3: For example, in the United States, the date format is MDY (month, day, year), while in most<sup>15</sup> of the rest of the world, day-month-year (DMY) prevails. Some countries (e.g. China) even follow the YMD format. In addition to the ordering of the three identifiers, separation and representation can differ. Months can be represented numerically (as in 1–12), verbose (e.g. April) or abbreviated (e.g. Apr. or apr. or APR). Most preprocessing routines will parse dates into separate, standardised fields.

Finally, step 4 can only be executed if a reference database is available. This is particularly important for address data that can be validated this way.

## 1.5.2 Choice of identifiers

Naturally, the choice of Record Linkage variables is limited to the common set of identifiers available in both datasets. Other considerations include the number of missing values in both datasets: if there are very few entries, the variable will not be a good choice. The amount of information gained by including the variable to identify an entity in the data is essential as well: an address, for example, is a better variable to discriminate entities than, say, the sex of a target person. Finally, the number of errors that can be expected (address fields, for example, can change over time, leading to missed links) is very important as well. Identifier choice and the best way to choose identifiers will be discussed in Section 4.2.

---

<sup>15</sup>See [https://en.wikipedia.org/wiki/Date\\_format\\_by\\_country](https://en.wikipedia.org/wiki/Date_format_by_country) (Last accessed: 09.02.2019).

### 1.5.3 Deduplication

A special case of Record Linkage is deduplication, which is the process of finding duplicate representations of the same entity in a set of data by matching the data file with itself. Multiple matches for a single representation then indicate duplicates in the data. As duplicates add no information to the data but enlarge the size of a database, they are routinely removed before the data is merged or blocked as a part of the preprocessing.

### 1.5.4 Blocking

Record Linkage is computationally expensive. As the number of pairs to compare is the Cartesian product of the number of records in both files ( $N = n_A \cdot n_B$ ), the computational effort required increases exponentially with the number of records in the files to compare. At some point, the computational time needed to match two records becomes very high, making a full linkage infeasible. To counteract this, all the records are grouped according to a variable – this is called *Blocking* or *Indexing* (Schnell 2016b). To match the datasets, only the records agreeing on the so-called *blocking variable* will be compared. This considerably reduces the number of candidate pairs to compare. However, according to Christen (2012), there are several things to consider when selecting a blocking variable:

- A missing value in any record will lead to a missed pair. Therefore, the blocking variable should have as few missing values as possible in both the datasets. For a recent publication concerned with the effect of missing values on blocking, see Anindya et al. (2019).
- The variable used should be as error-free as possible, as disagreements between two records that refer to the same entity will lead to a missed pair.
- Ideally, the variable should divide the data into sufficiently heterogeneous categories to effectively reduce the number of candidate pairs for the linkage.

The records agreeing on the blocking variable should then be compared using the matching algorithms chosen by the user. The creation of blocks will always have to balance the “trade-off between the number and the size of

blocks” (Christen 2012: 72): Blocks with a substantial number of records in each block will yield a lower reduction in computational time than not using blocking at all, while blocks with very few record pairs will inevitably miss true matches disagreeing on the blocking variable. This would be particularly true for blocks based on numeric values (such as birth weights). Applications with large amounts of records will use blocking. One of the largest (privacy-preserving) application linked 114 million records to around 370,000 health records in 9 days by using a supercomputer (Dantas Pita et al. 2018). The blocking used by Dantas Pita et al. (2018) included names, mother’s names and a municipality code, leading to comparatively small blocks. Without blocking (and the use of a special multi-threading software on a supercomputer), this operation would have been computationally infeasible.

Often, identifiers such as zip- or postcodes are used. However, another option is to use phonetic codes to improve the error-tolerance while creating sufficiently small blocks (Christen 2012). For a discussion on phonetic codes, see Section 2.3.4.

## 1.5.5 Matching algorithms

To compare candidate record pairs to classify them as matches or non-matches, several methods can be used. An overview can be found in Christen (2012) or Vatsalan et al. (2013), on which the following list is based.

### 1.5.5.1 Exact matching

The most basic variant is to use exact matching, also known as merging (see Section 1.3). Only record pairs that agree exactly on every variable used for the exact linkage are classified as matches (Herzog et al. 2007). This variant can be used in every software for analysis, linkage or programming and is synonymous to the merge command in R and Stata, Python’s `pandas.merge` and the join command in SQL, as well as the `MATCH FILES` command in SPSS. Merging files can be done natively on Unix-based systems by using the join macro as well.

Exact matching is not error-tolerant, as a simple deviation in any of the variables used will result in a misclassification (a *false negative*, see Section 1.5.6). Most datasets *will* contain errors, which is why exact matching is a

poor choice when linking using all but unique personal identifiers such as questionnaire IDs.

### 1.5.5.2 Errors in data

“Real-world data is dirty” is the title of the paper by Hernandez/Stolfo (1998), highlighting the problem of errors in the data as the primary source of missed linking pairs (particularly when using exact matching only). Therefore, according to Winkler (2009), preprocessing is one of the most vital steps to improve linkage quality. However, a closer look at the types of errors is warranted, because even though the central role of preprocessing is to reduce avoidable errors, most preprocessing will not be able to remedy more than simple typos. Winkler (2009) reports that for many applications, exact matching misses many correct links. Increasing non-linked pairs may introduce bias for medical settings (Harron et al. 2014). However, bias introduced by Record Linkage is a sparsely researched topic (Bohensky 2016). Without error-tolerant linkage methods, low linkage quality is certain and linkage-induced bias a real possibility.

Table 1.2: Types of errors in data fields. Fraction for each error type of all the errors from the experiments by Damerau (1964) and Peterson (1986).

File A – Error-free				
first	last			
Sarah	Peter			
File B – With errors				
first	last	Error type	% of errors	
			Damerau 1964	Peterson 1986
Sarah	Petet	Substitution	59%	40.0%
Sarha	Peter	Transpositions	2%	2.6%
Sara	Peter	Deletions	16%	31.6%
Sarah	Petter	Insertions	10%	18.7%
Sara	Petter	Multiple errors	13%	7.1%
Sarah	Peter	OCR error	<i>special case of a substitution error</i>	

In this light, special consideration has to be given to misspellings. Damerau (1964), a researcher for IBM, reported that most errors are single characters that are either wrong (substitutions), transposed (transpositions), omitted (deletions) or added (insertions). These errors comprised a majority of his

reported results. From a sample of 964 spelling errors collected at IBM, only 12.7% exhibited multiple errors. Table 1.2 gives an example for each error type with the percentages of errors reported by Damerau (1964) and a second experiment by Peterson (1986), where students were asked to retype the “word division list” of the U.S. Government Printing Office (they made a total of 155 errors). Here too, single errors were the most prominent error type, with substitution errors making up the biggest proportion.

In a large evaluation of several databases (with  $n = 52,963$  misspellings) as part of their introduction to the SPEEDCOP function, Pollock/Zamora (1984) reported further details on errors. They too reported that the overwhelming majority of errors are single errors. Among the optical recognition of scanned documents (OCR) errors, transposition errors are reported to be the most common type of spelling error. They also report that a typo in the first letter is less likely (7.8%) than typos at the other letter positions.

### 1.5.5.3 Rule-based matching

To counteract the weaknesses of exact matching, *rule-based* or *deterministic* matching can be used. As the name suggests, the general idea of exact matching is extended by introducing a set of rules that have to be fulfilled to classify a record pair as a match. An example would be to classify a pair of records as a match even if either the day or month of birth does not match. If the rules are carefully selected, the number of pairs missed by a simple exact matching algorithm can be reduced. If multiple identifiers are erroneous but very similar, as with small typing errors in the first and last name, rule-based matching can still perform poorly as compared to more sophisticated methods. This is why for most real-world applications, matching variables include phonetic codes of names (see Section 2.3.4), to achieve improved error-tolerance.

### 1.5.5.4 Probabilistic matching

Probabilistic Record Linkage (PRL) consists of a set of steps to implement an optimal linkage solution which enables linking to be error-tolerant and gives weight to the relative importance of the identifiers. Linkage approaches using a probabilistic linkage model do not depend on fixed rules or exact matches.

The first heuristics were devised by Newcombe et al. (1959), based on which Fellegi/Sunter (1969) formalised a Record Linkage model. Later work by Winkler (1988) and Jaro (1989) extended the standard model, using an automated approach to estimate optimal parameters for the linkage model, using the expectation-maximisation (EM) algorithm. Both the original model and the EM algorithm will be described in detail, as they are of central importance for both automated identifier choice (see Section 4.2) and optimal parameter choice (see Section 4.4.1).

### *Assumptions and preliminaries*

All the notations closely follow the slightly modified notations by Winkler (1988: 3ff), which are close to the original formulation (Fellegi/Sunter 1969).

The Fellegi-Sunter (FS) model is based on the assumption of the *conditional independence* of identifiers: The similarity of an identifier is independent of the similarity of the other identifiers. For example, the first name similarity of a random record pair is assumed to be independent of the last name similarity. Of course, this assumption is usually wrong: names influence many covariates, as a name can tell a lot about ethnicity, socio-economic status or religion (Mateos 2014: 2ff). Another example is the month and day of birth, where persons birth days when born in February are restricted to be between the 1st and 29th. In this case, the probability of having specific day of birth is dependent on the value of the month of birth. The same is true for first names and gender, for example. This is why extensions have been developed that allow for relaxing the assumption by introducing conditional dependence between identifiers (Schürle 2005). Despite the drawback of the violation of the independence assumption, the “classic” FS model has been widely used with great success (Christen 2012).

As a preliminary, let  $A$  and  $B$  denote two populations with their elements  $a$  and  $b$ . The central assumption is that there is an overlap between  $A$  and  $B$ . The set of all possible record pairs is given as follows:

$$A \times B = (a, b) \text{ where } a \in A, b \in B. \quad (1.1)$$

All the possible pairs are a disjoint set of either matching pairs:

$$M = (a, b); a = b \text{ where } a \in A, b \in B. \quad (1.2)$$

or non-matching (“unmatched”) pairs:

$$U = (a, b); a \neq b \text{ where } a \in A, b \in B. \quad (1.3)$$

For each pair of elements in both records  $a_i, b_i$ , they can either be part of  $M$  or  $U$ . However, as an identifier can be erroneous, a record pair can be a matching pair and not agree on the identifier (this concept will be discussed in more detail in Section 1.5.7). This is usually attributed to errors or missing values in the data.

As not all identifiers are perfectly discriminating for all records, unmatched pairs can have the same value for an identifier, despite not referring to the same entity. This is particularly true for identifiers with few possible values, such as gender.

Therefore, the probability that matched record pairs have the same value (index  $A = \text{“Agree”}$ ) can be expressed as follows:

$$m_A = P(a_i = b_i | (a, b) \in M). \quad (1.4)$$

Furthermore, matched pairs not having the same value (index  $D = \text{“Disagree”}$ ) can be calculated as:

$$m_D = P(a_i \neq b_i | (a, b) \in M). \quad (1.5)$$

For randomly chosen unmatched pairs, they can either agree on an identifier:

$$u_A = P(a_i = b_i | (a, b) \in U), \quad (1.6)$$

or disagree:

$$u_D = P(a_i \neq b_i | (a, b) \in U). \quad (1.7)$$

For example, using the month of birth as an identifier will yield 12 possible values. Given two random record pairs, disagreeing unmatched pairs are expected 11 out of 12 times, i.e.  $u_D = 11/12 = 0.917$ . Unmatched pairs can have the same value for the date of the month as in one of the 12 cases, the same value

is to be expected. Therefore,  $u_A = 1/12 = 0.083$ . Assuming an error rate of 5% and no missing values, the probability that matched records have the same value will be 95%, while (because of the errors) 5% of the matched records will not have the same value. This gives  $m_A = 0.95$  and  $m_D = 0.05$ , which is the expected error rate.

These probabilities are then used to weight the identifier fields with respect to their  $m$ - and  $u$ -probabilities<sup>16</sup>; this is known as *field weighting*.

### *Field weighting*

Using the  $m$ - and  $u$ -probabilities, we can calculate the agreement and disagreement weights of an identifier. They are the likelihood ratio of a match or non-match given the identifier match status ( $\gamma_i = 0$  (non-match) or  $\gamma_i = 1$  (match)). They are calculated as follows (Winkler 1988: 3ff):

$$\text{agreement weight} = \log_2 \left( \frac{m_A}{u_A} \right), \quad (1.8)$$

and

$$\text{disagreement weight} = \log_2 \left( \frac{m_D}{u_D} \right). \quad (1.9)$$

The agreement weight is used if the identifiers of two record pairs match ( $\gamma_i = 1$ ); the disagreement weight is used otherwise ( $\gamma_i = 0$ ).

Field weighting must be separated from “frequency weighting”, which is another form of weighting in the PRL framework. In the frequency weighting scheme, the agreement/disagreement weights are not based on the relative discriminatory power of the identifiers, but of the uniqueness of the identifier value itself: “Smith” is a considerably more common name than, say “Ottovordemgentschenfelde”.<sup>17</sup> As names (and rare names particularly) tell a lot about a person (Lasker 1985), matching rare values will gain a higher weight than common values that match. This way, the weights are calculated for each record separately.

The calculation of agreement/disagreement weights requires knowledge about the conditional probabilities of the identifier agreement of matching/non-

<sup>16</sup>For the rest of this thesis,  $m$  will refer to  $m_A$  for simplicity, while  $u$  will refer to  $u_A$ .

<sup>17</sup>This is the longest German last name, which is currently the last name of only a single person <https://www.telegraph.co.uk/news/worldnews/europe/germany/8117697/Bernd-Ottovordemgentschenfelde-the-man-with-the-longest-name-in-Germany.html> (Last accessed: 11.02.2019).

matching pairs. This can only be done if the true matching status is known. In the case of operating without a gold standard, the EM algorithm can be used.

### *The EM algorithm*

The expectation and maximisation (EM) algorithm is an iterative process trying to estimate the maximum likelihood parameters for latent classes given an empirical distribution (Dempster et al. 1977). It was first adapted for probabilistic Record Linkage by the Record Linkage Staff of the Statistical Research Division of the U.S. Bureau of the Census, where the first publications of the algorithm were the papers by Winkler (1988) and Jaro (1989).

As the name suggests, it is based on two steps: the expectation step and the maximisation step. For the first step, agreement patterns in the data are counted first. Following the idea from Enamorado et al. (2018b), Table 1.3 gives an example of such patterns.

Table 1.3: Example link file and resulting agreement patterns.

File A				File B			
first	last	dob	sex	first	last	dob	sex
PETER	JOERNS	20.01.1987	m	PETER	JOERNS	20.01.1987	m
JAMES	JONES	21.12.1970	m	JAMES	JONES	21.12.1970	m
SARAH	PETERS	01.01.1946	f	SAHRA	PETERS	01.01.1946	f
WOLF	YELLOW	11.11.2010	m	PETE	BLACK	20.02.2019	m

Agreement patterns				
first	last	dob	sex	$n_j$
1	1	1	1	2
0	1	1	1	1
0	0	0	1	1

Following Yancey (2002), these counts will be denominated as  $n_j$ , while all the counts  $C_i$  are part of the full count data  $C$ . The probability distribution given by the pattern counts is the basis on which the EM algorithm will estimate the underlying latent classes, which are the  $m$ - and  $u$ -probabilities in our case. In other words, the likelihood that a match or a non-match results given an agreement pattern probability distribution is estimated as a likelihood given the underlying data. Under the assumption that agreement patterns are binary ( $\gamma = 1$  for agreement and  $\gamma = 0$  for disagreement), there are  $2^k$  agreement patterns, where  $k$  is the number of fields considered.

According to Yancey (2002), the complete data likelihood can then be expressed as follows:

$$L = \prod_{(a,b)} \Pr(\gamma(a,b)) = \prod_{j=1}^{2^k} (\Pr(\gamma^j))^{n_j}, \quad (1.10)$$

where  $(a,b)$  are the pairwise comparisons of the elements of the data sets  $A$  and  $B$  and  $n_j$  are the agreement pattern counts.

To estimate  $m$  and  $u$ , initial values are set by the researchers (in the original implementation (Dempster et al. 1977), the value is set randomly), as, for example, in the linkage program MergeToolbox (Schnell et al. 2005). This should speed up the convergence, as the distance to the optimal solution is minimised.

Following Yancey (2002: Appendix A), the E step estimates the underlying data values ( $\hat{z}_{ij}$ ) given the frequency counts  $C_i$  and the probability of a pattern  $\gamma_j$  to be either coming from the latent class of matches  $M$  or unmatched pairs  $U$ :

$$\hat{z}_{ij} = \frac{\Pr(\gamma^j|C_i)\Pr(C_i)}{\Pr(\gamma^j|M)\Pr(M) + \Pr(\gamma^j|U)\Pr(U)}. \quad (1.11)$$

The M step then attempts to maximise the complete data log likelihood as follows (Yancey 2002):

$$\Pr(C_i) = \frac{1}{|C_i|} \sum_{j=1}^{2^k} n_j \hat{z}_{ij}, \quad (1.12)$$

$$\Pr(\gamma_l|C_i) = \frac{\sum_{j=1}^{2^k} n_j \hat{z}_{ij} \gamma_l^i}{\sum_{j=1}^{2^k} n_j \hat{z}_{ij}}. \quad (1.13)$$

$\gamma_l^i$  is the component  $l$  of agreement pattern  $j$ , with  $\gamma \in \{0, 1\}$ . Both steps are iterated, estimating a new  $\hat{z}_{ij}$  until convergence for  $m$  and  $u$  is reached.

### *String similarity functions*

The agreement of identifiers does not have to be binary (“match” and “non-match”) outside of the EM algorithm.  $\gamma_i$  can also take all values between 0 and 1 (e.g. 0.741), depending on the degree of similarity of two identifiers given

by a similarity measure, usually of strings.<sup>18</sup>

Thus,  $\gamma_i$  will be equal to the similarity value of the strings. An in-depth overview of the similarity measures used in practice is presented in Christen (2012). One example is the  $q$ -gram similarity (also called the  $n$ -gram similarity). To calculate a similarity based on, e.g. bigrams ( $q = 2$ ), identifiers are split into subsets of length  $q$ . Here, the overlap of two subsets of both the strings is taken as a metric for similarity. For example, the names  $s_1 = \text{JOHNA}$  and  $s_2 = \text{JONAS}$  give the bigram sets  $b_1 = \{\text{JO, OH, HN, NA}\}$  and  $b_2 = \{\text{JO, ON, NA, AS}\}$ .

Example similarity measures include the Dice coefficient (Dice 1945) and the Jaccard coefficient (Jaccard 1901). The overlap of  $b_1$  and  $b_2$  is  $b_1 \cap b_2 = \{\text{JO, NA}\}$ . This gives the Jaccard and Dice coefficient  $D$  as follows:

$$D = \frac{2|b_1 \cap b_2|}{|b_1| + |b_2|} = \frac{2 \cdot 2}{4 + 4} = 0.5, \quad (1.14)$$

and

$$J = \frac{|b_1 \cap b_2|}{|b_1| + |b_2| - |b_1 \cap b_2|} = \frac{2}{4 + 4 - 2} = 0.33. \quad (1.15)$$

For the final weight calculation – if using the Dice coefficient (equation 1.14) – the agreement weight for this one record pair  $(a_i, b_i)$  would be  $\gamma_i = 0.5$ .

As an important alternative, edit-distance-based measures are another set of methods widely used, which depend on the number of (editing) operations (such as deletions or substitutions) that are required to transform one string into another string, such as the Levenshtein similarity<sup>19</sup> (Levenshtein 1966).

Another important string similarity measure includes the Jaro-Winkler similarity (Winkler 1990), which combines the edit-distance approach with the  $q$ -gram similarities. The simple Jaro-similarity (Jaro 1989) is calculated as follows:

$$\text{sim}_{\text{Jaro}} = \frac{1}{3} \left( \frac{m}{|b_1|} + \frac{m}{|b_2|} + \frac{m-t}{m} \right), \quad (1.16)$$

where  $m$  is the number of agreeing characters within half the length of the string and  $t$  is the number of transpositions (switching string positions that would lead to the strings matching) in  $m$ . The modifications by Winkler (1990)

<sup>18</sup>Similarity measures for numerical values are also available; for further details, see Christen (2012).

<sup>19</sup>Which is a normalised inverse of the Levenshtein distance.

increase the similarity if the first (up to four) characters of the two strings agree. This is because errors at the beginning of strings have been shown to occur with a lower frequency (see Section 1.5.5.2). The similarity is also increased when the substitutions of two differing characters are very common (e.g. “s” and “z” or “e” and “a”). Thus, the Jaro-Winkler similarity takes several error mechanisms into account and adjusts the Jaro similarities upwards accordingly.

#### *Calculating the final weight*

Newcombe et al. (1959: 956) first suggested the use of a probability (a positive likelihood ratio) for denoting the agreement or disagreement of a record pair, where the agreement ratio was calculated as  $\log_2 p_L - \log_2 p_F$ , where  $p_L$  corresponds to  $m_A$  and  $p_F$  corresponds to  $u_A$ , respectively.

A weight  $w_i$  is calculated for each field of each record and is either negative (if both records disagree) or positive (if both records agree), corresponding to the agreement and disagreement weights:

$$w_i = \begin{cases} \log_2 \left( \frac{m_A}{u_A} \right), & \text{if } a_i = b_i \\ \log_2 \left( \frac{m_D}{u_D} \right), & \text{if } a_i \neq b_i \end{cases} \quad (1.17)$$

To calculate the final similarity score, an individual score is calculated for each pair  $(a_i, b_i)$  as follows:

$$\text{similarity score}(a_i, b_i) = \sum_{j=1}^k w_i \gamma_i, \quad (1.18)$$

where  $k$  is the number of identifiers and  $\gamma_i$  is the similarity for a given pair  $(a_i, b_i)$ . Note that in the simplest case of using exact matching only,  $\gamma_i$  will be either zero (unmatched pair) or one (matched pair). The use of string similarity measures allows for values between 0 and 1, depending on the similarity coefficient.

The classification of record pairs as matches or non-matches then depends on this final similarity score.

### 1.5.6 Classification of linkage pairs

All algorithms classify record pairs out of all the possible combinations as either *matches* (M) or *non-matches*<sup>20</sup> (NM). This classification can be either wrong or right with respect to the true state. Figure 1.4 shows the possible classification states for each record pair. The numbers of true positives (TP), false positives (FP) and false negatives (FN) will be central in the course of this work.

		True State	
		Match	Non-Match
Classification	Link	True Positive	False Positive
	Non-Link	False Negative	True Negative

Figure 1.4: Relationships between true matching states and classifications.

With comparisons based on probabilistic models (which are essentially Naïve-Bayes classifiers (Wilson 2011)), a threshold of the similarity score for each record pair will determine its classification. For decisions based on thresholds, a second, more conservative threshold can be used to denote *possible matches* (PM). For clear-text linkage operations, records classified as possible matches will usually be subject to a clerical review, where the classification into matches or non-matches is done manually (Herzog et al. 2007). This has no bearing on Privacy-preserving Record Linkage, as a clerical review of the encrypted identifiers is nonsensical by definition.

The classifications shown in Figure 1.4 are widely used for applications outside of Record Linkage as well. A false positive classification is sometimes referred to as a *Type-I error* or  $\alpha$ -error, while false negatives are called *Type-II errors* or  $\beta$ -errors (Schnell et al. 2013).<sup>21</sup> Applications include probes at airports, cancer screenings, drug testing and anti-virus software.

<sup>20</sup>Or: unmatched pairs. Throughout the rest of this thesis, these will be referred to as non-matches.

<sup>21</sup>In a narrow field of German computer science, these types of errors are known as “Synonymfehler” (false negative errors) and “Homonymfehler” (false positive errors). As both of these are mentioned in only 29 (as of April 2019) exclusively German publications, they will not be considered here. They are also misleading in terms of terminology. However, this is out of the scope of this thesis.

### 1.5.6.1 Cost-based classifications

For each of the examples above, the “costs” of making a false positive or false negative decision vary considerably: while a spam mail not classified as spam (a false negative classification by the spam filter) is an annoyance at most, giving a negative cancer screening result to a cancer-ridden patient is potentially life-threatening. The incorrect classification by anti-virus software of a file as virus-ridden (a false positive) is at most time-consuming for the user, while missing a virus (false negative) can threaten the security of computers or even entire networks.

The same is true for real-time algorithms in self-driving cars: In 2016, the self-braking emergency mechanism of a Tesla Model S malfunctioned, as its systems falsely classified a tractor-trailer as an overhead road sign.<sup>22</sup> In terms of the auto-brake system, this was a false negative (braking) event: The system did not brake even though it should have. As the driver did not intervene, he was killed. The CEO of Tesla at the time, Elon Musk, claimed that this particular system was designed to “avoid false braking events”<sup>23</sup> (a *false positive* braking event). This event highlights the problem of the “costs” of false decisions: False positive emergency braking could be considered to be extremely risky, particularly at high speeds, while a false negative braking event (not braking in case of an emergency) is life-threatening, particularly if the driver relies on the autopilot system.

As the costs for making false positive or false negative classifications can differ, optimal classification methods with different loss functions have been proposed (Verykios et al. 2003). For each matching state and (mis)-classification<sup>24</sup>, a cost is associated. This gives a “cost matrix” which is used to modify the decision thresholds for the scores from a probabilistic link. Thus, different linkage applications can modify their decisions on the basis of the requirements of their respective linkage scenarios.

---

<sup>22</sup>The full story, including the causes, can be found here: <https://electrek.co/2016/07/01/understanding-fatal-tesla-accident-autopilot-nhtsa-probe/> (Last accessed: 03.04.2019).

<sup>23</sup><https://twitter.com/elonmusk/status/748625979271045121> (Last accessed 03.04.2019).

<sup>24</sup>In the original publication, costs are associated with possible matches as well.

### 1.5.6.2 Machine learning for classifications: Decision trees

If record pairs from a similar population with a known link status are available, these can be used to train a machine learning-based model for finding the optimal classifications (for an overview, see Christen (2012)).

The most prominent methods to do this are *decision trees* and *support vector machines* (SVMs; Christen 2012).

Table 1.4: Example link file and resulting agreement states for DOB/sex and Jaro-Winkler similarities for names with a true match state (M is a match; NM a non-match) as an input into a decision tree.

File A				File B			
first	last	dob	sex	first	last	dob	sex
PETER	JOERNS	20.01.1987	m	PETER	JOERNS	20.01.1987	m
JAMES	JONES	21.12.1970	m	JAMES	JONES	21.12.1970	m
SARAH	PETERS	01.01.1946	f	SAHRA	PETERS	01.01.1946	f
WOLF	YELLOW	11.11.2010	m	PETE	BLACK	20.02.2019	m

Jaro-Winkler similarity (on names) and true match state				
first	last	dob	sex	Match state
1.00	1.00	1.00	1.00	M
1.00	1.00	1.00	1.00	M
0.87	1.00	1.00	1.00	M
0.00	0.46	0.00	1.00	NM

The former needs an input similar to Table 1.4, where the pairwise similarities of all the identifiers used for linkage are put together with the true matching state (this is called *labelled data* (Christen 2012)), which serves as the training data. Decision trees use this input to derive a set of rules that would give the best possible linkage quality (in terms of the trade-off between true positives and false positives) given the training data.

Figure 1.5 gives an example of a (crude) decision tree with only few leaves built by the very small input from Table 1.4.

### 1.5.6.3 Machine learning for classifications: Support vector machines

Support vector machines (SVMs) work on binary classifications (only matches and non-matches are possible; possible matches are not). The algorithm attempts to find a multi-dimensional plane (a hyperplane) that best segregates

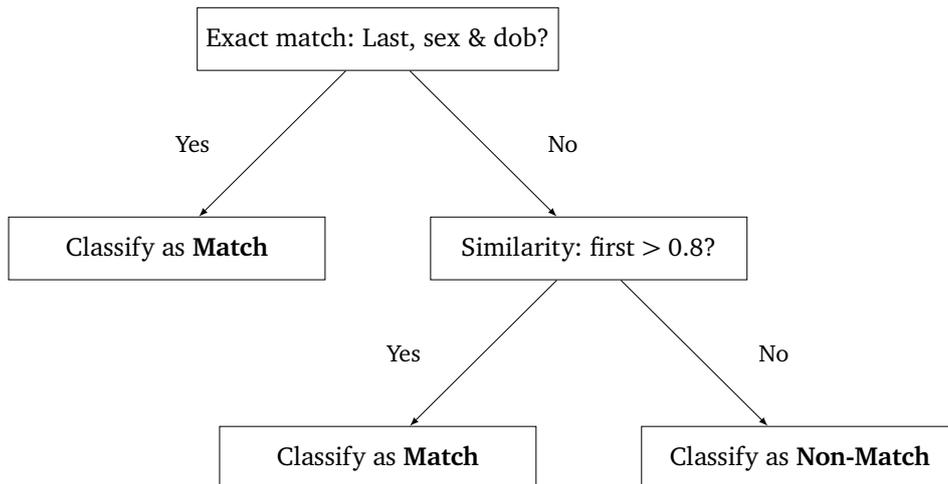


Figure 1.5: Example of a small decision tree that could be built for the example link files shown in Table 1.4.

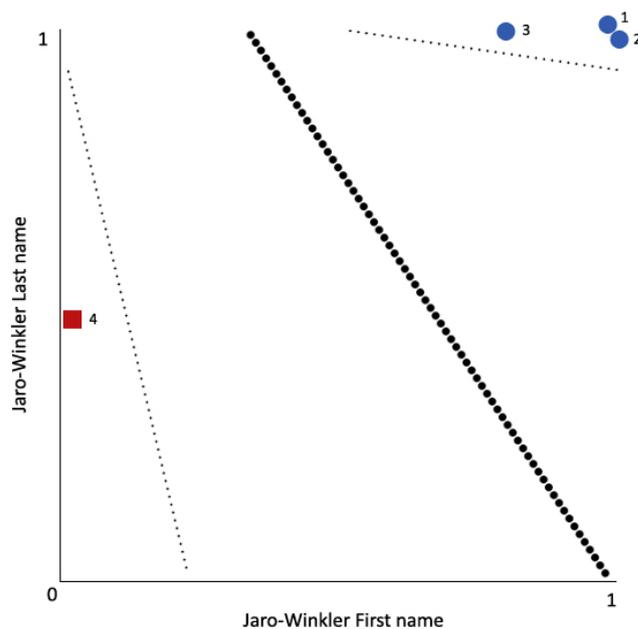


Figure 1.6: Example of a SVM with the examples from Table 1.4, where blue circles are the matching pairs and the red square is the non-matching pair. The small dotted lines are the support vectors, and the fat dotted line is the (hyperplane) support vector machine.

the given data (Christen 2012). In our case, it attempts to find a hyperplane which gives the best threshold for classifying matches and non-matches. For this, measures of similarity between several variables can be computed. These are then treated like coordinates. The central goal is to separate the data points

in all the dimensions with the maximum margin possible. Figure 1.6 gives an example with the data presented above. Here, three support vectors for segregating the data into matches and non-matches are shown. The one with the maximal margin between the nearest data points (points 3 and 4) is the SVM solution to the classification problem, dotted in bold.

Current extensions allow for non-linear SVM separation, based on kernels, which is essential particularly for image-based machine learning tasks (Campbell/Ying 2011).

However, applications of both decision trees and SVMs show that no single approach is superior to the standard Naïve-Bayes classifier (Elfeky et al. 2002). Other, more recent machine learning applications have been suggested as well, which promise to outperform the standard EM algorithm.

#### 1.5.6.4 Other machine learning classification methods: Neural networks and deep learning

*Deep learning* is a relatively new technique that uses several learning *layers* to classify an input layer into output layers (Kassambara 2018). Deep learning has been used in AlphaZero<sup>25</sup> to learn and master the games of Chess, Go and Shogi, outperforming all prior programs (Silver et al. 2018). Recently, it was used to learn a real-time strategy game (Star Craft 2), beating a team of professional players (Vinyals et al. 2019). Recently, a publication by Wilson (2011) suggested that a neural network with a single-layer perceptron works better for the match classification tasks in Record Linkage than the EM algorithm. This has not been systematically tested yet.

Using more than one learning layer, deep learning for data linkage has been successfully implemented by Kooli et al. (2018), reporting superior performance to that of standard methods, SVMs and decision trees. Figure 1.7 shows the general idea of using deep learning for linkage. However, this particular application was based on the entity resolution of professional databases, which is not a traditional Record Linkage application. Nevertheless, the algorithm should be adaptable to the standard PRL setting with ease. Given the recency of this publication, it has neither been replicated nor re-tested independently.

---

<sup>25</sup>By the Google company DeepMind: <https://deepmind.com/> (Last accessed: 04.04.2019).

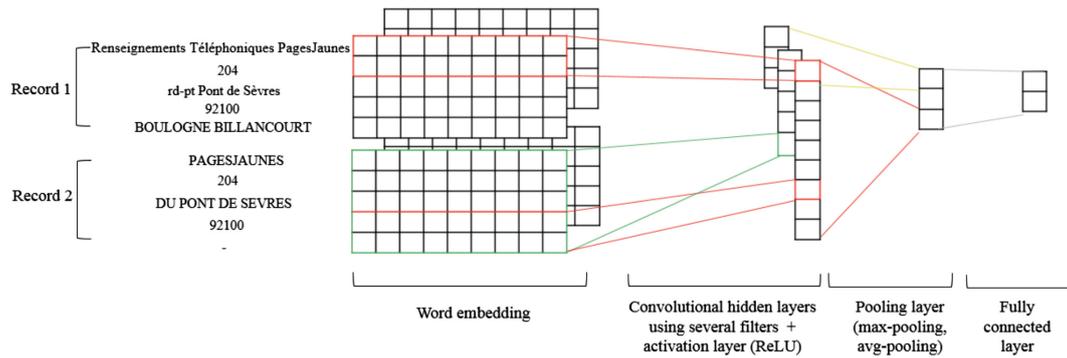


Figure 1.7: Example deep learning application for Record Linkage. Image taken from Kooli et al. (2018: 9).

### 1.5.7 Evaluation

After the classification of record pairs as matches or non-matches and with the classification outcomes of Figure 1.4 in mind, the evaluation of the quality of classifications can be done by calculating several metrics. This thesis will follow the description of the key metrics provided by Christen (2012).

According to Christen (2012), *accuracy* is commonly used to classify outcomes in machine learning-based applications. It can be calculated as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}. \quad (1.19)$$

It suffers from a major problem with respect to assessing the linkage quality for linking data: If the overlap of two files is small, the number of true negative pairs will be very large, heavily inflating the accuracy, even when the number of true positive classifications is small.

A second measure widely used, particularly in medical science settings, is *specificity* (also known as the true negative rate (TNR)). It can be calculated as follows:

$$\text{Specificity} = \text{TNR} = \frac{TN}{TN + FP}, \quad (1.20)$$

which can be useful for medical screenings. Pregnancy tests are an example. A high specificity means that a person with a negative pregnancy result very probably is not pregnant. This is a desirable property for many applications. However, the number of true negatives inflates this measure as well, making it less critical for Record Linkage applications.

The *false positive rate* (FPR) is a measure that is the inverse of specificity and can be calculated as follows:

$$\text{FPR} = 1 - \text{Specificity} = \frac{FP}{TN + FP}, \quad (1.21)$$

which suffers from the same drawbacks as specificity itself. A false positive rate is vital for many applications, for example, as a performance measure of anti-virus software detection rates.

The considerably more popular measures widely used in computer science are *precision* (also known as the positive predictive value (PPV)) and *recall* (also known as sensitivity). The former can be calculated as follows:

$$\text{Precision} = \text{PPV} = \frac{TP}{TP + FP}. \quad (1.22)$$

It gives a measure of how precise a classifier is when making classifications. As a classifier can classify all records as matches, giving the maximal number of true positives, precision shows at what “cost” the true positives are classified. recall can be calculated as follows:

$$\text{Recall} = \text{Sensitivity} = \frac{TP}{TP + FN}, \quad (1.23)$$

It is the fraction of actual matches found by the classifier and all true matching pairs. A recall of 0.9 means that 90% of all the possible true pairs were correctly classified as matches.

Together, precision and recall give a good overview of the quality of a linkage classification. Naturally, a method to combine the two can be used. The *F-measure* (also known as F1-Score or F-Score) is the harmonic mean of precision and recall:

$$\text{F-Measure} = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}. \quad (1.24)$$

As higher values for precision and recall are indicative of better linkage quality, conversely, a higher F-measure is desirable.

#### *Critique of the F-measure*

More recently, the F-measure has come under scrutiny for its inflexibility in terms of weighting recall and precision according to the costs associated with an increase in the number of false positives or negatives (Hand/Christen 2018).

Hand/Christen (2018: 546) suggested to simply use the simple arithmetic mean of precision and recall as a new convention, which is why the F-measure is renamed as mean of precision and recall (MPR or mean prec./rec.) and calculated as the arithmetic mean of precision and recall throughout this thesis (unless otherwise noted):

$$\text{Mean prec./rec. (MPR)} = \frac{1}{2} (\text{Recall} + \text{Precision}). \quad (1.25)$$

# Privacy-preserving Record Linkage (PPRL)

Current EU data privacy regulations (Council of the European Union 2016) recommend encrypting the identifiers used for linking data. Record linkage using encrypted identifiers is called Privacy-preserving Record Linkage (PPRL), which will be of increasing interest in the future because of these new data protection requirements. Everywhere else in the world, where national unique PIDs are unavailable, PPRL enables new opportunities to link data that would otherwise not be available for research purposes.

In some settings, privacy-protection laws or institutional rules prohibit the use of identifiers such as the first or the last name for Record Linkage. In such contexts, the identifiers used must be encrypted in a way that varies from scenario to scenario. In Germany, the so-called *de-facto anonymity* is needed (see below), which calls for encryption methods breakable only with infeasible effort or computational time (brute force attacks would be an example).

To clarify this central topic of the thesis, preliminaries are discussed first: Encryption, hash functions and PPRL methods are explained before Bloom filter-based methods are tackled.

## 2.1 Definition of PPRL

Privacy-preserving Record Linkage is defined as Record Linkage using encrypted identifiers. It is not important whether identifiers are encrypted separately

or together for the linkage. The encrypted variables can, in theory, be linked using the same matching algorithms as those described in Section 1.5.5.

The definition of the term *privacy-preserving* is considerably more complicated and diverse. Current EU regulation (Council of the European Union 2015; Council of the European Union 2016) demands that the encryption must guarantee *de-facto anonymity*, which is defined as follows:

“To ascertain whether means are reasonably likely to be used to identify the natural person, account should be taken of all objective factors, such as the costs of and the amount of time required for identification, taking into consideration the available technology at the time of the processing and technological developments” (Council of the European Union 2016: paragraph 26).

For most linkage operations in the EU countries, this will lead to an increase in the need for PPRL solutions, which should require a considerable amount of time and effort to break. The central provision for guaranteeing privacy is to be able to avoid the re-identification of the records of natural persons after encryption. Therefore, theoretical methods of attacking are not sufficient, unless they can be shown to lead to the identification of private records. However, it is not required to generate a wholly decrypted dataset, as long as natural persons can be re-identified by other means. For example, if a birth date was decrypted through an attack on the encryption, and the population in the data is very limited, this could identify the person – an example would be hospital records, where there might be only one person born on 29 February 1978. A person with knowledge about the patients could then infer that this person must be the encrypted record.

To prevent this, an encryption must be secure against re-identification. A wide-spread method to encrypt data is to apply *hash functions* to a (clear-text) input.

## 2.2 Prelude: Hash functions and HMACs

Encrypting data to obscure the underlying information can be done in several ways. Early applications were secret messages by spies or communication of troops amongst their units. Encryption methods are manifold and could easily

be covered in a second thesis. For PPRL, central concepts when talking about encryptions are *hashes* and *hash functions*.

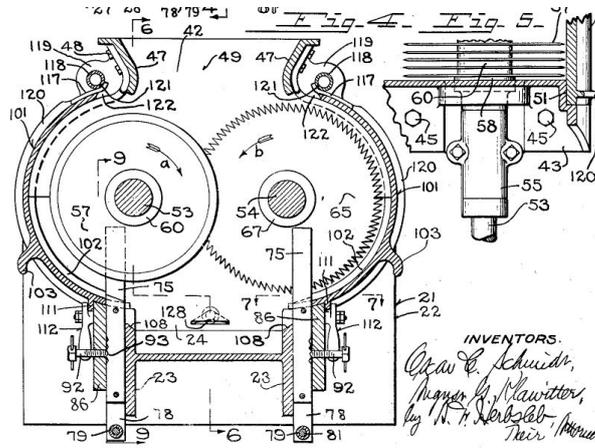


Figure 2.1: Part of a hashing machine, as patented by Schmidt/Klawitter (1927).

The origin of the term, as described by Knott (1975) is from an older usage of the words “to chop” or “to cut”, where the *hash* is the (random-looking) output of a so-called hashing machine (see Figure 2.1), often consisting of multiple rotating blades. The older patents (Schmidt/Klawitter 1927) for such machines that look like what could now be considered “shredders” were used to shred documents or meat leftovers. In its modern use as a non-physical method, *hashing* refers to the “[...] randomising scrambling of [a] given key-value [...]” (Knott 1975: 265). *How* this is achieved depends on the *hash function* used to generate the *hashes* from the input. Modern hash functions use several combinations of operations<sup>1</sup> to achieve the goal of the efficient scrambling of an input (Carter/Wegman 1979). The goal of a hash function is to generate random-appearing output from any input that is evenly distributed in the space of the possible hash values (Stallings 2014). This also means that any change in the input, however small, must lead to a completely different output.

A *HMAC* (keyed-hash message authentication code; Krawczyk et al. 1997) is an application of using (keyed) hash functions for checking the integrity of either messages or files in electronic communication. It is also called *cryptographic checksum* or *integrity check value* (Krawczyk et al. 1997: 3). Both parties need to own a secret shared key (a form of “password”), with which a keyed hash function of the sent message or file is generated. The resulting

<sup>1</sup>Examples include the bit operations AND, OR, XOR and bit shifting; for details, see Stallings (2014).

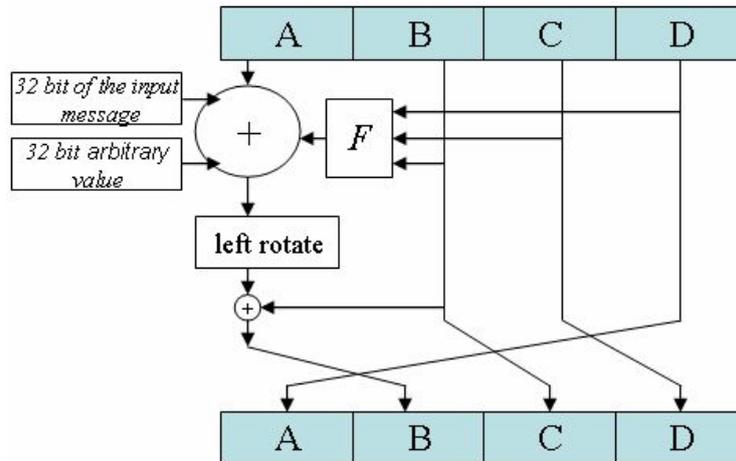


Figure 2.2: Illustration of a single round of the 16 rounds of the MD5 algorithm. Figure taken from Abualsaud et al. (2008: 293).

hash is the HMAC that can be sent along with messages, files or archives; for example, server certificates use “fingerprints” for server/client identification, which is a form of HMAC.

The most popular method of building HMACs is to use *cryptographic hash functions*. In addition to the requirements for a “regular” hash function, cryptographic hash functions must be secure against the identification of the specific value that leads to the given hash value (called one-way property) and against finding two inputs that map to the same value (*collisions*, called the two-way property (Stallings 2014)). Two widespread examples for such functions are MD5 (Rivest et al. 1978) and SHA-1 (National Institute of Standards and Technology 2002), which are still occasionally used for checking file integrity in operating systems. `md5sum` is still available as a Unix-command, while the file checksum integrity verifier (FCIV)<sup>2</sup> is the equivalent for older Windows systems, using MD5 or SHA-1.

Figure 2.2 illustrates the MD5 algorithm (Rivest et al. 1978). The single step shown in the figure is repeated 15 more times. MD5 uses four 32-bit blocks (A–D in the figure) as a buffer and the output hash (resulting in an output length of 128 bits). The input is split into 16 32-bit blocks, giving a 512-bit input. A–D are initialised with constants. The MD5 algorithm now uses 16 rounds (one for each 32-bit input block) of 16 operations to modify the values of A–D.

<sup>2</sup>For more details, see <https://support.microsoft.com/de-de/help/841290/availability-and-description-of-the-file-checksum-integrity-verifier-u> (Last accessed: 02.04.2019).

MD5 uses a combination of left rotations, additions and one of four functions  $F$ .  $F$  changes each round and can be an XOR, OR, NOT or AND operation.<sup>3</sup> The concatenated states of A–D after 16 rounds are the output hash.

It is important to note that MD5 and SHA-1 are considered cryptographically insecure and should be avoided in security applications (Stallings 2014). The first collision attack on MD5 was posted in 2004 (Wang et al. 2004). One year later, an automated, more efficient implementation running on standard office hardware was implemented (Klima 2005). SHA-1 was broken only recently (Stevens et al. 2017).<sup>4</sup> Therefore, alternative hash functions should be used. More recently, a group of cryptographic hash functions (among them MD6, the successor to MD5) was evaluated by the National Institute of Standards and Technology (NIST), with the winner (Keccak) being implemented as the new SHA-3 algorithm (National Institute of Standards and Technology 2012). SHA-3 and its predecessor, SHA-2, are both currently viewed as cryptographically secure (Stallings 2014).

With hash functions and cryptographic properties covered, the next section will deal with an overview of their application in the framework of Privacy-preserving Record Linkage.

## 2.3 Overview of PPRL methods

An extensive overview of all aspects of PPRL is given in Vatsalan et al. (2013). Figure 2.3 gives an overview. All of the methods mentioned thus far are two-party protocols or parts of three-party protocols: two parties encrypt their data and link the encrypted data either by themselves (two-party) or through a trustee (three-party). Much has been published about secure multi-party protocols. However, as not a single secure multi-party protocol is currently in use for linking large real-world databases, multi-party protocols are out of the scope of this thesis.

Encryptions used for PPRL in practice (Schnell/Borgs 2016a) can be divided

---

<sup>3</sup>Given two binary input bits A and B, AND requires both to be 1 to output a 1, OR requires any of either A or B to be 1, and XOR (exclusive OR) requires that *only* A or B are set to 1, but not both. NOT inverts the input value.

<sup>4</sup>There is a website explaining the attack: <https://shattered.io/> (Last accessed: 01.04.2019).

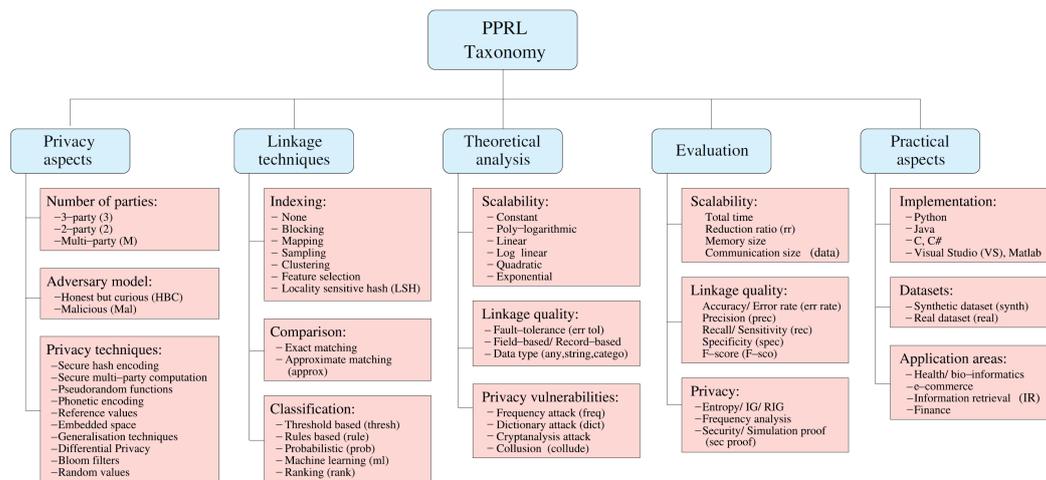


Figure 2.3: Aspects of Privacy-preserving Record Linkage. Image from Vatsalan et al. (2013: 953).

into at least three categories: phonetic codes, string hashing and Bloom filters.<sup>5</sup> Each will be discussed in detail.

### 2.3.1 Matching by encrypted PID

If a unique ID can be used for linkage, encrypting the ID using a strong HMAC (see Section 2.2), such as SHA-3 (National Institute of Standards and Technology 2012), is the easiest way to conduct PPRL. Linking data privately is trivial in this case. It will be discussed in Section 2.3.5.1.

### 2.3.2 Trusted third-party designs

In scenarios where a trustworthy institution conducts the linkage operations (see Figure 2.4), only the trustee will use (encrypted) personal identifiers for the linkage. A statistical agency or another research institution will only receive the pseudo-ID pairs which are classified as matches, along with the merged data. This is rarely done in practice (Schnell/Borgs 2016a) because of the high organisational demands. However, established trusted third-party designs for linking health data exist in some countries, e.g. Australia.<sup>6</sup>

<sup>5</sup>Recently, using multiple match keys was proposed by Randall et al. (2019). Due to the recency of this publication (23.05.2019), it will not be discussed here.

<sup>6</sup>Australian linkage organisations are an example. More examples are given in the Introduction.

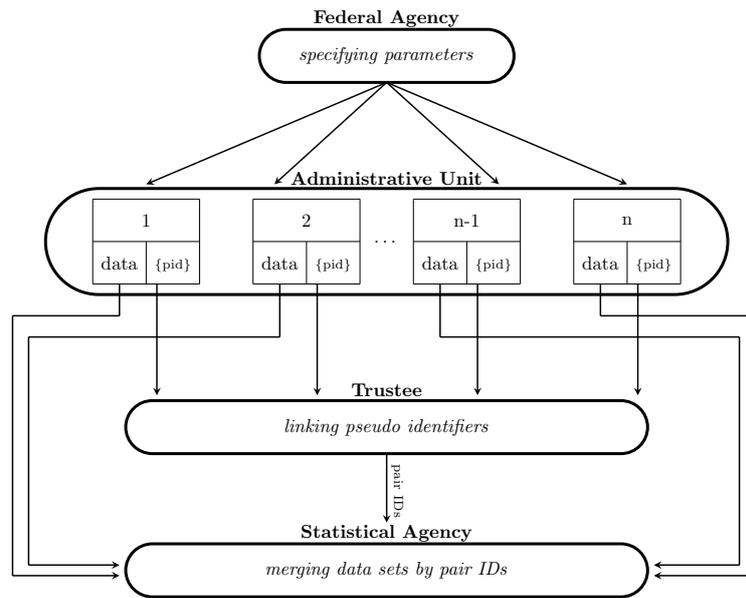
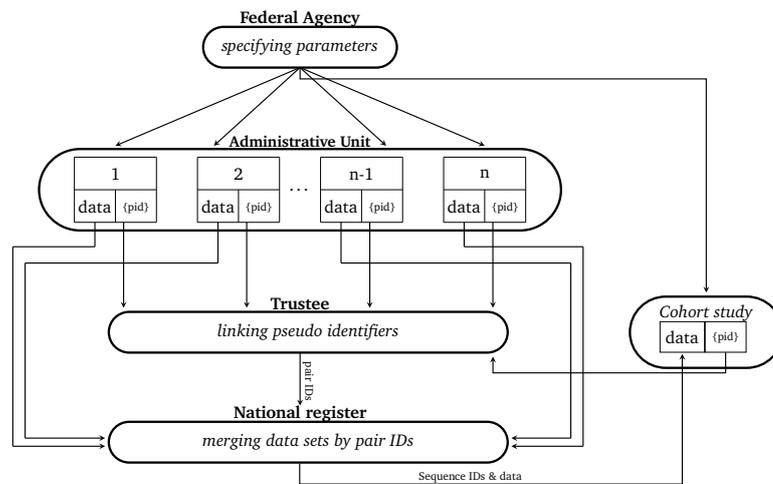


Figure 2.4: Design of linking hospital data using a trusted third party. Figure taken from Schnell (2016a).



Note: Simplified protocol assuming same parameter settings for all parties.

Figure 2.5: Design of linking external data (here: cohort data) to a private mortality register using a trusted third party. Figure from Schnell/Borgs (2018b).

Another scenario would be medical studies. Both follow-up studies and cohort studies may require current information about whether or not a patient is still alive. The private querying of a private mortality register is shown in Figure 2.5. Here, the organisation will encrypt its clear-text identifiers, sending the

pseudo-identifiers to the trustee. The cohort study will only receive data for the matched PIDs.

### 2.3.3 Secure multi-party protocols

Secure multi-party protocols have been used for non-linkage operations, but theoretical PPRL settings, even with Bloom filters, have been published (Vatsalan/Christen 2016). However, they usually depend on iterative online communications between multiple parties, where encrypted subsets of identifiers are interchanged to estimate the similarity between record pairs. This can have two disadvantages (Schnell 2016a): It is often computationally expensive and difficult to communicate. Most data protection officers will not allow remote connections at all, particularly if they communicate sensitive information (even when encrypted). In some settings, data holders might even be (technically) unable to communicate remotely for security reasons. Furthermore, the lack of portable, user-friendly implementations can be an obstacle.<sup>7</sup> Thus far, to the best of the authors knowledge, there has been no large-scale PPRL application with real-world data.

### 2.3.4 Phonetic codes

Phonetic codes are usually used to find similar names by assigning same-sounding names (phonetically equal names, hence the name) to the same phonetic code. Some examples include the German *Kölner Phonetik* (Postel 1969), *Metaphone* (Philips 1990) and *Soundex*.

Often wrongly attributed to Knuth (1977) who arguably gave rise to its popularity, the Soundex phonetic predates most phonetic systems in use today by several decades.<sup>8</sup> It was patented in 1918 by Robert C. Russel (1918) as an indexing system for cabinet folders used in offices.<sup>9</sup> An excerpt can be seen in Figure 2.6.

---

<sup>7</sup>One of the few publicly available implementations, for example, only works on Unix systems (<https://github.com/cryptobiu/libscapi>; last accessed: 20.04.2019). Most administrative data owners will most often use Windows.

<sup>8</sup>For this piece of knowledge, I have to thank Rainer Schnell, who shared this information with me. This fact was first published in Schnell (2016b).

<sup>9</sup>The original patent document can be found online: <https://patentimages.storage.googleapis.com/31/35/a1/f697a3ab85ced6/US1261167.pdf> (Last accessed: 29.03.2018).

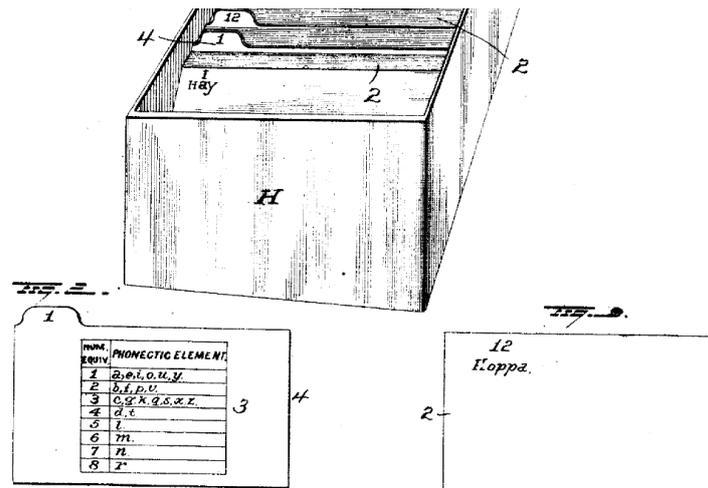


Figure 2.6: Excerpt of the original patent (Russel 1918: 1) which later became Soundex, including the phonetic code replacements for letters.

The system was later adopted to re-sort the files for the past American censuses as part of the job initiatives by then-U.S. President Roosevelt, known as the “New Deal”. The *Works Progress Administration* introduced Soundex in the 1930 U.S. Census.<sup>10</sup>

While all of these methods aggregate strings in a way that makes it impossible to directly re-identify the original names and serve as an error-tolerant base for applying a strong encryption (as small spelling errors will still possibly lead to the same phonetic code), the risk of a false positive classification increases, as many names are mapped to the same codes (which will, in turn, lead to identical hashes). This increases the number of duplicate linkage strings.

For example, the German last names MEIER, MAIER, MAYER and MEYER are all pronounced the same and are all mapped to the same Soundex code (M600), which would, in turn, lead to false positives if the first names are similar as well. To counteract this drawback, most PPRL methods make use of the date of birth as well. One of these (the Swiss ALC) is discussed in the next section.

### 2.3.5 String hashing

Encryptions using this method are widely used. Variants of string hashing methods are also called *anonymous linkage keys* (ALCs).

<sup>10</sup>See [https://www.census.gov/history/www/genealogy/decennial\\_census\\_records/soundex\\_1.html](https://www.census.gov/history/www/genealogy/decennial_census_records/soundex_1.html) (Last accessed: 29.03.2018).

### 2.3.5.1 Encrypted PIDs

If a nationally available, unique personal identification number is available, it still must be encrypted for linkage in many cases. An example can be seen in Figure 2.7. This is a special case of string hashing, because real PIDs are hashed, which should lead to neither false positive classifications nor missed record pairs. Although this method would be the preferred way to link data anonymously, PIDs are rarely available for linking outside of Scandinavian countries. Therefore, other methods must be used.

```
41234, James Sullivan, M, 20.1.1976  
41234  
0ff0bff6307805ce90ac578b9811ce7b5b766f55
```

Figure 2.7: Hashing the unique PID 41234 using SHA-1 as an HMAC.

### 2.3.5.2 Basic ALC

The most straightforward alternative is the *Basic ALC* (Herzog et al. 2007). All the identifiers used (usually: first and last name, sex and date of birth) are preprocessed and concatenated. The resulting concatenated string is encrypted using a cryptographic hash function (HMAC) such as MD5 or SHA-1 (Krawczyk et al. 1997).

An example can be seen in Figure 2.8. As the choice of the HMAC should not influence the linkage quality, more secure HMACs can be used (such as SHA-3).

```
James Sullivan, M, 20.1.1976  
JAMESSULLIVANM20011976  
c358a9721d899a1911f59ba83a56544b21c070d3
```

Figure 2.8: Basic ALC construction using the entry “James Sullivan, M, 20.1.1976” and SHA-1 as an HMAC.

### 2.3.5.3 Swiss ALC

First devised by the Swiss military and adapted for the Swiss Federal Statistics Office (Borst et al. 2001; Office fédéral de la statistique 1997), the Swiss ALC

is a variation of the basic ALC. While still concatenating the date of birth, sex and names, this method phonetically encodes the names first (usually using Soundex, see Section 2.3.4). This ensures a moderate amount of error-tolerance as long as the errors in the identifiers do not affect the phonetic code itself. A misspelling of the first character, for example, would lead to a missed link.

The two phonetic codes for the first and the last name are concatenated to the date of birth and sex. This forms the input string for an HMAC, which is applied in the same way as the basic ALC (see Figure 2.9).

```
James Sullivan, M, 20.1.1976  
J520, S415, M, 20011976  
J520S415M20011976  
640d982f29d266f8e34a38ef4fa6319b49481340
```

Figure 2.9: Swiss ALC construction using the entry “James Sullivan, M, 20.1.1976”, Soundex and SHA-1 as an HMAC.

#### 2.3.5.4 Encrypted Statistical Linkage Key (ESL)

The *Encrypted Statistical Linkage Key*, also called 581-key, was developed in Australia (Karmel 2005), which is why it is sometimes called Australian ALC. Instead of using the Soundex code of the first and the last names, only subsets of both these names are used. By merely trying which character positions give the best linkage quality, Karmel (2005) decided to use the second and the third positions of the first name concatenated with the second, third and the fifth positions of the last name, full date of birth and sex. The resulting string is encrypted using an HMAC again (see Figure 2.10). While the 581-key is arguably the most advanced string hashing variant in terms of linkage quality, it is currently seen as deprecated because of its limited recall and security (Randall et al. 2016).

```
James Sullivan, M, 20.1.1976  
AM, ULI, M, 20011976  
AMULIM20011976  
c6151183dd1335e5dc2c1312e7e9c28de9e13fd2
```

Figure 2.10: ESL construction using the entry “James Sullivan, M, 20.1.1976” and SHA-1 as an HMAC.

### 2.3.6 Bloom filters

First proposed by Burton H. Bloom (1970), Bloom filters were initially designed to efficiently check whether a record can be found in a dataset of reference records (*set membership*). The method was used primarily in chemoinformatics, where checking molecules against large molecule databases is commonly done using this data format (called binary fingerprints in Chem-informatics (Swamidass/Baldi 2007)).

They were first proposed to be used for Privacy-preserving Record Linkage (PPRL) by Schnell et al. (2009). The method has received widespread attention up to the point where “the [...] Bloom filter approach [...] has become almost a de-facto standard for Privacy-preserving Record Linkage” (Smith 2017: 272).

```
James Sullivan, M, 20.1.1976  
JA AM ME ES SU UL LL LI IV VA AN M 2 0 0 1 1 9 7 6  
0101011001010011100010010000010100001110
```

Figure 2.11: Bloom filter construction using the entry “James Sullivan, M, 20.1.1976”. The length of the filter is  $l = 40$  bits.

As Bloom filters are the main topic of this thesis, Bloom filter construction will be discussed in detail in Chapter 3. A short application is shown in Figure 2.11, where the names are split into sub-strings of two (names) and one (DOB/Sex). These are then encrypted into a binary bit vector with a length of  $l = 40$  bits. This bit vector is the Bloom filter used for linking. The choice of the length of the sub-strings ( $q$ ) and the number of elements set to 1 for each input sub-string ( $k$ ) as well as the length of the bit vector ( $l$ ) are the crucial parameters to choose. More details will be given in Chapter 3.

#### *Obstacles for implementing Bloom filters*

While Bloom filters are being increasingly used in research-oriented settings, their use outside of academics is rare. This is because of the following main reasons:

- Before the PPRL-Package for R (see Section 4.3.2), there were no user-friendly implementations of state-of-the-art Bloom filter methods, limiting the scope of potential users.

- Missing data will usually lead to non-matches. Imputation or post-estimation methods to solve this problem using Bloom filters have not been published thus far.
- Furthermore, the security of a single Bloom filter for each identifier designated for linkage is debatable, and composite Bloom filters are more challenging to implement into existing probabilistic models.
- Finally, without expertise and extensive practical knowledge, the selection of optimal identifiers and encryption parameters is difficult.

Research into better methods to prevent attacks on Bloom filters is ongoing (see Section 3.3), and a newly released R-package (see Section 4.3.2) fulfils the need for easy-to-use implementations. However, as the choice of parameters can have a significant effect on the data linkage quality (Brown et al. 2017a), identifier and parameter choices should be determined automatically, depending on the input data. This could help spread the use of Bloom filters for Privacy-preserving Record Linkage.

### 2.3.7 Parameter choice for Bloom filters

Currently, there is only literature on the topic of Bloom filters citing the need to address the issue of parameter choice:

“Nevertheless, choice of  $k$  is still an issue that needs to be addressed. This (and choice of size of Bloom filter) are affected by token set sizes. A choice that is optimal for small token sets might result in a large proportion of set bits for larger token sets” (Smith/Shlomo 2014: 8).

There are currently no solutions to this problem, other than best-practice recommendations (see Section 4.5.1 and Schnell (2016a)) and one brief publication by Izakian (2018), where a narrow parameter space was tested for optimal similarity-preserving properties, which is not the same as optimal linkage quality. Therefore, finding a way to automate identifier and parameter choice is the central topic of this thesis.

## 2.4 Privacy-preserving blocking and scalability

Privacy-preserving blocking attempts to accomplish the goals of standard (un-encrypted) blocking, as described in Section 1.5.4, but by using encrypted blocking variables. The simplest method to block privately would be to use a hash function (such as SHA-2) on any clear-text blocking variable, such as date of birth. The resulting hash string is then used as a blocking variable. Advanced methods such as canopy clustering (McCallum et al. 2000), sorted nearest neighbourhood (SNN) blocking (Hernandez/Stolfo 1995) or Single- and Multibit trees<sup>11</sup> (Kristensen et al. 2010) use different approaches, but all of them try to generate homogeneous groups of encrypted fields in terms of different similarity measures. Multibit trees are described in more detail in Section 3.4.2.

### *Sorted nearest neighborhood blocking*

To build blocks using sorted nearest neighbourhood (SNN) blocking (Hernandez/Stolfo 1995), all the records are first sorted by a sort key.<sup>12</sup> If binary vectors are used, the number of bits set to one is the sorting criterion. With a user specifying its size, a window now “slides” over the sorted records. All the entries inside the window form a block. SNN blocking is very fast (Schnell 2016a) but results in decreased linkage quality as compared to other methods.

### *Canopy clustering*

A special type of clustering algorithm, canopy clustering (McCallum et al. 2000) first determines a random starting point. A user-set distance *loose* is then used to build “canopies”, where all records within the distance *loose* form a cluster. Canopies are a particular type of cluster because they are allowed to overlap, meaning a record can be part of multiple clusters. A second user-defined distance *tight*, where  $tight < loose$ , defines the set of records to be discarded from the current canopy. The second step again uses a random starting point, includes all records within the distance *loose* and discards all records within the distance *tight*. This process is continued until all the records

<sup>11</sup>For an overview of privacy-preserving blocking techniques see Schnell (2015).

<sup>12</sup>An example for a sort key is given in Christen (2012: 82), where the first four letters of the first and last names are concatenated and used as a sort key.

are part of at least one canopy. The canopies are then used as blocks. The single greatest disadvantage of canopy clustering is the very long run-time required when linking large files (Schnell 2016a).

#### *Multibit trees*

The use of tree-based methods for blocking bit vectors was first proposed by Kristensen et al. (2010). Given input bit vectors, many trees are built according to several match bits. For each sub-tree, an upper similarity bound can be computed for any given query vector. All the sub-trees below a certain similarity threshold can be excluded from the comparison space. This reduces the search space considerably. For an in-depth look at Multibit tree blocking, see Section 3.4.2.

#### *PPJoin*

PPJoin was first developed by Sehili et al. (2015). It works on clear-text data and Bloom filters. For the latter, all the records are sorted by cardinality (number of bits set to one). For every record, the bit positions set to one are reordered according to their frequency of being set to one in the dataset. In the second step, a prefix of a dynamic length is constructed, consisting of the first few bit positions of the reordered bit vector. The comparison space is reduced by discarding all the record pairs with a high discrepancy in the cardinality and a sufficiently high difference in the prefix length. In empirical tests, PPJoin was outperformed by Multibit trees (Sehili et al. 2015). However, PPJoin can be implemented to work with GPU cores<sup>13</sup> in modern computer graphics cards, which potentially speeds up the linkage operation more than the multi-threading (CPU) capabilities of Multibit trees can. However, this has not been tested yet.

In the first application of using GPUs and CPUs together with PPRL, Boratto et al. (2019) showed a marked decrease in the required linkage time. If PPJoin can exploit the proposed architecture of Boratto et al. (2019), splitting the linking tasks by CPU threads and GPU cores, it might reduce the computational time required considerably.

---

<sup>13</sup>GPU is short for *graphics processing unit* and is the processor of a graphics card, as the CPU is the “universal” *central processing unit* of a computer. A graphics card will have hundreds or thousands of GPU cores running in parallel. However, GPUs can only perform very limited processing tasks.

## 2.5 PPRL applications

Privacy-preserving Record Linkage has been used to link databases in Brazil for epidemiological research into the causes of leprosy (Rocha et al. 2015), to link 114 million records of the Brazilian national cohort to health records (Dantas Pita et al. 2018), to develop a system to manage health insurance claims by merging data (Kimura et al. 2010), linking local heart attack registries to insurance data (Maier et al. 2015), and for privately linking data to national cancer and mortality registries (Eycken et al. 2000; Pålman et al. 2007).

Outside of medical research, Christen (2012) lists fraud detection, anti-terrorism surveillance and creating terrorism watchlists as examples of the areas of applications. PPRL methods were also modified by Farrow/Schnell (2015) to encrypt geographic information in a distance-preserving manner. In software programming, Bloom filters are used as part of some network protocols (Broder/Mitzenmacher 2003) and as a part of the Google Chrome browser (Erlingsson et al. 2014). Finally, Bloom filters are part of the protocols of at least two prominent crypto-currencies<sup>14</sup>, Bitcoin and Ethereum (Wood 2014).

---

<sup>14</sup>As the name suggests, these virtual “currencies” depend on decentralised, encrypted authentication when used for transactions, for which Bloom filters are part of the protocol.

## Bloom filter-based methods

As the central topic of the thesis is concerned with Bloom filter-based methods only, they will be covered in more detail in this chapter. First, the construction of Bloom filters will be explained, followed by an overview of all current attacks and hardening methods. Following this, methods to link and block Bloom filters will be discussed. Finally, the properties of Bloom filters when changing the parameters in terms of linkage quality and variance will be evaluated thoroughly. The implications for the next chapter will be discussed as well.

### 3.1 Constructing Bloom filters

A Bloom filter is a binary vector of length  $l$ , where each element (bit) of the vector has a starting value of 0. To store information in the bit vector, several bit positions are set to the value of 1. Determining which and how many bit positions are set to 1 differs<sup>1</sup>, depending on what is to be stored in the vectors. For molecules in chemistry, the existence or non-existence of atoms and bonds determines the bit position's values (Swamidass/Baldi 2007). In criminology, actual fingerprints are stored in bit vectors determined by the fingerprint features.

For storing personal information, as is the goal in Privacy-preserving Record Linkage, Schnell et al. (2009) suggested splitting an identifier into subsets of length  $q$ , resulting in so-called  $q$ -grams. Similarity measures based on  $q$ -grams are widely used in language-based information systems such as spell-checking,

---

<sup>1</sup>See page 51.

language classification, dictionaries and text compression (Robertson/Willett 1998), as well as unencrypted Record linkage (see Section 1.5.5.4). For string elements such as names,  $q = 2$  is often used. Sub-strings of  $q = 2$  are called bigrams, while for  $q = 1$ ,  $q = 3$  and  $q = 4$ , the respective denominations are unigrams, trigrams and quadgrams. For two sets of  $q$ -grams, the similarity between the sets can be computed as follows:

$$D_{A,B} = \frac{2|A \cap B|}{|A| + |B|}. \quad (3.1)$$

This gives the Dice coefficient (Dice 1945), which is the doubled intersect of the two sets of bigrams divided by the sum of unique bigrams in both sets. To make an approximation of the Dice coefficient possible by using Bloom filters, each unique element in a set of  $q$ -grams sets  $k$  different bit positions in the Bloom filter to a value of 1. Each one of the  $k$  mappings of an element to a bit position is called a hash function. An example is shown in Figure 3.1.

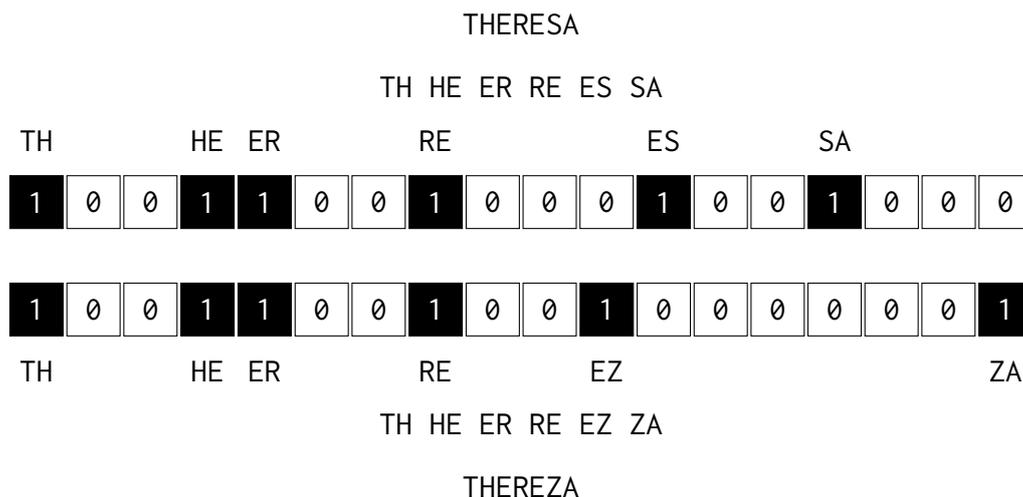


Figure 3.1: Example of a Bloom filter constructed from two different names using  $l = 18$  bits and  $k = 1$  hash function for each bigram. Note that the bit positions are set to 1 in the same order as the bigrams for visualisation purposes only.

In this example case, a single character is different, giving an edit distance of 1 and a Jaro-Winkler similarity of 0.905.<sup>2</sup> The Dice coefficient of the clear-text bigrams would be

$$D_{A,B} = \frac{2 \cdot 4}{6 + 6} \approx 0.667, \quad (3.2)$$

<sup>2</sup>In R using the stringdist package: `stringsim("THERESA", "THEREZA", method="jw")`.

while the Bloom filter representations have four bits in common, resulting in

$$D_{A,B} = \frac{2 \cdot 4}{6 + 6} \approx 0.667. \tag{3.3}$$

Note that collisions – bit positions set to 1 by more than one element – will lead to less exact similarity approximations, since more shared bigrams will then lead to fewer common bits set to one.

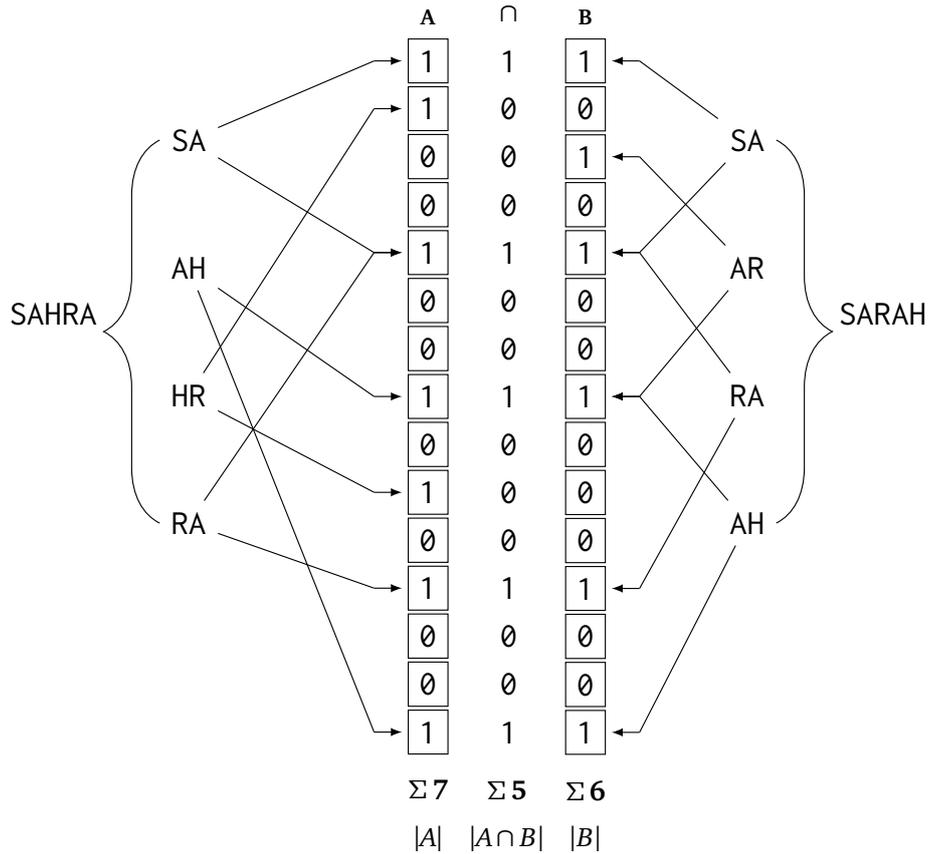


Figure 3.2: Second example of the Bloom filter construction for SAHRA and SARAH using  $l = 15$  bits and  $k = 2$  hash functions for each bigram. Image first published in Schnell/Borgs (2018c).

For the second example, see Figure 3.2. Here, the names SAHRA and SARAH both share three out of four bigrams (SA, AH and RA) and differ on a single bigram (HR and AR, respectively). The unencrypted Dice coefficient for the bigrams of both the names is calculated as follows:

$$D(A,B) = \frac{2 \cdot |A \cap B|}{|A| + |B|} = \frac{2 \cdot 3}{4 + 4} = 0.75. \tag{3.4}$$

Encrypting them using Bloom filters, both BFs have 7 resp. 6 bit positions set to 1, while having 5 common bits. Thus, the (encrypted) Dice coefficient can be estimated as follows:

$$D(A, B) = \frac{2 \cdot |A \cap B|}{|A| + |B|} = \frac{2 \cdot 5}{7 + 6} \approx 0.77. \quad (3.5)$$

This way, using Bloom filters for PPRL allows estimating the clear-text  $q$ -gram similarity using encrypted identifiers.

#### *Determining the bit positions set by $q$ -grams*

The original implementation of Bloom filters for PPRL as suggested by Schnell et al. (2009) used the *double-hashing scheme* as proposed by Kirsch/Mitzenmacher (2006) to determine the  $k$  bit positions set by a single  $q$ -gram. The scheme gives a bit position  $h_i$  for every  $i \in \{1, \dots, k\}$  by encrypting a  $q$ -gram  $x$  using two independent hash functions. Schnell et al. (2009) used MD5 and SHA-1 (see Section 2.2), converting their resulting hashes into a (large) numeric value and linearly combining them by multiplying one of the hash functions with a running number  $i$  and forming the sum of this operation. To give a bit position inside the bounds of the Bloom filter, the result is mapped to the length of the Bloom filter  $l$  by applying the modulo operation on it:

$$h_i = (\text{SHA-1}(x) + i \cdot \text{MD5}(x)) \bmod l. \quad (3.6)$$

This gives a maximum of  $k$  bit positions in the Bloom filter that are set to a value of 1. However, this approach is flawed in terms of security, as will be discussed in Section 3.2. In short, as only  $10^6$  different combinations of bit patterns are possible (Niedermeier et al. 2014), a bit pattern-based frequency attack can be successful. Therefore, the current best practice is to use full random hashing (Schnell 2016a), as the number of possible combinations increases to approximately  $10^{46}$ , making a frequency attack very difficult (Niedermeier et al. 2014). More on random hashing can be found in Section 3.1.4.2, while the security of Bloom filters is discussed in Section 3.2.

### 3.1.1 Steps when encrypting records using Bloom filters

The basic method to apply any Bloom filter-based encryption usually follows the pattern described in the subsections below: (1) The input strings are standardised depending on the type of information they contain. (2) Sometimes, a blank space is added to the standardised strings' beginning and end (*padding*), which gives more weight to the correct first and last characters. The resulting string is then split into subsets of length  $q$  to form the  $q$ -grams, with  $q$  being one of the user-made parameter choices. (3) Each element created in step (2) gives several ( $k$ ) bit positions in the Bloom filter. The method used to determine these bit positions depends on the implementation details. Currently, the use of double hashing to hash elements into BFs (Kirsch/Mitzenmacher 2006) is considered to be deprecated and has been replaced by random hashing, which is strongly recommended (Schnell/Borgs 2017). (4) In the initially empty Bloom filter (meaning all  $l$  bit positions have a value of 0), the resulting bit positions (depending on the parameter choices as discussed in 3.5.2) are set to the value of 1. Steps (3) and (4) are repeated for every element created in step (2). The resulting bit vector including only the elements of a single identifier is called a (standard) Bloom filter. Details on each step of the creation will be discussed in the following paragraphs.

#### *Cryptographic longterm keys (CLKs)*

The Cryptographic Longterm-key (CLK) is an extension of the Bloom filter algorithm first proposed in Schnell et al. (2011) and Schnell (2014). It is the result of all the input identifiers encrypted into a single Bloom filter. Several extensions and possible modifications of the algorithm are discussed in Schnell (2015). CLKs, which are a form of *composite Bloom filters*, are the result of an OR operation on several separate Bloom filters for each identifier. As the exact method of implementation varies, this chapter will present an overview of the exact procedure to create CLKs and Bloom filters, albeit with varying parameters and protections against cryptographic attacks.

### 3.1.2 Standardisation and preprocessing

As previously mentioned, every identifier to be encrypted needs to be standardised in one way or another. Omitting this step would drastically reduce the linkage quality, as real-world data is prone to errors (Hernandez/Stolfo 1998) or is represented differently (this is particularly true for date formats). As previously discussed in Section 1.5.1, two central steps are part of any preprocessing: Removing unwanted characters and words, and expanding abbreviations and correcting misspellings. For these, the `StandardizeString()`-routine available in the R-package PPRL (see Section 4.3.2) was chosen to remedy the problem. Table 3.1 presents an overview and some examples of the routines used.

Table 3.1: Preprocessing used in the PPRL package and throughout this thesis.

Preprocessing routine	Used on	Example	Description
Uppercase	Strings	Peter → PETER	All strings are set to uppercase
Resolve special characters	Strings	BJÖRN → BJOERN	Resolve special/foreign characters like Umlauts or Accents
Blanks	Strings	_BJÖRN → BJOERN	Leading and following blanks are deleted
Convert to ASCII	Strings	Jâ\$rg → JRG	The remaining string is converted to ASCII. All non-ASCII characters are omitted
Delete Non-Numeric	Numerics	0A2 → 02	All non-numerical values are deleted from the numeric fields

### 3.1.3 Padding and $q$ -gram partitioning

Padding is the addition of blanks to the beginning and the end of an input string. This way, the first and the last letters receive an additional bigram for encoding. If the assumption holds that the first and the last letters of the input strings are less error-prone than the letters in the middle of a string, padding can increase the linkage quality. However, it may be a security risk, as the frequency distribution of the first and the last letters is very skewed – some first and last letters are far more frequent (S and T) than others (Y and X).

With or without padding, all clear-text strings need to be split into characters of length  $q$ , forming sets of  $q$ -grams. Usually, strings are split to bigrams ( $q = 2$ ). Sometimes, dates of birth and sex are divided using unigrams ( $q = 1$ ). To the

best of the authors knowledge, thus far, trigrams ( $q = 3$ ) or quadgrams ( $q = 4$ ) have not been used in any published Bloom filter application. This will be covered in Section 4.3.1.

### 3.1.4 Mapping elements to bit vectors

Every  $q$ -gram needs to be stored in a Bloom filter in a deterministic way. As a Bloom filter is essentially a bit vector initially consisting of zeroes, positions of bits that will be set to the value of 1 have to be chosen. The number of hash functions  $k$  to be used can be the same for all the identifiers, it can vary for every identifier or it can be different even on the  $q$ -gram level. The choice of the number of hash functions ( $k$ ) chosen to encrypt clear text data is expected to be crucial in terms of the resulting linkage quality. To determine the actual bit positions to be set to 1, either double hashing or random hashing can be used.

#### 3.1.4.1 Double hashing

As previously discussed in Section 2.3.6, double hashing, as proposed by Kirsch/Mitzenmacher (2006), can be used to set  $k$  bit positions to 1, where each bit position  $h_i$  for  $i \in \{1, \dots, k\}$  is determined by the numeric MD5 and SHA-1 hash values of the input  $q$ -gram  $x$ :

$$h_i = (\text{SHA-1}(x) + i \cdot \text{MD5}(x)) \bmod l. \quad (3.7)$$

This gives  $10^6$  different combinations of possible bit patterns when using a 26-letter alphabet (a–z), which is considered to be attackable using frequency and bit pattern-based attacks (Niedermeier et al. 2014). Therefore, the use of random hashing is recommended.

#### 3.1.4.2 Random hashing

Following the attacks on Bloom filters, Niedermeier et al. (2014) suggested abandoning the double hashing scheme in favour of full random hashing. To determine the bit positions set to 1 ( $h_i$ ) for each element  $x$ , for each possible element,  $k$  random numbers  $h_i \in \{1, \dots, l\}$  for  $i \in \{1, \dots, k\}$  are drawn using a cryptographically secure random stream generator, such as Salsa20 (Bernstein

2008). This generates a table with all the elements  $x_i$  of all the possible elements  $X$  and  $k$  random integers. For each element  $x$ , the bit positions to be set to 1 are then looked up in the encryption table. This table will require a seed to be set so that both the data holders generate the same lookup table. The use of this method increases the number of possible combinations to approximately  $6.8 \cdot 10^{32}$  (Schnell 2016a).

In practice, random hashing is implemented using a seeded pseudo-random number generator (PRNG<sup>3</sup>, Stallings 2014) to generate a sequence  $X$  with length  $k$  for each  $q$ -gram. The seed is shared as a secret between the data holders. Note that the use of different seeds for each identifier used is possible and recommended, particularly for security reasons. As Privacy-preserving Record Linkage does not offer any advantages over the standard Record linkage procedures if it is not secured against privacy breaches, a discussion on the security of Bloom filter-based methods is in order.

## 3.2 Bloom filter-based encryptions and security

As Bloom filters are based on mapping bigrams to a bit vector in a similarity-preserving manner, they are theoretically prone to frequency attacks. However, attacks on them have been reported only recently. Each attack will be presented briefly.

### 3.2.1 Constraint satisfaction solver-based attack

The first published attack on Bloom filters consisting of a single identifier was conceived by Kuzu et al. (2011). Kuzu et al. (2011) used a so-called “constraint satisfaction solver” (CSS) to align frequent identifier values with frequent Bloom filter patterns.

The underlying assumption was that the attacking party had access to the same clear-text data as those used for the encryption. In such a case, attacking the encryption would be useless, as the attacking party is already in possession

---

<sup>3</sup>A PRNG differs from a “true” random number generator insofar that it does not create *actual* random numbers, but numbers *appearing* random (and sometimes satisfying the criteria of special randomness tests).

of the desired data. Therefore, this assumption is not very realistic in practice (Christen et al. 2018). In a follow-up publication, Kuzu et al. (2013) used a more realistic setting. In this publication, the set of names to attack was just composed of 42 names. Even then, the attack was unable to find substantially more names than could be expected when guessing them. In practice, the attack was a simple frequency alignment of frequent Bloom filter patterns and the most common names. This attack has not been replicated yet.

### 3.2.2 Substitution cipher-based frequency attack

The first attack to result in actual clear-text values decrypted from Bloom filters was devised by Niedermeyer et al. (2014). It only works on Bloom filters created with the double hashing scheme, as it exploits the limited number of patterns that this hashing scheme can generate. An extension can attack composite Bloom filters (CLKs) as well (Kroll/Steinmetzer 2015).

When all identifiers are split into bigrams, each bigram will generate a Bloom filter pattern consisting of  $k$  bit positions set to 1. This is called an *atom*. A matrix  $D$  is built consisting of the relative frequencies of the co-occurrences of bit positions. A reference dataset consisting of training data with the relative bigram frequencies of multiple identifiers of the same population (not necessarily the same database) is required. This gives the matrix  $E$ . An optimisation algorithm (Jakobsen 1995) is then used to minimise the distance between  $D$  and  $E$ . Given the new distance matrix of the distance between  $D$  and  $E$ , the most frequent atoms are now aligned to the most frequent co-occurring bigrams. In the automated solution of this problem (Kroll/Steinmetzer 2015), possible combinations are automatically found.

Kroll/Steinmetzer (2015) reported 44% correctly identified attribute values for an attack. The method has two drawbacks (Christen et al. 2018): it is computationally expensive, requiring days of computing time and lots of fine-tuning, and it only works for Bloom filters using the double-hashing scheme. Simply using random hashing prevents the attack; furthermore, no single attribute can be decrypted using any tested hardening method (Schnell/Borgs 2016b), as will be described in Section 3.3.

### 3.2.3 Frequency-based and pattern mining-based attack

The newest attacks on Bloom filters were published by Christen et al. (2018), using pattern mining to decrypt Bloom filters based on pattern frequencies, and (Christen et al. 2017), where the correspondence of frequent identifier combinations and resulting bit vectors is exploited.

The latter (Christen et al. 2017) frequency-based attack uses the same principles as the earlier attacks: re-aligning frequent values of identifiers to frequent Bloom filters. This requires a particular frequency of Bloom filters with the same value, as well as frequent identifier values. Thus, only single BFs for single identifiers can be attacked, as most CLKs will create unique patterns unless the population is either very homogeneous or very large. Another obstacle for this attack is salted BFs: Using a salt considerably reduces the number of frequent BFs. This will prevent the attack if the identifier used for the salting has a sufficiently high discrepancy power.

To attack CLKs and even hardened BFs, the pattern mining-based attack by Christen et al. (2018) works on composite BFs and does not require frequent BF patterns to occur (it works even for unique BF patterns). It uses pattern-mining techniques to determine the frequent patterns in the bit vector. These are then aligned to the frequent  $q$ -grams in the reference data. The alignment then gives a clear-text decryption, which works well using training data. For a more realistic setting, Ranbaduge et al. (2018) reported results of the two latest attacks with respect to the standard methods as well as the hardening methods for actual data. First, the hardening methods will be covered in detail, while reporting on the results of the attacks they can or cannot prevent.

## 3.3 Hardening methods

Based on the security concerns discussed in the last section, several extensions and modifications have been proposed. Each method will be discussed briefly. Note that, except for Record-Level Bloom filters (see Section 3.3.6), all the methods work for both separate Bloom filters and CLKs.

While the attack by Niedermeyer et al. (2014) fails for every hardening method, as demonstrated by Schnell/Borgs (2016b), further hardening meth-

ods are required to prevent frequency- or pattern-based attacks (Christen et al. 2017; Christen et al. 2018).

Recently, several hardening methods presented below have been tested by Ranbaduge et al. (2018) using the most recent attacks (Christen et al. 2017; Christen et al. 2018). For balanced Bloom filters, vector/XOR-folded BFs, BLIP randomised response Bloom filters and salting, the attacks using the first attack were not successful at all (Christen et al. 2017), while only up to 5% of the attacked Bloom filters could be successfully decrypted by the current method (Christen et al. 2018). The exact numbers for each hardening method have not been reported yet. However, note that even though the results from Ranbaduge et al. (2018) may show no successful attacks, this could be attributed to implementation details, as theoretically, frequent co-occurrences of pattern frequencies will be difficult to prevent.

### 3.3.1 Random hashing

As discussed in Section 3.1.4.2, replacing double hashing with full random hashing will prevent the attack by Niedermeyer et al. (2014). However, when no other hardening method is used, it will not prevent the attack by Christen et al. (2018). As it will not change the expected linkage quality, random hashing should always be used instead of double hashing. In any instance where the hardening method modifies an existing Bloom filter, random hashing should be used preferably to double hashing before applying any further hardening methods.

### 3.3.2 Random bits

Random noise is frequently added to data in statistical disclosure control to improve privacy (Abril et al. 2012). A simple method of adding noise to Bloom filters is to set a certain number of bit positions to 1 randomly. This was first demonstrated by Schnell (2015). Schnell (2015) showed that adding up to 5% random bits (which would mean adding 50 random ones to a Bloom filter of a length of  $l = 1000$ ) still yielded feasible linkage quality. The addition of random bits has not been tested with any attack thus far. However, other methods may be more secure while ensuring better linkage quality.

### 3.3.3 Salting

In computer science, a *salt* is defined as an extension of a password using additional (sometimes random) strings. These extended passwords are used for hash functions and were first used to avoid dictionary attacks (Morris/Thompson 1979). “Salted” hash functions are essentially the same as “keyed” hash functions, as described in Section 2.2.

<b>41234</b> , James Sullivan, 20.01.1976	<b>41234</b> , James Sullivan, 20.01.1999
41234 1976	41234 1999
cf593781c6233a82ebdf0083c268d5b408aa	7deef9ed5b83e837a5310c69ec77f30b39c19

Figure 3.3: Hashing the unique PID 41234 using SHA-1 with the year of birth as a salt.

Figure 3.3 visualises the general idea: By simply concatenating the identifier with the salt, we can offset the resulting hash values. The idea to use the date of birth as a salt for Bloom filters was suggested by Niedermeyer et al. (2014) to increase the uniqueness of the bit patterns, which prevents the Niedermeyer attack. Salting makes the pattern-mining attack by Christen et al. (2018) more difficult (but will not prevent it) and leads to enhanced linkage quality because of higher precision, should the identifier used for salting be stable and free of errors. In practice, using a salt will modify the “password” used as a seed for a PRNG by concatenating the key and the identifier.

### 3.3.4 BLIP/RAPPOR

A more sophisticated approach was the idea to use a randomised response technique, for example, used in survey methodology to ask sensitive questions in a privacy-preserving manner (first devised by Warner (1965), for an overview, see Schnell (2012)), to alter the bit pattern of a standard Bloom filter. The first application was described in Alaggan et al. (2012). A variant of Bloom filters with randomised response modifications is used in the Google Chrome browser, where the method is called RAPPOR (Erlingsson et al. 2014). The idea to use this for PPRL was first discussed by Schnell/Borgs (2016b).

For each bit value  $B_i$  in the original Bloom filter, a new value  $B'_i$  is set, de-

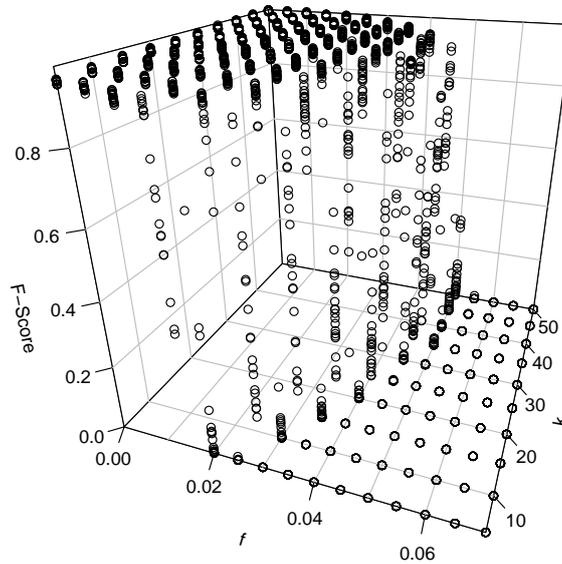


Figure 3.4: Linkage quality of BLIP/RAPPOR depending on the number of hash functions  $k$  and the bit-flipping probability  $f$ . Figure taken from Schnell/Borgs (2016b).

pending on a “flipping-probability”  $f$ :

$$B'_i = \begin{cases} 1 & \text{with probability } \frac{1}{2}f \\ 0 & \text{with probability } \frac{1}{2}f \\ B_i & \text{with probability } 1 - f \end{cases} \quad (3.8)$$

Of course, linkage quality and security are heavily dependent on the probability  $f$ , as can be seen in Figure 3.4. It is possible that acceptable privacy levels will yield unfavourable linkage quality because  $f$  has to be set very high. Randomised response Bloom filters were not successfully attacked in Ranbaduge et al. (2018).

Recently, an updated version of this hardening technique has been proposed by Vaiwsri et al. (2019). Here, the bit-flipping mechanism is modified to include a set of global reference values. For each identifier in the records, the most similar reference values are used as a seed for the random generator which determines the new bit value. Thus, the bit-flipping should be similarity-

preserving, reducing the problem of lower linkage quality when choosing higher flipping probabilities  $f$ . As the results by Vaiwsri et al. (2019) show, the linkage quality is markedly improved by using this approach, particularly when selecting higher values for  $f$ . Additionally, no current attack has been reported to break this method.

### 3.3.5 Rehashing

Rehashing, as discussed in Schnell (2016a), works by first generating a new, empty Bloom filter. This new Bloom filter is populated by setting a second set of randomly chosen bit positions to a value of 1. The pseudo-random number generator to do this uses a seed value that is determined by the numeric representation of substrings of the conventional BF. These sub-strings are generated by sliding a “window” over the Bloom filter of the first step (see Figure 3.5). The quality and privacy implications vary considerably, depending on the size of the window (window size  $w$ ), the number of bit positions to skip before constructing a new window (step size  $s$ ) and the number of bits set to 1 by a single window ( $k_{rehash}$ ). Rehashing has not been tested using any attack thus far.

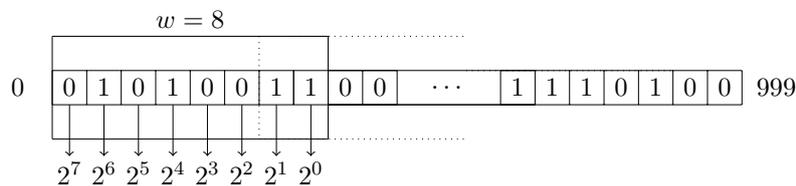


Figure 3.5: Rehashing using a sliding window of size  $w = 8$ . The window is used as a binary representation of an integer, which is used as a seed for a PRNG that samples  $k_{rehash}$  new bit positions that are set to 1 in an empty Bloom filter. Figure taken from Schnell (2016a).

The main drawback of this method is the varying linkage quality depending on the window size and the rehashed bits Schnell (2016a). As there is no ready recommendation for its parameters, optimising these parameters is an open research question.

### 3.3.6 Record-Level Bloom filters

Record-Level Bloom filters (RBFs) were first proposed by Durham (2012), combining multiple Bloom filters of several identifiers into a single bit vector. The idea to combine BFs into a single bit vector (in a different way) was already published at that time (Schnell et al. 2011). There are four variants of building RBFs, as described in Durham (2012):

- Static BF size, uniform sampling
- Static BF size, weighted sampling
- Dynamic BF size, uniform sampling
- Dynamic BF size, weighted sampling

The simplest version of building an RBF is to use a *static* Bloom filter size for all the identifiers to be encrypted into separate Bloom filters. This way, example identifiers such as the first and the last name and the date of birth would be built using the same Bloom filter length of, e.g.  $l = 500$ . From these Bloom filters, a RBF is built by sampling the same number of bits (*uniform*). The resulting samples are then concatenated and shuffled. *Dynamic* variants use the average number of  $q$ -grams in each identifier as a modifier to dynamically size the length of each identifier's Bloom filter. *Weighted* sampling uses the  $m$ - and  $u$ -weights from the EM algorithm (see Section 1.5.5.4) to determine the number of bits sampled from each original Bloom filter. In any case, the final concatenation of the sampled bits of the original Bloom filters is shuffled. No systematic evaluation of the linkage quality of the four methods has been published thus far. The same is true for their cryptographic properties and their resilience against attacks.

### 3.3.7 Balanced Bloom filters

Balanced Bloom filters work by appending a negated copy (a NOT operation on the bits, see Section 2.2) of itself to the original Bloom filter. The resulting Bloom filter of length  $2 * l$  is then shuffled using a seed shared between the data holders (Schnell/Borgs 2016b). This will give a constant Hamming weight (or cardinality) of  $l$ , which is the number of bit positions set to 1 in a Bloom filter (the concept of cardinality is discussed in Section 3.5.3). As the

attack by Niedermeyer et al. (2014) is in part dependent on differing Hamming weights, this simple variant is aimed at making the attack less efficient without sacrificing the linkage quality. However, a drawback is the doubled size of the Bloom filters, leading to higher memory use and computational time (see Section 4.3.1 for an evaluation of time and memory required when  $l$  is varied). While simple, balancing will make all the current attacks considerably more difficult, as reported by Ranbaduge et al. (2018).

### 3.3.8 Vector folding

First published in Schnell/Borgs (2016c) for use with Bloom filter (BF)s, vector folding is usually performed to compress bit vectors to speed up the searches of chemical databases (Baldi et al. 2008). The idea is to use XOR folding on Bloom filters, as is depicted in Figure 3.6.

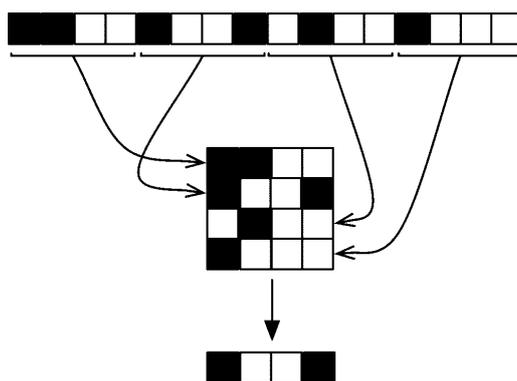


Figure 3.6: Four-time XOR folding of a bit vector of length  $l = 16$ . Figure taken from Schnell/Borgs (2016c).

Here, the original Bloom filter is split into four parts. The resulting splits are combined using the XOR operation.<sup>4</sup> The resulting folded BF is smaller in length. Schnell/Borgs (2016c) showed only a slight decrease in the linkage quality for a single folding step. This hardening method could not be successfully attacked according to Ranbaduge et al. (2018).

<sup>4</sup>For more information on bitwise operators (including XOR), see Section 2.2.

### 3.3.9 Markov chain Bloom filters (MCBFs)

All attacks known thus far are based on the underlying frequencies of the  $q$ -grams of the clear text. To make these attacks more difficult, artificial  $q$ -grams could be added. As adding random  $q$ -grams would be akin to merely inserting random bits, they must be added in a similarity-preserving manner, which would also preserve the linkage quality. For this, the follow-up probability for each given  $q$ -gram is approximated using a Markov-chain-based estimator. This idea was first presented at talks (Schnell 2017b) and first published in Schnell/Borgs (2018c). An example would be the name THEADORE. When the name is split into subsets of length  $q = 2$  (bigrams), starting with the first bigram, TH, the transition probabilities for all the possible follow-up bigrams are calculated.

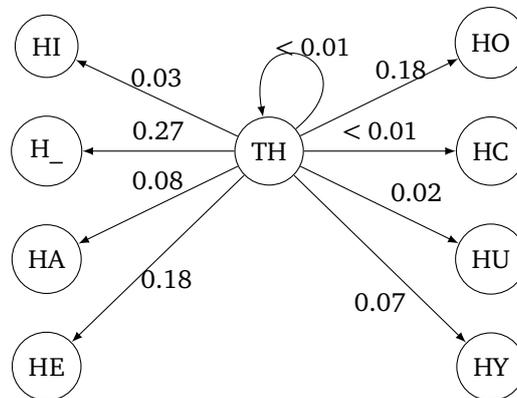


Figure 3.7: Subset of the transition graph for bigrams following TH. Figure taken from Schnell/Borgs (2018c).

Using the transition probabilities as a sampling fraction (see Figure 3.7), we can draw  $cl$  new bigrams for each bigram in the name. As an example, suppose four ( $cl = 4$ ) new bigrams are drawn this way, e.g. HO HI HY H\_. These are added to the list of actual bigrams. The final expanded bigram list is then hashed into the Bloom filter by using the standard approach. As this idea is relatively new, it has not been tested with any published attacks as yet.

### 3.3.10 Cellular automata

Wolfram (1986) first suggested using cellular automata for cryptography. A cellular automaton is a set of transformation rules for bit patterns depending

on the neighbouring states (0 or 1) of a cell. Repeating these transformations yields interesting new patterns (Wolfram 2002). Schnell (2017b) first suggested using one of the rules (Rule 90) on Bloom filters to enhance the security in Privacy-preserving Record Linkage. Rule 90 is non-reversible (meaning that the input cannot be derived from the output), which should enhance the resilience against bit pattern-based attacks. This has not been tested thus far.

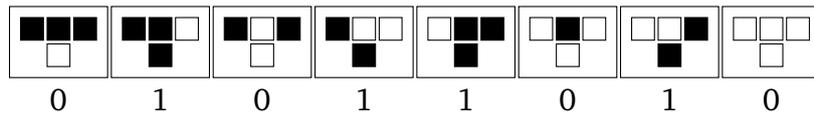


Figure 3.8: Replacement rules for Rule 90. Figure taken from Schnell/Borgs (2018a).

Figure 3.8 shows the replacement rules for Rule 90, where black boxes correspond to a bit position with a value of 1, and non-filled boxes denote an empty (0) bit position. Thus, similar input bit patterns in Bloom filters should lead to similar replacement patterns in the modified Rule 90 Bloom filter. This should be similarity preserving, which would minimise the impact on linkage quality. It was shown that Rule 90 shows good linkage quality (Schnell/Borgs 2018a), while the current attacks, as tested by Ranbaduge et al. (2018), were largely unsuccessful in decrypting the underlying clear-text records.

### 3.3.11 One-time pad-secured Bloom filters

This method was suggested by Schroeder (2012). After standardisation, a string is converted into a sequence of numbers that corresponds to the position of the characters in a reference alphabet. The reference alphabet can be a simple encoding table, mapping numeric values to characters. A *true* random generator will now generate a *one-time pad* of the string. One-time pads denote encryptions with a truly random number (the *pad*). Without knowledge of this random number, the encryption cannot be attacked (Denning 1982). The resulting string is again converted to a set of positions in the reference alphabet. For each position of the original string, the position numbers are now added to those of the one-time pad. The modulo of the length of the reference alphabet is then formed. The resulting numbers are now converted back to letters by using the reference alphabet.

The string generated in this way is denoted as  $a'$ , which is the result of an element  $a$  from the dataset  $A$ . The encrypted string  $b'$ , is formed from the

original element  $b$  from the dataset  $B$ . The Jaro-Winkler string similarity of  $a'$  and  $b'$  now corresponds to the similarity of  $a$  and  $b$  (Schroeder 2012: 3). This remarkable property can be used to either link the resulting strings directly using probabilistic Record Linkage methods or to encrypt them using Bloom filters, as the similarities of the strings are retained. This implies that any successful attack on Bloom filters will no longer reveal any privacy-relevant features.

The main caveat of this method is due to errors or name variants that lead to differing string lengths: the use of a one-time pad to offset the bit positions generated by the  $q$ -grams will lead to string similarities of zero (leading to a missed record pair) if the error type changes the string positions of the follow-up characters. This would be true for deletion, insertion and (most) phonetic errors. It is unclear whether this problem can be solved at all. If it can be solved, and if a true one-time pad is used, this method is provably similarity preserving and theoretically unbreakable.<sup>5</sup>

### 3.3.12 MinHash Bloom filters (MHBF)

First devised by Smith (2017), MHBFs are based on the possibility of estimating a Jaccard similarity of two strings by using several minwise hashes (MinHash, Smith/Shlomo 2014). To adapt MinHash to Bloom filters, several steps are required. First, a random key is hashed and represented as a bit vector. This bit vector is split into sub-parts of a certain length (i.e.  $4 * 8$  bit splits). For each subpart, a randomly filled lookup table is built. This table contains the hash values.

For each element (e.g. a single  $q$ -gram), this is repeated  $k$  independent times, where  $k$  is the desired number of hash functions. The minimum hash value of each of the  $k$  steps is retained and XORed with  $l$  random numbers, where  $l$  is the desired output bit vector length. The resulting least significant bit of each resulting XORed number is always either 0 or 1. This gives  $l$  independent but similarity-preserving bit values.

Empirical tests for this approach have not been published yet. It is theoretically secure if the independence assumption of each bit position holds. However,

---

<sup>5</sup>Of course, the Bloom filter encryption itself can be broken. However, the attacker would not learn any information from this, as the string creating the Bloom filter itself is encrypted in a way that makes it impossible to decipher.

as this method retains similarities, approaches based on the frequencies of the underlying clear-text data could still work. This is subject to further research.

### 3.3.13 Summary

Various hardening methods for Bloom filter-based methods have been proposed in the recent five years. Nearly all of them prevent the first-generation attacks. However, the new attacks by Christen et al. (2018) have the potential to decrypt almost all the possible hardening methods proposed thus far. This is still subject to more research, as a systematic evaluation of hardening methods, linkage quality and vulnerability to the new attacks has not been done yet.

With security in mind, a short overview of linking Bloom filter-encrypted methods in the PPRL framework will be given, before presenting an overview of the properties of Bloom filters relevant to the following chapters.

## 3.4 Linking encryptions based on Bloom filters

The most desirable property of Bloom filters is that encrypted record pairs (approximately) retain their similarity. This also enables them to be linked using different methods other than exact matching, as would be the case with hash strings, i.e. ALCs or ESLs. Therefore, the two most widespread techniques of incorporating Bloom filters into the Record Linkage process will be discussed briefly.

### 3.4.1 Using Bloom filters with probabilistic Record Linkage

As discussed in Section 2.3.6, the Dice coefficient can be used as an approximation for the clear-text Dice string similarity using the BF:

$$D_{A,B} = \frac{2|A \cap B|}{|A| + |B|}. \quad (3.9)$$

Another alternative is the Tanimoto similarity coefficient  $T$ , which is shown in Equation 3.10. Both can be used in conjunction with probabilistic linkage methods, as described in Section 1.5.5.4. For clear-text comparisons, a string similarity between two identifiers would be calculated using a similarity measure such as the Jaro-Winkler similarity. This can be approximated using the Dice or Tanimoto similarity on the Bloom filters. Thus, BFs can be used in any existing probabilistic linkage setup.

### 3.4.2 Multibit trees

First suggested by Kristensen et al. (2010) for chemoinformatics and adapted by Schnell (2013) for PPR, Multibit trees work exclusively with bit vectors.

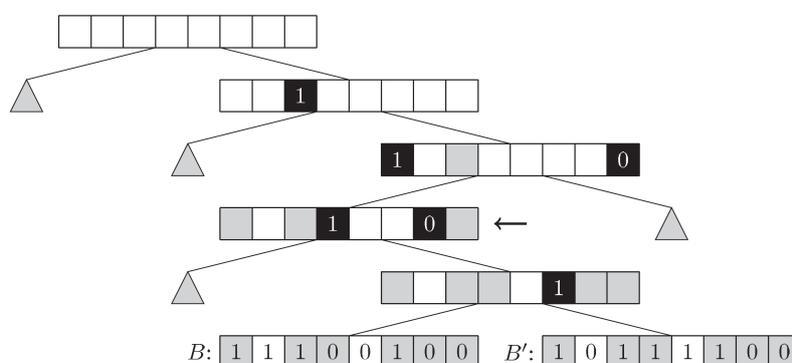


Figure 3.9: Multibit tree construction as shown in Kristensen et al. (2010: 13). Black squares show the match bits, and grey squares show the previous match bits from previous leaves. The grey triangles are place-holders for the respective other sides (branches) of the sub-trees.

To build a tree-based index, trees are created for each cardinality (number of bits set to 1). Each tree consists of *leaves*, where each branch consists of roughly half the records. Thus, the input vectors are split into two groups for every leaf built. The grouping is done on *match bits*, where approximately half of the records exhibit a value of one, while the other half of the records have a value of zero (this is called the *split-half technique* and is the method used by Kristensen et al. (2010)). This is repeated until the lowest leaf level consists of a user-set number of records (*leaf limit*; three in this thesis, see Table 4.3). A simplified version is shown in Figure 3.9, where two records  $B$  and  $B'$  are in different leaves after the fourth leaf level. Thus, only one record is part of a sub-leaf, which would correspond to a *leaf limit* of 1.

Searching for the nearest neighbours with Multibit trees is then restricted to trees with a similar cardinality (Kristensen et al. 2010). For any given query vector, an upper similarity bound for all the leaves in a tree based on the values of their respective match bit positions (the grey squares in Figure 3.9) can be estimated (Swamidass/Baldi 2007). Only trees with an upper bound higher than a user-defined similarity threshold will be queried.

This decreases the computational time required by reducing the search space (this will be covered experimentally in Section 4.4.2). Here, the Tanimoto similarity  $T$  is used as a similarity measure.  $T$  is calculated by dividing the number of bits set to 1 in both bit vectors  $A$  and  $B$  by the total number of 1-bits in  $A$  and  $B$ :

$$T(A, B) = \frac{\sum_i (A_i \wedge B_i)}{\sum_i (A_i \vee B_i)}. \quad (3.10)$$

The resulting record pairs above the user-defined Tanimoto similarity threshold are then given as an output. Thus, blocking and linking are combined into a single step. Lowering the threshold will consider more sub-trees and leaves that are more dissimilar to the query vector.

In practice, this will tolerate more differences in the bit vectors, leading to a higher tolerance of small deviations, such as typos or different birth years. Furthermore, lowering the threshold will increase the computing time required, as more candidate bit vectors will have to be queried. As less similar vectors will be considered matches, the false positive rate will increase with lower thresholds, while the number of true matches will usually increase as well. On average, MBTs can rapidly link large files with very good linkage quality (Brown et al. 2017a).

Prominent extensions of Multibit trees are Succinct Multibit trees (Tabei 2012), which supposedly reduce the memory required for linking and blocking data, and Symdex preprocessing, which is supposed to decrease the time taken to link by excluding dissimilar trees using a type of pre-filtering (Tai/Fang 2012). Tai/Fang (2012) also introduced UnionBit Trees, which, together with Symdex preprocessing, show slightly improved performance over standard Multibit trees in terms of the computing time when using lower similarity thresholds (see Figure 3.10<sup>6</sup>). In particular, Symdex preprocessing seems to considerably reduce the computing time required when lowering the Tanimoto

---

<sup>6</sup>This result has not been published yet.

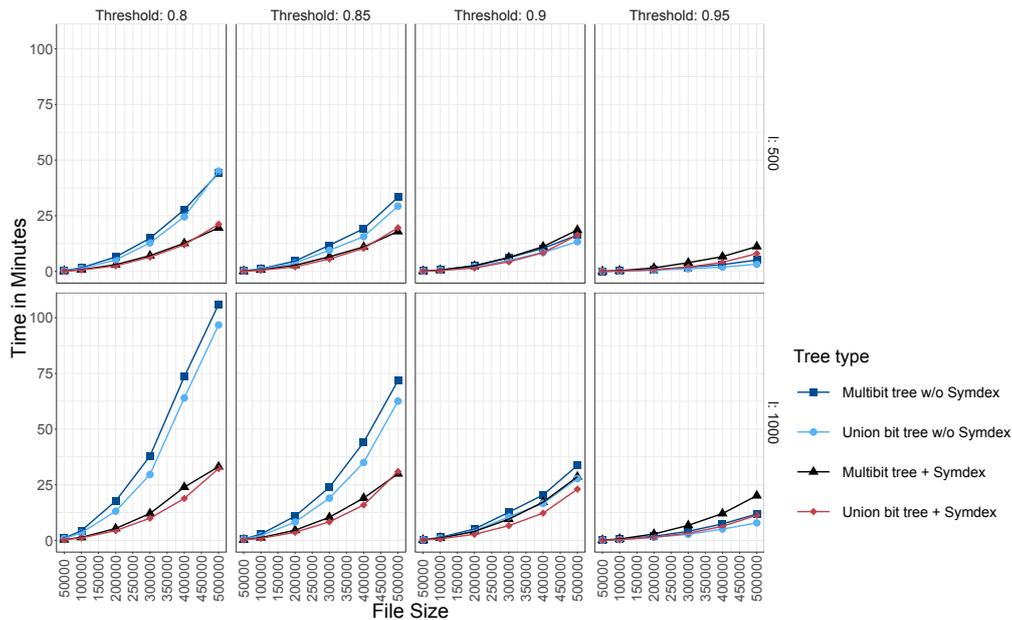


Figure 3.10: Multibit trees tested against union bit trees, both with and without Symdex preprocessing using synthetic data as described in Section 3.5.3.1, with differing lengths  $l$  and  $k_l = \frac{k=20}{l=1000}$  hash functions.

threshold below 0.9. This warrants further research, which is out of the scope of this thesis, however.

With the security discussion mind, and after the short overview of linking methods, a few properties of Bloom filters will have to be tested before tackling the optimal parameter choice itself.

## 3.5 Properties of Bloom filters

This thesis is concerned with finding optimal parameter choices for encrypting Bloom filters so that the best possible linkage quality can be achieved. For this, some properties of the Bloom filters are important:

1. Will repeating the generation of Bloom filters with different passwords for the same data lead to varying linkage quality? If so, the results would have to be reported with confidence intervals to consider the variance of the linkage quality.
2. What effects do different identifier and the parameter choices have on linkage quality?

3. With the given parameter choices, how many bits are set to 1 in a Bloom filter and how does this affect the linkage quality? Which fraction of bits set to 1 is optimal?

For the first property (1), a small experiment will be conducted. The importance of parameter choice (2) will be highlighted again in the subsection following (1), with the effect of the number of bits set to 1 (3) following last. This too will be part of an experiment.

Finally, the implications of the properties of Bloom filters for the optimal parameter choice will be discussed.

### 3.5.1 Variance of linkage quality with repeated Bloom filter generation

Bloom filters are created by hashing elements into a bit vector using  $k$  hash functions. Usually, these hash functions are *keyed* using a secret shared key (a password) that all the parties involved share, guaranteeing the same bit patterns in the Bloom filters for the same input values, meaning that the same records will result in the same bit vectors as long as the password is the same. If the linkage quality varies considerably when using different passwords, this would have to be kept in mind while reporting the linkage quality results. It is possible that the number of collisions (bit positions set to 1 by more than one input) varies when different passwords are used, giving different linkage quality for different passwords.

Table 3.2: Mean precision/recall and confidence intervals for three  $k$  hash functions for linking Bloom filters ( $l = 500$ ) using the same data encrypted using different passwords  $n = 30$  times.

$k$	Mean	CI <sub>low</sub>	CI <sub>low</sub>	SE	$n$
5	0.923	0.922	0.924	0.00038	30
10	0.934	0.933	0.935	0.00040	30
15	0.943	0.943	0.944	0.00024	30

Thus far, no publication has quantified the variance of the results in terms of the linkage quality when replicating them using different passwords for each replication. To remedy this shortcoming, with the synthetic data described in Section 3.5.3.1, linkage using Multibit trees was conducted  $n = 30$  times using

the same input data and the same encryption parameters. If different passwords lead to vastly different bit patterns, more hash functions could amplify this mechanism. To check for a systematic effect of the number of hash functions  $k$  on the variance of the resulting mean prec./rec., three values for  $k$  (5, 10 and 15) were tested on Bloom filter (BF)s with a length of  $l = 256$ .

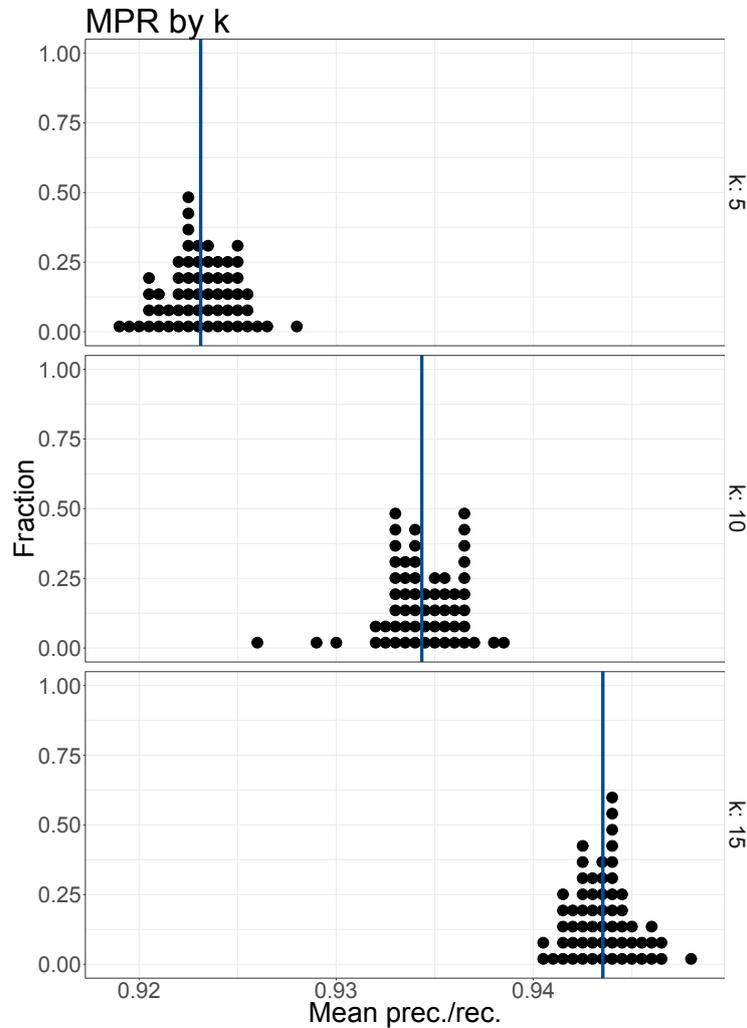


Figure 3.11: Distribution of the mean prec./rec. when using  $n = 30$  different passwords to hash the same data into Bloom filters using three different  $k$ . The blue lines are the respective MPR means for each  $k$ .

Figures 3.11 and 3.12 show the distribution of the mean prec./rec. and Q-Q plots of the resulting  $n = 30$  differing values for the mean of precision/recall. The distribution of these values obtained by linking the same files with different passwords follows a normal distribution with small variance.

As Table 3.2 shows, the variance caused by multiple encryptions is negligible, irrespective of the number of hash functions. This results in very small con-

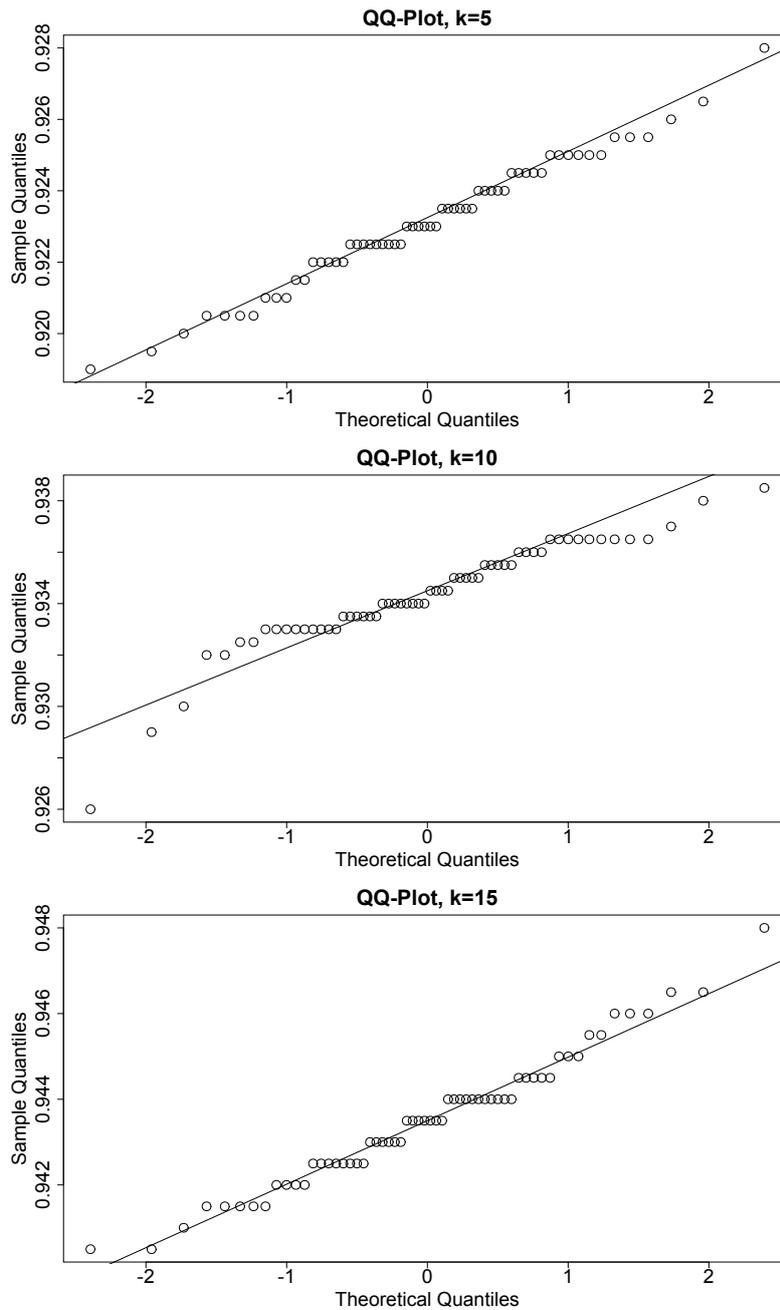


Figure 3.12: Q-Q plots of the mean prec./rec. when using  $n = 30$  different passwords to hash the same data into Bloom filters using three different  $k$ .

fidence intervals, irrespective of  $k$ . Therefore, reporting confidence intervals for single linkage quality results will be omitted. Of course, when multiple datasets are used with the same technique, intervals will be reported.

### 3.5.2 Parameter and identifier choices

As was first discussed in the Introduction, Bloom filter-based methods are susceptible to parameter and identifier choices (Schnell/Borgs 2015). An example using Australian health data can be seen in Figure 3.13, where different identifier sets yield very different linkage qualities. An overview of the identifiers used in Figure 3.13 for the encryption is given in Table 3.3.

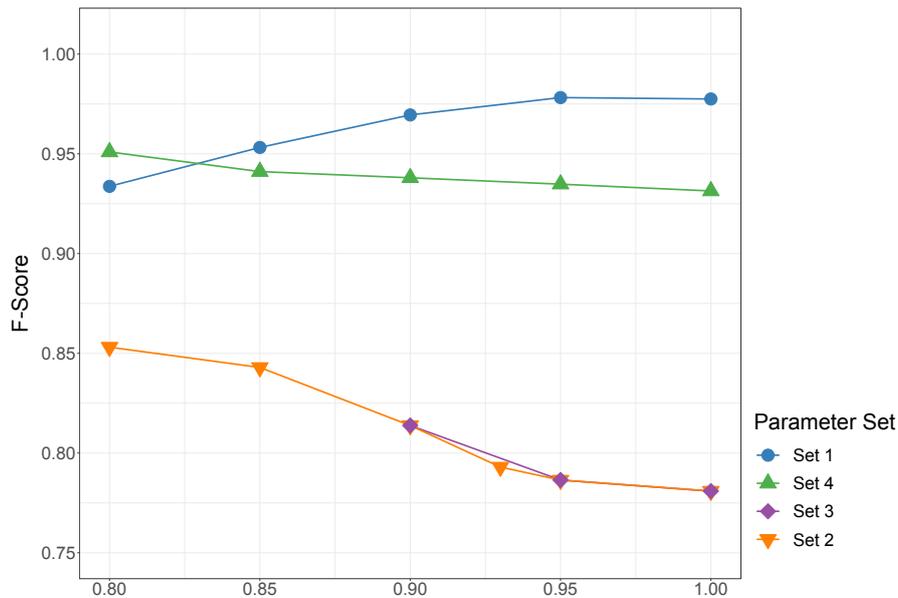


Figure 3.13: Different identifier sets used to encrypt CLKs can yield very different results in terms of linkage quality. Results from Brown et al. (2017a). Note that the F-Score used by Brown et al. (2017a) is the harmonic mean of precision and recall (see Section 1.5.7).

The drastic differences highlight the potential problem of merely adding as many identifiers as possible to the identifier set: choosing unstable identifiers prone to error or exhibiting large fractions of missing data will lead to reduced recall (Schnell et al. 2017). In some special cases, where identifiers contain many elements ( $q$ -grams) with only a few variations across the data, an increase in false positives is possible as well, decreasing the precision. In this particular case, the identifier set with the *least* number of identifiers performed the best. This implied that adding more information into the Bloom filter by encrypting more identifiers did *not* increase the linkage quality. This could be attributed to the fact that address data is notoriously unstable as people may have different addresses in different databases. This is particularly true

Table 3.3: Identifiers used for each parameter set presented in Figure 3.13. Table from Brown et al. (2017a).

Identificator	Parameter Sets			
	Parameter Set 1	Parameter Set 2	Parameter Set 3	Parameter Set 4
First Name	✓	✓	✓	✓
Middle Name	✗	✓	✗	✓
Last Name	✓	✓	✓	✓
Date of birth	✓	✓	✓	✓
Sex	✓	✓	✓	✓
Suburb	✗	✓	✓	✗
Address	✗	✓	✗	✗
Postcode	✗	✓	✓	✗

when there is some time between the data recording process between the two databases, as people may have moved their places of residence. Moreover, the storage formats of addresses may vary wildly. Finally, identifier set 4 shows that adding the middle name reduced the linkage quality. This may be attributed to the fact that people’s middle names are not recorded in some databases or abbreviations of these names were recorded.

This highlights another main focus of this work: Choosing appropriate identifiers for linking data is at least as important as predicting the right parameter choices (such as the number of hash functions).

It has to be kept in mind that even with the same identifier and parameter choices, datasets can differ in linkage quality, as shown in Figure 3.14. This can be attributed to more errors in the data, less entropy in the identifiers, a large number of very frequent combinations of names and birth dates (which would yield many false positive matches) or missing data problems. Different pre-linkage standardisations can also be a factor.

In the case of Figure 3.14, a possible explanation is that the error generation of FEBRL (Christen 2008) leads to more errors in the corrupted data than the error generation of the German dataset. One of the major obstacles in determining the best possible parameters and identifier choices is to find a way to estimate the number of errors in the data without any a priori knowledge of the data generation process. This will be discussed in the next chapter. Another vital measure when comparing the effect of the number of hash functions on the linkage quality is the resulting fractions of bits set to 1, as there is a theory

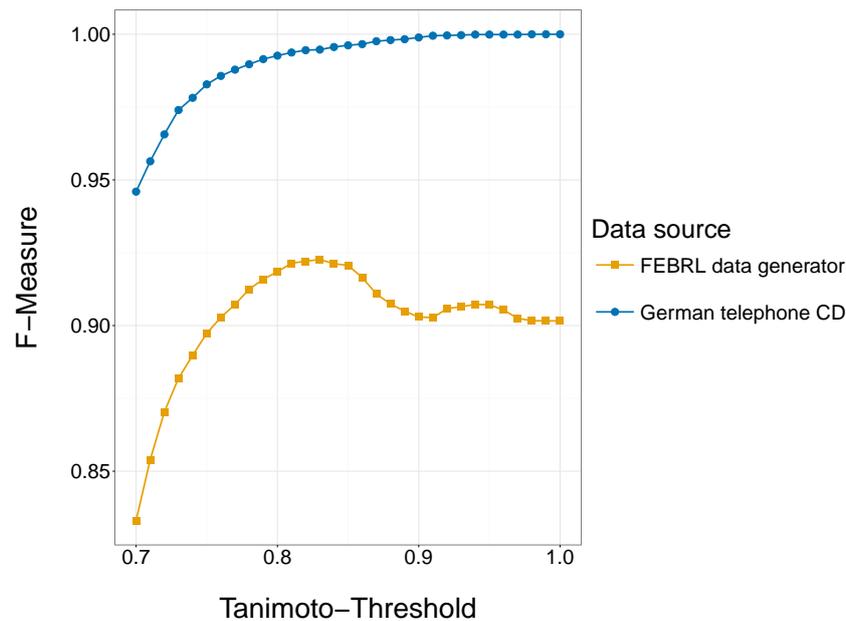


Figure 3.14: FEBRL-generated data and German telephone CD data encrypted with the same parameters yield different linkage quality based on their respective characteristics. Image first used in Borgs/Schnell (2017). Note that the F-Score used by Borgs/Schnell (2017) is the harmonic mean of precision and recall (see Section 1.5.7).

about the optimal ratio. This will be discussed in the following section.

### 3.5.3 Bits set and optimal parameters of Bloom filters

In theory, having 50% of bits set to 1 in a Bloom filter should be optimal in terms of the false positive rate (Bloom 1970: 422; Kirsch/Mitzenmacher 2006). The fraction of bits set to 1 is sometimes referred to as “fill”. In technical terms, the number of bits set to 1 is the *Hamming weight* (or *cardinality*) of a Bloom filter, as, for example, used in building sub-trees for Multibit trees (Kristensen et al. 2010).

#### *Optimality and optimal linkage quality*

Two problems are important to keep in mind when discussing optimal linkage quality and optimality in terms of the false positive rate. First, the calculation of the optimal number of hash functions  $k$  required to have approximately half the bit positions filled with ones requires two parameters: The length

of the bit vectors  $l$  and the number of elements  $m$  to be hashed into the BF (Kirsch/Mitzenmacher 2006):

$$k_{opt} = \frac{l}{m} * \log(2). \quad (3.11)$$

While the length of a Bloom filter is usually constant for all records, the number of elements varies in the usual Privacy-preserving Record Linkage setting: Names have different lengths, addresses vary considerably in length, and some fields (such as academic titles) may even be empty for most records. Thus, an average has to be calculated, which may be non-optimal for many input records.

The second problem is that, despite having half of the bit positions set to 1 in a Bloom filter, optimal linkage quality may not be achieved, as the false positive rate (see Section 1.5.7) is given as the fraction of false positive matches divided by the sum of true negatives and false positives ( $FPR = \frac{FP}{TN+FP}$ ). This implies that it is primarily dependent on the number of false positive classifications. Limiting the FPR is therefore implicitly done by keeping the precision close to 1, as this would mean making next to no false positive classifications. In the original application of Bloom filters, checking set membership, this makes sense: falsely classifying an element as belonging to the same subgroup should be avoided. In Privacy-preserving Record Linkage, however, the goal is to avoid false matches *and* to allow for some error-tolerance (see Section 1.5.5.2). By limiting  $k$  to a setting where false positives are avoided as far as possible, recall could suffer, leading to sub-optimal linkage quality.

This hypothesis seems to be a topic not covered in the literature thus far. As the only example, the author is aware of, Smith (2017: 8) noted that “[experiments] showed that values designed to minimise the false positive rate [...] tended to be very poor for Jaccard score estimation”, without giving further details on either the experiments or the cause for this discrepancy.

To test whether setting 50% of the bits to 1 is not optimal in terms of the linkage quality, a small test setup using synthetic data was devised, before testing the results of this synthetic experiment on actual data. Additionally, the relationship between the number of hash functions  $k$  and the Hamming weight was checked, as one of the central parameter choices is to find a  $k$  with good linkage quality.

### 3.5.3.1 Testing optimality using synthetic data

After the overview of the test setup,  $k$  and the bits set to 1 are evaluated. After this, the effect on linkage quality is evaluated using synthetic data. This is checked using two more datasets (for an overview of all the datasets used, see Section 4.3.3). Finally, the implications of the findings are discussed.

### 3.5.3.2 Test setup

For the tests, the R package `fastdigest` (Becker/Jenkins 2015) was used to create 32-bit hashes of randomly drawn letters. This was repeated  $n = 1,000$  times. The second dataset was a copy of the first with 20% of the rows containing errors, where certain characters of the hash string were replaced. IDs were generated for checking the true match state.

The hashes built this way were split into bigrams ( $q = 2$ ) and hashed into Bloom filters of a length of  $l = 256$  bits using varying numbers of hash functions from  $k = 1$  to  $k = 30$ .

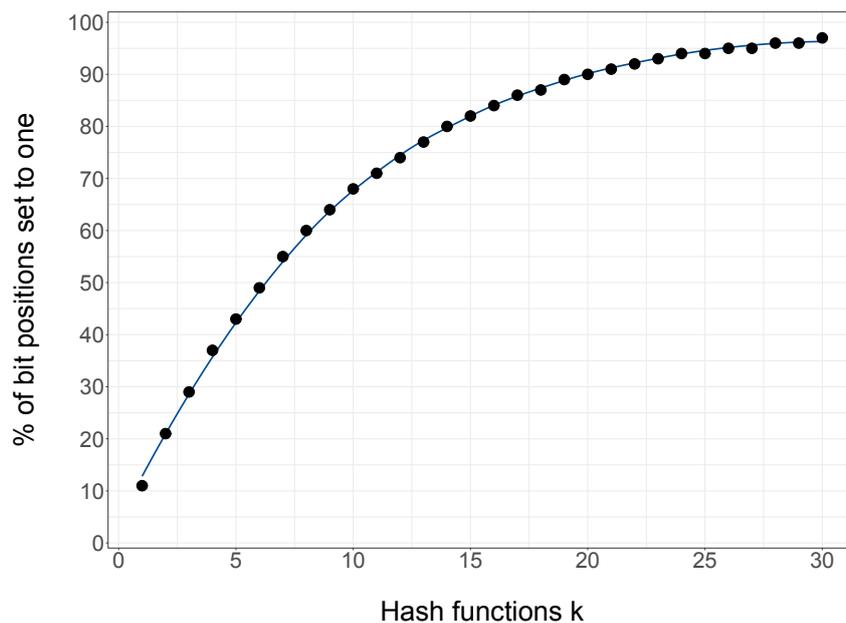


Figure 3.15: Number of hash functions used ( $k$ ) and the resulting percentage of bits set to 1 in a Bloom filter of length  $l = 256$ . The blue line is a loess smoother.

This resulted in the percentage of bit positions set to 1 shown in Figure 3.15. The curve shows a logarithmic growth of the fraction of bits set to 1, caused by the higher number of collisions, where bit positions were set to 1 by multiple

hash functions. Thus, even with  $k = 30$  hash functions and a constant number of 31 hashed bigrams (from the 32-bit fastdigest hash bits), a fill of only approximately 95% was reached.

With this in mind, the effect of the fraction of bit positions set to 1 on the linkage quality will be evaluated. For this, the two synthetic files were linked using Multibit trees with a Tanimoto threshold of  $T = 0.85$ , giving recall, precision and the mean prec./rec.. These were plotted against the percentage of bit positions set. The results were checked with varying datasets to see whether any patterns emerged.

### 3.5.3.3 Results

The assumption stated above was that having 50% of bits set to 1 is the supposed optimal fill for checking the set membership (Bloom 1970: 422). The following hypothesis was put forward: for Bloom filters in PPRL, this will lead to sub-optimal linkage quality as recall could be lower than optimal.

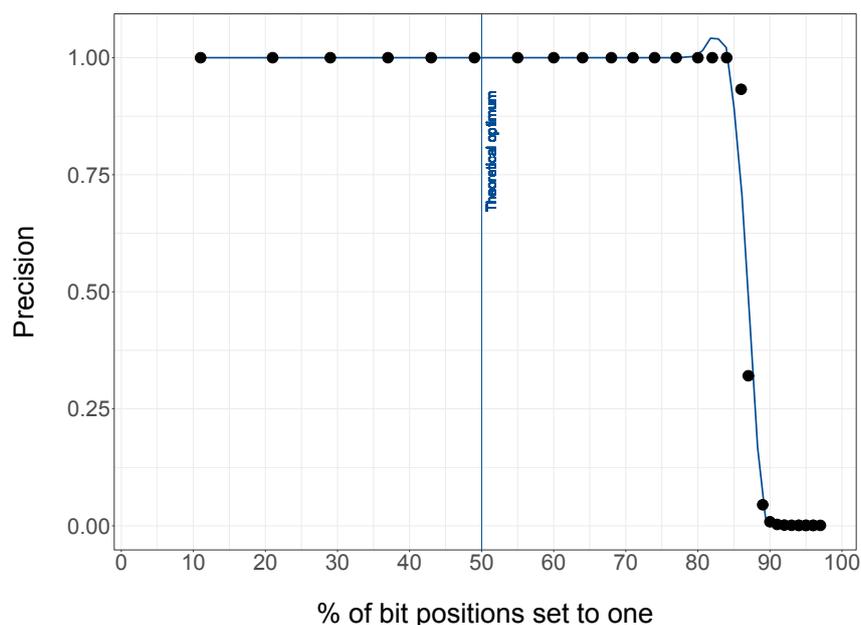


Figure 3.16: Using synthetic data, percentage of bits set to 1 in a Bloom filter plotted against precision. The blue line is a loess smoother.

As Figure 3.16 shows, precision stays stable for a considerably higher percentage of bit positions set to 1 than 50%. At around 85% of bits set to 1, it drops sharply before dropping to approximately zero before 90% fill is reached.

In contrast, as expected, recall increases monotonously with increased fill until it eventually reaches the maximum value of 1 (at around 90% fill), as shown in Figure 3.17.

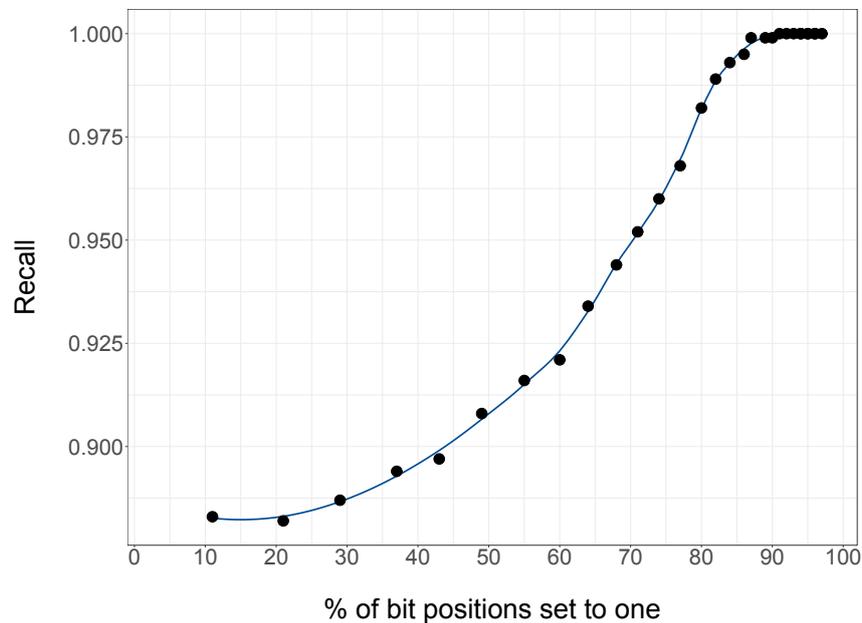


Figure 3.17: Using synthetic data, percentage of bits set to 1 in a Bloom filter plotted against recall. The blue line is a loess smoother.

The unweighted mean of both the measures is plotted in Figure 3.18. As precision is stable up to the point where the number of collisions produces false positive matches, the mean prec./rec. increases slightly with an increasing fill. The peak mean prec./rec. is at 82% fill, after which it drops sharply.

Predicting an optimal fill requires avoiding crossing this particular threshold. Finally, Figure 3.19 clearly demonstrates the problem: an inefficient  $k$  will lead to near-useless results.

The central problem of this thesis is the differing characteristics of datasets, which, using the same parameters, can yield different results. Therefore, the fill,  $k$  hash functions and linkage quality results will be compared using non-synthetic datasets to demonstrate the problem further.

#### 3.5.3.4 Testing using non-synthetic data

For this, the first and last names and dates of birth of the FEBRL and German Telephone CD data (see Section 4.3.3 for an overview of the datasets) were concatenated and used for the encryption (with  $q = 2$  and  $l = 256$ ). Figure 3.20

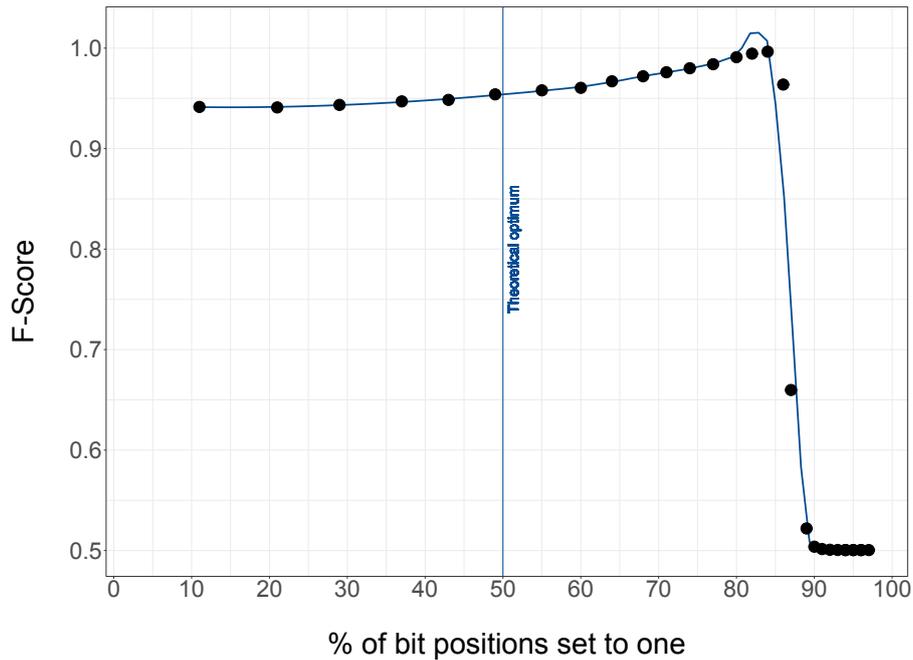


Figure 3.18: Using synthetic data, percentage of bits set to 1 in a Bloom filter plotted against mean prec./rec.. The blue line is a loess smoother.

plots the mean of precision and recall against the percentage of bit positions set to 1. To evaluate the mean prec./rec., Multibit trees were again used with a Tanimoto threshold of  $T = 0.85$ .

The point at which the linkage quality drops differs slightly. At above 75% to 85%, the linkage quality drops significantly because of the increased false positive matches. At 90% bits set to 1, the precision approaches 0, leading to mean prec./rec. of 0.5. Again, for every dataset tested, the ideal fill of the Bloom filter is considerably higher than 50%.

As expected, the same is true for the number of hash functions. For FEBRL data, synthetic data and Telephone CD data, Figure 3.21 shows the optimal  $k$  for  $q = 2$  to be approximately 17, 15 and 19, respectively. Choosing a  $k$  that is too conservative will lead to a reduced recall, while a  $k$  just slightly higher will reduce the linkage quality drastically. As is apparent, setting 50% of the bits to 1 is a very conservative choice, which is *not* optimal for linking population data as done in Privacy-preserving Record Linkage.

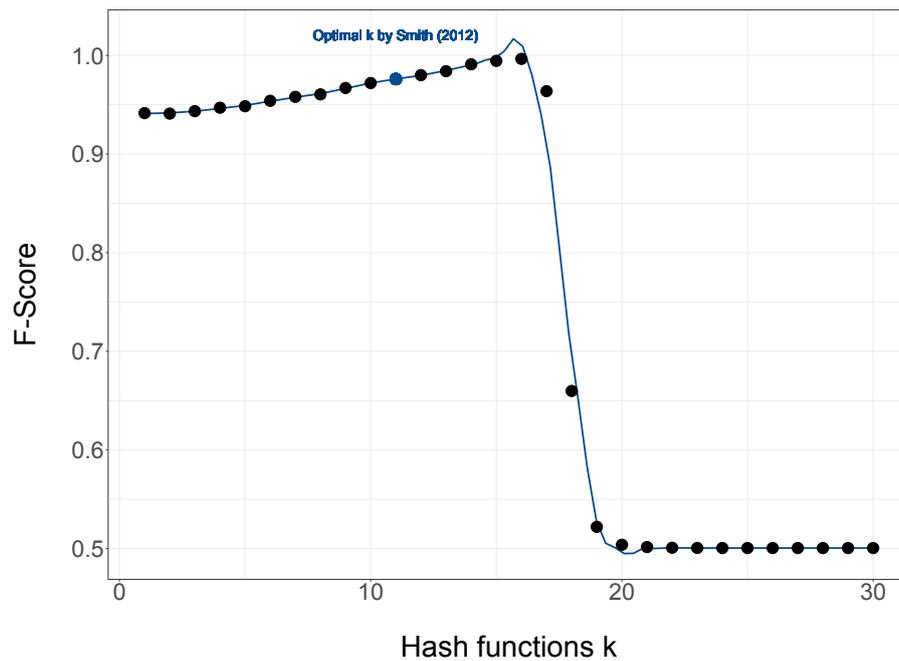


Figure 3.19: Using synthetic data, number of hash functions  $k$  plotted against mean prec./rec.. The blue line is a loess smoother.

### 3.5.4 Implications

Several things are of importance for the remainder of this thesis. First, the repeated generation of Bloom filters with different passwords will lead to negligible variance in the linkage quality. Second, the characteristics of the datasets themselves seem to be essential (this will be discussed in Section 4.4.1), as the same parameter choice will lead to drastically different linkage quality with different datasets. Finally, as was shown in the last subsections, choosing the number of hash functions in a way that leads to 50% of the bit positions to be set to 1 leads to sub-optimal linkage quality. Therefore, a different approach is needed, which will be described in detail in the next chapter.

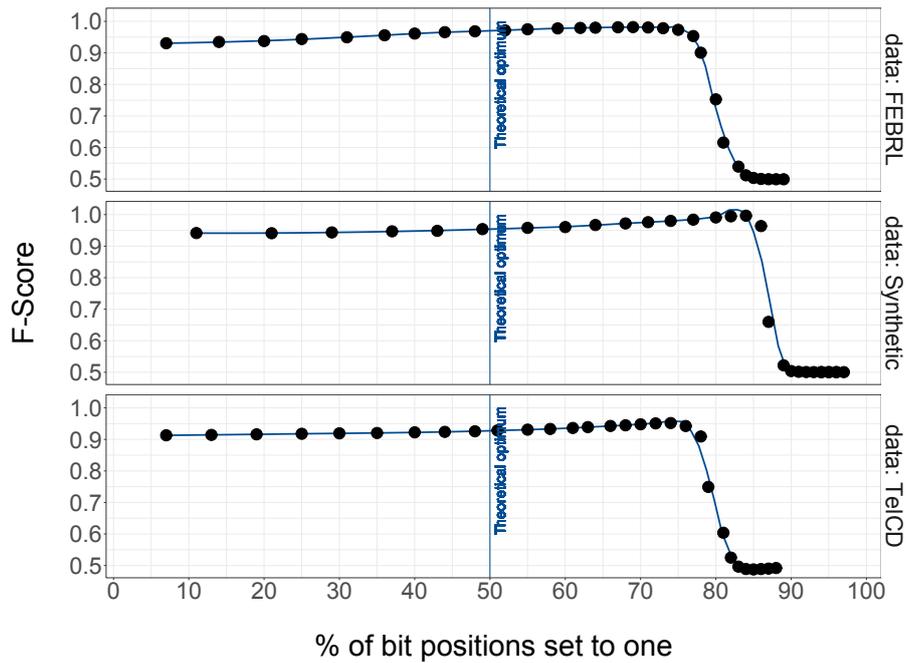


Figure 3.20: Percentage of bits set to 1 in a Bloom filter of length  $l = 256$  and resulting mean prec./rec. for several datasets using Multibit trees with a Tanimoto threshold  $T = 0.85$ . At above 75% to 85%, the linkage quality drops significantly because of the increased number of false positive matches. The blue line is a loess smoother.

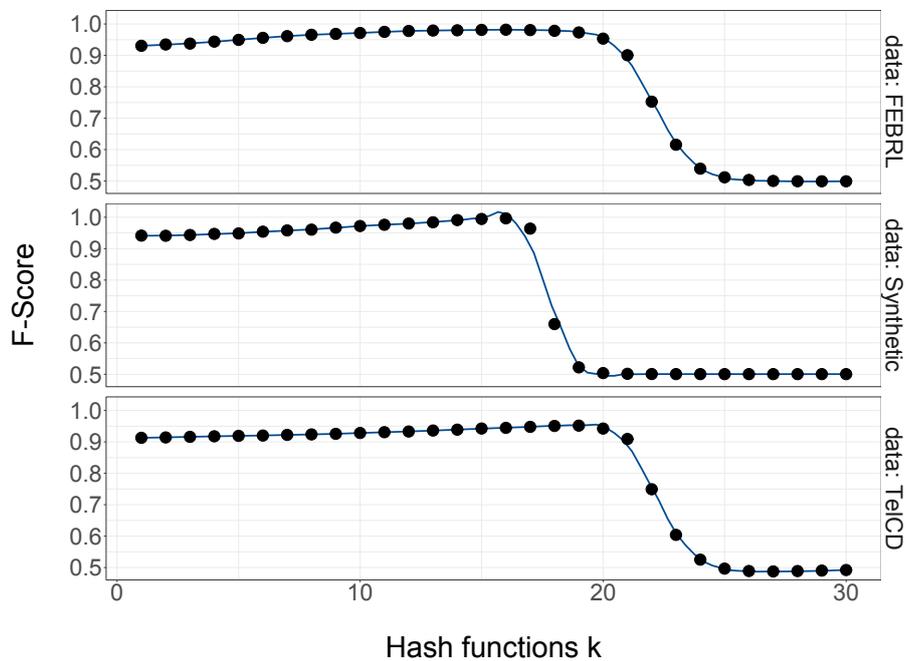


Figure 3.21: Using synthetic data, number of hash functions  $k$  plotted against mean prec./rec. for several datasets using Multibit trees with a Tanimoto threshold  $T = 0.85$ . The blue line is a loess smoother.

# Optimal identifier and parameter choice

Automating the parameter and identifier choice is the central goal of this thesis. Therefore, this chapter will, besides giving an overview of the data used (Section 4.3.3), be primarily concerned with the models for the optimal identifier (see Section 4.2) and parameter choice.

The current PPRL solutions are customised depending on the data to be encrypted. This is because of the sensitivity of many PPRL techniques – wrong parameter settings or identifier choices will often result in reduced linkage quality.

First, the optimal identifier choice is handled. Then, all the possible parameter combinations are used to generate Bloom filters. This is used to fit several models. For every future use, only a few input data metrics need to be computed, which are used to determine the optimal parameter choices given the clear text data. These can then be used to encrypt the data accordingly. This chapter will present an overview of this process.

## 4.1 Overview of the chapter

After a general overview of the data and the parameter space, a method for the optimal identifier choice is presented. For the optimal parameter choice, all the metrics recorded are discussed – some are results of the tests presented in the last chapter. Following this, the implementation of the model creation is explained. Implementation details are given. The results of the optimisation

and all the test runs with the model-based parameter choices are subject of the next chapter.

## 4.2 Optimal identifier choice

The optimal identifier choice for non-privacy-preserving Record Linkage was subject of the work of Quantin et al. (2004). In their publication, they used the agreement weight of identifiers in the PRL framework as a proxy for the discriminating power of identifiers. They found that first names, last names and date of birth worked the best as identifiers for their linkage setup.

Thus, following Quantin et al. (2004), we can make use of the  $m$ - and  $u$ -weights as proxies for the discriminating power and error rate, respectively (for more on these weights, see Section 1.5.5.4). The most important metric to combine both the measures is the agreement weight, which is calculated according to Equation 4.1:

$$\text{Agreement weight} = \log_2\left(\frac{m}{u}\right). \quad (4.1)$$

While Quantin et al. (2004) used a summary table to assess the optimal identifier choice by looking at the weights, the approach taken in this thesis focuses on the maximal agreement weight formed by the global  $m$  and  $u$ . The optimal identifier choice is the set of identifiers out of all the possible combinations that gives the highest agreement weight.

To test this, all the datasets were tested in this regard (for an overview of all the datasets, see Section 4.3.3). The agreement weights were computed using the  $m$  and  $u$  values as reported by the R package `fastLink` (Enamorado et al. 2018a). Table 4.1 gives an overview of all the data sets and their optimal identifier combination by using this approach. Note that the FEBRL WA data and the NC Voter data were sampled for computational reasons.<sup>1</sup> For many datasets, the standard set of identifiers (names and date of birth) seems to be an optimal solution. For both the mortality data files, this standard set ranked second in terms of the agreement weight.

---

<sup>1</sup>As the agreement weight needs to run the EM algorithm, this is computationally expensive. The NC Voter and FEBRL WA datasets were sampled with a fraction of 25% and 5%, respectively.

Table 4.1: Optimal identifier choice for all datasets used. Identifiers include first names (FN), middle names (MN), last names (LN), date of birth (DOB), sex and residential addresses. For the NC Voter data, DOB was unavailable and substituted with age. Only the FEBRL and the NC Voter datasets contained middle names.

Data	Optimal identifier combination
German telephone CD (20% errors)	FN, LN, DOB
FEBRL-generated data	FN, MN, LN, DOB
FEBRL-generated WA data	FN, MN, LN, DOB, suburb
NC Voter data	FN, MN, LN, age, sex, address
Mortality data & hospital data	FN, LN
Mortality register data & Commercial data	FN, LN

For example, the FEBRL WA dataset (more on this dataset in Section 4.3.3) has the following potential identifiers: First, middle and last name, Date of birth, Sex, Suburb, Postcode and Address. Table 4.2 shows a few possible combinations and their agreement weights with the optimal identifier choice highlighted. As can be seen, the agreement weight closely follows the actual results in terms of the linkage quality.

Table 4.2: Some possible identifier combinations and their resulting agreement weights for the FEBRL WA data. The optimal identifier choice is highlighted in blue. Note that the FEBRL WA data were sampled. Therefore, the true overlap is lower.

First	Last	Middle	<i>Identifiers</i>					Agreement weight	Mean prec./rec.
			DOB	Sex	Suburb	Postcode	Address		
✓	✓	✗	✗	✗	✗	✗	✗	14.86	0.09
✓	✓	✓	✗	✗	✗	✗	✗	15.67	0.11
✓	✓	✓	✓	✗	✗	✗	✗	16.50	0.27
✓	✓	✓	✓	✓	✗	✗	✗	15.42	0.12
✓	✓	✓	✓	✓	✓	✗	✗	16.52	0.33
✓	✓	✓	✓	✓	✓	✓	✗	14.65	0.10
✓	✓	✓	✓	✓	✓	✓	✓	15.28	0.10

The agreement weights are a good proxy for the expected linkage quality. Every optimal identifier choice presented in Table 4.1 also leads to the highest mean prec./rec. when linking. It has to be noted that `fastLink` produced a number of false positive matches, which is why the mean of precision and recall is lower than usual. This may also be due to sampling.<sup>2</sup>

<sup>2</sup>Sampling will lead to varying true positive matches, as the overlap between two files can differ. Even if only some very similar pairs are falsely classified as matches, the low number

Optimising identifier choice should be the first step taken before the next step, the optimal parameter choice, is undertaken.

## 4.3 Optimal parameter choice

First, the test setup for generating the training data is described. After an overview of the R package used, an overview of all the datasets follows. All the models for finding the optimal parameter choice are described in detail. Then, the resulting estimated optimal choices are presented for each model, as well as the best-practice approaches. The resulting linkage quality when encrypting and linking data with the model-based approaches is then tested against the best-practice suggestions in Chapter 5.

### 4.3.1 Test setup

The main criterion for a successful model-based optimal parameter choice estimation for a given dataset for encrypting Bloom filters is the improved linkage quality in terms of mean of precision and recall as compared to the current best-practice methods. This means that the parameters for encryption have to result in better linkage quality when linking the Bloom filters encrypted this way as compared to the baseline.

Finding such an optimal parameter choice requires several steps:

1. Generating training data
2. Fitting models
3. Predicting optimal parameters
4. Evaluation and diagnostics

Chapter 5 will deal with the evaluation of linkage quality, while the rest of this chapter is dedicated to the other three parts.

To generate the training data, several datasets (see Section 4.3.3) are designated as training datasets. Their properties with respect to several variables (see Section 4.4.1) are recorded. This will be designated as *meta data*, containing all the measures for the characteristics of a dataset.

---

of true positives in the equation  $\text{Precision} = TP / (TP + FP)$  will lead to a disproportionately low precision.

### Generating training data

With the training datasets, all the feasible parameters in the parameter space are used to encrypt data into Bloom filters. Some parameters were fixed for all the training data (and evaluations), as they were considered inconsequential for the linkage quality.

Table 4.3: Variables with fixed values used for the encryption and testing of the training data.

Variable	Value	Description
$l$	500	Length of the Bloom filter
$leaflimit$	3	Leaf limit setting for the MBT
$cores$	7	Number of threads used for MBT searching

These fixed-value parameters are shown in Table 4.3. The length of the Bloom filters is fixed, so the Hamming weight only depends on the number of hash functions  $k$ , not on the fraction  $\frac{k}{l}$ . In the literature, a length of  $l = 1000$  is often used (see Section 4.5.1). However, there are several reasons for using  $l = 500$ .

The primary reason is computational efficiency: Using smaller values for  $l$  uses less RAM and results in less computing time, as shown in Figures 4.1 and 4.2.

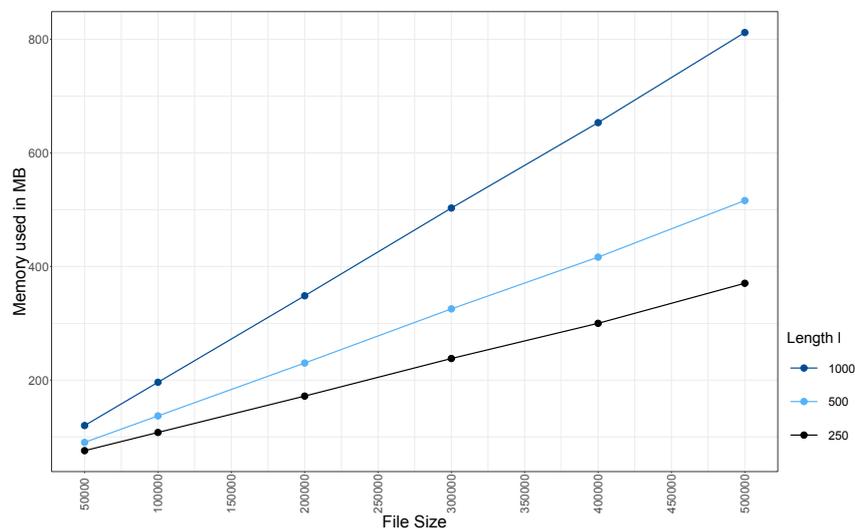


Figure 4.1: Memory used (in MB) for linking files of different file sizes for  $l = 250$ ,  $l = 500$  and  $l = 1000$ .

In particular, for larger files, using  $l = 500$  saves a considerable amount of memory. Interestingly enough, halving the length a second time only improves

the RAM required marginally.

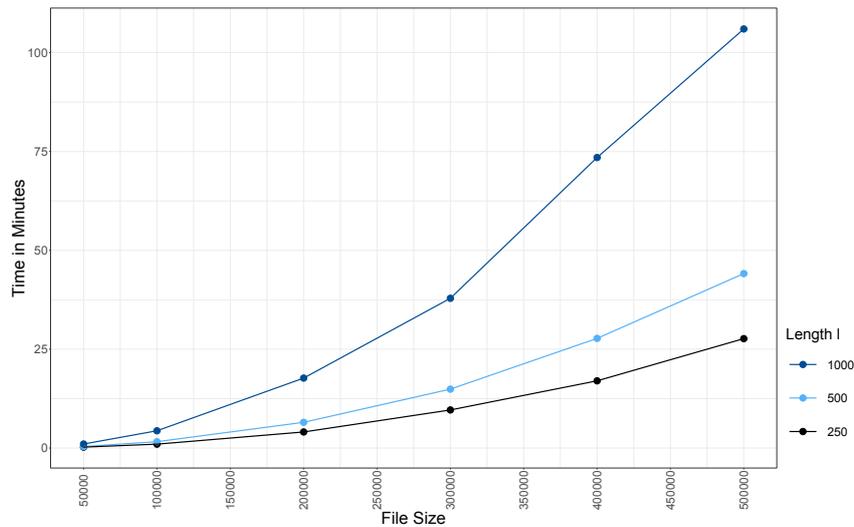


Figure 4.2: Time taken (in minutes) when linking files of different file sizes for  $l = 250$ ,  $l = 500$  and  $l = 1000$ .

The same pattern emerges when looking at the computing time, where the time taken is reduced considerably when choosing half the length ( $l = 500$ ) of the standard  $l = 1000$  bits.

Finally, as long as  $\frac{k}{l}$  remains constant, the linkage quality will not differ beyond the variance caused by differing bit positions that might collide. This should amount to approximately the amount of variance caused by different passwords, as was discussed in Section 3.5.1. Therefore,  $l$  will be fixed to a value of 500. The number of cores used depends on the threads that the computers CPU can supply and will remain constant as well. The *leaf limit* of Multibit trees (see Section 3.4) will remain at a constant value of three. The reasoning behind this is explained in Section 5.3.

With the fixed parameters in mind, the parameter space with varying parameters is shown in Table 4.4. The interplay between data characteristics and choosing  $q$  for  $q$ -grams and the number of hash functions ( $k$ ) will be central to achieving good linkage quality.

Table 4.4: Variables and their parameter space used for training data generation.

Variable	From	To	By	Description
$k$	1	40	1	Number of hash functions used to map $q$ -grams to Bloom filters
$q$	1	4	1	Splitting strings in to uni-, bi-, tri- or quadgrams.
$Tani$	1.0	0.8	-0.05	Tanimoto threshold for Multibit tree linking.

All the training data is encrypted using various numbers of hash functions. As was first described in Figure 3.20 in the last chapter, a Bloom filter (BF) with more than 90% of its bit positions set to 1 will probably yield a very low precision. Figure 3.15 in Section 3.5.3 shows the number of hash functions needed to approach a ratio of ones of more than 90% was  $k = 20$  for a length of  $l = 256$ . As our experimental approach showed, if a length of  $l = 500$  is used,  $k = 39.0625$ ; that is, approximately  $k = 40$  hash functions would be needed to have 90% of bits set to 1. Therefore, all the data will be encrypted using a range of  $k$  hash functions between 1 and 40.

The values for  $q$  in terms of splitting input strings into subsets of  $q$  are restricted to unigrams ( $q = 1$ ), bigrams ( $q = 2$ ), trigrams ( $q = 3$ ) and quadgrams ( $q = 4$ ).

All the encrypted files are linked to each other using Multibit trees (see Section 3.4 for details), with Tanimoto thresholds between 0.8 and 1.0. With these parameters in mind, the models can be used on the results generated by encrypting all the training data sets using the parameter space as described above.

#### *Fitting models and predicting optimal parameters*

Using the parameters described above and linking them yield various linkage quality measures (see Section 1.5.7), with the mean prec./rec. being the primary dependent measure. Thus, the models proposed in Section 4.4.2 can be used to predict the parameter combination giving the highest predicted mean prec./rec.. The  $q$  and  $k$  values of the predicted maximal mean prec./rec. are then used as the optimal parameter choice. The models themselves will make use of the *meta data* and the other variables dependent on the parameters that will be discussed in Section 4.4.1.

To implement all the four steps described above, R (R Core Team 2019) was used. The central package written by the author's research group is essential to producing the results of this thesis. This is why it will be described briefly.

### 4.3.2 The PPRL package for R

The core functionality of using PPRL methods in R was provided by the PPRL package: PPRL is a package by Schnell/Rukasz (2019). It can be downloaded

for free on CRAN.<sup>3</sup>

It offers the functionality to preprocess, encrypt and link data using state-of-the-art PPRL methods, making it possible to go through the entire Record Linkage process (see Figure 1.3) in R. After the required files are read, preprocessing can be handled by `StandardizeString()`. Table 4.5 shows a few replacement rules for input strings. Differing representations of dates can be resolved using the primitive R-function as `.POSIX()` and as `.date()`.

Encrypting identifiers is possible using ALCs, ESLs, Bloom filters or CLKs. Several hardening methods are available: salting (by modifying the CLK call), balancing, Markov chain-based BFs (MCBFs), RBFs, cellular automata (Rule 90) for CLKs and BLIP/RAPPOR randomised response BFs. For linking, deterministic and probabilistic Record Linkage are possible. Multibit trees can be used with the corresponding R package `multibitTree`, which is, as of 07/2019, not yet available on CRAN<sup>4</sup>, but available on request from the author. Evaluation of the results was done in R as well. Plots were generated by `ggplot2`<sup>5</sup> (Wickham 2016).

A complete session overview with most of the packages, R versions and operating systems used for generating the results of this thesis can be found in Appendix F.3.

Table 4.5: Replacement rules implemented in PPRL.

Æ	→	AE
æ	→	AE
Ä	→	AE
ä	→	AE
Å	→	A
å	→	A
.....		
ß	→	SS
ü	→	UE
Û	→	UE
Û	→	U
û	→	U
Û	→	U

### 4.3.3 Datasets

For the training and evaluation, several datasets<sup>6</sup> were considered:<sup>7</sup>

- A sample of a German telephone CD (TelCD) with and without errors

<sup>3</sup>See <https://CRAN.R-project.org/package=PPRL> (Last accessed 22.02.2019).

<sup>4</sup>The package was created for the research unit of the author by Markus Koetter for Rainer Schnell.

<sup>5</sup>See <https://CRAN.R-project.org/package=ggplot2> (Last accessed: 31.01.2019).

<sup>6</sup>FEBRL and TelCD were created by the author; all the other datasets were acquired from other people. I have to thank Adrian Brown for the FEBRL WA data, Peter Christen for the NC Voter data and Rainer Schnell for the Mortality dataset.

<sup>7</sup>The short names for the result tables and plots are given in braces.

- FEBRL-generated data (FEBRL)
- FEBRL-generated Western Australian data (FEBRL WA)
- North Carolina Voter data (NC Voter)
- Real-life mortality register data merged to commercial data or hospital data (Mortality commercial and Mortality hospital)

Table 4.6 presents an overview of all the datasets. Some were used to train the model-based approaches, while the others were used to test the models against best-practice solutions. As is obvious, no non-western language group was used to train the model. This is a caveat that could be subject to further research.

Table 4.6: All the datasets used for either training the models (Training) or evaluating the models (Testing) against best-practice parameter settings, with their populations' main language and their data type (real-world or simulated).

Data	Population	Type	Usage
Mortality data & hospital data	German	Real-world	Training
Telephone CD 20% err.	German	Simulated	Training
Telephone CD 0% err.	German	Simulated	Training
FEBRL	Australian	Simulated	Training
NC Voter data (2014)	American	Real-world	Testing
Mortality & commercial data	German	Real-world	Testing
FEBRL WA	Australian	Simulated	Testing

In the following section, all the datasets will be described briefly.

#### 4.3.3.1 German telephone CD (TelCD)

The German telephone CD dataset is taken from a telephone database (from the commercial organisation “klicktel AG”<sup>8</sup>) for the year 2012. In Germany, there used to be a de-facto mandatory system for a public phone number register, including names and addresses. Häder (2015: 1), without reference, gives 1992 as an end-date for the mandatory public registering of numbers. Since then, publicly listing your telephone number has become optional. Therefore, the coverage of the German population in these registers is dropping considerably

<sup>8</sup>In September 2016, Klicktel was renamed 11880.com; see <https://www.klicktel.de/klicktel-markenrelaunch> (Last accessed: 31.03.2019).

(Häder 2015). Using this database will probably lead to a slight bias towards “older” names than the general population. For this thesis, a sample of  $n = 10,000$  records was used. Years of birth were generated according to birth year statistics, while days and months were generated randomly.

A corrupted version of the resulting file was produced using a custom error generation program, written by the author and devised by Rainer Schnell. This was done by sampling 20% of all the rows and inducing at least one error in the name or DOB field. Table 4.7 presents an overview of the error types<sup>9</sup> (for an overview of the errors in data and several error types, see Section 1.5.5.2).

Table 4.7: Overview of error types and probabilities for each error type for the corrupted German telephone CD file.

Error type/probability (%)	First name	Last name	DOB
Selection probability	40	40	20
Spelling error	32	32	0
Insertion	5	10	0
Deletion	15	10	0
Replacement	35	35	20
Transposition	5	4	30
Swap values	2	2	15
Swap names	2	2	0
Insert blanks	1	1	0
Delete blanks	1	1	0
Set to missing	2	2	5

If a row is designated to contain errors, names have a 40% chance each of being picked for the error generation, while the date of birth (DOB) has a 20% chance of containing errors. For each field, the error probabilities are shown.<sup>10</sup> Apart from the insertions, deletions, transpositions and replacements of characters or digits (see Section 1.5.5.2), other more specific errors were included: Spelling errors replaced a name with a different name that had the same Soundex code; swapping first and last names or day, month or year, and inserting or removing blanks. A small fraction of fields were set to missing values (i.e. empty fields).

<sup>9</sup>The error probabilities are very similar to the FEBRL error generation (see Section 4.3.3.3), with small modifications to allow for German particularities such a higher frequency of double surnames joined with a hyphen.

<sup>10</sup>Therefore, the probability of a spelling error in an erroneous row in the first name would be  $0.40 * 0.32 = 0.128$  or 12.8%.

Finally, there was an independent chance of 3.5% of a complete change in the last name. This was supposed to simulate differing last names after marriage, which would be possible if the datasets to be merged were from a period before and after the marriage of a person.

#### 4.3.3.2 Real-life mortality register data (Mortality & commercial/hospital data)

First described in Schnell/Borgs (2018c), the larger file is an administrative database of deaths in one year of a large German town ( $n = 14,003$ ). This serves as a real-life administrative database. The file contains names and birth dates. The administrative data are linked against two files: (1) deaths of a local hospital in that town of the same year ( $n = 2,466$ ) and (2) deaths of residents of that town in this year as recorded by a commercial credit rating agency ( $n = 7,635$ ).

Naturally, dataset (1) did not cover all the deaths of the administrative data of a year, as it only included people who died at this one hospital in the city. Additionally, not all the people dying in this hospital were people living in this town. This was why the true overlap was only 896 cases. The records from this dataset were mostly free of errors.

Dataset (2) shares 6,592 cases with the administrative database. As this commercial credit rating data owner does not update the records on persons often, there will be a lag in the reporting of deaths, leading to fewer reported deaths in a year than the administrative data suggest. The high discrepancy between the overlap and the file size can only be explained by outdated records on the place of residence of persons in the registry. More than half of the deaths of a year were not recorded in the database, and roughly 1,000 cases were wrongly classified as deaths in the considered town. Many identifiers were missing or wrong in dataset (2), making it ideal for the evaluation of optimal parameter choices.

#### 4.3.3.3 FEBRL-generated data (FEBRL)

The FEBRL Record Linkage software by Christen (2008) contains `dsgen`, a data generation tool which can produce synthetic data and a corrupted copy of that same synthetic dataset. It uses samples of the real first, middle and last

names from an Australian population and gives random birth dates. FEBRL can generate erroneous duplicates with the well-researched error probabilities of different error types, such as mistyping, character replacements or random inserts and deletions (see Section 3.1.2 for an overview). Table 4.8 presents an overview of the three most important identifiers and their respective error types.

Table 4.8: Overview of error types and probabilities for each error type for the FEBRL-generated files. Note that there are more (unused) fields in the FEBRL generator, which is why the sum of the selection probabilities is not 100%.

Error type/probability (%)	First name	Last name	DOB
Selection probability	10	9	5
Spelling error	30	30	0
Insertion	5	10	0
Deletion	15	10	0
Replacement	35	35	50
Transposition	5	4	30
Swap values	2	2	5
Swap word	2	2	5
New value	2	2	5
Swap names	2	2	0
Phonetic error	30	3	5
OCR errors	12	12	12
Set to missing	2	2	0

The probabilities and error types are very similar to the error generation of the German telephone CD dataset (see Table 4.7) with some more error types added. OCR errors mimic the typical substitutions or transpositions that occur when trying to use optical character recognition (OCR) on scanned text. Phonetic errors replace values with a value with a similar phonetic code. Swapping words is slightly different from swapping values in the sense that it applies to a single field only. This is particularly useful for records with multiple space-separated strings (sometimes called *tokens*) in a single field or for fields with many words (e.g. addresses or suburbs).

#### 4.3.3.4 FEBRL-generated Western Australian data (FEBRL WA)

The Centre for Data linkage (CDL)<sup>11</sup> at the Faculty of Health Sciences at the Curtin University in Perth, Australia, uses a slightly modified version of FEBRL, where the sampling base consists of administrative data from Western Australia. According to personal communication, some error-generating parameters were adjusted as well.

The original datasets consisted of one million records each, where one was an error-free dataset, while the other was a corrupted copy, as with the standard FEBRL dataset. Moreover, 10% of the rows of the corrupted copy of the original dataset contained errors. As one million records would be computationally expensive to link, both datasets were sampled down to  $n = 100,000$  records.

As the name and error distribution differed slightly from the regular FEBRL dataset, this dataset was used to test the linkage quality of the model-based optimal parameter choices against best-practice solutions.

#### 4.3.3.5 North Carolina Voter data (NC Voter)

Some US states publish their voter registration data. In order to vote in elections, voters must be registered in such a voter register. One such register is the North Carolina Voter database. It has been used for Record Linkage purposes before (Vatsalan 2014). The datasets used here were preprocessed by Peter Christen, based on two snapshots<sup>12</sup> for the year 2014. A detailed description of the process to generate the files is given in a technical report (Christen 2014). A sample ( $n = 50,000$ ) of the overlap of the two snapshots is used for the evaluation of the model-based approaches.

---

<sup>11</sup>For more information on the linkage institution, see <https://healthsciences.curtin.edu.au/health-sciences-research/research-institutes-centres/centre-for-data-linkage/> (Last accessed: 04.04.2019).

<sup>12</sup>A *snapshot* is a time-stamped copy of a database that continually changes. Thus, snapshots from different times will differ to a varying degree, primarily on the basis of the amount of time passed between snapshots.

## 4.4 Models for optimal parameter choice

After the selection of a set of identifiers for linkage and an overview of the datasets that will be used, parameter choice is the next obstacle. For this, models will be fit using the linkage results of the training data encrypted with all the possible parameters to estimate the optimal choice given the evaluation datasets. The metrics used for fitting the model is the objective of this section. First, the rationale for variable choices is highlighted. Finally, the resulting models are presented.

### 4.4.1 Variable selection

The aim of the models (which will be introduced in Section 4.4.2) should be to maximise the linkage quality. Therefore, the dependent variable in all models will be the mean of precision and recall (mean prec./rec.).

Table 4.9 presents an overview of all the independent variables used in the models, describes them and provides the reasoning behind selecting them.<sup>13</sup> Selecting optimal  $k$  and  $q$  as the encryption parameters is the primary goal. Therefore,  $k$  and  $q$  will be included in the models. In Section 3.5.3, the fraction of bits set to 1 was shown to affect the linkage quality. This is reflected by the Hamming weight of the bit vectors. Of course, the Hamming weight itself is directly influenced by  $k$ ,  $q$  and the number of elements to be hashed. Therefore, the interaction effect between  $k$ ,  $q$  and the Hamming weight, as well as the Hamming weight and the mean number of elements hashed (mean  $q$ -grams) will be modelled. The latter is expanded by also recording the number of unique  $q$ -grams, giving a mean uniqueness estimation when dividing this number by the number of records.

Another consideration is the actual distribution of these  $q$ -grams. An earlier analysis shows that the bigram frequencies and their frequency distributions differ considerably between language groups. Figure 4.3 shows the distribution of the 50 most frequent Polish and German last names in comparison. The most frequent bigrams differ considerably by language group.

---

<sup>13</sup>At this point, I am indebted to Rainer Schnell, who had the ideas for most of the independent variables: Gini coefficient as a measure of inequality, skewness, Hamming weight, and  $m$  and  $u$ , as the idea was to use the EM algorithm. The use of  $k$  and  $q$  is reasonably apparent. To the best of my knowledge, my own ideas were the fraction of the 90<sup>th</sup> percentile versus the 10<sup>th</sup> percentile, unique patterns and uniqueness and the mean number of hashed elements.

Table 4.9: Independent variables hypothesised to influence the predicted linkage quality (mean prec./rec.).

Model Variable	Description	Underlying Hypothesis
$k$	Number of hash functions	Optimal choice is the goal
$q$	Number of subsets to split the identifier into ( $q$ -grams)	Optimal choice is the goal
errorest	Ratio of the 90 <sup>th</sup> and 10 <sup>th</sup> percentile of bigram frequencies ( $p_{90}/p_{10}$ )	Extreme inequality in bigram frequency can lead to loss of precision if few frequent bigrams dominate
skew	Skewness of the bigram frequency distribution	Low skewness will increase precision, high skewness will, like the percentile ratio, increase the number of false positives
uniqueness	Mean unique $q$ -grams generated by the identifiers	The higher the uniqueness, the better the precision
hamming	Hamming weight of CLKs created with parameters $k$ and $q$	Demonstrated to influence linkage quality
uniquepattA	Unique CLK patterns in the data created with parameters $k$ and $q$	See uniqueness
gini	Mean Gini coefficient of identifier frequencies	precision will be better for a low Gini coefficient
meanngr	Mean elements ( $q$ -grams) hashed into a Bloom filter for a file	Main factor; together with $q$ and $k$ will decide the number of bits set
$m$	Global $m$ -weight from the EM algorithm	Should give an estimation of discriminatory power
$u$	Global $u$ -weight from the EM algorithm	Should give an estimation of error rate

Differing ranks of  $q$ -gram frequencies between datasets should not be a problem. However, the presence of very predominant  $q$ -grams could lead to changes in the linkage quality, as these bit positions are not sufficiently discriminating. In Figure 4.3, the top 5 bigrams for German nationals make up a larger fraction of the top 50 than the top 5 bigrams of the Polish nationals. Two measures are introduced to check for this distributional discrepancy: the skewness of the  $q$ -gram distribution, given by the sum of the cube of the difference between the frequency values ( $x_i$ ) and the mean frequency, divided by the third central moment of the bigram frequencies:

$$\text{skewness} = \frac{\frac{1}{n} \cdot \sum_{i=1}^n (x_i - \bar{x})^3}{s_x^3}. \quad (4.2)$$

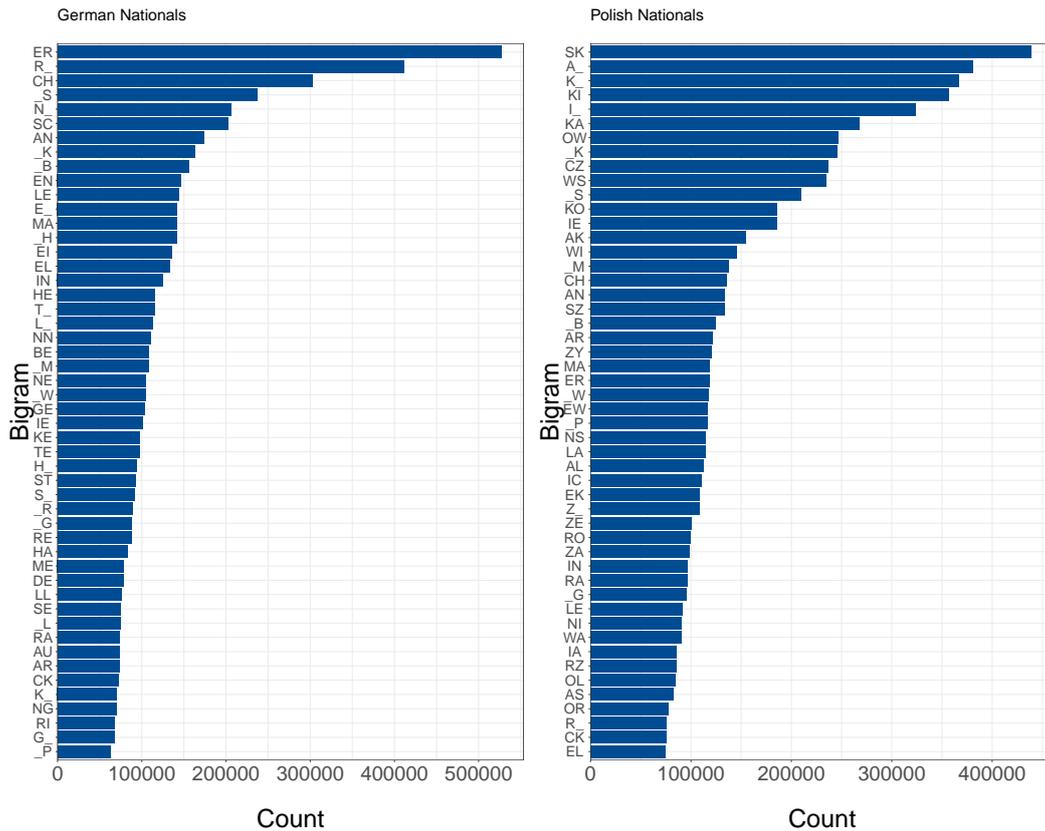


Figure 4.3: Bigram counts of last names of German (left) and Polish nationals (right) of a large administrative database. Note the skewed distribution with few very frequent bigrams.

Another related measure is used in income inequality research: The ratio of high versus low percentiles of a distribution (Cowell 2000). In this case, the frequency distribution of bigrams is used. Here, the frequency value of the 90<sup>th</sup> percentile will be divided by the frequency value of the 10<sup>th</sup> percentile. This will help determine whether a given dataset has a large number of very common bigrams, which might lead to a loss in precision through lesser discriminating power.

The skewness and the percentile ratio are correlated, as the ratio of the frequencies in the 90<sup>th</sup> and the 10<sup>th</sup> percentile of the bigram distribution increases the skewness. This will be modelled as well.

Another measure of income inequality is used for the distribution of the frequent values of all *identifiers*: The Gini coefficient (Gini 1912).<sup>14</sup> The coefficient  $G$  is given by subtracting the area under a Lorenz curve (the cumulative pro-

<sup>14</sup>Often misquoted as Gini (1921), since the original publication was written in Italian.

portional frequencies of all the values of an identifier) from the area under the “line of equality” (the cumulative proportional frequencies if all the identifier values were equally frequent):<sup>15</sup>

$$G = \frac{1}{2n^2\bar{x}} \sum_{i=1}^n \sum_{j=1}^n |x_i - x_j|. \quad (4.3)$$

For modelling purposes, the mean Gini coefficient of the frequency distribution of all the identifiers was used. A high Gini coefficient hints at a low discriminatory power of the identifiers used.

Following the logic of the EM algorithm as discussed in Section 1.5.5.4 and the insight gained from the optimal identifier choice (Section 4.2), the global  $m$  and  $u$  weights were used as a proxy measure for the discriminating power of the identifiers used for the encryption ( $m$ ) and as an approximate error measure ( $u$ ).

Finally, precision is almost guaranteed to be worse if there are duplicate Bloom filter patterns after the encryption, as they will always be classified as a match, which will lead to an increase in the number of false positive classifications if these are not the same records. Therefore, all the identifiers will be encrypted into Bloom filters and checked for duplicates. The number of unique patterns will be part of the model.

#### *Properties of all datasets*

After the overview of the variables used, Table 4.10<sup>16</sup> presents an overview of all the datasets used with their respective properties and file sizes ( $n$ ). Note that varying variables such as the Hamming weight are excluded, as they are not fixed for all choices of  $k$  and  $q$ .

After the selection of the appropriate independent variables and reporting on each dataset’s properties, an overview of all the models used for estimating the optimal parameter choice is presented.

<sup>15</sup>This formulation of the calculation is based on Sen (1997).

<sup>16</sup>The values for  $u$  were all very close to 1, which is why they were rounded. The variable *bigrams* represents the mean number of bigrams hashed into Bloom filters.

Table 4.10: Properties of all datasets used for modelling: file sizes ( $n$ ), true matches (overlap), entropy (entrp.), mean number of bigrams, skewness (skew), Gini coefficient (gini), ratio of the percentiles  $p_{90}$  and  $p_{10}$ , percent unique patterns (% uniq) and the weights given by the EM algorithm ( $u$  and  $m$ ). All the values shown are based on the use of bigrams ( $q = 2$ ).

	$n_A$	$n_B$	overlap	entrp.	bigrams	skew	gini	$p_{90}/p_{10}$	% uniq	m	u
<i>Training data</i>											
Mortality/Hospital	14003	2466	896	0.554	22.551	8.242	0.448	339	97.2	$2.66 \times 10^{-5}$	1
Telephone CD 20% err.	10000	10000	10000	0.221	37.544	11.005	0.215	294	77.6	$9.76 \times 10^{-5}$	1
Telephone CD 0% err.	10000	10000	10000	0.221	37.544	11.005	0.215	294	77.6	$1.00 \times 10^{-4}$	1
FEBRL	10000	10000	10000	0.697	33.306	13.263	0.413	320	76.3	$9.98 \times 10^{-5}$	1
<i>Evaluation data</i>											
NC Voter	50000	50000	50000	0.926	17.433	11.364	0.628	1187	98.6	$3.88 \times 10^{-5}$	1
FEBRL WA	100000	100000	100000	0.751	21.386	12.127	0.541	2890	97.7	$9.99 \times 10^{-6}$	1
Mortality/Commercial	14003	7635	6592	0.385	35.551	11.226	0.307	353	76.8	$6.30 \times 10^{-5}$	1

## 4.4.2 Models

The optimal parameter choice will be found by recording the properties of the variables presented above and encrypting the datasets with all feasible choices of  $k$  and  $q$ . The resulting linkage quality in terms of mean of precision and recall is then used to build several models, with which the optimal parameter choice for the evaluation data will be predicted. These resulting best parameter estimates will be tested against the current best-practice solutions, which will be described after the models themselves have been presented. The following models will be discussed and evaluated:

- Simple linear model (LM)
- BIC-based linear model selection (BIC)
- Response surface model (RSM)
- Random Forest-based regression (RF)

These four models will yield different optimal parameters (see Section 4.5.3) leading to different linkage quality when linking, which will be tested against two best-practice methods: (1) using the 50% rule (see Section 3.5.3) to estimate an optimal  $k$  and (2) using parameters currently recommended or used in the literature (see Section 4.5.1). The actual results will be reported at the end of this chapter.

First, however, some thought has to be given to the choice of the similarity threshold, as the optimal parameter choices will have to be optimised for a certain threshold.

#### *Fixing the Tanimoto threshold*

All the models were built using a fixed Tanimoto threshold of  $T = 0.9$  for the following reasons:

1. This way, the performance is optimal for a resulting parameter choice at  $T = 0.9$ .
2. This threshold is computationally feasible even for large files.
3. Optimal Tanimoto threshold selection is no longer an additional issue.
4. This threshold should provide sufficient error-tolerance if the parameters are set accordingly.

In particular, points 1 and 2 are worth stressing. Lowering the threshold of Multibit trees increases the time taken to link markedly: Following the description of Multibit trees in section 3.4, the primary advantage in terms of the computing time of using them to link data is the ability to give an upper similarity bound for any given input Bloom filter knowing only the values of the match bits. All the leaves in a tree below the set Tanimoto threshold  $T$  can be removed from the comparison space, as their similarities cannot satisfy the minimal similarity given by  $T$ . This also means that the higher the threshold is, the more restrictive is the filtering of the search space, as higher upper similarity bounds are required. Conversely, lowering the threshold will increase the search space, increasing the search time for candidate pairs. This will also require more memory, as the similarity calculations of all the pairs is done in the computer's RAM (Kristensen et al. 2010).

Figure 4.4 shows the time taken for linking two records with 1 million records each, with the Tanimoto threshold plotted on the x-axis. As can be seen, the time taken doubles from approximately 2 minutes to 4 minutes when lowering the threshold below  $T = 0.9$ . The effect is even stronger when lowering the threshold below 0.85.

The same is true for varying file sizes in general, as shown in Figure 4.5. Selecting  $T = 0.90$  reduces the time taken to link considerably. This is the

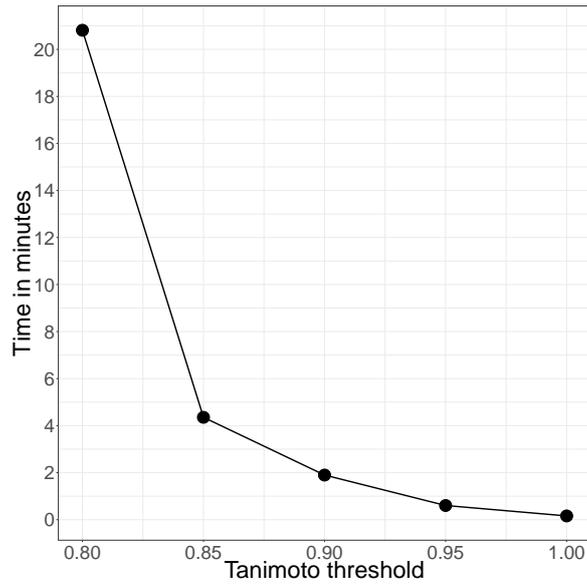


Figure 4.4: Tanimoto threshold and time taken to link (in minutes) 1 million by 1 million records using Multibit trees in R. Idea for the graph first published in Schnell (2016a).

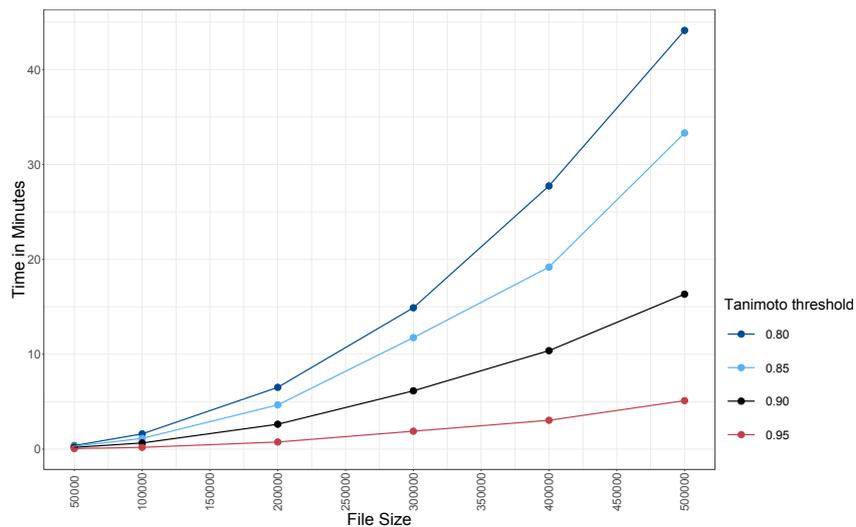


Figure 4.5: Tanimoto thresholds and time taken to link (in minutes) records with different file sizes using Multibit trees in R.

primary reason for fixing  $T$  at  $T = 0.90$ .<sup>17</sup>

With this in mind, an overview of the models themselves will follow.

<sup>17</sup>Note that the code allows for this to be changed: optimising parameter choice for a different  $T$  is not a problem.

### 4.4.2.1 Simple linear model (LM)

A simple linear multivariate regression model is the most basic option of estimating an optimal parameter set for the metric target variable. It should always be outperformed by other methods. Moreover, some assumptions needed to produce correct estimates might not hold (more on this later).<sup>18</sup>

A simple linear model tries to estimate values for a dependent variable  $y$  given several independent input variables  $x_i$ . The estimated values are denoted as  $\hat{y}$  for a linear model with  $k$  independent variables and an error term  $\epsilon$ . The notation of  $k$  independent variables explaining the dependent variable  $y$  is as follows:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k + \epsilon. \quad (4.4)$$

In the matrix form, this can be expressed as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{1_1} & x_{2_1} & \dots & x_{k_1} \\ 1 & x_{1_2} & x_{2_2} & \dots & x_{k_2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{1_n} & x_{2_n} & \dots & x_{k_n} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}. \quad (4.5)$$

This when simplified gives the following:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon, \quad (4.6)$$

where the coefficient  $b$  is given as follows:

$$\beta = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}. \quad (4.7)$$

To estimate a model's fit to the data,  $R^2$  can be used (Myers et al. 2010: 22f):

$$R^2 = \frac{SS_R}{SS_T}, \quad (4.8)$$

<sup>18</sup>The notation will follow Myers et al. (2010), while Venables/Ripley (2002) is referred to when R is applied.

where the explained sum of squares is<sup>19</sup>

$$SS_T = \mathbf{y}'\mathbf{y} - \frac{\left(\sum_{i=1}^n y_i\right)^2}{n}, \quad (4.9)$$

while the sum of squares of the residuals is as follows:

$$SS_R = \mathbf{b}'\mathbf{X}'\mathbf{y} - \frac{\left(\sum_{i=1}^n y_i\right)^2}{n}. \quad (4.10)$$

#### *Model specification*

With the choice of independent variables ( $\beta$ s) in mind, as discussed in Section 4.4.1, the simple linear model predicting the mean prec./rec. can be expressed as follows:

```
LM <- lm(Fmeasure ~ k + q + errorestimation * skew +
uniquepatternsA * uniqueness + ginicoefficient +
k * hammingweight + hammingweight * meanengrams +
uniquepatternsA + log(u), data=fixed)
```

As was previously discussed, several interaction effects must be considered. One problem arose which required selecting either  $m$  or  $u$ : Both these measures are completely co-linear. This is because  $m$  and  $u$  are de-facto only transformations of each other. Therefore, only  $u$  was retained in this model as an error approximation. As  $u$  was very skewed between the datasets, it was logarithmised. Table 4.11 presents an overview of all the variables used in the linear regression as compared to the BIC-based model selection (which will be discussed in the next section). Thus, only approximately 78% of the variance of mean of precision and recall can be explained by this model. This may be attributed to a violation of some assumptions of the regressions:

The estimation of  $\hat{y}$  with a linear model (ordinary least squares (OLS) estimation) requires the following assumptions to be made:

- Linearity

<sup>19</sup>Note that  $\mathbf{y}$  and  $\mathbf{y}'$  are the vector of the dependent variable and the transposed vector of the dependent variables, which is why this notation is equivalent to the classic notation where the explained sum of squares is  $SS_R = \sum_{i=1}^n (y_i - \hat{y}_i)^2$ . The same is true for Equation 4.10.

- Homoscedasticity
- Uncorrelated residuals
- The expected value of the residuals is 0

### Model diagnostics

Nearly all of the abovementioned assumptions can be checked using the diagnostic plots provided by R, as seen in Figure 4.6. The following interpretations are loosely based on Everitt/Hothorn (2010).

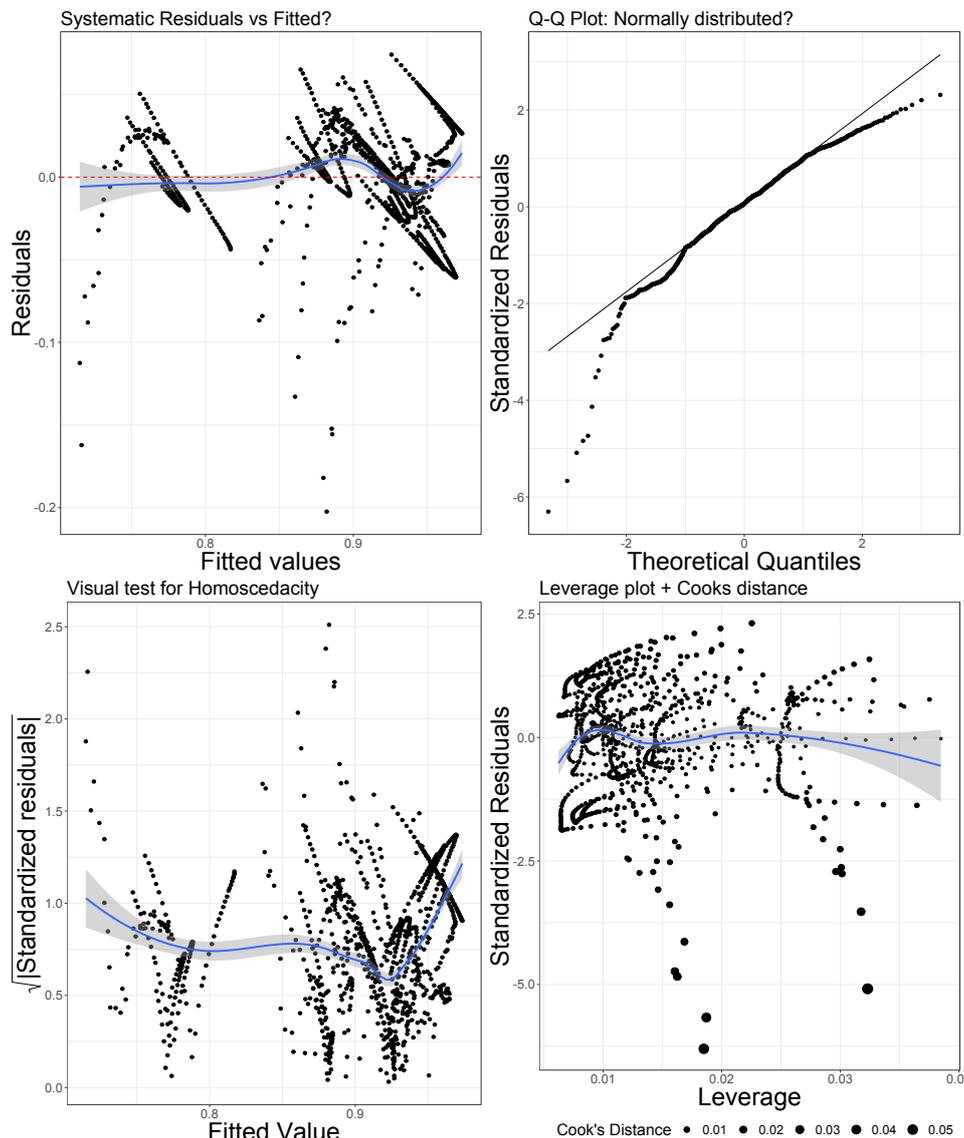


Figure 4.6: Diagnostic plots for the simple linear model (LM).

The residuals versus fitted plot is used to check for the linearity assumption. If there is a systematic fitted/residual pattern, the linearity assumption could be violated. As the fit line is not perfectly straight, this assumption may be violated, even though the patterns suggest that different datasets and different  $q$  values have independent fit/residual characteristics (see Figure 5.6). As previously shown in Section 3.5.3, not all the identifiers have a linear relationship with the mean of precision and recall. It seems that the linear model may be unable to predict the non-linear drop-off in mean of precision and recall (see Section 3.5.3).

The line generated by the loess smoother is close to zero, and the expected value of the residuals seems to be zero.

The Q-Q plot shows an expected line following a normal distribution and the empirical values against each other. As this is not an assumption important for the OLS estimation, it will not be of further importance.

The third plot is a visual test for homoscedasticity. As the standardised residuals are larger at low and high fitted values, this assumption seems to be violated. This will likely result in unstable estimates at the lowest and highest predictions, leading to sub-optimal parameter choices.

The final plot is a leverage plot. Cases with high leverage values considerably influence the regression results. There are some cases with a larger Cook's distance, where the predicted linkage quality diverges from the actual linkage quality. This could be a sign of either over-fitting or a result of the model being inadequate for the lack of linearity in the data.

Finally, to check for the correlation of independent variables, the variance inflation factor (VIF) can be used:

$$VIF = \frac{1}{1 - (r^2)} \quad (4.11)$$

According to Fox/Weisberg (2011), inflation factors should not exceed  $\sqrt{VIF} > 2$ . For the linear model, all the square roots of the VIFs were larger than two, suggesting multicollinearity between the variables.

To sum up, some assumptions hold, but very central requirements are violated: linearity and homoscedacity. This will, at the least, lead to less-than-optimal results.

### 4.4.2.2 Bayesian model selection based on BIC (BIC)

While the linear model (LM) is based on the selection of variables by the researcher, optimal model coefficients can be selected using information criteria. One such approach is the Bayesian information criterion (BIC; Schwarz 1978). For this, the `stepAIC` function from the MASS package was used (Venables/Ripley 2002). The advantages of using the BIC against using the often-used Akaike information criterion (AIC; Akaike 1974) are that BIC penalises the addition of more variables to the model more harshly, usually resulting in fewer variables in the model (Venables/Ripley 2002), which might reduce the chance of selecting an over-fitting model.

In practice, a minimal model and a maximal model are formulated. Here, the minimum set of variables consisted only of  $k$  and  $q$  as the independent variables, while the full set of variables described in Section 4.4.1 (and used for the simple linear model (LM)) was used as the maximum model. The final optimal model returned was as follows:<sup>20</sup>

```
BIC <- lm(formula = Fmeasure ~ k + q + errorestimation +
skew + uniqueness + meanentropy + log(u) +
ginicoefficient + uniquepatternsA + hammingweight +
meanngrams + meanmissing + log(m) +
k:q + k:hammingweight + q:hammingweight +
k:q:hammingweight, data = fixed)
```

Compared with the linear model, this model includes slightly different variables. More interaction effects are included, as can be seen in the regression Table 4.11. With these variables, more than 90% of the variance of the mean prec./rec. can be explained by the model. Note that the selection of variables, even if optimal, does not remedy the violated assumptions of the linear estimation, as discussed in the previous section. To avoid an over-fitted model which will perform unsatisfactorily given new data, a different approach may be needed.

<sup>20</sup>The colon signals an interaction effect that includes all interactions between all variables included. For example, `k:q:hammingweight` can be written as the interaction terms `k*q`, `q*hammingweight`, and `k*hammingweight`, as well as the three-way interaction effect of the three variables.

Table 4.11: Regression tables for the simple linear model (LM) and the BIC-selected linear model (BIC). Confidence intervals are shown in braces.

Variables	Mean of precision and recall	
	Simple linear model (1)	BIC-selected linear model (2)
k	0.006* (0.004;0.007)	-0.014* (-0.018;-0.011)
q	-0.015* (-0.024;-0.007)	0.013* (0.006;0.019)
errorestimation	0.000* (0.000;0.000)	0.000* (0.000;0.000)
skewness	-0.000 (-0.001;0.000)	-0.001 <sup>+</sup> (-0.001;-0.000)
uniquepattA	-0.000* (-0.000;-0.000)	0.001* (0.000;0.001)
uniqueness	0.034 (-0.009;0.077)	-0.099* (-0.130;-0.068)
meanentropy		257.435* (219.256;295.614)
gini	-0.346* (-0.399;-0.294)	-739.042* (-848.302;-629.782)
hammingweight	-0.001* (-0.001;-0.000)	0.001* (0.001;0.001)
meanngr	-0.012* (-0.013;-0.010)	-5.532* (-6.338;-4.726)
meanmissing		26,496.220* (22,775.620;30,216.830)
log(m)		21.221* (18.066;24.376)
k*q		0.002* (0.001;0.003)
log(u)	-1,412.271* (-1,568.769;-1,255.773)	189,044.400* (157,114.600;220,974.200)
errorest*skewness	-0.000* (-0.000;-0.000)	
uniquepattA*uniqueness	0.000* (0.000;0.000)	
k*hammingweight	-0.000* (-0.000;-0.000)	0.000 (-0.000;0.000)
hammingweight*meanngr	0.000* (0.000;0.000)	
q*hammingweight		-0.000* (-0.000;-0.000)
k*q*hammingweight		0.000* (0.000;0.000)
Constant	1.349* (1.278;1.421)	520.200* (443.333;597.066)
Observations	1,120	1,120
R <sup>2</sup>	0.782	0.911
Adjusted R <sup>2</sup>	0.779	0.910
Residual Std. Error	0.032 (df = 1105)	0.021 (df = 1102)

Note: <sup>+</sup>p<0.05  
\*p<0.001

### 4.4.2.3 Response surface model (RSM)

Response surface model (RSM) designs, originally proposed by Box/Wilson (1951), are concerned with optimising a response variable with several controllable factors and independent variables (Myers et al. 2016).<sup>21</sup> The general premise of RSMs is that the unknown response function  $f$  for the factor variables that produce the output values for  $y$  is unknown and must be experimentally tested. For the notation, Myers et al. (2016) will be followed closely. Given a response function  $f$  with  $k$  factor variables  $\xi_i$  and an error term ( $\epsilon$ ), a response surface model can be written as follows:

$$y = f(\xi_1, \xi_2, \dots, \xi_k) + \epsilon. \quad (4.12)$$

To estimate a response function  $\eta$ , the factor variables will be standardised in some way<sup>22</sup>, giving  $x_i$  as the standardised values of  $\xi_i$ :

$$\eta = f(x_1, x_2, \dots, x_k). \quad (4.13)$$

As the response function  $f$  is unknown, it must be estimated. This is done using either first-order (FO) or second-order (SO) models. The first-order model for  $k$  independent variables is written as follows:

$$\eta = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k. \quad (4.14)$$

The second-order model can be expressed as follows:

$$\eta = \beta_0 + \sum_{j=1}^k \beta_j x_j + \sum_{j=1}^k \beta_{jj} x_j^2 + \sum_{i < j} \sum_{=2}^k \beta_{ij} x_i x_j. \quad (4.15)$$

According to Myers et al. (2016), if a strong curvature is expected in the response surface, the second-order model is more appropriate. Based on the figure and the example from Myers et al. (2016: 5), Figure 4.7 shows the example RS for the following second-order model:

$$\eta = 50 + 8x_1 + 3x_2 - 7x_{11}^2 - 3x_{22}^2 - 4x_1 x_2. \quad (4.16)$$

<sup>21</sup>The idea to use RSMs for the particular problem of this thesis was devised by Rainer Schnell.

<sup>22</sup>E.g. internally studentised (normalised by dividing the values by their estimated standard distribution) or mean-centered.

As can be seen, the resulting response surface resembles the curvature seen in most plots concerning linkage quality. An example would be Figure 3.19 on page 82. The mean prec./rec. value increases steadily, before falling rapidly at a certain threshold. For such an expected curvature, the second-order model seems to be appropriate. Myers et al. (2016: 282ff) calls these response functions a *ridge system*.<sup>23</sup>

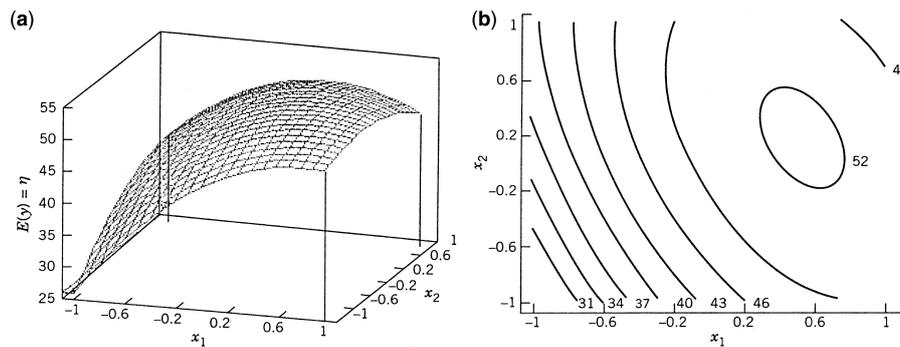


Figure 4.7: Example response surface (a) and contour plot (b) for the second-order model given in Equation 4.16. Figure from Myers et al. (2016: 5).

Response surface methods require experimental designs with several *phases* of the model fitting (Myers et al. 2016):

- Phase zero: Variable selection. This has essentially been covered by the variable selection for the linear models. The so-called *region of operability* (parameter space) is determined as well.
- Phase one: Finding the area of optimal response with respect to the current parameters. This is usually done by the first-order model using the method of the steepest ascent. This is not required here, as the *region of operability* is tested in full.
- Phase two: Finding the actual optimal values using the second-order model in a constrained parameter space. This is done here by fitting the model to the training data.
- Phase three: Doing a confirmatory experiment. Here, this is done evaluating the test datasets as described in the next chapter.

The final objective of phase two is to find the optimal parameters. For our application, the fitting of the actual second-order model using the training

<sup>23</sup>As will be shown later, the response function will resemble a *stationary ridge system* (Myers et al. 2016: 282f).

data is visualised in Figure 4.8. The visual analysis shows values of  $q = 2$  and  $k$  between 1 and 5 to be optimal for the full set of training data. Automating this visual decision for a given input dataset is the central task. To implement RSMs in R, the package `rsm` was used (Lenth 2009).

The main disadvantage of the RSM model building was the strictness of the implementation: Interaction effects, even if made explicit (as was the case for the interaction between  $k$ ,  $q$  and the resulting hamming weight) often led to non-convergence for aliased identifiers. Sadly, this problem could not be resolved. Therefore, the RSM was built as a very minimal model, which was the only one that converged with all the variables possible.

Therefore, the final model output for the second-order model (SO) with a two-way interaction (TWI) between  $q$ ,  $k$  and the hamming weight was as follows:

```
rsm.train <- rsm(Fmeasure ~ SO(k, q) +
  TWI(q, k, hammingweight) + u, data = RSMdata)
```

Table 4.12: Regression table for the response surface method.

<i>Variables</i>	Estimate	Std. Error	<i>t</i> -value	Pr(>   <i>t</i>  )
Constant	0.928	0.013	66.83	$<2.200 \times 10^{-16}$ *
k	-0.208	0.061	-3.36	$8.055 \times 10^{-4}$ *
q	0.227	0.045	5.05	$5.059 \times 10^{-7}$ *
k:q	1.463	0.135	10.80	$<2.200 \times 10^{-16}$ *
$k^2$	-1.129	0.125	-9.01	$<2.200 \times 10^{-16}$ *
$q^2$	-0.276	0.036	-7.55	$8.746 \times 10^{-14}$ *
q*hammingweight	-1.293	0.141	-9.12	$<2.200 \times 10^{-16}$ *
k*hammingweight	1.161	0.152	7.63	$5.031 \times 10^{-14}$ *
u	-0.259	0.008	-30.35	$<2.200 \times 10^{-16}$ *
Multiple R-squared:	0.517			
Adjusted R-squared:	0.513			
F-statistic:	148.6 on 8 and 1111 degrees freedom			
p-value:	$< 2.2 \times 10^{-16}$			

Note: \* $p < 0.001$

Table 4.12 gives a more detailed look at the model's independent variables. The RSM explains only approximately 51% of the variance of the training data.

This might lead to poor performance when a parameter estimation of a data set that is radically different from the training data is needed.

The resulting surface plot for finding the optimal parameters is shown in Figure 4.8.

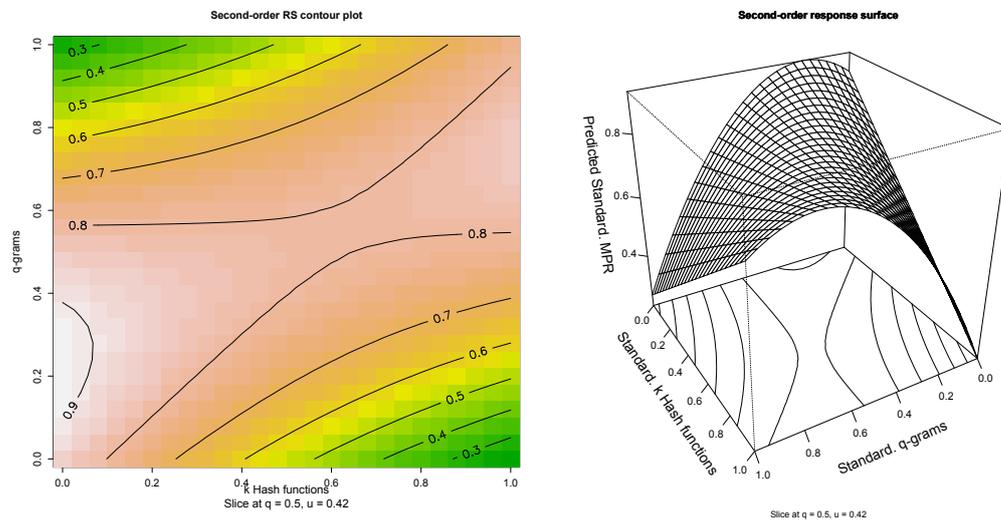


Figure 4.8: Response surface plot and contour plot of the final RSM-based prediction of the training datasets. Note that all the values are standardised to be between 0 and 1, as this is required to fit the model appropriately (Myers et al. 2016).

As can be seen, the response surface follows a static ridge model. The optimal parameter space can be seen in the contour plot below the surface. The estimates of the RSM model for the datasets are presented in Section 4.5.3.

#### 4.4.2.4 Random Forest-based approach (RF)

Random Forests (RF) were first proposed by Ho (1995) as *random decision forests* as an extension to “regular” decision trees. Decision trees were covered in Section 1.5.6.2 as part of the machine learning approaches for classifying matches and non-matches. Here, trees will be used for building a regression model. Therefore, the explanation of Random Forests will concentrate on Random Forest-based regression and regression trees.

To understand Random Forests, it is more efficient to first look at single regression trees. Table 4.13 shows a training data file input into a decision tree. It contains four independent variables ( $k$ ,  $m$ ,  $u$  and hamming) that are thought to influence the mean of precision and recall. The goal is to best estimate the

dependent variables given the independent variables. This is done by split lines, or, in the case of multiple variables, a hyperplane that best splits the data into two categories (Ho 1995). A simple linear regression is then done separately for each of the (two) splits. Figure 4.9 shows a very crude regression tree using just two split points for a more straightforward visualisation.

Table 4.13: Example training data file that can be input into a regression tree.

Small training data with true resulting metric				
k	m	u	hamming	mean prec./rec.
10	0.02	1.00	20	0.95
5	0.04	0.99	12	0.94
20	0.01	1.00	38	0.70
16	0.06	0.99	30	0.99

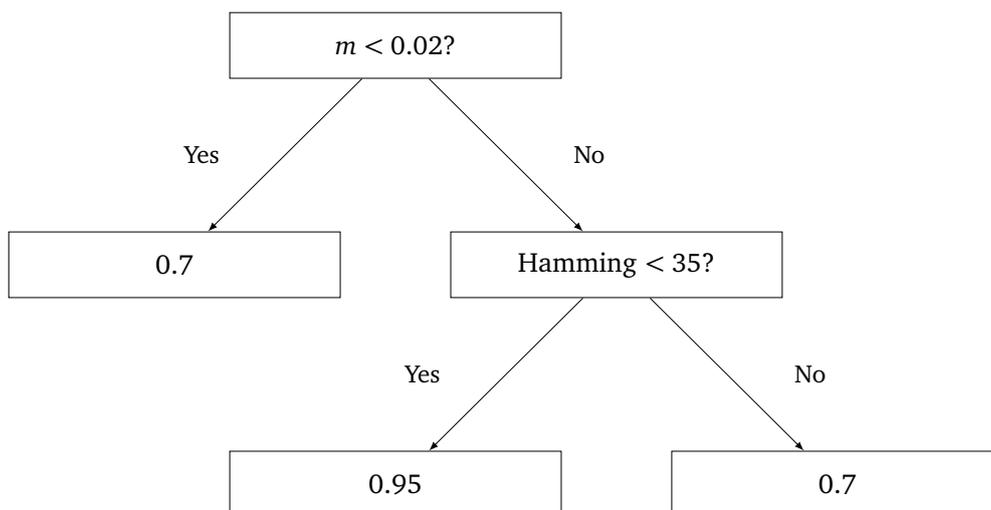


Figure 4.9: Example of a single regression tree with the estimates for the mean prec./rec. that could be built from the the example training data shown in Table 4.13.

In reality, the hyperplanes would be fitted optimally, generating two new sub-leaves with cases on either side of the hyperplane in the multi-dimensional space. For each leaf, fitting a hyperplane and splitting the remaining records are repeated until the terminal leaves contain only a single record. This gives a single regression tree with many separate linear estimates for several data points in the parameter space. However, a single tree will always be optimal for

making perfect decisions with the given data, possibly overfitting the training data (Ho 1995). To counteract this, Ho (1995) proposed building multiple trees (forming a “forest”).

In practice (Liaw/Wiener 2002), this is done by drawing  $B$  bootstrap samples<sup>24</sup> from the training data. For each sample  $B_i$ , a tree is built as described above. This will give several different trees, as each sample should have differing characteristics. All the trees together form the Random Forest.

For a given input (like Table 4.13), all the values of every single record will be predicted (or classified) by every single tree in parallel. This will give  $B$  regression estimates (or, in the case of classifications, decisions). These are averaged to provide a regression estimate. Decision trees for classification usually use majority voting to generate a singular classification (Liaw/Wiener 2002). For both, the process is visualised in Figure 4.10 with several trees forming the random forest.

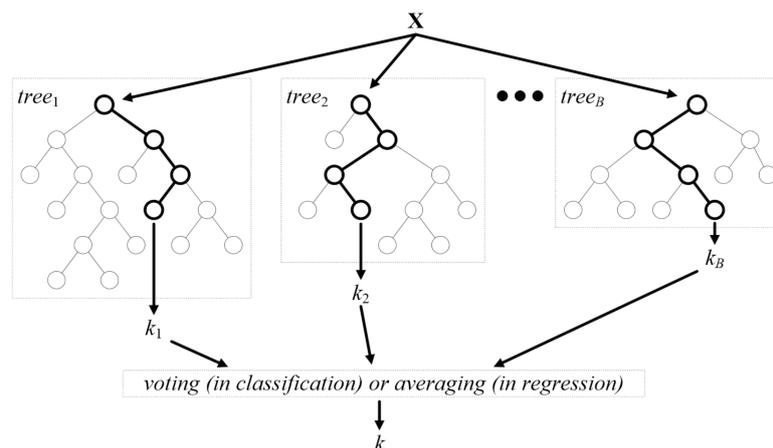


Figure 4.10: The Random Forest approach for a single input ( $x$ ), with one output ( $k$ ) and  $B$  random decision trees. Figure by Verikas et al. (2016: 601).

The advantage of a Random Forest-based regression is that by splitting the regression into multiple smaller spaces in the entire parameter space, the drop points where the linkage quality drops sharply (see Section 3.5.3) may be better estimated, as split points can predict the drop-offs. On average, the optimal solution should give a reasonable estimate of the best fit. In an application

<sup>24</sup>*Bootstrapping* means re-sampling a given dataset or sample  $B$  times with replacement and the sample size of the original data or sample. Usually, this is done to obtain confidence interval estimates for distributions with unknown parameters (Myers et al. 2010).

for linking addresses, Comber/Arribas-Bel (2019) found that this led to better performance when compared with the standard regression techniques.

Another advantage of Random Forests is their nonparametric nature. While a linear regression requires linearity, Random Forests do not. Figure 4.11 shows this by plotting the true values of a logarithmic transformation of 1000 random numbers and the estimates by a linear regression and a Random Forest-based regression. The forest-based solution closely follows the logarithmic curve.

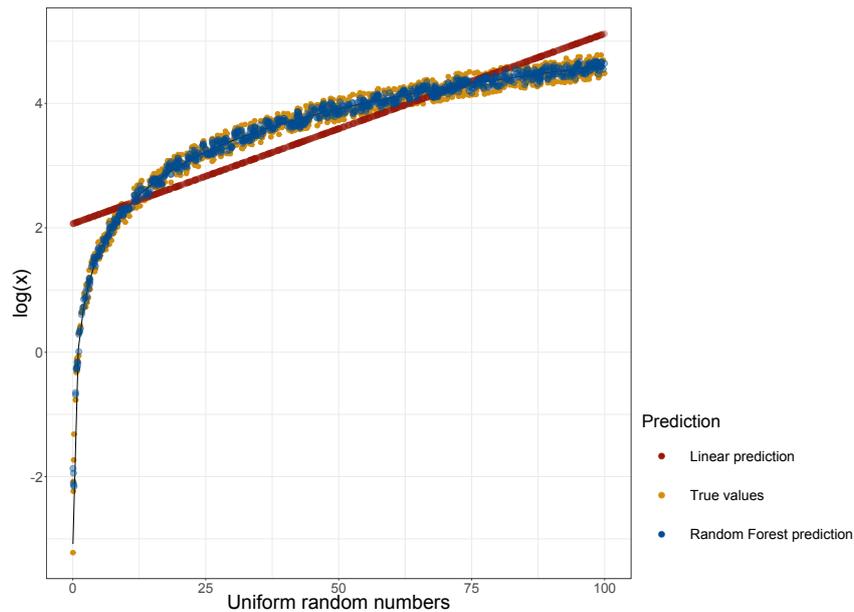


Figure 4.11: Approximating a logarithmic transformation of uniform random numbers using a linear regression and Random Forest-based regression.

For the actual implementation, the R package `randomForest` (Liaw/Wiener 2002) was used. The model specification follows the formulation of the simple linear model (LM, see Section 4.4.2.1), with a few parameters unique to Random Forests:

```
RF <- randomForest(Fmeasure ~ k + q + errorestimation +
  skew + uniqueness + ginicoefficient + hammingweight +
  meanengrams + u + uniquepatternsA, data = fixed,
  ntree = 500, nodesize = 20,
  importance = TRUE, nPerm = 5, keep.forest = TRUE)
```

Several of these standard parameters were changed from their default: As more trees should lead to better estimates (Liaw/Wiener 2002), the number of trees

(`ntree`) was set from 500 to 3000. This led to massive overfitting. Therefore, it was set back to 500. The minimal node size was comparable to the *leaf limit* for Multibit trees (see Section 3.4): The lower the limit was, the larger was the number of sub-leaves generated. For regressions, using larger values is recommended (Liaw/Wiener 2002). Therefore, the node size was set to a value of 20. Importance weighting (parameter `importance`) sampled variables from the variable set to estimate the relative importance of them in the model (see Figure 4.12). The estimates were weighted by this importance estimator. This led to more stable estimates, particularly when this process is repeated (i.e. `nPerm` is higher than one). Changing the number of permutations or the number of trees to more than five changed nothing in terms of prediction or estimation, but increased the time taken for the construction of the model.

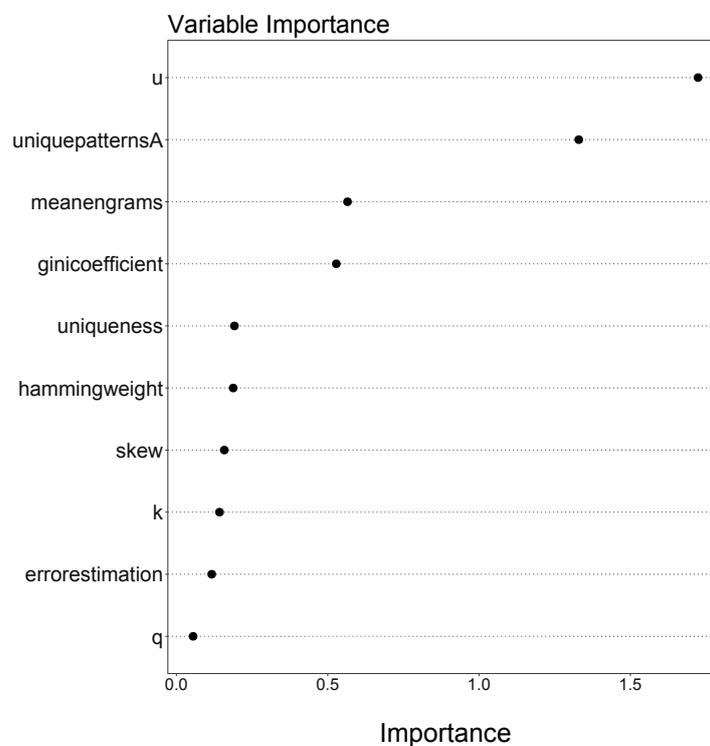


Figure 4.12: Variable importance for the Random Forest model.

As can be seen in the variable-importance plot, *u* is deemed to be the most important variable in the model. The model explains approximately 96% of the variance of the mean prec./rec. of the training data. This very high  $r^2$  value may be problematic if the model overfits the data.

## 4.5 Resulting estimated and best-practice parameter choices

Before the central evaluation of the models described above against the current best-practice solutions, the actual parameters used in the literature, the “optimal” parameters resulting in half of the Bloom filters bits to be set to 1 and the estimated optimal parameters given by each of the four models will be reported for each dataset.

### 4.5.1 Using recommendations from the literature as best-practice suggestions

As the parameter choice of Bloom filter-based methods was never formally settled, several best-practice suggestions have been given over time. Table 4.14 provides an overview of the selected works and their respective parameter choices. Some of these are implicit choices used in evaluations (marked with <sup>E</sup>), and the others are explicit recommendations (marked with <sup>R</sup>).

Table 4.14: Selected publications and parameters tested or suggested.

Source	$k$	$q$	$l$	$k_{l=500}$	Note
Izakian (2018) <sup>R</sup>	50	1	1000	100	Parameter sets for optimal approximation of the clear-text similarity ( <i>not</i> for optimal linkage quality)
	20	1	500	20	
	50	2	1000	25	
	50	3	500	50	
Schnell/Borgs (2018c) <sup>E</sup>	20	2/1	1000	10	
Dantas Pita et al. (2018) <sup>E</sup>	2	2	180	≈ 5	Optimised for memory/speed
Lazrig et al. (2018) <sup>E</sup>	15	2	1000	≈ 8	Actual $k$ (15) only hinted
Brown et al. (2017a) <sup>E</sup>	10	2/1	1000	5	
Schnell (2016a) <sup>E</sup>	20	2	1000	10	
Schnell (2014) <sup>R</sup>	10–20	2	500/1000	5–10	Only explicit range published
Schnell (2014) <sup>E</sup>	20	2	1000	10	
Sehili et al. (2015) <sup>E</sup>	20	2	1000	10	
Randall et al. (2014) <sup>E</sup>	3	2	100	15	Very large database
Vatsalan et al. (2014) <sup>E</sup>	30	2	1000	15	
Schnell et al. (2014) <sup>E</sup>	10	2	1000	5	
Kuzu et al. (2011) <sup>E</sup>	15	2	500	15	First published attack
Schnell et al. (2009) <sup>E</sup>	30	2	1000	15	Original publication

<sup>E</sup>: Evaluation with these parameters only.

<sup>R</sup>: Recommendation to use these parameters.

Therefore, the two most prominent parameter choices were tested using  $q = 2$  (splitting the identifiers into bigrams), which was predominantly used in the literature. As the length of the Bloom filters in the evaluations was  $l = 500$  (see Section 4.3.1), the number of hash functions  $k$  was adjusted to maintain the fraction  $\frac{k}{l}$  constant, which led to the same number of bits set to 1, resulting in equivalent linkage quality. This is shown in Table 4.14 with  $k_{l=500}$ . The choices for  $k$  varied. The most frequently used choice was  $k_{l=500} = 10$  (which was also the median of the distributions of  $k$  used for  $q = 2$ ). Both  $k_{l=500} = 5$  and  $k_{l=500} = 15$  were used the same number of times. However, as the explicit range given by Schnell (2014) was between  $k_{l=500} = 5$  and  $k_{l=500} = 10$ ,  $k_{l=500} = 5$  was the second best-practice parameter tested. Moreover,  $k_{l=500} = 15$  was very close to what the 50% rule (see the next sub-section) predicted as the optimal  $k$  (see Table 4.15).

## 4.5.2 Estimating an optimal $k$ using the 50% rule

As a fraction of set 1-bits of 0.5 is supposedly optimal (for a discussion of this assumption see Section 3.5.3 and Kirsch/Mitzenmacher (2006)), following the implementation of Smith (2017), the optimal number of  $k$  hash functions was calculated as follows (Kirsch/Mitzenmacher 2006):

$$k_{opt} = \frac{l}{m} * \log(2), \quad (4.17)$$

where  $m$  is the number of elements to be hashed into the Bloom filter. As this value is not fixed in most PPRL applications, the mean number of elements ( $q$ -grams) was taken to obtain a global optimal  $k$ -value. For simplicity, the value of  $q$  was fixed to  $q = 2$ , optimising  $k$  for bigrams, as suggested by the settings in the literature (see Table 4.14). This did not negatively affect the linkage quality as compared to the other choices. The resulting optimal  $k$  can be seen in Table 4.15 below.

## 4.5.3 Optimal parameters chosen by the models

With the models described in Section 4.4.2 and linking the datasets presented in Section 4.3.3, optimal parameter estimates for  $k$  (number of hash functions) and  $q$  (sub-string length) were derived. Table 4.15 shows the resulting param-

Table 4.15: Optimal parameter choice estimations for all datasets as given by each model for a Bloom filter length of  $l = 500$  bits.

Data	$k_{50\%}$	$k_{LM}$	$k_{RF}$	$k_{BIC}$	$k_{RSM}$	$q_{50\%}$	$q_{LM}$	$q_{RF}$	$q_{BIC}$	$q_{RSM}$
<i>Training</i>										
Telephone CD 0% err.	17	12	29	1	1	2	1	3	1	1
Telephone CD 20% err.	17	12	16	1	1	2	1	1	1	1
FEBRL	18	15	8	7	1	2	2	1	1	1
Mortality & Hospital	17	12	10	11	1	2	2	1	1	1
<i>Testing</i>										
NC Voter	27	30	26	12	1	2	2	1	1	1
Mortality & Comm.	17	27	17	40	1	2	4	1	4	1
FEBRL WA	20	24	29	18	1	2	2	1	1	1

eters given by the models and the 50% rule. The best choices for  $k$  might seem counter-intuitive at first but produced better quality than any best-practice recommendation, as will be demonstrated later.

The results obtained using the linear model and the BIC-based selection model produced many equivalent choices, where values for  $k$  and  $q$  were chosen in a way that they would produce the approximately same number of bits set to 1 and similar linkage quality. Despite this, the best estimates were used.

For the four models, the optimal  $q$  was estimated along with the optimal  $k$ . For the 50% rule,  $q$  was fixed at 2.

Using these parameters, the training and evaluation datasets were encrypted and linked, comparing the resulting linkage quality for each model or best-practice suggestion. This will be the central part of the next chapter.

# Results of the optimal parameter choice

The central goal of this thesis is the optimal parameter choice estimation. Given the training and test datasets as described in Section 4.3.3, all the results testing the model-based parameters against best-practice methods will be presented in this chapter.

## 5.1 Overview of the chapter

First, the training and test dataset strategy is discussed. The terminology and measures are recapped before presenting the central results of the comparison between the models and the best-practice suggestions of the parameters for Bloom filter-based encryption methods. Lastly, variance estimations for the models using cross-validation will be given. The chapter closes with a summary of the results.

## 5.2 Recap: Training and test data

To understand the plots, all the datasets used for training and the evaluation will be briefly reiterated. A more extensive discussion about the datasets can be found in Section 4.3.3.

### 5.2.1 Training data

The data used for building the models and finding the response function for the RSM are shown in Table 5.1.

Table 5.1: All the training datasets used to train the models with their main language population and their data type (real-world or simulated).

Data	Population	Type	Usage
Mortality data	German	Real-world	Training
Telephone CD 20% err.	German	Simulated	Training
Telephone CD 0% err.	German	Simulated	Training
FEBRL	Australian	Simulated	Training

As these were used to train the models, the evaluation concentrated on the test datasets. However, the use of the estimated parameters on them and the resulting linkage quality are reported for completeness.

### 5.2.2 Test data

The data used for evaluating the optimal parameters given by the models against the current best-practice solutions are shown in Table 5.2.

Table 5.2: All the test datasets used for evaluating the model against commonly suggested settings with their main language population and their data type (real-world or simulated).

Data	Population	Type	Usage
NC Voter data	American	Real-world	Testing
Mortality & Commercial data	German	Real-world	Testing
FEBRL WA	Australian	Simulated	Testing

Before testing these, the settings of the Multibit trees used for linkage and the model diagnostics are reported.

## 5.3 Multibit tree settings

For all the evaluations, Multibit trees were used with several Tanimoto thresholds. For further details on the construction of MBTs, see Section 3.4. The Tanimoto thresholds tested were  $T \in \{1.00, 0.95, 0.90, 0.85, 0.80\}$ . The *leaf limit*,

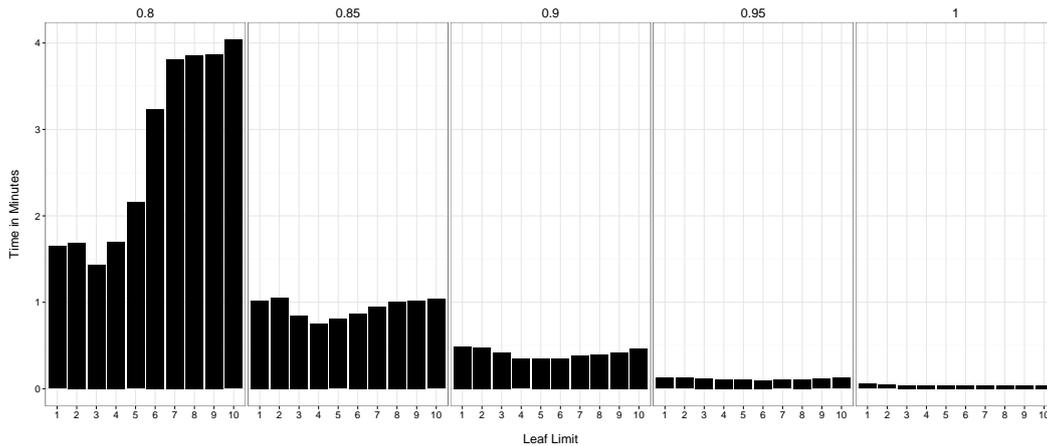


Figure 5.1: Leaf limits and time taken to link  $n = 100,000$  CLKs by Tanimoto threshold (0.8 to 1.0). Leaf limits of three or four seem to result in faster linkage times for lower thresholds than the standard setting by Kristensen et al. (2010) – they proposed a leaf limit of 8.

the final number of elements in each leaf before building further sub-leaves, was set to a value of three, as tests (see Figure 5.1<sup>1</sup>) showed this to be the computationally most advantageous setting.

Multibit trees allow for multi-threading. Given the constraints of the computer used for testing, the number of threads to use was set to 7 (out of 8). Neither Symdex preprocessing nor union bit trees (see Section 3.4) were used. The package returns ID pairs considered to be matched pairs. These are evaluated as described in Section 1.5.7, with the mean of precision and recall (MPR) being the central measure for linkage quality.

## 5.4 Model diagnostics

Before looking at the central results of model-based approaches to best-practice recommendations, the models themselves will be evaluated, and their resulting response surfaces will be visualised, where possible. Possible drawbacks resulting from this will be discussed as well.

<sup>1</sup>There is an unpublished working paper about the details of the tests (Klingwort et al. 2015):  $n = 100,000$  FEBRL-generated records were encrypted using CLKs containing bigrams of first and last name and unigrams of the full date of birth, hashed into a CLK of length  $l = 1000$  using  $k = 20$  hash functions. The time taken to link the files using Multibit trees was recorded for each Tanimoto threshold and leaf limit. This result has not been published yet.

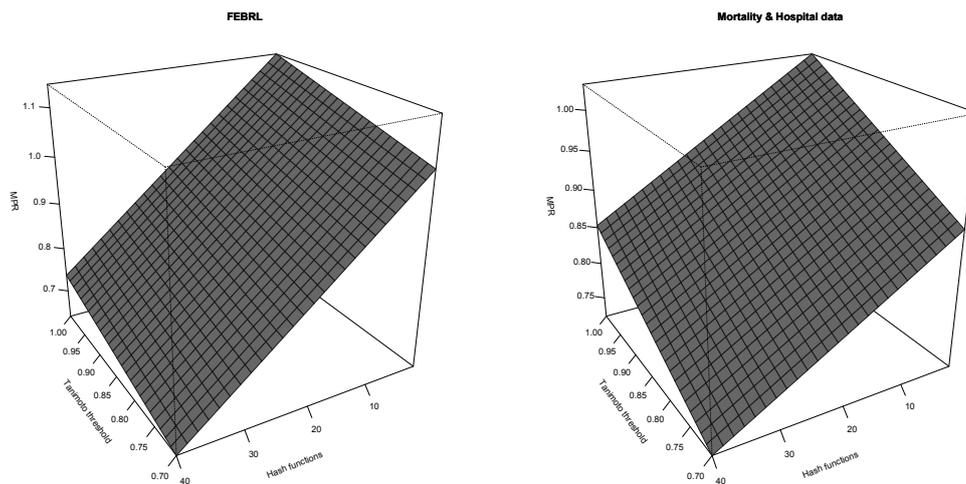


Figure 5.2: Surface plots for the FEBRL data and the NC voter data. Both show the Tanimoto threshold and the number of  $k$  hash functions along with the resulting mean of precision and recall.

As can be seen in the surface plots in Figure 5.2, different data characteristics lead to different results, even with the same identifiers and parameter choices. To check whether the assumptions for the models (see also Section 4.4.2) hold, the diagnostic plots are recapped here.

### 5.4.1 Assumption violations

Figure 5.3 shows the diagnostic plots for the LM model, Figure 5.4 shows the same plots for the BIC model, Figure 5.5 shows the diagnostics for the RSM and Figure 5.6 gives the diagnostics for the Random Forests.

The discussion of the diagnostics for the linear model was already part of Section 4.4.2.1. Note that each of the “curves” in the diagnostic plots is a setting for  $q$  for different datasets. The conclusion was mixed: Some assumptions hold, but the required linearity and co-linearity are violated. Furthermore, the assumption of homoscedasticity may be violated, as the standardised residuals are larger at low and high fitted values. There are also cases with high leverage where the model prediction and the true linkage quality diverge sharply.

The same results are observed for the BIC model. The same assumptions are violated, which was expected. A small difference is that there are fewer high-leverage outliers, but these are more pronounced.

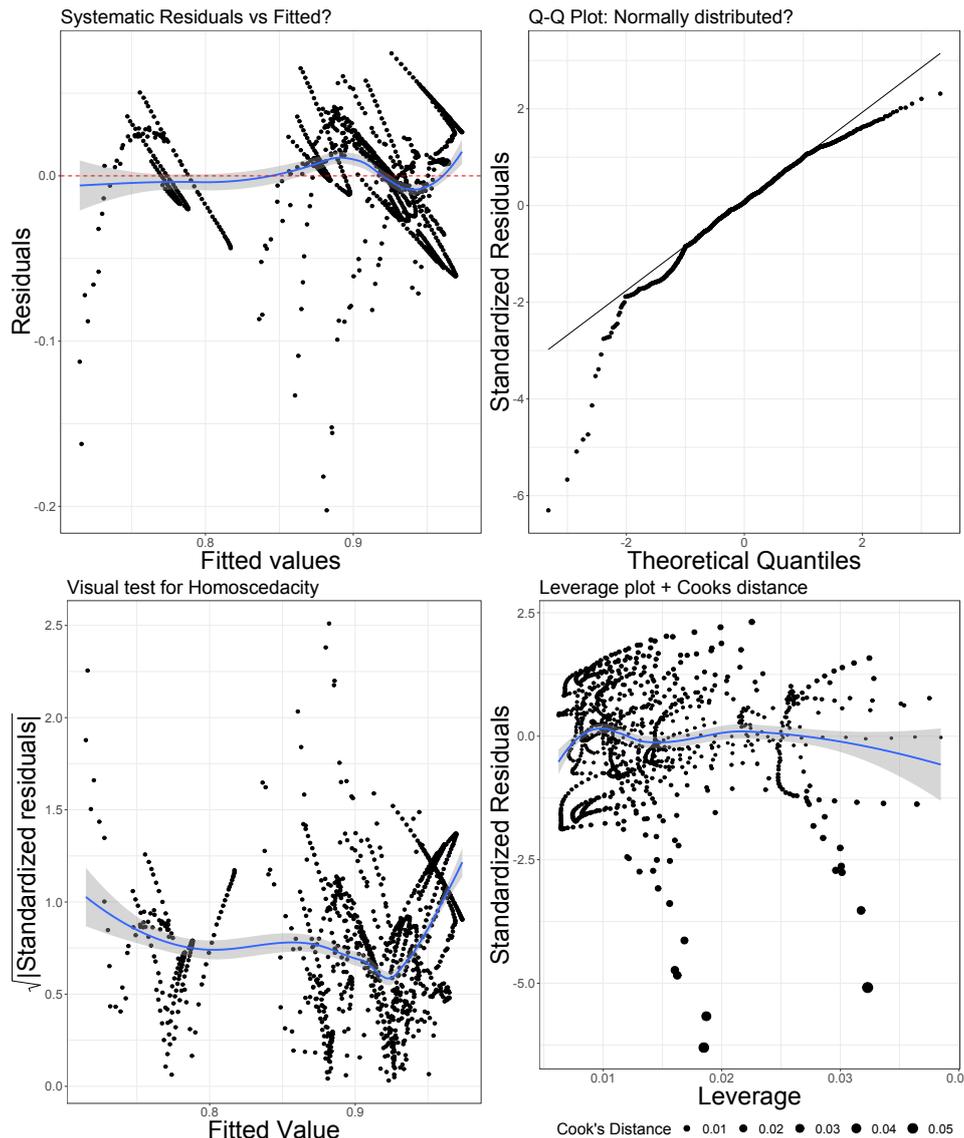


Figure 5.3: Diagnostic plots for the simple linear model (LM).

For the RSM, most of the linear assumptions do not hold. There is heteroscedasticity, as the root standardised residuals seem to get smaller with higher fitted values. Linearity is violated for higher fitted values. The leverage is very high for some data points, but its distribution is very different from that of the linear models.

The outliers in the diagnostic plots for BIC and LM is from the mortality and commercial dataset, where the predicted values differed greatly from the actual mean prec./rec. at the extreme ends of around  $k = 1$  and  $k = 40$  hash functions. This is because, as can be seen from the Q-Q plot, there is no perfect

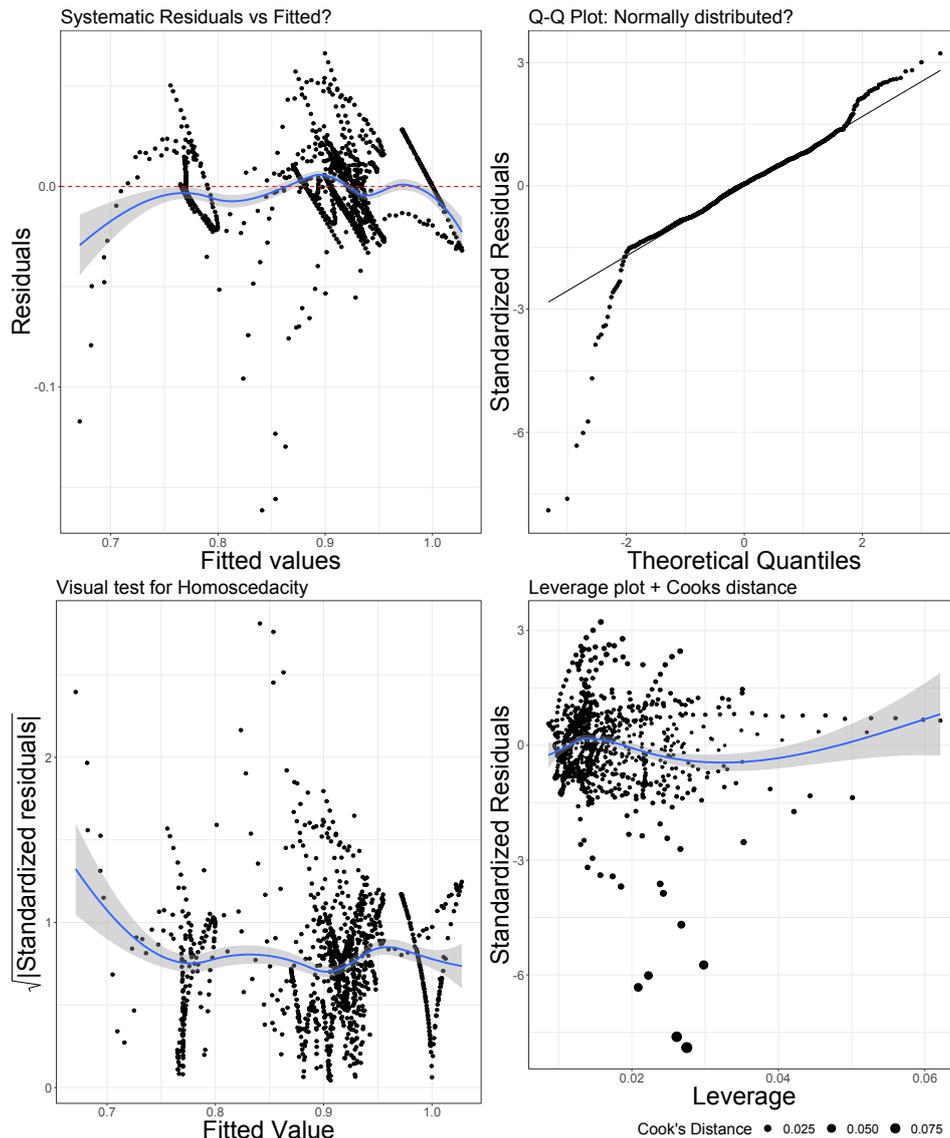


Figure 5.4: Diagnostic plots for the linear model based on variable selection by the stepAIC function using the BIC as the criterion (BIC).

linear correlation between the variables and the mean of precision and recall. This is particularly true for the endpoints, where the resulting linkage quality is not linear in nature, as the linkage quality drops sharply after a certain number of hash functions, as discussed in Section 3.5.3.

The diagnostic plot of the Random Forest-based regression (Figure 5.6) provides more details: Here, the datasets and the differing  $q$  values are shown as well. As can be seen, the residuals are still systematically smaller when approaching higher fitted values, but the effect is considerably stronger for a sub-string length of  $q = 1$  (unigrams); the strength of the effect differs for

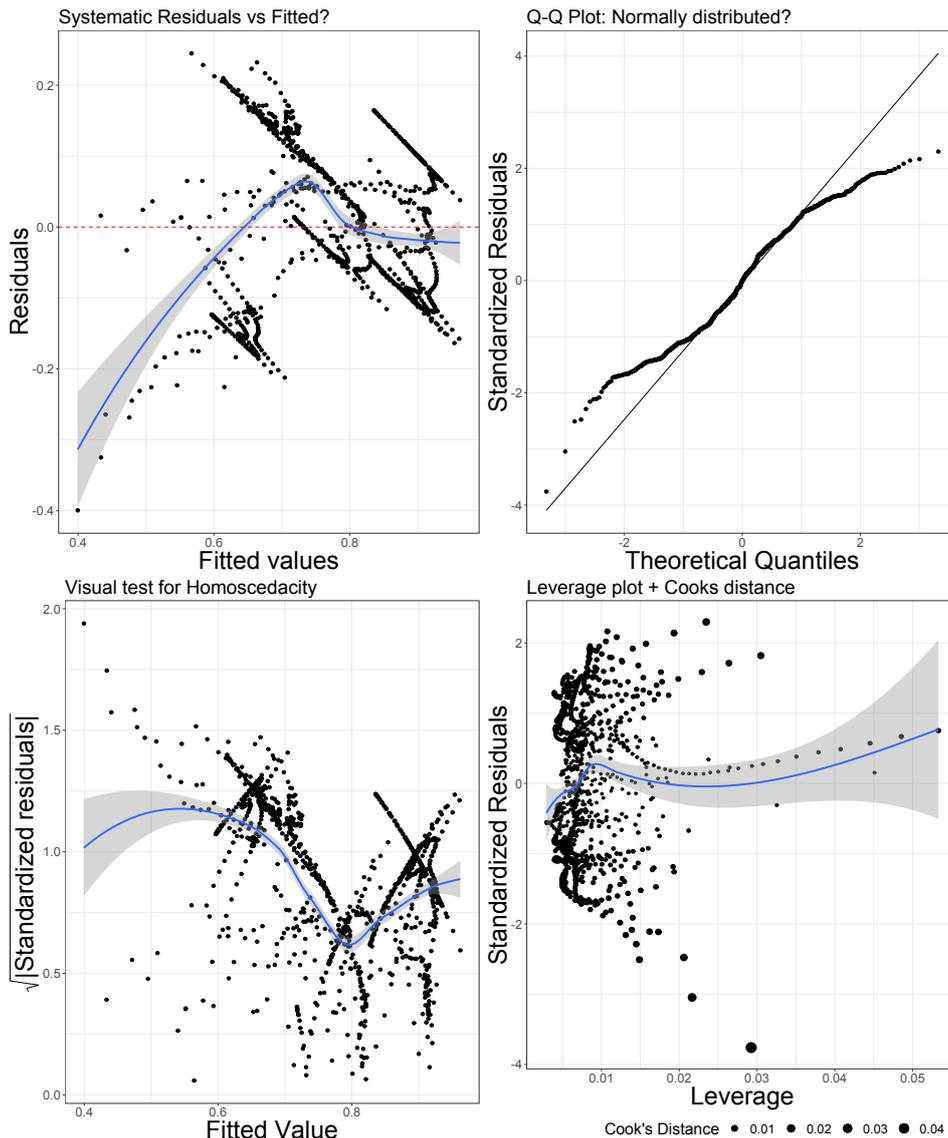


Figure 5.5: Diagnostic plots for the response surface model (RSM).

each dataset. The root standardised residuals are smaller for higher predicted mean of precision and recall – here, the effect of the datasets is even more pronounced than that seen in the other models.

To sum up, all the models violate some of the assumptions for linear regressions (see Section 4.4.2.1), which will possibly be detrimental to the performance of the models.

However, for most of the datasets and models, the predicted linkage quality was strongly associated with the true linkage quality.<sup>2</sup> Figure 5.7 presents an

<sup>2</sup>Note that  $r$  or  $r^2$  are inadequate measures to determine (non-linear) model quality

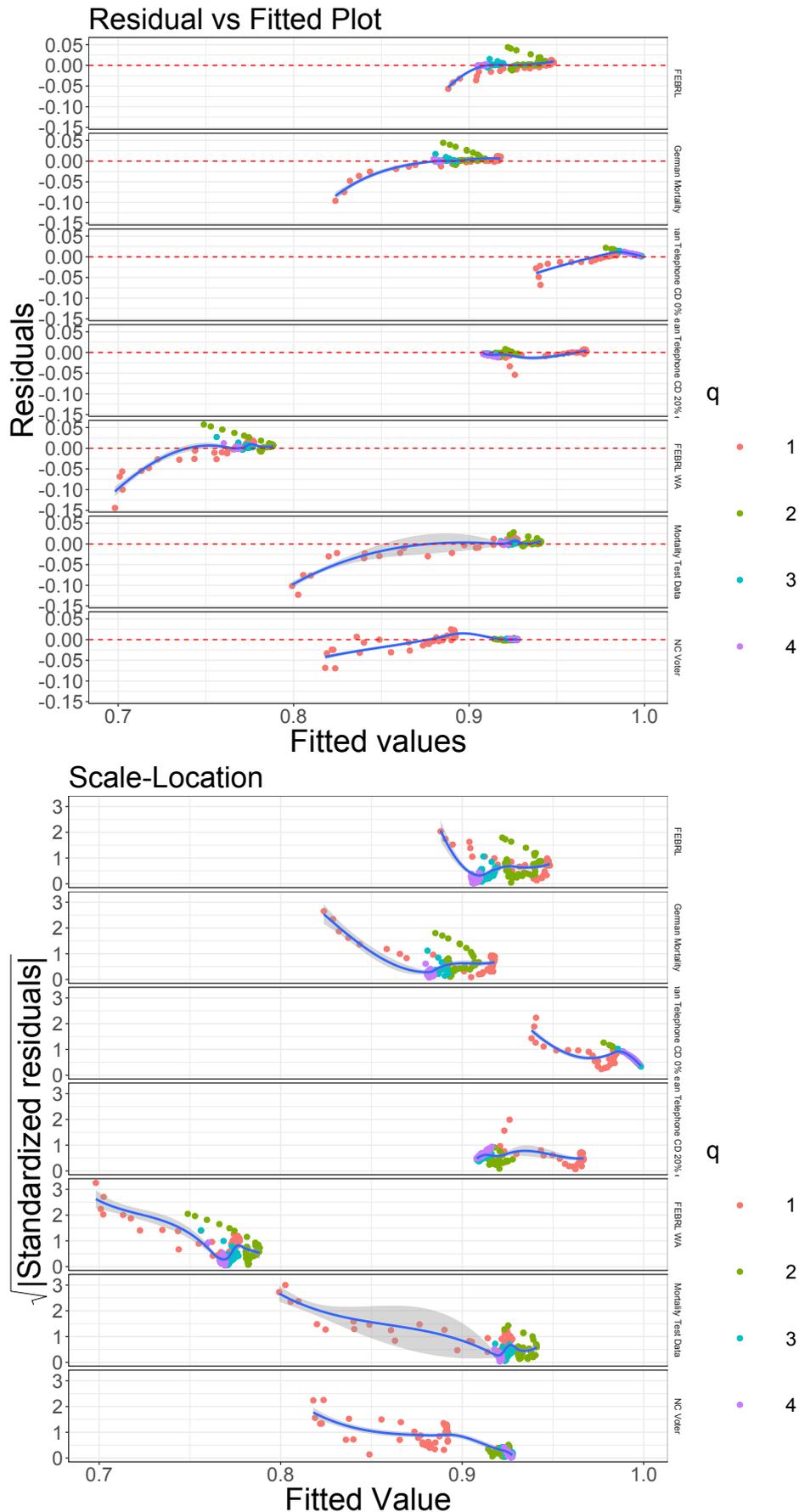


Figure 5.6: Diagnostic plots of the Random Forest-based regression (RF) for each dataset and for all the settings of  $q$ .

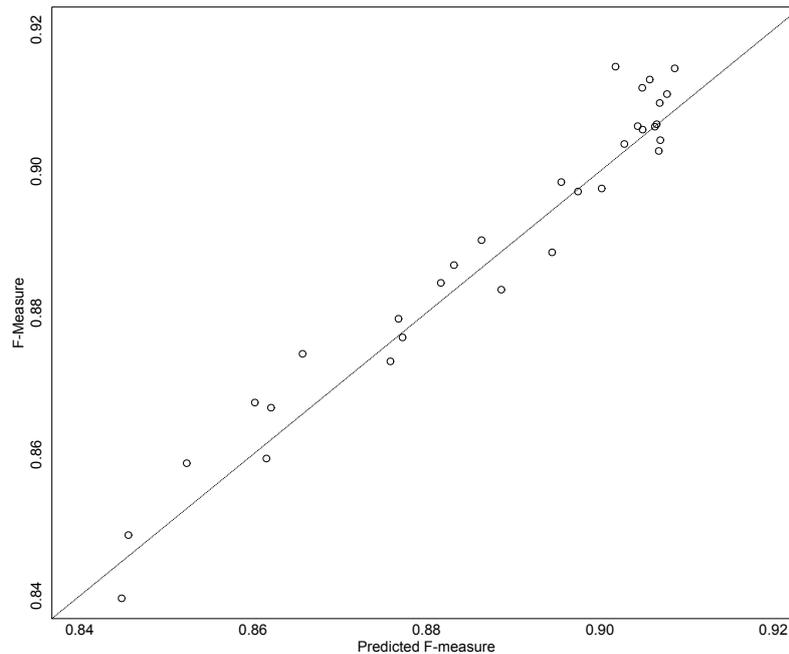


Figure 5.7: Predicted mean prec./rec. and actual mean prec./rec. for different  $k$  with  $q = 1$  for the NC Voter data using the BIC-selected linear model. Correlation coefficient is  $r = 0.983$ . The dotted line represents  $r = 1$ .

example for the NC Voter dataset, where the BIC-selected model prediction and the true values increase monotonically. This was not true for the other datasets, such as the FEBRL WA data, where the prediction shows the same characteristics before predicting a falling mean prec./rec. for the endpoints of the parameter space (which would have been optimal). This would lead to a maximum predicted mean prec./rec. that in turn would lead to a sub-par parameter choice.

In contrast, the Random Forest model fits the data very well, even trying to estimate the drop-off in the linkage quality at certain points, as can be seen in Figure 5.8. Here, the same problem as that mentioned above is observed: The actual optimal parameter choice is estimated to be behind the drop-off, resulting in it not being selected. However, the best-estimated parameter choice is still an excellent choice, as it is one of the best actual settings. Therefore, as described in the previous chapter, all the models used the parameters of the maximum predicted mean of precision and recall as the optimal parameter choice estimate.

---

(Spiess/Neumeyer 2010). Therefore, a cross-validation of the models was done (see Section 5.4.2).

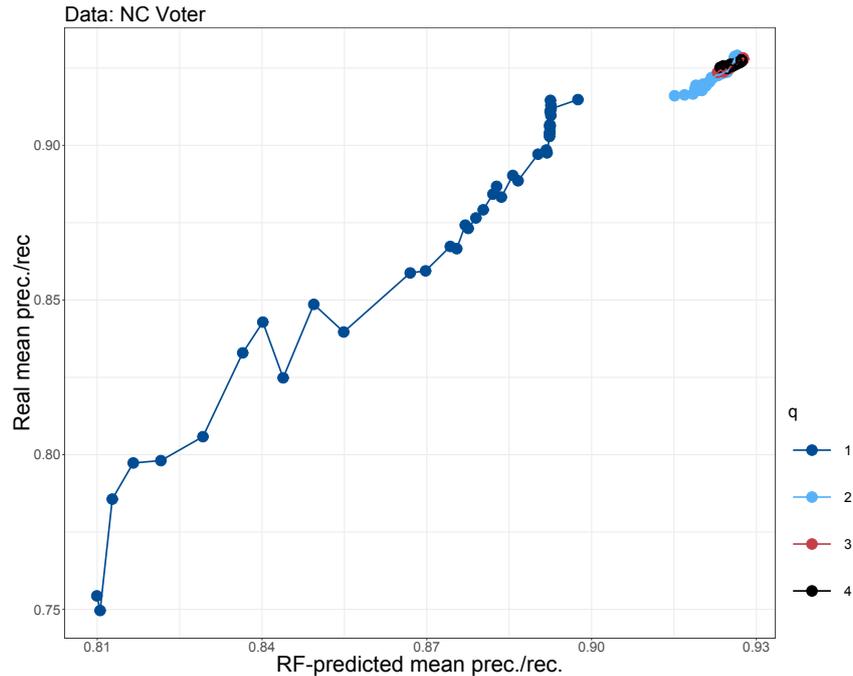


Figure 5.8: Predicted mean prec./rec. and actual mean prec./rec. for the NC Voter data using Random Forest-based regression for the four choices of  $q$ .

## 5.4.2 Cross-validation and root mean squared error (RMSE)

As discussed in Sections 4.4.2 and 5.4, the assumptions of the linear models do not hold. This may lead to problems with the estimation of the optimal parameters. This is why cross-validation, root mean squared error (RMSE) and standard error (SE) estimations will be considered as well. Cross-validation and standard error estimation were performed using the `cvTools` package for R (Alfons 2012).

The mean squared error (MSE) for data with  $n$  records was calculated as the mean squared difference of the predicted values of a model ( $\hat{y}_i$ ) and the true values ( $y_i$ ):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2. \quad (5.1)$$

The RMSE is, consequently, the square root of the MSE.

For validating models, K-fold validation can be used (James et al. 2013). This works by estimating the mean squared error (MSE) by splitting the observation

data into  $K$  subsets of equal size. These are called *folds*. Usually,  $K = 5$  or  $K = 10$  folds are used (James et al. 2013: 183). One fold is used as a validation dataset, while the other folds are used to train the model. This gives  $K$  different estimates and error estimations. For each repetition, the MSE is estimated. To obtain the reported coefficient of variation (CV) estimates and the standard errors (SE), the mean of all the MSEs is used (James et al. 2013):

$$CV_{(K)} = \frac{1}{K} \sum_{i=1}^K \text{MSE}_i. \quad (5.2)$$

For this application,  $K = 5$ -fold validation was replicated 30 times.

The resulting measures are shown in Table 5.3. The model-based errors and their variation are comparatively small, with the notable exception of the BIC-selected linear model and the simple linear model, which show marked increases in the RSME for the evaluation data. This could suggest that for the RSM and the RF-based regressions, despite the assumption violations, the results are neither biased strongly nor do they vary considerably when replicated. With this (and the assumption violations) in mind, the central evaluation will follow.

## 5.5 Results

Cross-validation has shown that the coefficient of variation and the standard error of the predictions of the models are small, as is the root mean squared error (RSME), whereas the diagnostic plots show that the models have several problems – most of them might be explained by the difficulties of predicting the correct drop-off point before the linkage quality drops. This makes the estimates somewhat unreliable at the fringes of the parameter space. To check whether the results are still superior to both choosing the number of hash functions  $k$  according to the 50% rule and the current best-practice suggestions, their resulting linkage quality will be tested against all the best estimates from the four models (see Section 4.5.3) here.

Table 5.3: Means, cross-validation-based coefficients of variation (CV), standard errors (SE) and RMSEs for the point estimates (MPR) for each model and dataset.

<b>Simple linear model</b>				
Data	Mean MPR	$CV_{LM}$	$SE_{LM}$	$RMSE_{LM}$
Telephone CD 0% errors	0.996	$9.262 \times 10^{-3}$	$1.199 \times 10^{-4}$	0.021
Telephone CD 20% errors	0.971	$7.810 \times 10^{-3}$	$1.288 \times 10^{-4}$	0.020
FEBRL	0.924	$1.318 \times 10^{-2}$	$2.053 \times 10^{-4}$	0.019
Mortality & Hospital	0.930	$1.754 \times 10^{-2}$	$3.688 \times 10^{-4}$	0.025
NC Voter	0.906	$1.199 \times 10^{-2}$	$3.592 \times 10^{-4}$	2120.309
Mortality & Commercial	0.882	$2.461 \times 10^{-2}$	$3.994 \times 10^{-4}$	1920.670
FEBRL WA	0.801	$2.010 \times 10^{-2}$	$2.962 \times 10^{-4}$	1657.558
<b>BIC-selected model</b>				
Data	Mean MPR	$CV_{BIC}$	$SE_{BIC}$	$RMSE_{BIC}$
Telephone CD 0% errors	0.998	$4.379 \times 10^{-3}$	$1.170 \times 10^{-4}$	0.018
Telephone CD 20% errors	0.962	$4.780 \times 10^{-3}$	$1.348 \times 10^{-4}$	0.019
FEBRL	0.962	$6.149 \times 10^{-3}$	$1.670 \times 10^{-4}$	0.010
Mortality & Hospital	0.938	$9.736 \times 10^{-3}$	$3.021 \times 10^{-4}$	0.016
NC Voter	0.924	$5.255 \times 10^{-3}$	$1.754 \times 10^{-4}$	80.312
Mortality & Commercial	0.885	$1.274 \times 10^{-2}$	$2.574 \times 10^{-4}$	98.314
FEBRL WA	0.785	$1.068 \times 10^{-2}$	$3.382 \times 10^{-4}$	11.965
<b>Response surface model</b>				
Data	Mean MPR	$CV_{RSM}$	$SE_{RSM}$	$RMSE_{RSM}$
Telephone CD 0% errors	0.998	$6.022 \times 10^{-3}$	$1.673 \times 10^{-4}$	0.286
Telephone CD 20% errors	0.962	$8.218 \times 10^{-3}$	$1.308 \times 10^{-4}$	0.163
FEBRL	0.957	$8.982 \times 10^{-3}$	$2.022 \times 10^{-4}$	0.213
Mortality & Hospital	0.945	$1.302 \times 10^{-2}$	$2.721 \times 10^{-4}$	0.240
NC Voter	0.926	$8.332 \times 10^{-3}$	$2.286 \times 10^{-4}$	0.380
Mortality & Commercial	0.926	$1.642 \times 10^{-2}$	$2.923 \times 10^{-4}$	0.259
FEBRL WA	0.795	$1.352 \times 10^{-2}$	$2.424 \times 10^{-4}$	0.270
<b>Random Forests</b>				
Data	Mean MPR	$CV_{RF}$	$SE_{RF}$	$RMSE_{RF}$
Telephone CD 0% errors	1.000	$4.379 \times 10^{-3}$	$2.006 \times 10^{-4}$	0.010
Telephone CD 20% errors	0.971	$3.616 \times 10^{-3}$	$1.882 \times 10^{-4}$	0.007
FEBRL	0.963	$8.169 \times 10^{-3}$	$2.248 \times 10^{-4}$	0.011
Mortality & Hospital	0.938	$9.754 \times 10^{-3}$	$2.886 \times 10^{-4}$	0.017
NC Voter	0.906	$5.823 \times 10^{-3}$	$2.129 \times 10^{-4}$	0.039
Mortality & Commercial	0.929	$1.224 \times 10^{-2}$	$3.956 \times 10^{-4}$	0.042
FEBRL WA	0.742	$1.085 \times 10^{-2}$	$3.774 \times 10^{-4}$	0.136

### 5.5.1 Recall and precision

First, recall and precision are evaluated separately before looking at the mean of precision and recall.

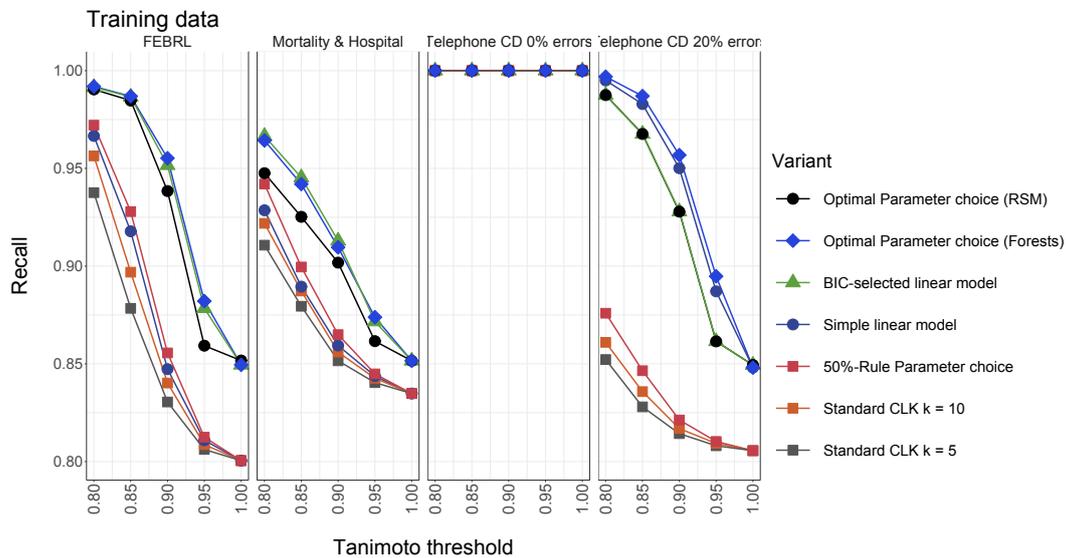


Figure 5.9: Tanimoto threshold, parameter choice-method and resulting recall for the training data.

Recall for all the methods and all the training datasets is shown in Figure 5.9. For all the methods, recall increased with decreasing thresholds. As lower thresholds consider more dissimilar bit vectors as matching pairs, this was expected. Recall was 1 for all the methods in the case of the Telephone CD data without errors. As the models were trained with these datasets, better linkage quality was expected. In terms of recall, the model-based approaches outperformed the best-practice methods considerably.

Figure 5.10 shows similar results for the evaluation data. For all the datasets, model-based parameter choices resulted in a better recall than any of the best-practice parameter suggestions. For the NC Voter dataset, the best-practice suggestions were very close in terms of recall. However, all the model-based methods were on par with the best-practice solutions with the Random Forest-based selection slightly outperforming all other methods. In contrast, for the mortality dataset, the improvement in the recall for the RSM and RF-based models was more pronounced. The RF-based model outperformed every other model-based approach in terms of recall for all the datasets.

Precision for the training data is visualised in Figure 5.11. For all the model-

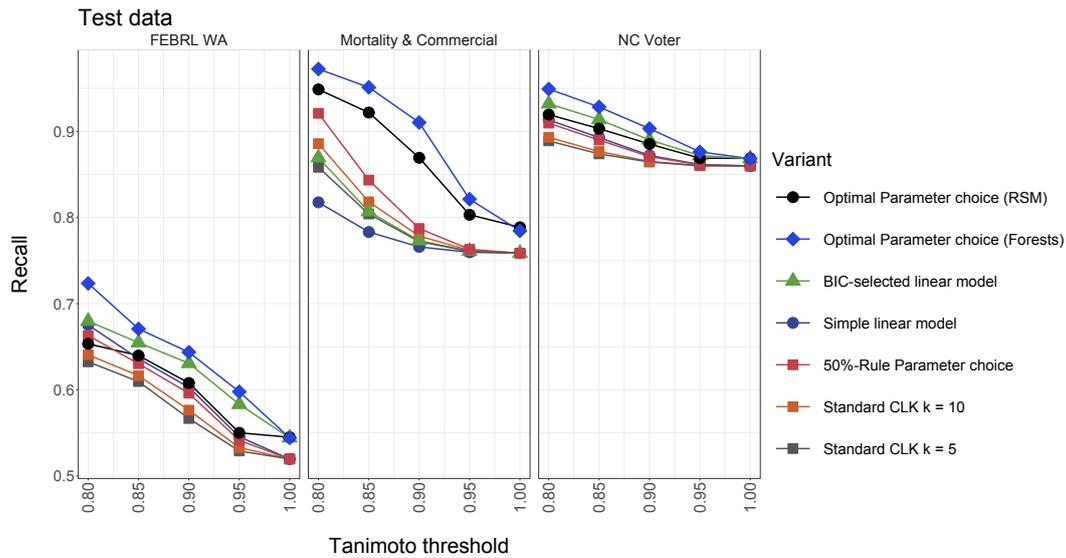


Figure 5.10: Tanimoto threshold, parameter choice method and resulting recall for the test data.

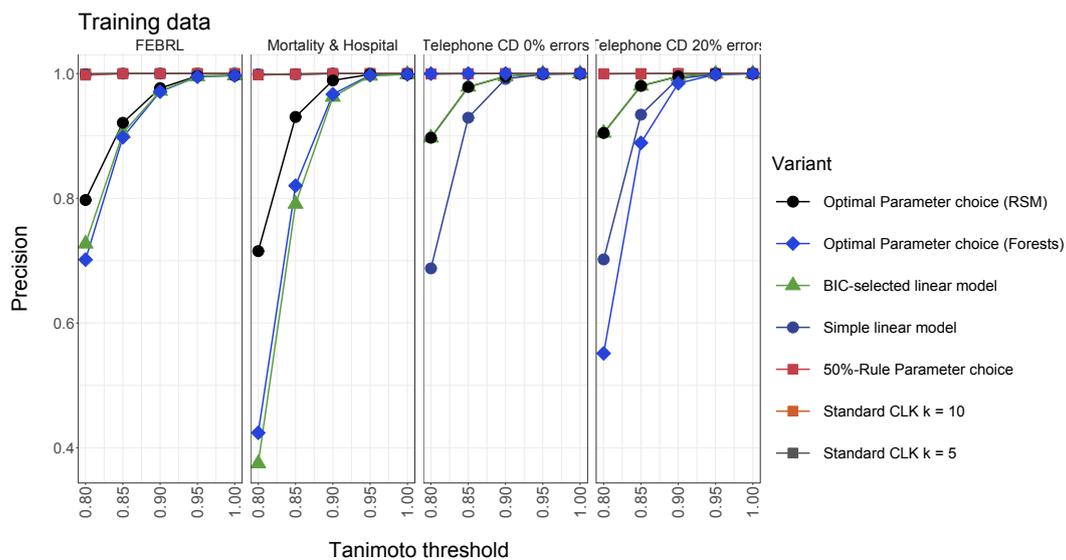


Figure 5.11: Tanimoto threshold, parameter choice method and resulting precision for the training data.

based methods, precision dipped at lower thresholds. The effect of considering differing bit patterns as matches led to more wrong classifications (false positives), which decreases precision. In general, all the methods were very stable in terms of precision. Only the RF- RSM- and BIC-based models showed a drop in precision when lowering the threshold below 0.9, for which it was optimised. At this threshold, all the methods performed comparatively well in terms of

precision.

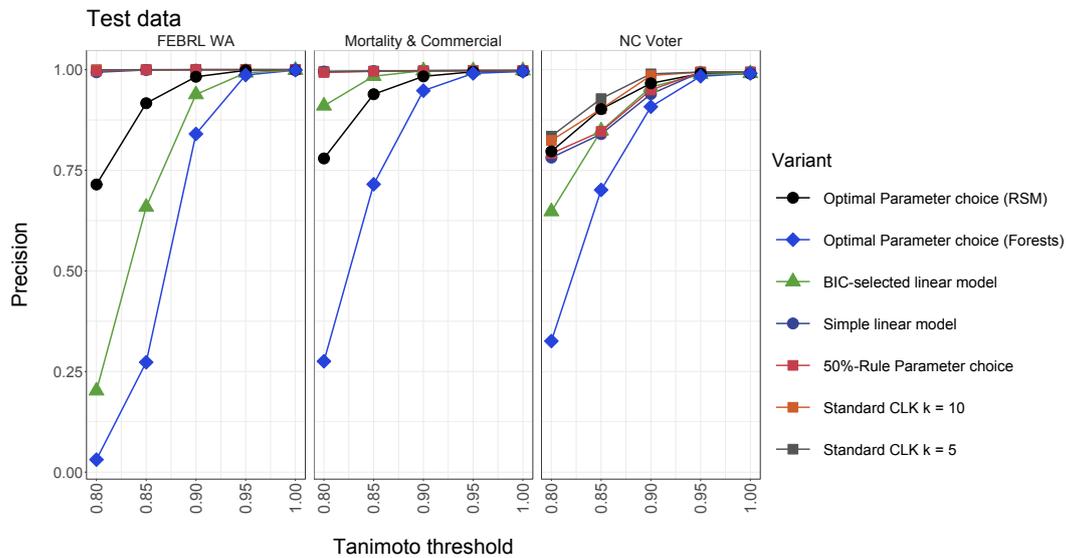


Figure 5.12: Tanimoto threshold, parameter choice method and resulting precision for the test data.

Figure 5.12 shows precision for the evaluation datasets. For the FEBRL WA dataset, Random Forests showed a slightly inferior performance, while the RSM-based approach performed the best overall. However, the best-practice solutions remained very stable irrespective of the threshold. Therefore, the mean of both the measures provides better insight into the model-based performance, particularly as all the models were trained for the mean of precision and recall, and not precision and recall separately.

## 5.5.2 Mean of recall and precision

Figures 5.14 and 5.13 show the mean of precision and recall for the training data and the test data separately. As all the models were trained using the training data, better results were expected.

Note that the models were optimised for a Tanimoto threshold of  $T = 0.9$ . For all the datasets, the mean prec./rec. dropped just below this threshold. As is evident from all the result plots, the approaches proposed here outperformed all the other methods as long as the threshold was exactly at  $T = 0.9$ .

Looking at the results for the test data (Figure 5.14) gives a more nuanced picture. Here, the models are slightly superior to the encryptions using the best-practice recommendations, but not as distinctively so as with the training data.

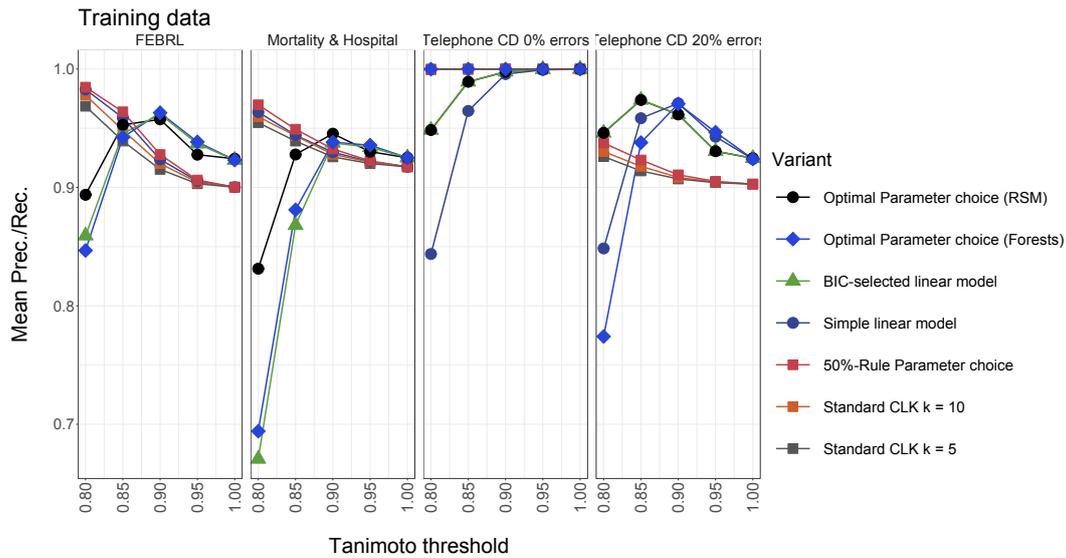


Figure 5.13: Result plot for the optimisation in terms of mean prec./rec. using the training data. As the models were trained with these datasets, better linkage quality was expected.

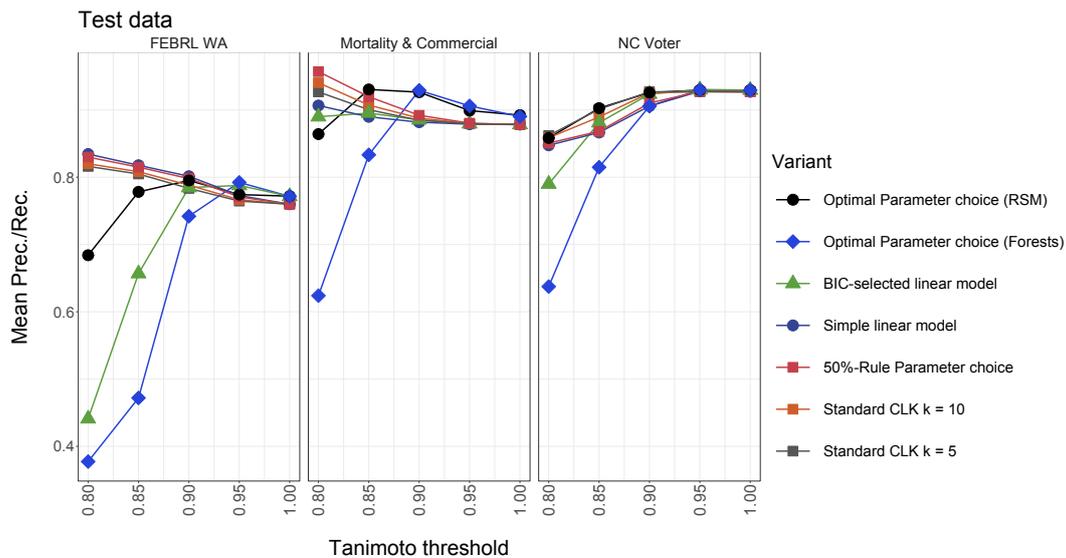


Figure 5.14: Result plot for the optimisation in terms of mean prec./rec. using the test data.

Furthermore, the FEBRL WA dataset shows the Random Forest-based approach as well as the BIC-based linear model estimating sub-optimal parameter choices. This was due to the lower precision, as discussed in the previous section. Here, the response surface method seems to work the best out of all the methods, particularly at the central Tanimoto threshold of  $T = 0.9$ . Without the FEBRL WA data, the Random Forest-based regression shows the best results.

### 5.5.3 Central results

Averaging the mean prec./rec. over all the datasets for each method led to the results presented in Table 5.4, which shows that over all the datasets, the mean mean prec./rec. of the model-based methods outperformed the best-practice parameters. The difference between the standard best-practice setting and the optimal parameter choice (OPC) methods was approximately 1–3%, depending on the model. For obtaining a confidence interval for each mean estimate, bootstrapping (see Section 4.4.2.4) was implemented by re-drawing 1000 samples with replacement.

Table 5.4: Mean resulting mean of precision and recall (MPR) by parameter choice method over all the datasets, as well as bootstrapped confidence intervals for the means.

Method	MPR	CI <sub>low</sub>	CI <sub>high</sub>
<i>Model-based approaches</i>			
Response surface model (RSM)	0.9299	0.9285	0.9314
BIC-selected linear model (BIC)	0.9217	0.9202	0.9233
Random Forest-based model (RF)	0.9212	0.9194	0.9231
Simple linear model (LM)	0.9156	0.9141	0.9170
<i>Best-practice</i>			
50% rule parameter choice	0.9102	0.9089	0.9115
Standard CLK $k = 10$	0.9083	0.9069	0.9096
Standard CLK $k = 5$	0.9062	0.9047	0.9076

In particular, the response surface method and the BIC-selected linear model worked very well. However, there was some variance in the results because of the different linkage quality for each dataset. Without the FEBRL WA dataset, the Random Forest-based approach would yield the best linkage quality.

As all the models were trained for a Tanimoto threshold  $T$  of 0.90, it is sensible to look at the mean of precision and recall for each dataset and method at this threshold. The central objective of this thesis was to improve linkage quality over all the other best-practice solutions for every database possible.

Figure 5.15 presents an overview of the mean prec./rec. for all the datasets at the fixed Tanimoto threshold  $T$  of 0.90. As can be seen, all the model-based approaches usually outperformed both the best-practice suggestions and the 50% rule. Unexpectedly, the difference was considerably large for some datasets (FEBRL and Telephone CD data), while it was very close where the optimal

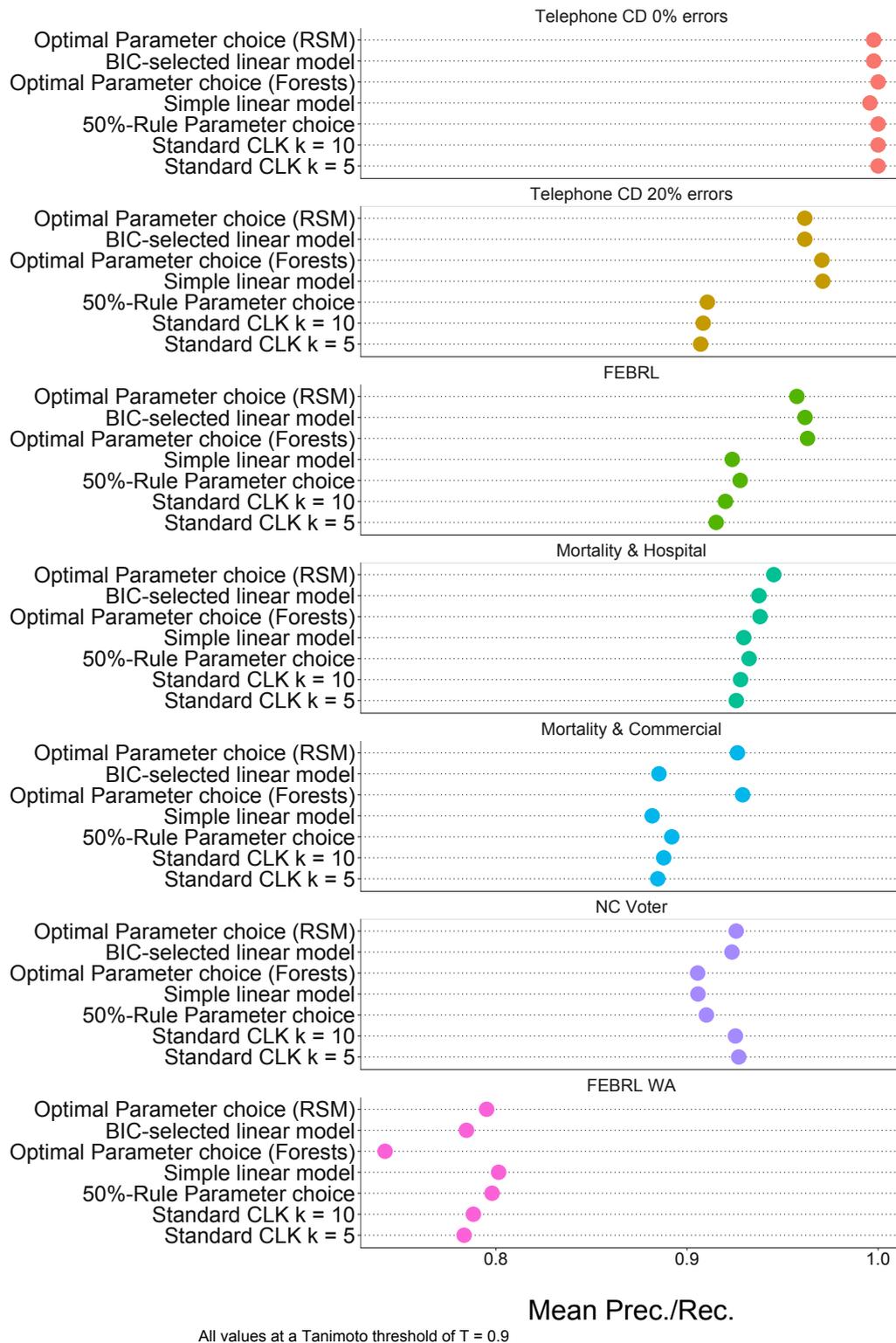


Figure 5.15: Mean of precision and recall for linking for several datasets with encryption parameters based on best-practice suggestions and different methods of choosing optimal parameters. All the results are shown for a Tanimoto threshold of 0.9.

parameter choice was close to the best-practice suggestions: for the Mortality/Hospital data, the linear model and the 50% rule were very close in terms of the choice of  $q$  (sub-string length) and  $k$  (number of hash functions).

One outlier was the performance of the Random Forest-based regression with the FEBRL WA dataset. Here, it performed substantially worse than all the other methods (even though the other methods were only on par with the best-practice solutions). This was probably due to overfitting, as for most of the other applications, it yielded very good results. Without the FEBRL WA data, it would be the best choice for parameter estimation.

Finally, for most of the datasets, the 50% rule worked better than the fixed values found in the literature.

## 5.6 Summary of the results

Next, the results for the comparison of the best-practice parameters and the estimations of the models are summarised.

The use of model-based approaches outperformed the best-practice parameters for nearly all the databases available, sometimes substantially. As the methods were only equal to the best-practice suggestions for one dataset, caution is advised, because the models may overfit the training data. For future use, however, all the seven datasets can be used as training data, which should increase the model performance substantially.

Together with the optimal identifier choice presented in Section 4.2, the automation of identifiers and parameters without further expert knowledge is now possible. Utilising the PPRL package for R (see Section 4.3.2), we can perform the entire PPRL process, including optimal encryption parameter choices, done in R. This will possibly help to further expand the use of Bloom filter-based methods.

## Conclusion and outlook

After the overview of Record Linkage in general and of each of the steps in the Record Linkage process, a summary of the state-of-the-art Privacy-preserving Record Linkage is put forth. Special consideration is given to Bloom filter-based methods, all the published attacks on them and all the published hardening methods to prevent them.

Variable selection, identifier choice and the models used to estimate the optimal encryption parameters are tackled, after which the implementations are discussed. Finally, Chapter 5 presents an overview of the results of simulations, model estimates and the results when using these to automate parameter choice against the current best-practice recommendations.

This chapter will summarise the key findings and the remaining open questions and will close with the updated best-practice recommendations for Bloom filter-based Privacy-preserving Record Linkage.

### 6.1 Key findings

This thesis was concerned with finding optimal identifier and parameter choices. None of the approaches is without drawbacks. Therefore, the key findings are split into insights gained from optimal identifier choice as the first step, followed by the central results and conclusions from the chapter on optimal parameter choice and its results. Finally, limitations and a few words of caution are presented.

### 6.1.1 Identifier choice

A practical method to find optimal parameter sets was proposed. It was based on the agreement weights as computed by the EM algorithm. It resulted in the best-possible mean of precision and recall while not requiring any actual linkage to take place. Interestingly, the standard set of identifiers often used for linkage (first, last and middle names and date of birth) was often optimal.

### 6.1.2 Parameter choice

The models proposed here outperformed the best-practice solutions found in the literature most of the time. Using them slightly improved the linkage quality on average with respect to the mean prec./rec., when compared to the fixed  $k$  solutions. For some applications, the current parameter choices cited in the publications are substantially worse than the model-based solutions. As to the model choice, the use of Random Forests (RF) for parameter selection showed the most promise, while the response surface methods (RSM) performed the best, with the linear model based on model selection through BIC yielding the second-best results. However, there are some drawbacks.

### 6.1.3 Limitations

For all the linear models, most assumptions required for the estimation of the parameters were violated. There were also signs of over-fitted models, as they were prone to outliers. This might make the models very sensitive to changes in the independent variables, leading to (possibly) unreliable best estimates. At best, these would still perform well. At worst, they would be inferior to the current best-practice solutions. For one dataset (FEBRL WA data), such behaviour was observed for the Random Forest regression. While nearly all the theoretical problems of model assumptions could be solved by using response surface model or Random Forests, their estimates might be flawed as well. However, in future applications, the current evaluation datasets could be used as additional training data, which would yield better training data for the models. This would be beneficial to the Random Forest model. The response surface method-based estimates already show consistently good results, but

their linearity assumption may lead to deteriorated performance when using other datasets.

## 6.2 Open research questions

A few follow-up research perspectives present themselves in light of the new findings. They will be discussed briefly.

### *Random Forest model tuning*

Random Forest-based regressions can be tuned in a variety of ways, as they have a number of adjustable parameters (see Section 4.4.2). Furthermore, there are extensions to the algorithm that have not been implemented, such as gradient boosting (Trevor et al. 2009). This may reduce overfitting, making the Random Forest-based approach the best method for optimal parameter choice.

### *Non-western languages*

The results should be replicated with data containing non-western names, to see whether the results can be reproduced with very different training data. As all the data characteristics should be very close, it would be interesting to evaluate the models with languages with longer names (such as Indian last names) or very short names (Chinese names).

### *Optimal parameter choices for hardening methods*

Another direction for further research is optimal parameter selection using hardened Bloom filters: This will probably require modified models or a second layer of models. Some hardening methods, such as rehashing or Markov chain BFs (see Section 3.3) need their own optimal parameter choices. Giving optimal parameter choices for hardening methods was out of the scope of this thesis.

### *Further ideas for modelling optimal choices*

The problem of different dataset characteristics when selecting optimal parameters could be reformulated as a latent-variable structural equation model (sem). The “data characteristics” could be a latent variable with measurable traits. This would give more weight to the characteristics of the data when selecting an optimal parameter set.

A different take using more modern methods can be imagined. The task of choosing optimal values can be re-formulated as a decision-making problem for a convoluted neural network: With a low starting value of  $k$ , the iteratively asked question for the decision-making process would be “will increasing  $k$  provide a better linkage quality?” Given independent input variables, the network could learn decision rules derived from the training data and each of the previous neural network decisions. The implementation of this idea was also out of the scope of this thesis.

Other ideas using machine-learning applications were not explored here, including a non-convoluted artificial neural network where the dataset characteristics are a “hidden layer”, and the parameter choices are the input for the model. Deep learning in the form of convoluted deep neural networks with an output layer that gives parameters as its labelled output could be used as well (see Section 1.5.6.4). All of this is yet to be tested.

## 6.3 Outlook and conclusion

The first step to overcome one of the practical obstacles of implementing Privacy-preserving Record Linkage has been presented here. A working identifier choice system has been implemented. Selecting the right parameters for Bloom filter-based encryptions can be done using the models provided. As these models will likely outperform the current best-practice solutions, their use as a guideline for optimal parameter selection will be beneficial to the linkage quality. Of course, evaluating and testing the recommended parameters is still advisable.

### *Updated best-practice recommendations for Bloom filter-based PPRL*

Given the results of this thesis, the best-practice recommendations need an update. Here is the new proposition for the list of recommendations for conducting Bloom filter-based PPRL as efficiently as possible:

- For linkage in general, use as many stable identifiers as possible.
- Find the optimal identifier choices by using the approach outlined in this thesis.

- With the optimal identifiers chosen, run the model-based approaches to find the optimal parameter choices. Use all the seven datasets for training the models.
- When in doubt, use the Random Forest-based approach for optimal parameter estimation. Tuning the Random Forest parameters is optional but may be very beneficial.
- Use random hashing instead of double hashing during encryption.
- For increased security, use CLKs instead of Bloom filters.
- Use a different password for each identifier.
- Using the R-package PPRL makes random hashing easier.
- The fastest linkage method for BFs thus far is the use of Multibit trees with Symdex preprocessing, using a leaf limit of three, and as many threads (or cores) as possible.
- For large databases, use additional external blocking variables, for example, encrypted year of birth.
- Use a stable external blocking variable as a salt for encryptions if high security is necessary.

### Outlook

The use of these recommendations will enable researchers to provide some level of privacy as demanded by many legal frameworks while optimising linkage quality. Of course, the security of Bloom filter-based methods will be an ongoing research field. Further improvements in the linking and encryption speed will improve the practicability of the approach. Moreover, finding optimal parameters is not *finally* solved with the ideas presented here.

Further research into other methods is needed. One promising approach is the use of convoluted neural networks for iterative decision making on the optimal parameters. Some other untested methods have been proposed as well.

### Conclusion

This thesis was concerned with finding optimal identifier and parameter choices for Bloom filter-based PPRL applications. Some viable ideas were presented and successfully tested. Despite the positive results, further research using a broader selection of data is warranted. Of course, the security, scalability and improvement of model-based decision making remain viable research topics to be explored in the future.

# References

- Abril, D./G. Navarro-Arribas/V. Torra (2012): Improving Record Linkage with Supervised Learning for Disclosure Risk Assessment. In: *Information Fusion* 13 (4): 274–284.
- Abualsaud, K./H. Mohamad Tahir/A. A. El-Zoghabi/M. Saleh (2008): Performance Evaluation of Secured versus Non-Secured EIGRP Routing Protocol. 292–297 in: *Proceedings of the 2008 International Conference on Security & Management, SAM 2008, July 14-17, 2008*. Las Vegas, Nevada, USA.
- Akaike, H. (1974): A new look at the statistical model identification. In: *IEEE Transactions on Automatic Control* 19 (6): 716–723.
- Alaggan, M./S. Gambs/A.-M. Kermarrec (2012): BLIP: Non-interactive Differentially-Private Similarity Computation on Bloom Filters. 202–216 in: Richa, A. W./C. Scheideler (Eds.): *Stabilization, Safety, and Security of Distributed Systems: 14th International Symposium, SSS 2012, Toronto, Canada, October 1–4, 2012*. Berlin: Springer.
- Alfons, A. (2012): *cvTools: Cross-validation tools for regression models*. R package version 0.3.2. URL: <https://CRAN.R-project.org/package=cvTools>.
- Anindya, I. C./M. Kantarcioglu/B. Malin (2019): Determining the Impact of Missing Values on Blocking in Record Linkage. 262–274 in: Yang, Q./Z.-H. Zhou/Z. Gong/M.-L. Zhang/S.-J. Huang (Eds.): *Advances in Knowledge Discovery and Data Mining*. Cham: Springer International Publishing.

- 
- Antoni, M./R. Schnell (2017): The Past, Present and Future of the German Record Linkage Center (GRLC). In: *Journal of Economics and Statistics* (ahead of print, published online: 2017-08-05).
- Baldi, P./D. S. Hirschberg/R. Nasr (2008): Speeding Up Chemical Database Searches Using a Proximity Filter Based on the Logical Exclusive OR. In: *Journal of Chemical Information and Modeling* 48 (7): 1367–1378.
- Becker, G./B. Jenkins (2015): *fastdigest: Fast, Low Memory-Footprint Digests of R Objects*. R package version 0.6-3. URL: <https://CRAN.R-project.org/package=fastdigest>.
- Bellahsene, Z./A. Bonifati/E. Rahm, eds. (2011): *Schema Matching and Mapping*. Berlin: Springer.
- Bender, S./J. Heining/T. Hethey/P. Scioch (2010): Combining Administrative and Survey Data in a German Research Data Centre: Linkage, Quality and Future Developments. 324 in: *Proceedings of Statistics Canada Symposium 2010: Social Statistics: The Interplay among Censuses, Surveys and Administrative Data*. Ottawa: Statistics Canada.
- Bernstein, D. J. (2008): The Salsa20 Family of Stream Ciphers. 84–97 in: Robshaw, M./O. Billet (Eds.): *New Stream Cipher Designs*. Berlin: Springer.
- Bloom, B. H. (1970): Space/Time Trade-offs in Hash Coding with Allowable Errors. In: *Communications of the ACM* 13 (7): 422–426.
- Bohensky, M. (2016): Bias in data linkage studies. 63–82 in: Harron, K./H. Goldstein/C. Dibben (Eds.): *Methodological Developments in Data Linkage*. Wiley.
- Boratto, M./P. Alonso/C. Pinto/P. Melo/M. Barreto/S. Denaxas (2019): Exploring hybrid parallel systems for probabilistic record linkage. In: *The Journal of Supercomputing* 75 (3): 1137–1149.
- Borgs, C./R. Schnell (2017): *Estimating Optimal Parameter and Identifier Choices for Bloom Filter-based PPRL using Model Estimates*. Data Linkage

- 
- Technical Forum: 04.10.2017; Curtin University, Perth, Western Australia.
- Borst, F./F.-A. Allaert/C. Quantin (2001): The Swiss Solution for Anonymous Chaining Patient Files. 1239–1241 in: Patel, V./R. Rogers/R. Haux (Eds.): *Proceedings of the 10<sup>th</sup> World Congress on Medical Informatics: 2–5 September 2001; London*. Amsterdam: IOS Press.
- Box, G. E. P./K. B. Wilson (1951): On the Experimental Attainment of Optimum Conditions. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 13 (1): 1–45.
- Broder, A./M. Mitzenmacher (2003): Network Applications of Bloom Filters: A Survey. In: *Internet Mathematics* 1 (4): 485–509.
- Brown, A. P./C. Borgs/S. M. Randall/R. Schnell (2017a): Evaluating Privacy-preserving Record Linkage Using Cryptographic Long-term Keys and Multibit Trees on Large Medical Datasets. In: *BMC Medical Informatics and Decision Making* 17 (83): 1–7.
- Brown, A./C. Borgs/S. Randall/R. Schnell (2017b): High quality linkage using Multibit Trees for privacy-preserving blocking. In: *International Journal for Population Data Science* 1 (1).
- Bundesverfassungsgericht (1983): *Urteil vom 15. Dezember 1983, Az. 1 BvR 209, 269, 362, 420, 440, 484/83*.
- Campbell, C./Y. Ying (2011): *Learning with Support Vector Machines*. London: Morgan & Claypool Publishers.
- Carter, J./M. N. Wegman (1979): Universal classes of hash functions. In: *Journal of Computer and System Sciences* 18 (2): 143–154.
- Chang, R. M./R. J. Kauffman/Y. Kwon (2014): Understanding the paradigm shift to computational social science in the presence of big data. In: *Decision Support Systems* 63: 67–80.

- 
- Christen, P. (2008): Febrl: A Freely Available Record Linkage System with a Graphical User Interface. 17–25 in: *HDKM '08 Proceedings of the second Australasian workshop on Health data and knowledge management*. Las Vegas, Nevada, USA: ACM.
- Christen, P. (2012): *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Berlin: Springer.
- Christen, P. (2014): *Preparation of a real temporal voter data set for record linkage and duplicate detection research*. Research School of Computer Science, Australian National University.
- Christen, P./T. Randaduge/A. Vidanage/R. Schnell (2018): Pattern-Mining based Cryptanalysis of Bloom Filters for Privacy-Preserving Record Linkage in: *The 22nd Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), June 3rd – 6th, 2018, Melbourne, Australia*.
- Christen, P./R. Schnell/D. Vatsalan/T. Ranbaduge (2017): Efficient Cryptanalysis of Bloom Filters for Privacy-Preserving Record Linkage. 628–640 in: Kim, J./K. Shim/L. Cao/J.-G. Lee/X. Lin/Y.-S. Moon (Eds.): *Advances in Knowledge Discovery and Data Mining: 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea, May 23-26, 2017*. Springer.
- Comber, S./D. Arribas-Bel (2019): Machine learning innovations in address matching: A practical comparison of word2vec and CRFs. In: *Transactions in GIS* 23 (2): 334–348.
- Council of the European Union (2015): *Proposal for a Regulation of the European Parliament and of the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation)*. Document 9565/15. URL: <http://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf>.
- Council of the European Union (2016): *Council Regulation (EU) No 679/2016*.

- 
- Cowell, F. (2000): Measurement of inequality. 87–166 in: Atkinson, A./F. Bourguignon (Eds.): *Handbook of Income Distribution*. Vol. 1. Elsevier.
- Damerau, F. J. (1964): A technique for computer detection and correction of spelling errors. In: *Communications of the ACM* 7 (3): 171–176.
- Dantas Pita, R./C. Pinto/S. Sena/R. Fiaccone/L. Amorim/S. Reis/M. Barreto/S. Denaxas/M. Barreto (2018): On the Accuracy and Scalability of Probabilistic Data Linkage over the Brazilian 114 Million Cohort. In: *IEEE Journal of Biomedical and Health Informatics* 22 (2): 346–353.
- Dempster, A. P./N. M. Laird/D. B. Rubin (1977): Maximum likelihood from incomplete data via the EM algorithm. In: *Journal of the Royal Statistical Society. Series B (Methodological)*: 1–38.
- Denning, D. (1982): *Cryptography and Data Security*. Boston: Addison-Wesley.
- Dice, L. R. (1945): Measures of the Amount of Ecologic Association between Species. In: *Ecology* 26 (3): 297–302.
- Doiron, D./P. Raina/I. Fortier (2013): Linking Canadian population health data: maximizing the potential of cohort and administrative data. In: *Canadian journal of public health* 104 (3): e258–e261.
- Durham, E. A. (2012): A Framework for Accurate, Efficient Private Record Linkage. PhD thesis. Nashville: Vanderbilt University.
- Elfeky, M. G./V. S. Verykios/A. K. Elmagarmid (2002): TAILOR: a record linkage toolbox. 17–28 in: *Proceedings 18th International Conference on Data Engineering*. San Jose, CA, USA: IEEE.
- Enamorado, T./B. Fifield/K. I. [aut] (2018a): *fastLink: Fast Probabilistic Record Linkage with Missing Data*. R package version 0.5.0. URL: <https://CRAN.R-project.org/package=fastlink>.
- Enamorado, T./B. Fifield/K. Imai (2018b): Using a probabilistic model to assist merging of large-scale administrative records. In: *American Political Science Review*: 1–19.

- 
- Erlingsson, Ú./V. Pihur/A. Korolova (2014): RAPPOR: Randomized Aggregatable Privacy-preserving Ordinal Response. 1054–1067 in: Ahn, G.-J. (Eds.): *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. New York: ACM.
- Everitt, B. S./T. Hothorn (2010): *A Handbook of Statistical Analyses Using R*. 2nd ed. Boca Raton: Chapman & Hall.
- Eycken, E. van/K. Haustermans/F. Buntinx/A. Ceuppens/J. Weyler/E. Wauters/H. V. Oyen/M. D. Schaefer/D. V. den Berge/M. Haelterman (2000): Evaluation of the Encryption Procedure and Record Linkage in the Belgian National Cancer Registry. In: *Archives of Public Health* 58 (6): 281–294.
- Farrow, J./R. Schnell (2015): *Locational Privacy Preserving Distance Computations with Intersecting Sets of Randomly Labelled Grid Points*. Discussion paper. Research Methodology Group, University of Duisburg-Essen.
- Fellegi, I. P./A. B. Sunter (1969): A Theory for Record Linkage. In: *Journal of the American Statistical Association* 64 (328): 1183–1210.
- Fienberg, S. E. (2013): *Record Linkage as a Statistical Procedure: Some History, Formal Frameworks, Applications, and Challenges*. Presentation: Department of Statistics, Machine Learning Department, Living Analytics Research Centre Carnegie Mellon University SAMSI Program on Statistical and Computational Methods in the Social Sciences. 18. August 2013.
- Foster, I./R. Ghani/R. S. Jarmin/F. Kreuter/J. L. (Hrsg.) (2017): *Big Data and Social Science: A Practical Guide to Methods and Tools*. Boca Raton: CRC Press.
- Fox, J./S. Weisberg (2011): *An R Companion to Applied Regression*. 2nd ed. Thousand Oaks: Sage.
- Giles, J. (2012): Computational social science: Making the links. In: *Nature* 488 (7412): 448–450.

- 
- Gini, C. (1912): Variabilità e mutabilità. In: *Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi.*
- Gini, C. (1921): Measurement of Inequality of Incomes. In: *The Economic Journal* 31 (121): 124–126.
- Groves, R. M. (2011): Three Eras of Survey Research. In: *Public Opinion Quarterly* 75 (5): 861–871.
- Grundy, E./C. Tomassini (2005): Fertility History and Health in Later Life: A Record Linkage Study in England. In: *Social Science & Medicine* 61 (1): 217–228.
- Habermann, H./C. Kennedy/P. Lahiri (2017): A Conversation with Robert Groves. In: *Statistical Science* 32 (1): 128–137.
- Häder, S. (2015): *Stichproben in der Praxis*. Mannheim: GESIS.
- Hall, D. L./J. Llinas (1997): An introduction to multisensor data fusion. In: *Proceedings of the IEEE* 85 (1): 6–23.
- Hamilton, L. C./M. D. Stampone (2013): Blowin’ in the Wind: Short-Term Weather and Belief in Anthropogenic Climate Change. In: *Climate Change. Weather, Climate, and Society* 5 (2): 112–119.
- Hand, D./P. Christen (2018): A note on using the F-measure for evaluating record linkage algorithms. In: *Statistics and Computing* 28 (3): 539–547.
- Harron, K./A. Wade/R. Gilbert/B. Muller-Pebody/H. Goldstein (2014): Evaluating bias due to data linkage error in electronic healthcare records. In: *BMC Medical Research Methodology* 14 (1): 36–45.
- Hernandez, M. A./S. J. Stolfo (1995): The merge/purge problem for large databases. 127–138 in: *SIGMOD ’95 Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. San Jose: ACM.

- 
- Hernandez, M. A./S. J. Stolfo (1998): Real-world Data Is Dirty: Data Cleansing and the Merge/purge Problem. In: *Data Mining and Knowledge Discovery* 2 (1): 9–37.
- Herzog, T. N./F. J. Scheuren/W. E. Winkler (2007): *Data Quality and Record Linkage Techniques*. New York: Springer.
- Ho, T. K. (1995): Random decision forests. 278–282 in: *Proceedings of 3rd international conference on document analysis and recognition*. IEEE.
- Izakian, H. (2018): Privacy preserving record linkage meets record linkage using unencrypted data. In: *International Journal of Population Data Science* 3 (4): 61.
- Jaccard, P. (1901): Étude comparative de la distribution florale dans une portion des Alpes et des Jura. In: *Bull Soc Vaudoise Sci Nat* 37: 547–579.
- Jakobsen, T. (1995): A fast method for the cryptanalysis of substitution ciphers. In: *Cryptologia* 19 (3): 265–274.
- James, G./D. Witten/T. Hastie/R. Tibshirani (2013): *An Introduction to Statistical Learning: with Applications in R*. New York: Springer.
- Jaro, M. A. (1989): Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. In: *Journal of the American Statistical Association* 84 (406): 414–420.
- Karmel, R. (2005): *Data Linkage Protocols Using a Statistical Linkage Key*. Canberra: AIHW.
- Kassambara, A. (2018): *Machine Learning Essentials: Practical Guide in R*. London: STHDA.
- Kimura, S./T. Sato/S. Ikeda/M. Noda/T. Nakayama (2010): Development of a Database of Health Insurance Claims: Standardization of Disease Classifications and Anonymous Record Linkage. In: *Journal of Epidemiology* 20 (5): 413–419.

- 
- Kirsch, A./M. Mitzenmacher (2006): Less Hashing Same Performance: Building a Better Bloom Filter. 456–467 in: Azar, Y./T. Erlebach (Eds.): *Algorithms-ESA 2006. Proceedings of the 14<sup>th</sup> Annual European Symposium: 11-13 September 2006; Zürich, Switzerland*. Berlin: Springer.
- Klima, V. (2005): Finding MD5 Collisions – a Toy For a Notebook. In: *IACR Cryptology ePrint Archive 2005*: 75–81.
- Klingwort, J./B. Buelens/R. Schnell (2018): *Capture-recapture Techniques for Transport Survey Estimate Adjustment Using Road Sensor Data*. BigSurv 2018, Barcelona, 25.–27. October 2018.
- Klingwort, J./N.-M. Lenz/C. Borgs (2015): *Studienprotokoll zu Versionstests, Leaf Limit-Tests und Geschwindigkeitszuwachs durch Parallelisierung mit dem MultibitTree-Paket*. Unpublished Research Methodology Group working paper, University of Duisburg-Essen.
- Knott, G. D. (1975): Hashing functions. In: *The Computer Journal* 18 (3): 265–278.
- Knuth, D. E. (1977): *The Art of Computer Programming Volume 3: Sorting and Searching*. Reading: Addison-Wesley.
- Kolb, L./A. Thor/E. Rahm (2012): Load Balancing for MapReduce-based Entity Resolution. 618–629 in: *Proceedings 28<sup>th</sup> International Conference on Data Engineering (ICDE)*.
- Kooli, N./R. Allesiardo/E. Pigneul (2018): Deep Learning Based Approach for Entity Resolution in Databases. 3–12 in: Nguyen, N. T./D. H. Hoang/T.-P. Hong/H. Pham/B. Trawiński (Eds.): *Intelligent Information and Database Systems. Asian Conference on Intelligent Information and Database Systems (ACIIDS 2018)*. Cham: Springer.
- Krawczyk, H./M. Bellare/R. Canetti (1997): *HMAC: Keyed-hashing for Message Authentication*. Internet RFC 2104.

- 
- Kristensen, T. G./J. Nielsen/C. N. S. Pedersen (2010): A Tree-based Method for the Rapid Screening of Chemical Fingerprints. In: *Algorithms for Molecular Biology* 5 (1): 9–20.
- Kroll, M./S. Steinmetzer (2015): Who Is 1011011111...1110110010? Automated Cryptanalysis of Bloom Filter Encryptions of Databases with Several Personal Identifiers. 341–356 in: Fred, A./H. Gamboa/D. Elias (Eds.): *Biomedical Engineering Systems and Technologies (BIOSTEC), 8<sup>th</sup> International Joint Conference, Lisbon, Portugal, January 12–15*. Cham: Springer.
- Kuzu, M./M. Kantarcioglu/E. A. Durham/C. Toth/B. Malin (2013): A Practical Approach to Achieve Private Medical Record Linkage in Light of Public Resources. In: *Journal of the American Medical Informatics Association* 20 (2): 285–292.
- Kuzu, M./M. Kantarcioglu/E. Durham/B. Malin (2011): A Constraint Satisfaction Cryptanalysis of Bloom Filters in Private Record Linkage. 226–245 in: Fischer-Hübner, S./N. Hopper (Eds.): *The 11<sup>th</sup> Privacy Enhancing Technologies Symposium: 27–29 July 2011; Waterloo, Canada*. Berlin: Springer.
- Lasker, G. W. (1985): *Surnames and Genetic Structure*. Cambridge Studies in Biological and Evolutionary Anthropology. Cambridge: Cambridge University Press.
- Lazrig, I./T. C. Ong/I. Ray/I. Ray/X. Jiang/J. Vaidya (2018): Privacy Preserving Probabilistic Record Linkage Without Trusted Third Party. 1–10 in: *2018 16th Annual Conference on Privacy, Security and Trust (PST)*.
- Lenth, R. V. (2009): Response-Surface Methods in R, Using rsm. In: *Journal of Statistical Software* 32 (7): 1–17.
- Levenshtein, V. I. (1966): Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady* ( 8): 707–710.
- Liaw, A./M. Wiener (2002): Classification and regression by randomForest. In: *R News* 2 (3): 18–22.

- 
- Ludvigsson, J. F./P. Otterblad-Olausson/B. U. Pettersson/A. Ekbom (2009): The Swedish Personal Identity Number: Possibilities and Pitfalls in Health-care and Medical Research. In: *European Journal of Epidemiology* 24 (11): 659–667.
- Maier, B./K. Wagner/S. Behrens/L. Bruch/R. Busse/D. Schmidt/H. Schühlen/R. Thieme/H. Theres (2015): Deterministic Record Linkage with Indirect Identifiers: Data of the Berlin Myocardial Infarction Registry and the AOK Nordost for Patients with Myocardial Infarction. In: *Gesundheitswesen* 77 (2): e15–e19.
- Marr, B. (2018): *Here's Why Data Is Not The New Oil*. Forbes contribution, <https://www.forbes.com/sites/bernardmarr/2018/03/05/heres-why-data-is-not-the-new-oil> (Last accessed: 13.04.2018).
- Martini, M./D. Wagner/M. Wenzel (2017): *Rechtliche Grenzen einer Personen- bzw. Unternehmenskennziffer in staatlichen Registern*. Deutsches Forschungsinstitut für öffentliche Verwaltung, Universität Speyer.
- Mateos, P. (2014): *Names, ethnicity and Populations*. Berlin: Springer.
- McCallum, A./K. Nigam/L. H. Ungar (2000): Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching. 169–178 in: Ramakrishnan, R./S. Stolfo/R. Bayardo/I. Parsa (Eds.): *Proceedings of the 6<sup>th</sup> ACM SIGKDD International Conference on Knowledge Discovery and Data Mining: 20–23 August 2000; Boston*. New York: ACM.
- Morris, R./K. Thompson (1979): Password security: A case history. In: *Communications of the ACM* 22 (11): 594–597.
- Münnich, R./S. Gabler/M. Ganninger/J. P. Burgard/J.-P. Kolb (2011): Das Stichprobendesign des registergestützten Zensus 2011. In: *Methoden – Daten – Analysen* 5 (1): 37–61.
- Myers, R. H./D. C. Montgomery/C. M. Anderson-Cook (2016): *Response surface methodology: process and product optimization using designed experiments*. 4th ed. Hoboken: Wiley.

- 
- Myers, R. H./D. C. Montgomery/G. G. Vining/T. J. Robinson (2010): *Generalized Linear Models: with Applications in Engineering and the Sciences*. 2nd ed. Hoboken: Wiley.
- National Institute of Standards and Technology (2002): *The Keyed-Hash Message Authentication Code (HMAC)*. FIPS PUB 198.
- National Institute of Standards and Technology (2012): *Secure Hash Standard (SHS)*. FIPS PUB 180-4.
- Newcombe, H. B./J. M. Kennedy/S. J. Axford/A. P. James (1959): Automatic Linkage of Vital Records. In: *Science* 130 (3381): 954–959.
- Niedermeyer, F./S. Steinmetzer/M. Kroll/R. Schnell (2014): Cryptanalysis of Basic Bloom Filters Used for Privacy Preserving Record Linkage. In: *Journal of Privacy and Confidentiality* 6 (2): 59–79.
- Office fédéral de la statistique (1997): *La protection des données dans la statistique médicale*. Neuchatel.
- Påhlman, L./M. Bohe/B. Cedermark/M. Dahlberg/G. Lindmark/R. Sjö Dahl/B. Öjerskog/L. Damber/R. Johansson (2007): The Swedish rectal cancer registry. In: *British Journal of Surgery* 94 (10): 1285–1292.
- Peterson, J. L. (1986): A note on undetected typing errors. In: *Communications of the ACM* 29 (7): 633–637.
- Philips, L. (1990): Hanging on the Metaphone. In: *Computer Language Magazine* 7 (12): 39–44.
- Pollock, J. J./A. Zamora (1984): Automatic spelling correction in scientific and scholarly text. In: *Communications of the ACM* 27 (4): 358–368.
- Postel, H. J. (1969): Die Kölner Phonetik – ein Verfahren zur Identifizierung von Personennamen auf der Grundlage der Gestaltanalyse. In: *IBM-Nachrichten* 19: 925–931.

- 
- Quantin, C./C. Binquet/K. Bourquard/R. Pattisina/B. Gouyon-Cornet/C. Ferdynus/J. B. Gouyon/A. François-André (2004): Which are the best identifiers for record linkage? In: *Medical informatics and the Internet in medicine* 29 (3-4): 221–227.
- R Core Team (2019): *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: <https://www.R-project.org/>.
- Ranbaduge, T./A. Vidanage/S. Vaiwsri/R. Schnell/P. Christen (2018): Evaluating hardening techniques against cryptanalysis attacks on Bloom filter encodings for record linkage. In: *International Journal of Population Data Science* 3 (4).
- Randall, S. M./A. M. Ferrante/J. H. Boyd/J. K. Bauer/J. B. Semmens (2014): Privacy-preserving Record Linkage on Large Real World Datasets. In: *Journal of Biomedical Informatics* 50: 205–212.
- Randall, S./A. P. Brown/A. M. Ferrante/J. H. Boyd (2019): Privacy preserving linkage using multiple dynamic match keys. In: *International Journal of Population Data Science* 4 (1): 15–26.
- Randall, S./A. Ferrante/J. Boyd/A. Brown/J. Semmens (2016): Limited Privacy Protection and Poor Sensitivity: Is It Time to Move on from the Statistical Linkage Key-581? In: *Health Information Management Journal* 45 (2): 71–79.
- Rässler, S. (2002): *Statistical Matching: A Frequentist Theory, Practical Applications, and Alternative Bayesian Approaches*. New York: Springer.
- Reisch, M. (2012): *Record Linkage im Zensus 2011*. Statistik-Tage Bamberg 2012.
- Rivest, R. L./A. Shamir/L. Adleman (1978): A method for obtaining digital signatures and public-key cryptosystems. In: *Communications of the ACM* 21 (2): 120–126.

- 
- Robertson, A. M./P. Willett (1998): Applications of n-grams in textual information systems. In: *Journal of Documentation* 54 (1): 48–67.
- Rocha, M. C. N./R. B. de Lima/A. Stevens/M. M. U. Gutierrez/L. P. Garcia (2015): Óbitos registrados com causa básica hanseníase no Brasil: uso do relacionamento de bases de dados para melhoria da informação (Deaths with leprosy as the underlying cause recorded in Brazil: use of data base linkage to enhance information). In: *Ciência & Saúde Coletiva* 20 (4): 1017–1026.
- Rosenbaum, P. R./D. B. Rubin (1985): Constructing a control group using multivariate matched sampling methods that incorporate the propensity score. In: *The American Statistician* 39 (1): 33–38.
- Russel, R. C. (1918): *Patent US1261167: Index. Filed February 4th, 1918.*
- Scannapieco, M./I. Figotin/E. Bertino/A. K. Elmagarmid (2007): Privacy Preserving Schema and Data Matching. 653–664 in: *Proceedings ACM SIGMOD International Conference on Management of Data.*
- Schmidt, O. C./A. G. Klawitter (1927): *Patent US1637965 : Index. Filed August 2nd, 1927.* <https://patents.google.com/patent/US1637965A/en> (Last accessed: 31.03.2019).
- Schnell, R. (2018): ‘Big Data’ aus wissenschaftssoziologischer Sicht: Warum es kaum sozialwissenschaftliche Studien ohne Befragungen gibt. 101–125 in: Arranz-Becker, O./D. Lois/D. Baron (Eds.): *Erklärende Soziologie und soziale Praxis.* Cham: Springer.
- Schnell, R. (2012): *Recent Developments in Privacy Preserving Record Linkage.* MEA Workshop ‘Linking Survey and Survey Data’: 19.11.-20.11.2012; Berlin, 19.11.2012.
- Schnell, R. (2013): Privacy-preserving Record Linkage and Privacy-preserving Blocking for Large Files with Cryptographic Keys Using Multibit Trees. 187–194 in: *Proceedings Joint Statistical Meetings, American Statistical Association.*

- 
- Schnell, R. (2014): An efficient Privacy-Preserving Record Linkage Technique for Administrative Data and Censuses. In: *Journal of the International Association for Official Statistics* 30 (3): 263–270.
- Schnell, R. (2015): Combining Surveys with Non-questionnaire Data: Overview and Introduction. 269–272 in: Engel, U./B. Jann/P. Lynn/A. Scherpenzeel/P. Sturgis (Eds.): *Improving Survey Methods: Lessons from Recent Research*. New York: Routledge, Taylor & Francis Group.
- Schnell, R. (2016a): Privacy Preserving Record Linkage. 201–225 in: Harron, K./H. Goldstein/C. Dibben (Eds.): *Methodological Developments in Data Linkage*. Chichester: Wiley.
- Schnell, R. (2016b): Record Linkage. 662–669 in: Wolf, C./D. Joye/T. W. Smith/Y. C. Fu (Eds.): *The Sage Handbook of Survey Methodology*. London: SAGE.
- Schnell, R. (2017a): *Matching-Verfahren*. Invited talk at the Bundesinstitut für Berufsbildung (BIBB); Bonn, 30. November 2017.
- Schnell, R. (2017b): *Recent Developments in Bloom Filter-based Methods for Privacy-preserving Record Linkage*. Curtin Institute for Computation, Curtin University, Perth, 12.9.2017.
- Schnell, R. (2019): *Survey-Interviews: Methoden standardisierter Befragungen*. 2nd ed. Wiesbaden: Springer VS.
- Schnell, R./T. Bachteler/J. Reiher (2005): MTB: ein Record-Linkage-Programm für die empirische Sozialforschung. In: *ZA-Information* 56: 93–103.
- Schnell, R./T. Bachteler/J. Reiher (2009): Privacy-preserving Record Linkage Using Bloom Filters. In: *BMC Medical Informatics and Decision Making* 9 (1): 41–52.
- Schnell, R./T. Bachteler/J. Reiher (2011): *A Novel Error-Tolerant Anonymous Linking Code*. Duisburg: German Record Linkage Center. WP-GRLC-2011-02.

- 
- Schnell, R./C. Borgs (2015): Building a National Perinatal Data Base without the Use of Unique Personal Identifiers. 232–239 in: *IEEE International Conference on Data Mining Workshop, ICDMW 2015, Atlantic City, NJ, USA, November 14-17, 2015*.
- Schnell, R./C. Borgs (2016a): Available Methods for Privacy Preserving Record Linkage on Census Scale Data in: *European Conference on Quality in Official Statistics (Q2016), June 1st*. Madrid.
- Schnell, R./C. Borgs (2016b): Randomized Response and Balanced Bloom Filters for Privacy Preserving Record Linkage. 218–224 in: Domeniconi, C./F. Gullo/F. Bonchi/J. Domingo-Ferrer/R. Baeza-Yates/Z.-H. Zhou/X. Wu (Eds.): *Proceedings of the 16<sup>th</sup> IEEE International Conference on Data Mining Workshops (ICDMW), 12–15 December 2016 Barcelona, Spain*. Barcelona: IEEE Publishing.
- Schnell, R./C. Borgs (2016c): *XOR-Folding for Bloom Filter-based Encryptions for Privacy-preserving Record Linkage*. German Record Linkage Center. WP-GRLC-2016-03.
- Schnell, R./C. Borgs (2017): *State of the Art Privacy-Preserving Record Linkage of Large Administrative Data Sets*. New Techniques and Technologies for Statistics (NNTS 2017): 13–17th March 2017; Brussels, 14.03.2017.
- Schnell, R./C. Borgs (2018a): Hardening Encrypted Patient Names Against Cryptographic Attacks Using Cellular Automata. 518–522 in: *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. Singapore: IEEE.
- Schnell, R./C. Borgs (2018b): Implementing Privacy-preserving National Health Registries. In: *Proceedings of Statistics Canada Symposium 2018: Combine to Conquer: Innovations in the Use of Multiple Sources of Data*. Ottawa, Canada.
- Schnell, R./C. Borgs (2018c): Protecting Record Linkage Identifiers Using a Language Model for Patient Names. In: *Studies in Health Technology and Informatics* 253: 91–95.

- 
- Schnell, R./P. B. Hill/E. Esser (2013): *Methoden der empirischen Sozialforschung*. 10th ed. München: Oldenbourg.
- Schnell, R./A. Richter/C. Borgs (2014): Performance of Different Methods for Privacy Preserving Record Linkage with Large Scale Medical Data Sets in: *2014 International Health Data Linkage Conference*. Vancouver.
- Schnell, R./A. Richter/C. Borgs (2017): A Comparison of Statistical Linkage Keys with Bloom Filter-based Encryptions for Privacy-preserving Record Linkage Using Real-world Mammography Data. 276–283 in: *Proceedings of the 10<sup>th</sup> International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2017)*.
- Schnell, R./D. Rukasz (2019): *PPRL: Privacy Preserving Record Linkage*. R package version 0.3.5.2. URL: <https://CRAN.R-project.org/package=PPRL>.
- Schroeder, A. (2012): Pad and Chaff: Secure Approximate String Matching in Private Record Linkage. 121–125 in: *IIWAS 12 Proceedings of the 14<sup>th</sup> International Conference on Information Integration and Web-based Applications & Services*. Bali.
- Schürle, J. (2005): A method for consideration of conditional dependencies in the Fellegi and Sunter model of record linkage. In: *Statistical Papers* 46 (3): 433–449.
- Schwarz, G. (1978): Estimating the Dimension of a Model. In: *Ann. Statist.* 6 (2): 461–464.
- Sehili, Z./L. Kolb/C. Borgs/R. Schnell/E. Rahm (2015): Privacy Preserving Record Linkage with PPJoin. 85–104 in: *Proceedings 16. GI-Konferenz für Datenbanksysteme in Business, Technologie und Web (BTW)*.
- Sen, A. (1997): *On Economic Inequality*. Oxford: Clarendon Press.
- Silver, D./T. Hubert/J. Schrittwieser/I. Antonoglou/M. Lai/A. Guez/M. Lantot/L. Sifre/D. Kumaran/T. Graepel/T. Lillicrap/K. Simonyan/D. Hassabis

- (2018): A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. In: *Science* 362 (6419): 1140–1144.
- Smith, D. (2017): Secure Pseudonymisation for Privacy-preserving Probabilistic Record Linkage. In: *Journal of Information Security and Applications* 34: 271–279.
- Smith, D./N. Shlomo (2014): *Privacy Preserving Record Linkage*. Data without Boundaries, Deliverable D11.2: Record Linkage Approaches for Dynamic Database Integration. Data without Boundaries (DwB).
- Soloff, C./A. Sanson/M. Wake/L. Harrison (2007): Enhancing Longitudinal Studies by Linkage to National Databases: Growing up in Australia, the Longitudinal Study of Australian Children. In: *International Journal of Social Research Methodology* 10 (5): 349–363.
- Spiess, A.-N./N. Neumeyer (2010): An evaluation of  $R^2$  as an inadequate measure for nonlinear models in pharmacological and biochemical research: a Monte Carlo approach. In: *BMC pharmacology* 10 (1): 6.
- Stallings, W. (2014): *Cryptography and Network Security: Principles and Practice*. 6th ed. New Jersey: Pearson.
- Statistisches Bundesamt (2015): *Zensus 2011: Methoden und Verfahren*. Wiesbaden: Statistische Ämter des Bundes und der Länder.
- Statistisches Bundesamt (2017): *Ein Blick in die Registerlandschaft in Deutschland*. Beistellung zum Gutachten “Mehr Leistung für Bürger und Unternehmen: Verwaltung digitalisieren. Register modernisieren.” im Auftrag des Nationalen Normenkontrollrates. Wiesbaden, Statistisches Bundesamt.
- Stausberg, J./A. Waldenburger/C. Borgs/R. Schnell (2017): Combining Different Privacy-preserving Record Linkage Methods for Hospital Admission Data. In: *Studies in Health Technology and Informatics* 235: 161–165.

- Stevens, M./E. Bursztein/P. Karpman/A. Albertini/Y. Markov (2017): The First Collision for Full SHA-1. 570–596 in: Katz, J./H. Shacham (Eds.): *Advances in Cryptology – CRYPTO 2017*. Cham: Springer International Publishing.
- Swamidass, S. J./P. Baldi (2007): Bounds and Algorithms for Fast Exact Searches of Chemical Fingerprints in Linear and Sublinear Time. In: *Journal of Chemical Information and Modeling* 47 (2): 302–317.
- Tabei, Y. (2012): Succinct Multibit Tree: Compact Representation of Multibit Trees by Using Succinct Data Structures in Chemical Fingerprint Searches. 201–213 in: Raphael, B./J. Tang (Eds.): *Algorithms in Bioinformatics*. Berlin: Springer.
- Tai, D./J. Fang (2012): SymDex: Increasing the Efficiency of Chemical Fingerprint Similarity Searches for Comparing Large Chemical Libraries by Using Query Set Indexing. In: *Journal of Chemical Information and Modeling* 52 (8): 1926–1935.
- Tokle, J./S. Bender (2017): Record Linkage. 71–92 in: Foster, I./R. Ghani/R. S. Jarmin/F. Kreuter/J. Lane (Eds.): *Big data and social science*. Boca Raton: CRC Press.
- Trevor, H./T. Robert/F. Jerome H. (2009): *The elements of statistical learning: data mining, inference, and prediction*. 2nd ed. New York: Springer.
- Vaiwsri, S./T. Ranbaduge/P. Christen (2019): Reference Values Based Hardening for Bloom Filters Based Privacy-Preserving Record Linkage. 189–202 in: Islam, R./Y. S. Koh/Y. Zhao/G. Warwick/D. Stirling/C.-T. Li/Z. Islam (Eds.): *Data Mining. Australasian Conference on Data Mining (AusDM)*. Vol. 996, Communications in Computer and Information Science. Singapore: Springer.
- Vatsalan, D. (2014): Scalable and Approximate Privacy-Preserving Record Linkage. PhD thesis. Canberra: Australian National University.

- 
- Vatsalan, D./P. Christen (2016): *Multi-Party Privacy-Preserving Record Linkage using Bloom Filters*. arXiv preprint, <https://arxiv.org/abs/1612.08835>.
- Vatsalan, D./P. Christen/C. O’Keefe/V. S. Verykios (2014): An Evaluation Framework for Privacy-preserving Record Linkage. In: *Journal of Privacy and Confidentiality* 6 (1): 35–75.
- Vatsalan, D./P. Christen/V. S. Verykios (2013): A Taxonomy of Privacy-preserving Record Linkage Techniques. In: *Information Systems* 38 (6): 946–969.
- Venables, W. N./B. D. Ripley (2002): *Modern Applied Statistics with S*. 4th ed. New York: Springer.
- Verikas, A./E. Vaiciukynas/A. Gelzinis/J. Parker/M. C. Olsson (2016): Electromyographic Patterns during Golf Swing: Activation Sequence Profiling and Prediction of Shot Effectiveness. In: *Sensors* 16 (5): 592–618.
- Verykios, V./G. Moustakides/M. Elfeky (2003): A Bayesian decision model for cost optimal record matching. In: *The VLDB Journal* 12 (1): 28–40.
- Vinyals, O./I. Babuschkin/J. Chung/M. Mathieu/M. Jaderberg/W. M. Czarnecki/A. Dudzik/A. Huang/P. Georgiev/R. Powell/T. Ewalds/D. Horgan/M. Kroiss/I. Danihelka/J. Agapiou/J. Oh/V. Dalibard/D. Choi/L. Sifre/Y. Sulsky/S. Vezhnevets/J. Molloy/T. Cai/D. Budden/T. Paine/C. Gulcehre/Z. Wang/T. Pfaff/T. Pohlen/Y. Wu/D. Yogatama/J. Cohen/K. McKinney/O. Smith/T. Schaul/T. Lillicrap/C. Apps/K. Kavukcuoglu/D. Hassabis/D. Silver (2019): *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>.
- Wang, X./D. Feng/X. Lai/H. Yu (2004): *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. IACR Eprint archive.

- 
- Warner, S. L. (1965): Randomized response: A survey technique for eliminating evasive answer bias. In: *Journal of the American Statistical Association* 60 (309): 63–69.
- Weinhardt, M./A. Meyermann/S. Liebig/J. Schupp (2017): The linked employer–employee study of the Socio-Economic Panel (SOEP-LEE): Content, design and research potential. In: *Jahrbücher für Nationalökonomie und Statistik* 237 (5): 457–467.
- Wickham, H. (2016): *ggplot2: elegant graphics for data analysis*. 2nd ed. Cham: Springer.
- Wilson, D. R. (2011): Beyond probabilistic record linkage: Using neural networks and complex features to improve genealogical record linkage. 9–14 in: *The 2011 International Joint Conference on Neural Networks*. San Jose, CA, USA: IEEE.
- Winkler, W. E. (1988): Using the EM algorithm for weight computation in the Fellegi-Sunter model of record linkage. 667–671 in: *Proceedings of the Section on Survey Research Methods, American Statistical Association*. Vol. 667. Alexandria, VA: : American Statistical Association.
- Winkler, W. E. (1990): String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. 354–359 in: *Proceedings of the Section on Survey Research Methods*. American Statistical Association: ERIC.
- Winkler, W. E. (2009): Record Linkage. 351–380 in: Pfeffermann, D./C. Rao (Eds.): *Handbook of Statistics 29A, Sample Surveys: Design, Methods and Applications*. Amsterdam: Elsevier, North-Holland.
- Wolfram, S. (1986): Cryptography with Cellular Automata. 429–432 in: Williams, H. C. (Eds.): *Advances in Cryptology – CRYPTO '85: Proceedings*. Berlin: Springer.
- Wolfram, S. (2002): *A New Kind of Science*. Champaign: Wolfram.

- Wood, G. (2014): Ethereum: A secure decentralised generalised transaction ledger. In: *Ethereum project yellow paper 151*: 1–32.
- Woodhams, J./C. Bennell, eds. (2015): *Crime Linkage: Theory, Research, and Practice*. Boca Raton: CRC Press.
- Yancey, W. E. (2002): *Improving EM algorithm estimates for record linkage parameters*. Washington D.C.: U.S. Bureau of the Census.

---

# Appendices

---

## Source code overview

All source code provided should be run in the given order. For a list of all source code, see page xiv. For the full replication routine, see Code A.1.

Before running any evaluation code, Code B.1 must be run to create the FEBRL WA sample. Code B.2 performs all evaluations and creates all plots in Section 3.5.

As first described in Section 4.3.1, *meta data* refers to the properties of all data sets used (see Section 4.3.3) for training and evaluation of the optimal parameter choice models. For this, Code B.3 evaluates all datasets (an excerpt is shown in Table 4.10) and saves the results so they can be used for training the models. The latter is done in Code C.1, where all training data sets are evaluated as described in Section 4.3.1. This code also contains the model formulation.

The models are then applied to the three evaluation data sets FEBRL WA (Code C.2), NC Voter data (Code C.3) and German Mortality (Hospital and Commercial) data (Code C.4). The resulting metrics are used to encrypt all datasets with the optimal parameter choice estimates and the best-practice suggestions, which is the core of Code C.5. For a full reproduction, this code needs to be run before any result plots can be produced.

For plotting, the Code for the bigram frequencies (Figure 4.3) is found in Code D.1. The plots showing the time taken to link by Tanimoto threshold (Section 4.3.1) are created by Code D.2. The plot used in the introduction for the term “Big Data” (Figure 1.2) is created by Code F.1, while the plot showing the publications for the term “Record Linkage” (Figure 1.1) is created by Code F.2.

Nearly all result plots in Chapter 5 are created by Code D.3, which can be run after all other code mentioned so far. The electronic appendix<sup>1</sup> contains all files, including the result files required for plotting. This way, all plots can be reproduced without the need to run any of the previous code.

To use the results of this work for new applications, two stand-alone codes are provided: estimating the optimal identifier choice for a new data set (Code E.1) and giving the estimated optimal parameter choice for a new input data set (Code E.2).

---

<sup>1</sup>Available at GitHub: <https://github.com/cborgs/thesis/> (Last accessed: 29.07.2019).

Finally, the full R version info for the two R versions used in this thesis is given by Code F.3 (R for Windows) and Code F.4 (R for Ubuntu).

Source Code A.1: Source code for the full replication, including local repositories.

```
1 #####
2 #
3 # Master file to reproduce results from the thesis
4 # All required packages were stored by packrat
5 #
6 #####
7
8 # Set working dir -----
9 #setwd("E:/Priv/Dropbox/Dissertation/Abgabe")
10
11
12 # Load libs -----
13
14 require(packrat)
15 # Only needed once
16 #packrat::init()
17 #devtools::install_github("germanrecordlinkage/multibitTree")
18 #packrat::snapshot()
19 #packrat::clean()
20 #packrat::bundle(file = "Diss_CB_Source.tar.gz",include.lib = TRUE)
21
22 # Load packrat
23 #packrat::unbundle(file = "Diss_CB_Source.tar.gz")
24 packrat::packify()
25
26 # Prepare data -----
27
28 print("Make 100k sample of FEBRL WA for evaluation")
29 source("06_Make_FEBRLWA_100kFile.R")
30
31
32 # Preliminaries and misc. plots -----
33
34 print("Plotting Big data and Record linkage plots (Fig. 1.1 and 1.2.)")
35 source("01_BigData_Scholar_RecordLinkage_medline_soc.R")
```

```
36
37 print("Plotting all preliminary plots: Variance of repeated BF generation, k/fill/f
   ↳ Pretests with sim and real data and random forest vs. linear regression plot")
38 source("02_Preliminaries.R")
39
40 print("Plot Bigram frequency plot (Fig 4.3)")
41 source("03_Plot_BigramFreqs.R")
42
43 print("Plot time by tanimoto threshold, l and tree type; Plot 1 mio vs 1 mio linkage time
   ↳ by threshold")
44 print("Warning: Requires compilation of the mbtSearch_2.0.1 folder: Navigate there and
   ↳ execute 'make'. Make sure the resulting program is executable (chmod -X).")
45 source("04_Plot_Times.R")
46
47
48 # Run main analysis files -----
49
50 # Meta data evaluation of training data
51 source("08_Datasets_Metadata.R")
52
53 # Training data full evaluation, including joining meta data
54 source("07_TestSetup.R")
55
56 # OPC of the three test data sets
57 source("09_Optimal_Parameter_Choice_FEBRLWA.R")
58 source("09b_Optimal_Parameter_Choice_NCVoter.R")
59 source("09c_Optimal_Parameter_Choice_Mortality.R")
60
61 # Evaluation of the parameter choices
62 source("11_OPTC_vs_TrainingResults.R")
63
64
65 # Plot results -----
66
67 # Plot all result plots
68 source("12_Plots_Chap5.R")
```

## Source code used for (data) preparation

Source Code B.1: Source code to generate the sampled FEBRL WA files.

```
1 library(data.table)
2
3 # Target n
4 targetsize <- 100000
5
6 # File names
7 inputFile <- "Daten/Adrian Synthetic Perfect.txt"
8 inputFileB <- "Daten/Adrian Synthetic 10 percent.txt"
9
10 # Read input
11 clearTextA <- fread(inputFile, colClasses = "character", stringsAsFactors = FALSE)
12 clearTextB <- fread(inputFileB, colClasses = "character", stringsAsFactors = FALSE)
13
14
15 ## Deduplicate by ID
16 clearTextA <- clearTextA[!duplicated(clearTextA$`Group Id`),]
17 clearTextB <- clearTextB[!duplicated(clearTextB$`Group Id`),]
18
19 # Sample for size
20 set.seed(42)
21 newfileA <- clearTextA[sample(nrow(clearTextA), targetsize, replace=FALSE),]
22
23 set.seed(42)
24 newfileB <- clearTextB[sample(nrow(clearTextA), targetsize, replace=FALSE),]
25
26 # Write new file
27 write.table(newfileA, "Daten/Adrian_Perfect_100k.csv", row.names = FALSE, sep="\t")
```

```
28 write.table(newfileB, "Daten/Adrian_10percent_100k.csv", row.names = FALSE, sep="\t")
```

Source Code B.2: Source code for the pre-evaluation of BF properties and all resulting plots.

```
1 #setwd("E:/Priv/Dropbox/Dissertation/Programme")
2
3 library(PPRL)
4 library(fastdigest)
5 library(xtermStyle)
6 library(tidyverse)
7 library(reshape2)
8 library(multibitTree)
9 library(gmodels)
10
11
12 # Pre-tests for properties of Bloom Filters
13
14
15
16 # Set number of test cases and BF length
17 n <- 1000
18 l <- 256
19 q <- 2
20 set.seed(69)
21
22 # Sim data by hashing sampled letters
23 dat <- replicate(n ,fastdigest(sample(LETTERS, 10000, replace = TRUE)))
24
25
26
27
28
29 # 20% errors
30 datB <- dat
31 datB[(n - round(0.2 * n)):n] <- gsub("2","y", datB[(n - round(0.2 * n)):n])
32 #datB[(n - round(0.2 * n)):n] <- gsub("1","x", datB[(n - round(0.2 * n)):n])
33
34
35
36 # Init result matrix
```

```

37 results <- NULL
38
39 # Look over k
40 for (k in seq(from = 1, to = 30)){
41
42   # Gen BF
43   testDataBF <- CreateBF(ID = as.character(1:n), dat,
44                          k = k, padding = 0, q = 2, l = 1, password = "(H]$6Uh04q")
45
46   testDataBFB <- CreateBF(ID = as.character(1:n), datB,
47                            k = k, padding = 0, q = 2, l = 1, password = "(H]$6Uh04q")
48
49
50   write.table(testDataBF, "BFA.csv", sep="\t", row.names=FALSE, quote=FALSE,
51              ↪ col.names=TRUE)
52
53   write.table(testDataBFB, "BFB.csv", sep="\t", row.names=FALSE, quote=FALSE,
54              ↪ col.names=TRUE)
55
56   # Empirical fill and k/l
57   fill <- mean(nchar(gsub("0","",testDataBF$CLKs))) / l
58   quotient <- k / l
59   elements <- k * (32/q)
60
61   # Link
62   multibitTree.load("BFB.csv",threads=7, leafLimit = 3)
63   res <- multibitTree.searchFile("BFA.csv", 0.85)
64
65   # Calc quality measures
66   tp <- sum(as.character(res$query) == as.character(res$fingerprint))
67   fp <- sum(as.character(res$query) != as.character(res$fingerprint))
68   fn <- n - tp
69   recall <- (tp/(tp+fn))
70   precision <- (tp/(tp+fp))
71   fscore <- (recall + precision) / 2
72
73   # Print result
74   cat("\n",style("\t  PPRL Test\t\t", bg = "black", fg = "white", font="bold"),
75       "\n Positives:\t\t", n,"\n k:\t\t\t", k,"\n Fill:\t\t\t", fill,

```

```

76     "\n True positives:\t", tp,
77     "\n False positives:\t", fp, "\n False Negatives:\t", fn,
78     "\n Recall:\t\t", (tp/(tp+fn)), "\n Precision:\t\t", (tp/(tp+fp)),
79     "\n", style("\t\t \t\t", bg = "black", fg = "white", font="bold"))
80
81 # Append results
82 results <- rbind(results, data.frame(data="Synthetic",k, q, l, fill,elements, quotient,
83   ↪ tp, fp, fn, precision, recall, fscore))
84
85 # Plotting
86 #plotdata <- results
87 write.table(results, "./results/fill_k_quality.csv", sep="\t", row.names=FALSE,
88   ↪ quote=FALSE, col.names=TRUE)
89
90 plotdata <- read.csv("./results/fill_k_quality.csv", head=TRUE, sep = "\t",
91   ↪ stringsAsFactors = FALSE)
92
93 # No. of elements hashed. Fastdigest returns 32 bit hashes, divided into q-grams, hashes k
94   ↪ times.
95
96
97 library(ggplot2)
98
99 p1 <- ggplot(data = plotdata, aes(y = round(fill * 100), x = k))
100 p1 <- p1 + geom_smooth(se=F, method="loess", color="#004c93") # method="loess"
101 p1 <- p1 + geom_point(size=6) #
102 p1 <- p1 + xlab("\nHash functions k") + ylab("% of bit positions set to one\n")
103 p1 <- p1 + scale_y_continuous(breaks = seq(0,100,10), limits= c(1,100), expand = c(0.02,
104   ↪ 0))
105 p1 <- p1 + scale_x_continuous(breaks = seq(0,30,5))
106 p1 + theme_bw() + theme(panel.spacing = unit(0.85, "lines"), text =
107   ↪ element_text(size=35), legend.justification=c(0.9,0.05), legend.key = element_blank(),
108   ↪ legend.key.size = unit(5,"char"), strip.background = element_rect(colour='white',fill
109   ↪ = 'white'))
110 ggsave("./results/k_vs_fill.pdf", height=11, width=15, units="in", device=cairo_pdf) #
111   ↪ Fig 3.15
112
113
114 p2 <- ggplot(data = plotdata, aes(y = fscore, x = round(fill * 100)))
115 p2 <- p2 + geom_smooth(se=F, method="loess", color="#004c93", span=.25) # method="loess"

```

```

108 p2 <- p2 + geom_point(size=6) # methos="loess"
109 p2 <- p2 + geom_vline(xintercept=50, colour="#004c93") +
110   geom_text(aes(x=50, label="\nTheoretical optimum", y=0.8), colour="#004c93", angle=90,
  ↪ size=6)
111 p2 <- p2 + scale_x_continuous(breaks = seq(0,100,10), limits= c(1,100), expand = c(0.02,
  ↪ 0))
112 p2 <- p2 + xlab("\n% of bit positions set to one") + ylab("F-Score\n")
113 p2 + theme_bw() + theme(panel.spacing = unit(0.85, "lines"), text =
  ↪ element_text(size=35),legend.justification=c(0.9,0.05), legend.key = element_blank(),
  ↪ legend.key.size = unit(5,"char"), strip.background = element_rect(colour='white',fill
  ↪ = 'white'))
114 ggsave("./results/fill_vs_f.pdf", height=11, width=15, units="in", device=cairo_pdf) #
  ↪ Fig 3.18
115
116
117
118 p2 <- ggplot(data = plotdata, aes(y = precision, x = round(fill * 100)))
119 p2 <- p2 + geom_smooth(se=F, method="loess", color="#004c93", span=.25) # method="loess"
120 p2 <- p2 + geom_point(size=6) # methos="loess"
121 p2 <- p2 + geom_vline(xintercept=50, colour="#004c93") +
122   geom_text(aes(x=50, label="\nTheoretical optimum", y=0.8), colour="#004c93", angle=90,
  ↪ size=6)
123 p2 <- p2 + scale_x_continuous(breaks = seq(0,100,10), limits= c(1,100), expand = c(0.02,
  ↪ 0))
124 p2 <- p2 + xlab("\n% of bit positions set to one") + ylab("Precision\n")
125 p2 + theme_bw() + theme(panel.spacing = unit(0.85, "lines"), text =
  ↪ element_text(size=35),legend.justification=c(0.9,0.05), legend.key = element_blank(),
  ↪ legend.key.size = unit(5,"char"), strip.background = element_rect(colour='white',fill
  ↪ = 'white'))
126 ggsave("./results/fill_vs_prec.pdf", height=11, width=15, units="in", device=cairo_pdf) #
  ↪ Fig 3.16
127
128
129
130
131 p2 <- ggplot(data = plotdata, aes(y = recall, x = round(fill * 100)))
132 p2 <- p2 + geom_smooth(se=F, method="loess", color="#004c93", span=.25) # method="loess"
133 p2 <- p2 + geom_point(size=6) # methos="loess"
134 #p2 <- p2 + geom_vline(xintercept=50, colour="#004c93") +
135 # geom_text(aes(x=50, label="\nTheoretical optimum", y=0.8), colour="#004c93", angle=90,
  ↪ size=6)

```

```

136 p2 <- p2 + scale_x_continuous(breaks = seq(0,100,10), limits= c(1,100), expand = c(0.02,
  ↪ 0))
137 p2 <- p2 + xlab("\n% of bit positions set to one") + ylab("Recall\n")
138 p2 + theme_bw() + theme(panel.spacing = unit(0.85, "lines"), text =
  ↪ element_text(size=35),legend.justification=c(0.9,0.05), legend.key = element_blank(),
  ↪ legend.key.size = unit(5,"char"), strip.background = element_rect(colour='white',fill
  ↪ = 'white'))
139 ggsave("./results/fill_vs_rec.pdf", height=11, width=15, units="in", device=cairo_pdf) #
  ↪ Fig 3.17
140
141 p3 <- ggplot(data = plotdata, aes(y = fscore, x = k))
142 p3 <- p3 + geom_smooth(se=F, method="loess", color="#004c93", span=.25) # method="loess"
143 p3 <- p3 + geom_point(size=6)
144 p3 <- p3 + geom_point(aes(y = 0.976, x = 11), color="#004c93", size=6)
145 p3 <- p3 + geom_text(aes(y = 1.02, x = 11, label="Optimal k by Smith (2012)"),
  ↪ color="#004c93", size=6)
146 p3 <- p3 + xlab("\nHash functions k") + ylab("F-Score\n")
147 p3 + theme_bw() + theme(panel.spacing = unit(0.85, "lines"), text =
  ↪ element_text(size=35),legend.justification=c(0.9,0.05), legend.key = element_blank(),
  ↪ legend.key.size = unit(5,"char"), strip.background = element_rect(colour='white',fill
  ↪ = 'white'))
148 ggsave("./results/k_vs_f.pdf", height=11, width=15, units="in", device=cairo_pdf) # Fig
  ↪ 3.19
149
150
151 p4 <- ggplot(data = plotdata, aes(y = fscore, x = elements))
152 p4 <- p4 + geom_smooth(se=F, method="loess", color="#004c93", span=.25) # method="loess"
153 p4 <- p4 + geom_point(size=6)
154 p4 <- p4 + xlab("\nNo. of Elements hashed") + ylab("F-Score\n")
155 p4 + theme_bw() + theme(panel.spacing = unit(0.85, "lines"), text =
  ↪ element_text(size=35),legend.justification=c(0.9,0.05), legend.key = element_blank(),
  ↪ legend.key.size = unit(5,"char"), strip.background = element_rect(colour='white',fill
  ↪ = 'white'))
156 ggsave("./results/elements_vs_f.pdf", height=11, width=15, units="in", device=cairo_pdf)
157
158
159
160 # AMount of hash functions for 50% fill acc to smith 2012:
161 (log(1-0.5) / log(1-(1/16))) / 16
162 # [1] 11.06868
163

```

```
164
165
166 # Init result matrix
167 results <- NULL
168 l = 500
169
170 set.seed(1)
171
172 # Bootstrap with pws
173 for (k in c(5,10, 15)){
174   for (i in seq(from = 1, to = 30)){
175
176
177     PWS <- replicate(length(1), fastdigest(sample(LETTERS, 10000, replace = TRUE)))
178     # Gen BF
179     testDataBF <- CreateBF(ID = as.character(1:n), dat,
180                           k = k, padding = 0, q = 2, l = l, password = PWS)
181
182     testDataBFB <- CreateBF(ID = as.character(1:n), datB,
183                             k = k, padding = 0, q = 2, l = l, password = PWS)
184
185     write.table(testDataBF, "BFA.csv", sep="\t", row.names=FALSE, quote=FALSE,
186               ↪ col.names=TRUE)
187     write.table(testDataBFB, "BFB.csv", sep="\t", row.names=FALSE, quote=FALSE,
188               ↪ col.names=TRUE)
189
190
191     # Empirical fill and k/l
192     fill <- mean(nchar(gsub("0","",testDataBF$CLKs))) / l
193     quotient <- k / l
194     elements <- k * (32/q)
195
196
197     # Link
198     multibitTree.load("BFB.csv", threads=7, leafLimit = 3)
199     res <- multibitTree.searchFile("BFA.csv", 0.9)
200
201
202     # Calc quality measures
203     tp <- sum(as.character(res$query) == as.character(res$fingerprint))
204     fp <- sum(as.character(res$query) != as.character(res$fingerprint))
205     fn <- n - tp
206     recall <- (tp/(tp+fn))
```



```

234
235
236
237 cairo_pdf(file = "Variation_Normality.pdf", width=9, height=15) # Fig 3.12
238
239 par(mfrow=c(3,1), mar=c(5,5,3,1))
240
241 qqnorm(results$fscore[results$k==5], main = "QQ-Plot, k=5", cex=2,
  ↪ cex.axis=2,cex.lab=2,cex.main=2.2)
242 qqline(results$fscore[results$k==5])
243 qqnorm(results$fscore[results$k==10], main = "QQ-Plot, k=10", cex=2,
  ↪ cex.axis=2,cex.lab=2,cex.main=2.2)
244 qqline(results$fscore[results$k==10])
245 qqnorm(results$fscore[results$k==15], main = "QQ-Plot, k=15", cex=2,
  ↪ cex.axis=2,cex.lab=2,cex.main=2.2)
246 qqline(results$fscore[results$k==15])
247
248 dev.off()
249
250
251
252 ##### Real data #####
253
254 ##### FEBRL
255
256 # Sim data by hashing sampled letters
257 dat <- read.csv("./Daten/FEBRL_10000_20_A.csv", head=FALSE, sep = "\t", stringsAsFactors
  ↪ = FALSE, colClasses = "character")
258 datB <- read.csv("./Daten/FEBRL_10000_20_B.csv", head=FALSE, sep = "\t", stringsAsFactors
  ↪ = FALSE, colClasses = "character")
259
260 # Generate full string to create BF
261 dat$linkkey <- paste0(dat$V2, dat$V3, dat$V10)
262 dat$linkkey <- StandardizeString(dat$linkkey)
263 datB$linkkey <- paste0(datB$V2, datB$V3, datB$V10)
264 datB$linkkey <- StandardizeString(datB$linkkey)
265
266 l <- 256
267 q <- 2
268 n <- nrow(dat)
269

```

```
270
271 # Init result matrix
272 #results <- NULL
273
274 # Look over k
275 for (k in seq(from = 1, to = 30)){
276
277   print(k)
278
279   # Gen BF
280   testDataBF <- CreateBF(ID = dat$V1, dat$linkkey,
281                          k = k, padding = 0, q = 2, l = 1, password = "(H]$6Uh04q")
282   print("First BF")
283
284   testDataBFB <- CreateBF(ID = datB$V1, datB$linkkey,
285                            k = k, padding = 0, q = 2, l = 1, password = "(H]$6Uh04q")
286   print("sec BF")
287
288   write.table(testDataBF, "BFA.csv", sep="\t", row.names=FALSE, quote=FALSE,
289              ↪ col.names=TRUE)
289   write.table(testDataBFB, "BFB.csv", sep="\t", row.names=FALSE, quote=FALSE,
290              ↪ col.names=TRUE)
291
292   # Empirical fill and k/l
293   fill <- mean(nchar(gsub("0", "", testDataBF$CLKs))) / l
294   quotient <- k / l
295   elements <- k * (mean(nchar(dat$linkkey))/q)
296
297   # Link
298   multibitTree.load("BFB.csv", threads=7, leafLimit = 3)
299   res <- multibitTree.searchFile("BFA.csv", 0.85)
300
301
302
303   # Calc quality measures
304   tp <- sum(as.character(res$query) == as.character(res$fingerprint))
305   fp <- sum(as.character(res$query) != as.character(res$fingerprint))
306   fn <- n - tp
307   recall <- (tp/(tp+fn))
308   precision <- (tp/(tp+fp))
```



```

347
348
349 # Init result matrix
350 #results <- NULL
351
352 # Look over k
353 for (k in seq(from = 1, to = 30)){
354
355
356 # Gen BF
357 testDataBF <- CreateBF(ID = dat$V1, dat$linkkey,
358                       k = k, padding = 0, q = 2, l = 1, password = "(H]$6Uh04q")
359
360 testDataBFB <- CreateBF(ID = datB$V1, datB$linkkey,
361                       k = k, padding = 0, q = 2, l = 1, password = "(H]$6Uh04q")
362
363
364 write.table(testDataBF, "BFA.csv", sep="\t", row.names=FALSE, quote=FALSE,
365            ↪ col.names=TRUE)
366 write.table(testDataBFB, "BFB.csv", sep="\t", row.names=FALSE, quote=FALSE,
367            ↪ col.names=TRUE)
368
369 # Empirical fill and k/l
370 fill <- mean(nchar(gsub("0","",testDataBF$CLKs))) / l
371 quotient <- k / l
372 elements <- k * (mean(nchar(dat$linkkey))/q)
373
374 # Link
375 multibitTree.load("BFB.csv",threads=7, leafLimit = 3)
376 res <- multibitTree.searchFile("BFA.csv", 0.85)
377
378
379 # Calc quality measures
380 tp <- sum(as.character(res$query) == as.character(res$fingerprint))
381 fp <- sum(as.character(res$query) != as.character(res$fingerprint))
382 fn <- n - tp
383 recall <- (tp/(tp+fn))
384 precision <- (tp/(tp+fp))
385 fscore <- (recall + precision) / 2

```



```

419 ggsave("./results/k_vs_fill_data.pdf", height=11, width=15, units="in", device=cairo_pdf)
    ↪ # Fig 3.20
420
421 p2 <- ggplot(data = plotdata, aes(y = fscore, x = round(fill * 100)))
422 p2 <- p2 + geom_smooth(se=F, method="loess", color="#004c93", span=.25) # method="loess"
423 p2 <- p2 + geom_point(size=6) # method="loess"
424 p2 <- p2 + geom_vline(xintercept=50, colour="#004c93") +
425   geom_text(aes(x=50, label="\nTheoretical optimum", y=0.8), colour="#004c93", angle=90,
    ↪ size=6)
426 p2 <- p2 + scale_x_continuous(breaks = seq(0,100,10), limits= c(1,100), expand = c(0.02,
    ↪ 0))
427 p2 <- p2 + xlab("\n% of bit positions set to one") + ylab("F-Score\n")
428 p2 + facet_grid(data ~ ., labeller = label_both) + theme_bw() + theme(panel.spacing =
    ↪ unit(0.85, "lines"), text = element_text(size=35), legend.justification=c(0.9,0.05),
    ↪ legend.key = element_blank(), legend.key.size = unit(5,"char"), strip.background =
    ↪ element_rect(colour='white',fill = 'white'))
429 ggsave("./results/fill_vs_f_data.pdf", height=11, width=15, units="in", device=cairo_pdf)
    ↪ # Fig 3.21
430
431 p3 <- ggplot(data = plotdata, aes(y = fscore, x = k))
432 p3 <- p3 + geom_smooth(se=F, method="loess", color="#004c93", span=.25) # method="loess"
433 p3 <- p3 + geom_point(size=6)
434 #p3 <- p3 + geom_point(aes(y = 0.976, x = 8), color="#004c93", size=6)
435 #p3 <- p3 + geom_text(aes(y = 1.02, x = 8, label="Optimal k by Smith (2012)"),
    ↪ color="#004c93", size=6)
436 p3 <- p3 + xlab("\nHash functions k") + ylab("F-Score\n")
437 p3 + facet_grid(data ~ ., labeller = label_both) + theme_bw() + theme(panel.spacing =
    ↪ unit(0.85, "lines"), text = element_text(size=35), legend.justification=c(0.9,0.05),
    ↪ legend.key = element_blank(), legend.key.size = unit(5,"char"), strip.background =
    ↪ element_rect(colour='white',fill = 'white'))
438 ggsave("./results/k_vs_f_data.pdf", height=11, width=15, units="in", device=cairo_pdf)
439
440
441 p4 <- ggplot(data = plotdata, aes(y = fscore, x = elements))
442 p4 <- p4 + geom_smooth(se=F, method="loess", color="#004c93", span=.25) # method="loess"
443 p4 <- p4 + geom_point(size=6)
444 p4 <- p4 + xlab("\nNo. of Elements hashed") + ylab("F-Score\n")
445 p4 + facet_grid(data ~ ., labeller = label_both) + theme_bw() + theme(panel.spacing =
    ↪ unit(0.85, "lines"), text = element_text(size=35), legend.justification=c(0.9,0.05),
    ↪ legend.key = element_blank(), legend.key.size = unit(5,"char"), strip.background =
    ↪ element_rect(colour='white',fill = 'white'))

```

```
446 ggsave("./results/elements_vs_f_data.pdf", height=11, width=15, units="in",
447 ↪ device=cairo_pdf)
448 # AMount of hash functions for 50% fill acc to smith 2012:
449 (log(1-0.5) / log(1-(1/256))) / 16
450 # [1] 11.06868
451
452 # Opt hash functions for 50% fill acc to smith 2012:
453 #(1/n) * log(2)
454 (256/16) * log(2)
455 #[1] 11.09035
456
457 ## TelCD
458 #mean(nchar(dat$linkkey))
459 # 20.0304
460
461 # AMount of hash functions for 50% fill acc to smith 2012:
462 (log(1-0.5) / log(1-(1/256))) / 20
463 # [1] 8.854944
464
465 # Opt hash functions for 50% fill acc to smith 2012:
466 #(1/n) * log(2)
467 (256/20) * log(2)
468 #[1] 8.872284
469
470
471 ## FEBRL
472 #mean(nchar(dat$linkkey))
473 # 20.4278
474
475 # AMount of hash functions for 50% fill acc to smith 2012:
476 (log(1-0.5) / log(1-(1/256))) / 20
477 # [1] 8.854944
478
479 # Opt hash functions for 50% fill acc to smith 2012:
480 #(1/n) * log(2)
481 (256/20) * log(2)
482 #[1] 8.872284
483
484
485 ##### Random forests vs. linear regression plot
```

```

486
487 library(ggplot2)
488 library(randomForest)
489
490 # Random numbers
491 x = runif(1000, min = 0, max = 100)
492 # Log of x
493 y = log(x) + runif(1000, min=-0.2, max=0.2)
494
495 # Function for plotting
496 logfun <- function(x) log(x)
497
498 # Linear prediction
499 lin <- lm(y ~ x)
500 linPred <- predict(lin)
501
502 # RF prediction
503 r2 <- randomForest(y ~ x, ntree=3000, nodesize=5, importance=TRUE, nPerm=5,
504   ↪ keep.forest=TRUE)
505 rfPred <- predict(r2)
506
507 # Put all into DF
508 df <- data.frame(x,y, linPred, rfPred)
509
510 # Plot and save
511 ggplot(df, aes(x=x, y=y)) +
512   geom_point(aes(color="orange"), size = 3, alpha=1) +
513   stat_function(fun = logfun, color = "black", alpha = 1) +
514   geom_point(aes(x=x, y=linPred, color="blue"), size = 4, alpha=.2) +
515   geom_point(aes(x=x, y=rfPred, color="red"), size = 4, alpha=.4) +
516   xlab("Uniform random numbers") + ylab("log(x)") +
517   scale_color_manual(name="Prediction", values=c("#991607", "#d38e0e", "#004c93"), labels
518     ↪ = c("Linear prediction", "True values", "Random Forest prediction")) +
519   theme_bw() + theme(panel.spacing = unit(1, "lines"), text = element_text(size=27),
520     axis.text.x = element_text(size = 21, angle = 0, vjust = 0.5),
521     ↪ axis.title=element_text(size=30),
522     legend.justification=c(0.9,0.05), legend.key = element_blank(),
523     ↪ legend.key.size = unit(5,"char"),
524     strip.background = element_rect(colour='white',fill = 'white'))
525 ggsave("linear_vs_RF.pdf",height=13, width=18, units="in", device=cairo_pdf) # Fig 4.11

```

## Source Code B.3: Source code for the meta data.

```
1 # Clear workspace, set working dir
2 #rm(list=ls())
3 #setwd("E:/Priv/Dropbox/Dissertation/Programme")
4
5
6 # Libs used
7 library(PPRL)
8 library(fastdigest)
9 library(stringdist)
10 library(entropy)
11 library(data.table)
12 library(fastLink)
13 library(ggplot2)
14 library(moments)
15 library(ineq)
16
17 ##### File names #####
18 ewodataFN <- "Daten/Abgleich_UKE_SCHUFA_EMA.csv"
19 ncvoterAFN <- "Daten/ncvoter-20140619-temporal-balanced-ratio-1to1-a.csv"
20 ncvoterBFN <- "Daten/ncvoter-20140619-temporal-balanced-ratio-1to1-b.csv"
21 telCDAFN <- "Daten/A_10000_python_R0_C0_0100_percent.csv"
22 telCDBFN <- "Daten/B_10000_python_R20_C50_0100_percent.csv"
23 Feb1AFN <- "Daten/FEBRL_10000_20_A.csv"
24 Feb1BFN <- "Daten/FEBRL_10000_20_B.csv"
25
26 ##### Loop vars #####
27 # Set data sets for loop
28 datasets <- c("German Mortality", "German Telephone CD 20% errors", "German Telephone CD 0%
  ↪ errors", "FEBRL")
29
30 # Set to standard identifier set for CLK
31 v <- "Standard CLK"
32 # Amount of hash functions to loop over
33 K <- seq(from = 1, to = 40, by = 1) # Amount of Hash functions
34 # q-grams to loop over
35 Q <- c(1,2,3,4)
36 # Length of the BF
37 l <- 500
```

```
38 # No loopable errors in data
39 fehler <- c(0)
40 # Set padding to false
41 padding <- FALSE
42 # Size of NC Voter data subsample
43 nNCV <- 50000
44 # Thresholds to loop over
45 tanimotoThresholds <- seq(from= 1.0, to = 0.8, by = -0.01)
46 # Set cores and leaf limit
47 cores <- 7
48 leaflimit <- 3
49
50
51
52 # identifiers <- c("FN","LN", "SEX", "YEAR", "DAY", "MONTH","PLACE")
53 #
54 # # Build Identifier Sets from all combinations (except for nonsense two-ID cases, only
55   ↪ FNLN FNYEAR LNYEAR count)
56 # identifierSets <- c(paste(identifiers[c(1,2)], collapse=""), paste(identifiers[c(1,4)],
57   ↪ collapse=""),
58   ↪ paste(identifiers[c(2,4)], collapse=""))
59 # identifierSets <- c(identifierSets, apply(t(combn(identifiers,3)), 1, paste,
60   ↪ collapse=""))
61 # identifierSets <- c(identifierSets, apply(t(combn(identifiers,4)), 1, paste,
62   ↪ collapse=""))
63 # identifierSets <- c(identifierSets, apply(t(combn(identifiers,5)), 1, paste,
64   ↪ collapse=""))
65 # identifierSets <- c(identifierSets, apply(t(combn(identifiers,6)), 1, paste,
66   ↪ collapse=""))
67 # identifierSets <- c(identifierSets, paste(identifiers, collapse=""))
68
69 # Init Counter to count progress
70 counter <- 0
71
72 # Calc number of combinations in total
73 numCombinations <- length(datasets) * length(Q)
74
75 # Init result matrix
76 meta <- NULL
77
```

```

73 cat("\n\n\nClassifications start!", format(Sys.time(), " (%a, %d-%m-%Y, %H:%M:%S)"), "\n")
74
75
76 ##### Function definitions #####
77
78 ##### Standardization #####
79 unwanted_array = list('Š'='S', 'š'='s', 'Ž'='Z', 'ž'='z', 'À'='A', 'Á'='A', 'Â'='A',
80   ↪ 'Ã'='A', 'Ä'='Ae', 'Å'='A', 'Æ'='A', 'Ç'='C', 'Ð'='D', 'È'='E', 'É'='E',
81   ↪ 'Ê'='E', 'Ë'='E', 'Ì'='I', 'Í'='I', 'Î'='I', 'Ï'='I', 'Ñ'='N',
82   ↪ 'Ö'='Oe', 'Ò'='O', 'Ó'='O', 'Ô'='O', 'Õ'='O', 'Õ'='O', 'Ø'='O',
83   ↪ 'Ù'='U',
84   ↪ 'Ú'='U', 'Û'='Ue', 'Ý'='Y', 'Þ'='B', 'ß'='Ss', 'à'='a',
85   ↪ 'á'='a', 'â'='a', 'ã'='a', 'ä'='ae', 'å'='a', 'æ'='a', 'ç'='c',
86   ↪ 'è'='e', 'é'='e', 'ê'='e', 'ë'='e', 'ì'='i', 'í'='i', 'î'='i',
87   ↪ 'ï'='i', 'ð'='o', 'ñ'='n', 'ò'='o', 'ó'='o', 'ô'='o', 'õ'='o',
88   ↪ 'ö'='oe', 'ø'='o', 'ü'='ue', 'ù'='u', 'ú'='u', 'û'='u', 'ý'='y',
89   ↪ 'ÿ'='y', 'þ'='b', 'ÿ'='y' )
90
91 normalize_string=function(string){
92   string = enc2utf8(string)
93   string=gsub("[ ]$*+.[^{|(\\#%&~/<=> '! , ; \\") }@]", "", string)
94   string=gsub("-", "", string)
95   string = chartr(paste(names(unwanted_array), collapse=''),
96     paste(unwanted_array, collapse=''),
97     string)
98   string=gsub("[\t\n\r\f\v]", "", string)
99   string=gsub("[0-9]", "", string)
100  string=toupper(string)
101  string=iconv(string, "utf8", "ASCII", sub="")
102  return(string)
103 }
104
105 normalize_dobs=function(string){
106   if(is.na(string)){
107     return("")
108   } else {
109     string = as.character(string)
110     if (nchar(string) < 2){
111       string = paste0("0", string)
112     }
113     return(string)
114   }
115 }

```

```
108 }
109 }
110
111
112
113
114
115 ##### Loop over datasets #####
116
117 for (d in datasets){
118
119
120   ##### Preprocess depending on data #####
121   if (d == "NC Voter"){
122
123     clearTextB <- fread(ncvoterBFN, sep=",")
124     clearTextA <- fread(ncvoterAFN, sep=",")
125
126     clearTextA <- clearTextA[,c(1,2,4,5)]
127     clearTextB <- clearTextB[,c(1,2,4,5)]
128
129     # Draw sample
130     set.seed(69)
131     clearTextA <- clearTextA[sample(1:nrow(clearTextA), nNCV, replace=FALSE),]
132     clearTextB <- clearTextB[sample(1:nrow(clearTextB), nNCV, replace=FALSE),]
133
134
135     clearTextA$Day <- ""
136     clearTextA$Month <- ""
137     clearTextA$Year <- as.character(2014-clearTextA$age)
138
139     clearTextB$Day <- ""
140     clearTextB$Month <- ""
141     clearTextB$Year <- as.character(2014-clearTextB$age)
142     clearTextA$age <- NULL
143     clearTextB$age <- NULL
144
145     ### names anpassen für alle datasets
146
147   } else if (d == "German Telephone CD 20% errors"){
148
```

```
149 clearTextB <- fread(telCDBFN, sep="\t")
150 clearTextA <- fread(telCDAFN, sep="\t")
151
152 # Reorder
153 clearTextA <- clearTextA[,c(1,7,8,5,4,3)]
154 clearTextB <- clearTextB[,c(1,7,8,5,4,3)]
155
156
157 } else if (d == "German Telephone CD 0% errors"){
158
159 clearTextB <- fread(telCDAFN, sep="\t")
160 clearTextA <- fread(telCDAFN, sep="\t")
161
162 # Reorder
163 clearTextA <- clearTextA[,c(1,7,8,5,4,3)]
164 clearTextB <- clearTextB[,c(1,7,8,5,4,3)]
165
166
167 } else if (d == "German Mortality"){
168 clearText <- fread(ewodataFN, sep=",")
169
170 clearTextB <- subset(clearText, quelle=="schufa")
171 clearTextA <- subset(clearText, quelle=="einwo")
172 clearTextA <- clearTextA[,c(14,2,3,5,6,7)]
173 clearTextB <- clearTextB[,c(14,2,3,5,6,7)]
174
175 } else {
176 clearTextB <- fread(FebrlBFN, sep="\t")
177 clearTextA <- fread(FebrlAFN, sep="\t")
178
179 clearTextA <- clearTextA[,c(1,2,3,10)]
180 clearTextB <- clearTextB[,c(1,2,3,10)]
181 clearTextA$Day <- (substr(clearTextA$V10,7,8))
182 clearTextA$Month <- (substr(clearTextA$V10,5,6))
183 clearTextA$Year <- (substr(clearTextA$V10,1,4))
184 clearTextA$V10 <- NULL
185
186 clearTextB$Day <- (substr(clearTextB$V10,7,8))
187 clearTextB$Month <- (substr(clearTextB$V10,5,6))
188 clearTextB$Year <- (substr(clearTextB$V10,1,4))
189 clearTextB$V10 <- NULL
```

```
190
191 }
192
193 # Read error-free file
194 names(clearTextA) <- c("ID", "Vorname", "Nachname", "Day", "Month", "Year")
195 names(clearTextB) <- c("ID", "Vorname", "Nachname", "Day", "Month", "Year")
196
197
198
199 ##### General preproc #####
200 clearTextA$ID <- as.character(clearTextA$ID)
201 clearTextB$ID <- as.character(clearTextB$ID)
202
203
204 # Preprocess A
205
206 ## Standardize data
207 clearTextA$Vorname <- normalize_string(clearTextA$Vorname)
208 clearTextA$Nachname <- normalize_string(clearTextA$Nachname)
209
210 clearTextA$Day <- sapply(clearTextA$Day, normalize_dobs)
211 clearTextA$Month <- sapply(clearTextA$Month, normalize_dobs)
212 clearTextA$Year <- as.character(ifelse(is.na(clearTextA$Year), "", clearTextA$Year))
213
214 # Generate full DOB
215 clearTextA$DOB <- paste(clearTextA$Day, clearTextA$Month, clearTextA$Year, sep="")
216
217
218 # Preprocess B
219
220 ## Standardize data
221 clearTextB$Vorname <- normalize_string(clearTextB$Vorname)
222 clearTextB$Nachname <- normalize_string(clearTextB$Nachname)
223
224 clearTextB$Day <- sapply(clearTextB$Day, normalize_dobs)
225 clearTextB$Month <- sapply(clearTextB$Month, normalize_dobs)
226 clearTextB$Year <- as.character(ifelse(is.na(clearTextB$Year), "", clearTextB$Year))
227
228 # Generate full DOB for salting
229 clearTextB$DOB <- paste(clearTextB$Day, clearTextB$Month, clearTextB$Year, sep="")
230
```

```

231
232 ## Deduplicate by ID
233 clearTextA <- clearTextA[!duplicated(clearTextA$ID),]
234 clearTextB <- clearTextB[!duplicated(clearTextB$ID),]
235
236
237 ## File sizes, overlap
238 filesizeA <- (nrow(clearTextA))
239 filesizeB <- (nrow(clearTextB))
240
241 pairs <- as.numeric(filesizeA) * as.numeric(filesizeB)
242 truepositives <- sum(clearTextA$ID %in% clearTextB$ID)
243 positives <- truepositives
244
245
246
247 cat("\n\n\nClear-Text files ready, analyzing files", format(Sys.time()," (%a, %d-%m-%Y,
↳ %H:%M:%S)"), "\n")
248
249
250
251 #####
252
253 # Get EM weights
254 ests <- fastLink(clearTextA, clearTextB, varnames = c("Vorname", "Nachname", "DOB"),
255               stringdist.match=c("Vorname", "Nachname"), stringdist.method="jw",
↳ n.cores = 7, estimate.only = TRUE)
256
257 m <- ests$p.m
258 u <- ests$p.u
259
260
261 for (q in Q){
262
263   cat("\n\n\nLine-by-line checking...", format(Sys.time()," (%a, %d-%m-%Y,
↳ %H:%M:%S)"), "\n")
264
265
266 # Generate full linkage key
267 keys <- apply(clearTextA, 1, paste0, collapse = "")
268 keys <- gsub(" ", "", keys)

```

```

269
270 # Split into bigrams
271 qgrams <- (sapply(keys,function(key)substring(key,first=seq(1,nchar(key)),last=seq(q,
  ↪ nchar(key)+q))))
272 # Count uniques and save measures
273 uniquengrams <- as.numeric(unlist(lapply(qgrams, function(x)length(unique(x)))))
274 # ngram Count
275 ngramsC <- as.numeric(unlist(lapply(qgrams, function(x)length(x))))
276 qgrams <- unlist(qgrams)
277 qgramlist <- as.character(na.omit(ifelse(qgrams == "" | nchar(qgrams)!= q,NA,qgrams)))
278
279 cat("\n\n\nLine-by-line checking complete, calculating measures...",
  ↪ format(Sys.time()," (%a, %d-%m-%Y, %H:%M:%S)"),"\n")
280
281 # Sum missings
282 missings <- sum(apply(clearTextA, 2, function(x)sum(is.na(x)))) +
  ↪ sum(apply(clearTextA, 2, function(x)sum(x=="")))
283
284 # Mean entropy calculation
285 meanentropy <- mean(apply(clearTextA, 2, function(x) entropy(table(x))))
286
287 # q90/q10-ratio
288 q90 <- round(quantile(as.numeric(table(qgramlist)), probs=seq(0,1,0.1))["90%"])
289 q10 <- round(quantile(as.numeric(table(qgramlist)), probs=seq(0,1,0.1))["10%"])
290
291 # Measures
292 meanengrams <- mean(ngramsC)
293 sdgrams <- sd(ngramsC)
294
295 skew <- skewness(table(qgramlist))
296 ginicoefficientqgrams <- Gini(table(qgramlist))
297
298 ginicoefficient <- mean(apply(clearTextA, 2, function(x) Gini(table(x))))
299
300 errorestimation <- q90/q10 # amount of rare combinations in data
301 missingamount <- missings
302 meanmissing <- missings/nrow(clearTextA)
303 uniqueness <- mean(uniquengrams/ngramsC)
304 sduniqueness <- sd(uniquengrams/ngramsC)
305
306

```

```
307
308 # Set counter plus one
309 counter <- counter + 1
310
311 #gc()
312 cat("\nClassification", counter, "of", numCombinations,"Finished (",format((counter /
  ↪ numCombinations)*100, digits=3),"%)!",format(Sys.time()," (%a, %d-%m-%Y,
  ↪ %H:%M:%S)"),": \nFilesizes:\t",format(filesizeA,scientific=FALSE),"Data
  ↪ source:\t",d,"\n q:\t\t",q,"\n\n")
313
314 # Append to result
315 meta <- rbind(meta, cbind(d, q, l, filesizeA, filesizeB, positives, meanentropy,
  ↪ meanengrams, sdgrams, skew, ginicoefficient,ginicoefficientqgrams,
  ↪ errorestimation , uniqueness,sduniqueness, meanmissing, missingamount,m,u))
316 write.table(meta, "tempResult2.csv", sep="\t", row.names=FALSE, quote=FALSE,
  ↪ col.names=TRUE)
317 }
318 }
319
320
321 # format result
322 result <- as.data.frame(meta)
323
324 # Write final result file
325 write.table(result, "fullResult_Metadata.csv", sep="\t", row.names=FALSE, quote=FALSE,
  ↪ col.names=TRUE)
```

# Source code of the model implementation and evaluation

Source Code C.1: Main code for evaluating the training data. Requires meta data source code to be run first.

```
1 # Clear workspace, set working dir
2 #rm(list=ls())
3 #setwd("E:/Priv/Dropbox/Dissertation/Programme")
4
5
6 # Libs used
7 library(PPRL)
8 library(fastdigest)
9 library(stringdist)
10 library(entropy)
11 library(data.table)
12 library(fastLink)
13 library(ggplot2)
14
15
16
17 ##### File names #####
18 ewodataFN <- "Daten/Abgleich_UKE_SCHUFA_EMA.csv"
19 ncvoterAFN <- "Daten/ncvoter-20140619-temporal-balanced-ratio-1to1-a.csv"
20 ncvoterBFN <- "Daten/ncvoter-20140619-temporal-balanced-ratio-1to1-b.csv"
21 telCDAFN <- "Daten/A_10000_python_R0_C0_0100_percent.csv"
22 telCDBFN <- "Daten/B_10000_python_R20_C50_0100_percent.csv"
23 FebrlAFN <- "Daten/FEBRL_10000_20_A.csv"
24 FebrlBFN <- "Daten/FEBRL_10000_20_B.csv"
25
```

```
26 ##### Loop vars #####
27 # Set data sets for loop
28 datasets <- c("German Mortality", "German Telephone CD 20% errors", "German Telephone CD
↳ 0% errors", "FEBRL")
29 # Set to standard identifier set for CLK
30 v <- "Standard CLK"
31 # Amount of hash functions to loop over
32 K <- seq(from = 1, to = 40, by = 1) # Amount of Hash functions
33 # q-grams to loop over
34 Q <- c(1,2,3,4)
35 # Length of the BF
36 l <- 500
37 # Size of NC Voter data subsample
38 nNCV <- 50000
39 # No loopable errors in data
40 fehler <- c(0)
41 # Set padding to false
42 padding <- FALSE
43 # Thresholds to loop over
44 tanimotoThresholds <- seq(from= 1.0, to = 0.8, by = -0.01)
45 # Set cores and leaf limit
46 cores <- 7
47 leaflimit <- 3
48
49
50 # Init Counter to count progress
51 counter <- 0
52
53 # Calc number of combinations in total
54 numCombinations <- length(tanimotoThresholds) * length(fehler) *
55   length(datasets) * length(K) * length(Q)
56
57
58 # Init result matrix
59 meta <- NULL
60
61 cat("\n\n\nClassifications start!", format(Sys.time(), "(%a, %d-%m-%Y, %H:%M:%S)"), "\n")
62
63
64 ##### Function definitions #####
65
```

```

66 ##### Standardization #####
67 unwanted_array = list('Š'='S', 'š'='s', 'Ž'='Z', 'ž'='z', 'À'='A', 'Á'='A', 'Â'='A',
68   ↪ 'Ã'='A', 'Ä'='Ae', 'Å'='A', 'Æ'='A', 'Ç'='C', 'Ð'='D', 'È'='E', 'É'='E',
69   ↪ 'Ê'='E', 'Ë'='E', 'Ì'='I', 'Í'='I', 'Î'='I', 'Ï'='I', 'Ñ'='N',
70   ↪ 'Ö'='Oe', 'Ò'='O', 'Ó'='O', 'Ô'='O', 'Õ'='O', 'Ö'='O', 'Ø'='O',
71   ↪ 'Ù'='U',
72   ↪ 'Ú'='U', 'Û'='Ue', 'Ý'='Y', 'Þ'='B', 'ß'='Ss', 'à'='a',
73   ↪ 'á'='a', 'â'='a', 'ã'='a', 'ä'='ae', 'å'='a', 'æ'='a', 'ç'='c',
74   ↪ 'è'='e', 'é'='e', 'ê'='e', 'ë'='e', 'ì'='i', 'í'='i', 'î'='i',
75   ↪ 'ï'='i', 'ð'='o', 'ñ'='n', 'ò'='o', 'ó'='o', 'ô'='o', 'õ'='o',
76   ↪ 'ö'='oe', 'ø'='o', 'ü'='ue', 'ù'='u', 'ú'='u', 'û'='u', 'ý'='y',
77   ↪ 'ÿ'='y', 'þ'='b', 'ÿ'='y' )
78
79 normalize_string=function(string){
80   string = enc2utf8(string)
81   string=gsub("[^$*+\\.?[^{|(\\#%&~_<=>!,:;\\\"')}@]", "",string)
82   string=gsub("-", "",string)
83   string =chartr(paste(names(unwanted_array), collapse=''),
84     ↪ paste(unwanted_array, collapse=''),
85     ↪ string)
86   string=gsub("[\t\n\r\f\v]", "",string)
87   string=gsub("[0-9]", "",string)
88   string=toupper(string)
89   string=iconv(string, "utf8", "ASCII", sub="")
90   return(string)
91 }
92
93 normalize_dobs=function(string){
94   if(is.na(string)){
95     ↪ return("")
96   } else {
97     ↪ string = as.character(string)
98     ↪ if (nchar(string) < 2){
99     ↪   ↪ string = paste0("0",string)
100     ↪ }
101     ↪ return(string)
102   }
103 }

```

```
101
102 ##### Loop over datasets #####
103
104 for (d in datasets){
105
106
107   ##### Preprocess depending on data #####
108   if (d == "NC Voter"){
109
110     # Read
111     clearTextB <- fread(ncvoterBFN, sep=",")
112     clearTextA <- fread(ncvoterAFN, sep=",")
113
114     # Subset identifiers
115     clearTextA <- clearTextA[,c(1,2,4,5)]
116     clearTextB <- clearTextB[,c(1,2,4,5)]
117
118     # Draw sample
119     set.seed(69)
120     clearTextA <- clearTextA[sample(1:nrow(clearTextA), nNCV, replace=FALSE),]
121     set.seed(69)
122     clearTextB <- clearTextB[sample(1:nrow(clearTextB), nNCV, replace=FALSE),]
123
124     clearTextA$Day <- ""
125     clearTextA$Month <- ""
126     clearTextA$Year <- as.character(2014-clearTextA$age)
127
128     clearTextB$Day <- ""
129     clearTextB$Month <- ""
130     clearTextB$Year <- as.character(2014-clearTextB$age)
131     clearTextA$age <- NULL
132     clearTextB$age <- NULL
133
134     ### names anpassen für alle datasets
135
136   } else if (d == "German Telephone CD 20% errors"){
137
138     clearTextB <- fread(telCDBFN, sep="\t")
139     clearTextA <- fread(telCDAFN, sep="\t")
140
141     # Reorder
```

```

142 clearTextA <- clearTextA[,c(1,7,8,5,4,3)]
143 clearTextB <- clearTextB[,c(1,7,8,5,4,3)]
144
145
146 } else if (d == "German Telephone CD 0% errors"){
147
148 clearTextB <- fread(telCDAFN, sep="\t")
149 clearTextA <- fread(telCDAFN, sep="\t")
150
151 # Reorder
152 clearTextA <- clearTextA[,c(1,7,8,5,4,3)]
153 clearTextB <- clearTextB[,c(1,7,8,5,4,3)]
154
155
156 } else if (d == "German Mortality"){
157 clearText <- fread(ewodataFN, sep=",")
158
159 clearTextB <- subset(clearText, quelle=="schufa")
160 clearTextA <- subset(clearText, quelle=="einwo")
161 clearTextA <- clearTextA[,c(14,2,3,5,6,7)]
162 clearTextB <- clearTextB[,c(14,2,3,5,6,7)]
163
164 } else {
165 clearTextB <- fread(Febr1BFN, sep="\t")
166 clearTextA <- fread(Febr1AFN, sep="\t")
167
168 clearTextA <- clearTextA[,c(1,2,3,10)]
169 clearTextB <- clearTextB[,c(1,2,3,10)]
170 clearTextA$Day <- (substr(clearTextA$V10,7,8))
171 clearTextA$Month <- (substr(clearTextA$V10,5,6))
172 clearTextA$Year <- (substr(clearTextA$V10,1,4))
173 clearTextA$V10 <- NULL
174
175 clearTextB$Day <- (substr(clearTextB$V10,7,8))
176 clearTextB$Month <- (substr(clearTextB$V10,5,6))
177 clearTextB$Year <- (substr(clearTextB$V10,1,4))
178 clearTextB$V10 <- NULL
179
180 }
181
182 # Read error-free file

```

```
183 names(clearTextA) <- c("ID", "Vorname", "Nachname", "Day", "Month", "Year")
184 names(clearTextB) <- c("ID", "Vorname", "Nachname", "Day", "Month", "Year")
185
186
187
188 ##### General preproc #####
189 clearTextA$ID <- as.character(clearTextA$ID)
190 clearTextB$ID <- as.character(clearTextB$ID)
191
192
193 # Preprocess A
194
195 ## Standardize data
196 clearTextA$Vorname <- normalize_string(clearTextA$Vorname)
197 clearTextA$Nachname <- normalize_string(clearTextA$Nachname)
198
199 clearTextA$Day <- sapply(clearTextA$Day, normalize_dobs)
200 clearTextA$Month <- sapply(clearTextA$Month, normalize_dobs)
201 clearTextA$Year <- as.character(ifelse(is.na(clearTextA$Year), "", clearTextA$Year))
202
203 # Generate full DOB
204 clearTextA$DOB <- paste(clearTextA$Day,clearTextA$Month,clearTextA$Year, sep="")
205
206
207 # Preprocess B
208
209 ## Standardize data
210 clearTextB$Vorname <- normalize_string(clearTextB$Vorname)
211 clearTextB$Nachname <- normalize_string(clearTextB$Nachname)
212
213 clearTextB$Day <- sapply(clearTextB$Day, normalize_dobs)
214 clearTextB$Month <- sapply(clearTextB$Month, normalize_dobs)
215 clearTextB$Year <- as.character(ifelse(is.na(clearTextB$Year), "", clearTextB$Year))
216
217 # Generate full DOB for salting
218 clearTextB$DOB <- paste(clearTextB$Day,clearTextB$Month,clearTextB$Year, sep="")
219
220
221 ## Deduplicate by ID
222 clearTextA <- clearTextA[!duplicated(clearTextA$ID),]
223 clearTextB <- clearTextB[!duplicated(clearTextB$ID),]
```

```

224
225
226 ## File sizes, overlap
227 filesizeA <- (nrow(clearTextA))
228 filesizeB <- (nrow(clearTextB))
229
230 pairs <- as.numeric(filesizeA) * as.numeric(filesizeB)
231 truepositives <- sum(clearTextA$ID %in% clearTextB$ID)
232 positives <- truepositives
233
234
235 cat("\n\n\nClear-Text files ready, analyzing files", format(Sys.time(), "(%a, %d-%m-%Y,
↳ %H:%M:%S)"), "\n")
236
237 ## Loop over combos ##
238 for (q in Q){
239   for (k in K){
240
241     # Placeholder measures
242     ngrams <- q * k
243     meanentropy <- 1
244     meanengrams <- 1
245     entropyIdentifier <- 1
246     skewness <- 1
247     ginicoefficient <- 1
248     errorestimation <- 1
249     missingamount <- 1
250     m <- 1
251     u <- 1
252
253     cat("\n\n\nEncrypting...", format(Sys.time(), "(%a, %d-%m-%Y, %H:%M:%S)"), "\n")
254
255     # Encrypt
256     encryptedA <- CreateCLK(ID = clearTextA$ID, clearTextA[, c(2,3,7)], k = k, padding =
↳ rep(0,3), q = c(q,q,q), l = 1, password = c("(H]$6Uh*-Z204q", "asd", "afsd"))
257     encryptedB <- CreateCLK(ID = clearTextB$ID, clearTextB[, c(2,3,7)], k = k, padding =
↳ rep(0,3), q = c(q,q,q), l = 1, password = c("(H]$6Uh*-Z204q", "asd", "afsd"))
258
259     # Calculate hamming weight and no of unique patterns
260     hammingweight <- mean(c(nchar(gsub("0", "", encryptedA$CLKs)),nchar(gsub("0", "",
↳ encryptedB$CLKs))))

```

```

261 uniquepatternsA <- length(unique(encryptedA$CLKs))
262 uniquepatternsB <- length(unique(encryptedB$CLKs))
263
264 # Tree data
265 write.table(encryptedA, "TreeA.csv", sep="\t", row.names=FALSE, quote=FALSE,
  ↪ col.names=FALSE)
266 write.table(encryptedB, "TreeB.csv", sep="\t", row.names=FALSE, quote=FALSE,
  ↪ col.names=FALSE)
267
268 ## Loop over thresholds
269 for (Tani in tanimotoThresholds){
270
271   # Run MBT
272   multibitTree.load("TreeB.csv",threads=cores, leafLimit = leaflimit)
273   result <- multibitTree.searchFile("TreeA.csv", Tani)
274
275
276   # Classify candidate pairs
277   tp <- sum((result$fingerprint) == (result$query))
278   fp <- sum((result$fingerprint) != (result$query))
279   fn <- positives - tp
280   tn <- pairs - (fn + tp + fp)
281
282   # Calculate central measures
283   ReductionRate <- 1 - ((tp+fp)/(tn+fn))
284   precision <- tp / (tp+fp)
285   recall <- tp / (tp+fn)
286   Fmeasure = (recall + precision) / 2
287
288   # Set counter plus one
289   counter <- counter + 1
290
291   cat("\nClassification", counter, "of", numCombinations,"Finished (",
  ↪ format((counter / numCombinations)*100, digits=3),"%)", format(Sys.time()),"
  ↪ (%a, %d-%m-%Y, %H:%M:%S)",": \nFilesizes:\t",
  ↪ format(filesizeA,scientific=FALSE), "\nData source:\t", d, "\nMethod:\t\t", v,
  ↪ "\nk/q:\t\t", k, "/", q, "\nTanimoto:\t", Tani, "\n\nPositives:\t",
  ↪ truepositives, "\nTrue Positives:\t", tp,"\nRecall:\t\t", recall, "\nFalse
  ↪ Positives:", fp, "\nPrecision:\t", precision, "\nFalse Negatives:", fn,
  ↪ "\nF-Score:\t", Fmeasure, "\n\n")
292

```

```

293     # Append to result
294     meta <- rbind(meta, cbind(d, v, q, k, l, Tani, filesizeA,filesizeB,positives, tp,
    ↪ fp, fn, tn, recall, precision,Fmeasure, ngrams, meanentropy, meanengrams,
    ↪ entropyIdentifier, skewness, ginicoefficient, errorestimation ,
    ↪ missingamount,hammingweight,uniquepatternsA,uniquepatternsB,m,u))
295     write.table(meta, "tempResult.csv", sep="\t", row.names=FALSE, quote=FALSE,
    ↪ col.names=TRUE)
296
297     }
298   }
299 }
300 }
301
302
303 # format result
304 result <- as.data.frame(meta)
305 colnames(result) <- c("data", "variant", "q", "k", "l", "Tani", "filesizeA", "filesizeB",
    ↪ "positives", "tp", "fp", "fn", "tn", "recall", "precision", "Fmeasure", "ngrams",
    ↪ "meanentropy", "meanengrams", "entropyIdentifier", "skewness", "ginicoefficient",
    ↪ "errorestimation", "missingamount", "hammingweight", "uniquepatternsA",
    ↪ "uniquepatternsB", "m", "u")
306
307 # Write final result file
308 write.table(result, "fullResult.csv", sep="\t", row.names=FALSE, quote=FALSE,
    ↪ col.names=TRUE)
309
310 rm(list=ls())
311 gc()
312
313
314 ##### Merging metadata to result data #####
315
316 # Read result file
317 result <- read.csv("fullResult.csv", sep="\t")
318
319
320 result2 <- read.csv(paste0("fullResult_Metadata.csv"), sep="\t")
321 names(result2)[1] <- "data"
322
323 result <- result[,!names(result) %in% names(result2)[7:22]]
324 result$entropyIdentifier <- NULL

```

```

325 result$skewness <- NULL
326 result$ngrams <- NULL
327 result2$l <- NULL
328 result2$filesizeA <- NULL
329 result2$filesizeB <- NULL
330 result2$positives <- NULL
331 result <- subset(result, Tani >= 0.7)
332
333
334 # Merge metadata with simulation results
335 plotdata <- merge(result, result2, by=c("data", "q"), all.x=TRUE)
336 rm(result, result2)
337
338 write.table(plotdata, "Complete_Results.csv", sep="\t", row.names=FALSE, quote=FALSE,
  ↪ col.names=TRUE)

```

Source Code C.2: Evaluating evaluation data set FEBRL WA for testing OPC.

```

1 #####
2 #
3 # Estimating optimal parameter choices for the FEBRL WA data
4 #
5 #####
6
7
8 #### User settings ####
9
10 # Set working dir for script
11 #setwd("E:/Priv/Dropbox/Dissertation/Programme")
12 # File to run the optimization over
13 inputFile <- "Daten/Adrian_Perfect_100k.csv"
14 # Identifiers
15 idents <- c("Given Name", "Family Name", "Date Of Birth")
16 # ID-column. If none, set to FALSE (WARNING: This will be detrimental to the model
  ↪ quality!)
17 IDcol <- "Group Id"
18 # String-Distance columns. For EM weighting
19 Stringidents <- c("Given Name", "Family Name")
20
21 # Desired BF length

```

```
22 l <- 500
23 # Type: BF or CLK
24 type <- "CLK"
25 # Bootstrap repetitions
26 bootstraps <- 1
27 # Desired Tanimoto threshold
28 tanimotoGoal <- 0.9
29 # Cores of the machine for multithreading
30 cores <- 7
31 # Mode: "Train" for using a subset of the original data set as training data, "Link" if
  ↪ both files are available
32 mode <- "Link"
33 # If mode "Link": relative size of the linking subsample (for large files)
34 linkSample <- 1
35 # If mode "Train": relative size of the training subsample
36 trainSample <- 0.2
37 # If mode "Link": file name of file to link to
38 inputFileB <- "Daten/Adrian_10percent_100k.csv"
39
40 #####
41
42 # Libs used
43 library(PURL)
44 library(fastdigest)
45 library(stringdist)
46 library(entropy)
47 library(data.table)
48 library(fastLink)
49 library(moments)
50 library(ineq)
51
52 ##### Static vars, change only if you know what you are doing #####
53
54 # Training data BF length
55 lTraining <- 500
56 # Training data file
57 trainFN <- "Complete_Results.csv"
58 # q-grams to test
59 Q <- c(1,2,3,4)
60 # Hash function test space
61 K <- seq(from = 1, to = 40, by = 1)
```

```

62 # MBT leaf limit
63 leaflimit <- 3
64 # Calc number of combinations in total
65 numCombinations <- length(K) * length(Q) * bootstraps
66 # Set counter for progress report
67 counter <- 0
68 # Result vector initialization
69 meta <- NULL
70
71 #####
72
73
74 # Read training file for model building
75 traindata <- read.csv(trainFN, sep="\t")
76
77 # Subset to fixed tanimoto threshold
78 fixed <- subset(traindata, Tani == tanimotoGoal)
79
80 # Fixed model calls
81
82 BIC <- glm(Fmeasure ~ k + q + errorestimation + skew + meanentropy +
83   log(u) + ginicoefficient + poly(k * q * meanengrams * hammingweight,
84   2) + meanmissing + uniquepatternsA,
85   data =fixed)
86 LM <- lm(
87   Fmeasure ~ k + q +
88     errorestimation*skew + uniqueness + ginicoefficient +
89     poly(k*q*meanengrams*hammingweight, 2) +
90     uniquepatternsA + m, data=fixed)
91
92 #### Work with data
93
94 # Read input
95 clearTextA <- fread(inputFile, colClasses = "character", stringsAsFactors = FALSE)
96
97 # If second file, read it. Otherwise, sample
98 if (mode == "Link"){
99   clearTextB <- fread(inputFileB, colClasses = "character", stringsAsFactors = FALSE)
100
101   clearTextA <- clearTextA[sample(nrow(clearTextA),round(nrow(clearTextA)*linkSample)),]
102

```

```

103 clearTextB <- clearTextB[sample(nrow(clearTextB),round(nrow(clearTextB)*linkSample)),]
104
105 } else {
106   clearTextB <- clearTextA[sample(nrow(clearTextA),round(nrow(clearTextA)*trainSample)),]
107 }
108
109 ## Deduplicate by ID
110 clearTextA <- clearTextA[!duplicated(clearTextA$`Group Id`),]
111 clearTextB <- clearTextB[!duplicated(clearTextB$`Group Id`),]
112
113
114
115 # Check file sizes
116 filesizeA <- nrow(clearTextA)
117 filesizeB <- nrow(clearTextB)
118 # Number of pairs
119 pairs <- as.numeric(filesizeA) * as.numeric(filesizeB)
120
121 # Pairs and positives
122 if (IDcol != FALSE){
123   # Calc true matches
124   truepositives <- sum(unlist(clearTextA[,..IDcol]) %in% unlist(clearTextB[,..IDcol]))
125
126   # Generate ID Col
127   IDA <- as.character(unlist(clearTextA[,..IDcol]))
128   IDB <- as.character(unlist(clearTextB[,..IDcol]))
129
130   # subset the data
131   clearTextA <- clearTextA[, ..idents]
132   clearTextB <- clearTextB[, ..idents]
133 } else {
134
135   # Generate crude TP estimate by using keys
136   clearTextA$key <- apply(clearTextA[,..idents],1,paste0, collapse="")
137   clearTextB$key = apply(clearTextB[,..idents],1,paste0, collapse="")
138
139   #Estimate TP
140   truepositives <- sum(clearTextA$key %in% clearTextB$key)
141
142   # Generate ID Col by generating same IDs for exact matches, and SHA1 random IDs for all
   ↪ other pairs

```

```

143 clearTextA$IDA <- ""
144 set.seed(42)
145 clearTextA[which(clearTextA$key %in% clearTextB$key), "IDA"] <-
  ↳ replicate(truepositives, fastdigest(sample(LETTERS, 1000, replace = TRUE)))
146 set.seed(42)
147 clearTextB$IDB <- ""
148 clearTextB[which(clearTextB$key %in% clearTextA$key), "IDB"] <-
  ↳ replicate(length(clearTextB$key %in% clearTextA$key), fastdigest(sample(LETTERS,
  ↳ 1000, replace = TRUE)))
149 clearTextA$IDA[clearTextA$IDA == ""] <-
  ↳ sha1(as.character(1:length(clearTextA$IDA[clearTextA$IDA == ""])), key = "A")
150 clearTextB$IDB[clearTextB$IDB == ""] <-
  ↳ sha1(as.character(1:length(clearTextA$IDA[clearTextA$IDA == ""])), key = "B")
151
152 # Generate ID Col
153 IDA <- as.character(clearTextA$IDA)
154 IDB <- as.character(clearTextB$IDB)
155
156 # subset the data
157 clearTextA <- clearTextA[, ..idents]
158 clearTextB <- clearTextB[, ..idents]
159 }
160
161
162 ##### Start working #####
163
164 # Get EM weights
165 ests <- fastLink(clearTextA, clearTextB, varnames = idents,
166                 stringdist.match=Stringidents, stringdist.method="jw", n.cores = cores,
167                 ↳ estimate.only = TRUE)
168
169 save(ests, file="EM_estimates.Rdata") # In case of crash
170 load("EM_estimates.Rdata")
171
172 # Weights
173 m <- ests$p.m
174 u <- ests$p.u
175
176 # Loop over parameter space
177 for (q in Q){

```

```

178
179   cat("\n\n\nLine-by-line checking...", format(Sys.time(), " (%a, %d-%m-%Y,
    ↪   %H:%M:%S)"), "\n")
180
181
182   # Generate full linkage key
183   keys <- apply(clearTextA, 1, paste0, collapse = "")
184   keys <- gsub(" ", "", keys)
185
186   # Split into bigrams
187   qgrams <- (sapply(keys, function(key) substring(key, first=seq(1, nchar(key)), last=seq(q,
    ↪   nchar(key)+q))))
188   # Count uniques and save measures
189   uniquengrams <- as.numeric(unlist(lapply(qgrams, function(x) length(unique(x)))))
190   # ngram Count
191   ngramsC <- as.numeric(unlist(lapply(qgrams, function(x) length(x))))
192   qgrams <- unlist(qgrams)
193   qgramlist <- as.character(na.omit(ifelse(qgrams == "" | nchar(qgrams) != q, NA, qgrams)))
194
195   cat("\n\n\nLine-by-line checking complete, calculating measures...",
    ↪   format(Sys.time(), " (%a, %d-%m-%Y, %H:%M:%S)"), "\n")
196
197   # Sum missings
198   missings <- sum(apply(clearTextA, 2, function(x) sum(is.na(x)))) +
    ↪   sum(apply(clearTextA, 2, function(x) sum(x[!is.na(x)]=="")))
199
200   # Mean entropy calculation
201   meanentropy <- mean(apply(clearTextA, 2, function(x) entropy(table(x))))
202
203   # q90/q10-ratio
204   q90 <- round(quantile(as.numeric(table(qgramlist)), probs=seq(0,1,0.1))["90%"])
205   q10 <- round(quantile(as.numeric(table(qgramlist)), probs=seq(0,1,0.1))["10%"])
206
207   # Measures
208   meanengrams <- mean(ngramsC)
209   sdgrams <- sd(ngramsC)
210
211   skew <- skewness(table(qgramlist))
212   ginicoefficientqgrams <- Gini(table(qgramlist))
213
214   ginicoefficient <- mean(apply(clearTextA, 2, function(x) Gini(table(x))))

```

```

215
216 errorestimation <- q90/q10 # amount of rare combinations in data
217 missingamount <- missings
218 meanmissing <- missings/nrow(clearTextA)
219 uniqueness <- mean(uniquengrams/ngramsC)
220 sduniqueness <- sd(uniquengrams/ngramsC)
221
222
223
224 for (k in K){
225
226   for (i in seq(1:bootstraps)){
227
228     cat("\n\n\nEncrypting...", format(Sys.time(), "(%a, %d-%m-%Y, %H:%M:%S)"), "\n")
229
230     ## Generate password vector for BF encoding, resample for boot-strapping
231     PWs <- replicate(length(identfs), fastdigest(sample(LETTERS, 10000, replace =
232       ↪ TRUE)))
233
234     encryptedA <- CreateCLK(ID = IDA, clearTextA[, ..identfs], k = k, padding =
235       ↪ rep(0,length(identfs)), q = rep(q,length(identfs)), l = l, password =
236       ↪ PWs)#,"13124","124124"))
237
238     encryptedB <- CreateCLK(ID = IDB, clearTextB[, ..identfs], k = k, padding =
239       ↪ rep(0,length(identfs)), q = rep(q,length(identfs)), l = l, password =
240       ↪ PWs)#,"13124","124124"))
241
242     # Calculate hamming weight and NO of unique patterns
243     hammingweight <- mean(c(nchar(gsub("0", "", encryptedA$CLKs)), nchar(gsub("0", "",
244       ↪ encryptedB$CLKs))))
245     uniquepatternsA <- length(unique(encryptedA$CLKs))
246     uniquepatternsB <- length(unique(encryptedB$CLKs))
247     #
248     # # Tree data
249     write.table(encryptedA, "TreeA_OPC.csv", sep="\t", row.names=FALSE, quote=FALSE,
250       ↪ col.names=FALSE)
251     write.table(encryptedB, "TreeB_OPC.csv", sep="\t", row.names=FALSE, quote=FALSE,
252       ↪ col.names=FALSE)
253     #
254     # Run MBT

```

```

248 multibitTree.load("TreeB_OPC.csv",threads=cores, leafLimit = leaflimit)
249 result <- multibitTree.searchFile("TreeA_OPC.csv", Tani)
250 #
251 #
252 # # Classify candidate pairs
253 tp <- sum((result$fingerprint) == (result$query))
254 fp <- sum((result$fingerprint) != (result$query))
255 fn <- truepositives - tp
256 tn <- pairs - (fn + tp + fp)
257 #
258 # # Calculate central measures
259 ReductionRate <- 1 - ((tp+fp)/(tn+fn))
260 precision <- tp / (tp+fp)
261 recall <- tp / (tp+fn)
262 Fmeasure = (recall + precision) / 2
263
264 #tp <- fp <- fn <- tn <- precision <- recall <- Fmeasure <- 0
265
266 # Save RAM
267 gc()
268
269 # Set counter plus one
270 counter <- counter + 1
271
272
273 cat("\nClassification", counter, "of", numCombinations,"Finished
  ↳ (",format((counter / numCombinations)*100, digits=3),"%)!",
  ↳ format(Sys.time()," (%a, %d-%m-%Y, %H:%M:%S)"), ": \nFilesizes:\t",
  ↳ format(filesizeA,scientific=FALSE),"\nData source and mode:\t" , inputFile, "
  ↳ ", mode,"\nBootstrap:\t\t", i, "/", bootstraps, "\nk/q:\t\t", k, "/",q
  ↳ ,"\nTanimoto:\t", tanimotoGoal, "\n\nPositives:\t", truepositives, "\nTrue
  ↳ Positives:\t", tp, "\nRecall:\t\t", recall, "\nFalse Positives:", fp,
  ↳ "\nPrecision:\t", precision, "\nFalse Negatives:", fn, "\nF-Score:\t",
  ↳ Fmeasure, "\n\n")
274
275 # Append to result
276 meta <- rbind(meta, data.frame(data="FEBRL WA", type, q, k, i, l,
  ↳ Tani=tanimotoGoal,Fmeasure, filesizeA, filesizeB, meanentropy, meanengrams,
  ↳ sdgrams, skew, ginicoefficient, ginicoefficientqgrams, errorestimation ,
  ↳ hammingweight, uniqueness, sduniqueness, meanmissing, missingamount,
  ↳ uniquepatternsA,uniquepatternsB,m,u))

```

```
277     write.table(meta, "tempResult.csv", sep="\t", row.names=FALSE, quote=FALSE,
278               ↪ col.names=TRUE)
279   }
280 }
281
282
283 # Prepare resulting data to predict from it
284 newdata <- data.frame(meta, stringsAsFactors = FALSE, row.names = NULL)
285 names(newdata)[names(newdata) == "tanimotoGoal"] <- "Tani"
286 write.table(newdata, "Trainresults.csv", sep="\t", row.names=FALSE, quote=FALSE,
287             ↪ col.names=TRUE)
288
289 # Predict
290 newfitted <- data.frame(newdata, predicted = predict(LM, newdata, type = "response"))
291 newfittedBIC <- data.frame(newdata, predicted = predict(BIC, newdata, type = "response"))
292
293 # Get optimal parameter estimate
294 parameters <-
295   ↪ newfitted[which.max((newfitted$predicted)),c("Tani", "q", "k", "l", "predicted")]
296
297 parameters2 <-
298   ↪ newfittedBIC[which.max((newfittedBIC$predicted)),c("Tani", "q", "k", "l", "predicted")]
299
300 # Write out parameters for use in loop
301 write.table(parameters, "Optimal_Parameters.csv", sep="\t", row.names = FALSE)
302 write.table(parameters2, "Optimal_Parameters_BIC.csv", sep="\t", row.names = FALSE)
303
304 cat("\n All done!")
```

Source Code C.3: Evaluating evaluation data set NC Voter for testing OPC.

```
1 #####
2 #
3 # Estimating optimal parameter choices for NC Voter data
4 #
5 #####
6
7
8 ##### User settings #####
9 # Set working dir for script
10 #setwd("E:/Priv/Dropbox/Dissertation/Programme")
11 # File to run the optimization over
12 inputFile <- "Daten/ncvoter-20140619-temporal-balanced-ratio-1to1-a.csv"
13 # Identifiers
14 idents <- c("first_name","last_name","Year")
15 # ID-column. If none, set to FALSE (WARNING: This will be detrimental to the model
16   ↪ quality!)
17 IDcol <- "voter_id"
18 # String-Distance columns. For EM weighting
19 Stringidents <- c("first_name","last_name")
20
21 # Desired BF length
22 l <- 500
23 # Type: BF or CLK
24 type <- "CLK"
25 # Bootstrap repetitions
26 bootstraps <- 1
27 # Desired Tanimoto threshold
28 tanimotoGoal <- 0.9
29 # Cores of the machine for multithreading
30 cores <- 7
31 # Mode: "Train" for using a subset of the original data set as training data, "Link" if
32   ↪ both files are available
33 mode <- "Link"
34 # If mode "Link": relative size of the linking subsample (for large files)
35 linkSample <- 1
36 # If mode "Train": relative size of the training subsample
37 trainSample <- 1
38 # If mode "Link": file name of file to link to
```

```
37 inputFileB <- "Daten/ncvoter-20140619-temporal-balanced-ratio-1to1-b.csv"
38
39 #####
40
41 # Libs used
42 library(PPRL)
43 library(fastdigest)
44 library(stringdist)
45 library(entropy)
46 library(data.table)
47 library(fastLink)
48 library(moments)
49 library(ineq)
50
51 ##### Static vars, change only if you know what you are doing #####
52
53 # Training data BF length
54 lTraining <- 500
55 # Training data file
56 trainFN <- "Complete_Results.csv"
57 # q-grams to test
58 Q <- c(1,2,3,4)
59 # Hash function test space
60 K <- seq(from = 1, to = 40, by = 1)
61 # MBT leaf limit
62 leaflimit <- 3
63 # Calc number of combinations in total
64 numCombinations <- length(K) * length(Q) * bootstraps
65 # Set counter for progress report
66 counter <- 0
67 # Result vector initialization
68 meta <- NULL
69
70 #####
71
72
73 # Read training file for model building
74 traindata <- read.csv(trainFN, sep="\t")
75
76 # Subset to fixed tanimoto threshold
77 fixed <- subset(traindata, Tani == tanimotoGoal)
```

```

78
79 # Fixed model calls
80
81 BIC <- glm(Fmeasure ~ k + q + errorestimation + skew + meanentropy +
82   log(u) + ginicoefficient + poly(k * q * meanengrams * hammingweight,
83   2) + meanmissing + uniquepatternsA,
84   data =fixed)
85 LM <- lm(
86   Fmeasure ~ k + q +
87     errorestimation*skew + uniqueness + ginicoefficient +
88     poly(k*q*meanengrams*hammingweight, 2) +
89     uniquepatternsA + m, data=fixed)
90
91 ##### Work with data
92
93 # Read input
94 clearTextA <- fread(inputFile, colClasses = "character", stringsAsFactors = FALSE)
95
96 # If second file, read it. Otherwise, sample
97 if (mode == "Link"){
98   clearTextB <- fread(inputFileB, colClasses = "character", stringsAsFactors = FALSE)
99
100  # Subset overlapping records
101  clearTextA <- clearTextA[(clearTextA$voter_id %in% clearTextB$voter_id),]
102  clearTextB <- clearTextB[(clearTextB$voter_id %in% clearTextA$voter_id),]
103
104  # Deduplicate
105  clearTextA <- clearTextA[!duplicated(clearTextA$voter_id),]
106  clearTextB <- clearTextB[!duplicated(clearTextB$voter_id),]
107
108  # Draw sample
109  set.seed(42)
110  clearTextA <- clearTextA[sample(1:nrow(clearTextA), 50000, replace=FALSE),]
111  set.seed(42)
112  clearTextB <- clearTextB[sample(1:nrow(clearTextB), 50000, replace=FALSE),]
113
114
115  clearTextA$Day <- ""
116  clearTextA$Month <- ""
117  clearTextA$Year <- as.character(2014-as.numeric(clearTextA$age))
118

```

```

119 clearTextB$Day <- ""
120 clearTextB$Month <- ""
121 clearTextB$Year <- as.character(2014-as.numeric(clearTextB$age))
122 clearTextA$age <- NULL
123 clearTextB$age <- NULL
124
125 } else {
126   clearTextB <- clearTextA[sample(nrow(clearTextA),round(nrow(clearTextA)*trainSample)),]
127 }
128
129 ## Deduplicate by ID
130 clearTextA <- clearTextA[!duplicated(clearTextA[,..IDcol]),]
131 clearTextB <- clearTextB[!duplicated(clearTextB[,..IDcol]),]
132
133
134
135 # Check file sizes
136 filesizeA <- nrow(clearTextA)
137 filesizeB <- nrow(clearTextB)
138 # Number of pairs
139 pairs <- as.numeric(filesizeA) * as.numeric(filesizeB)
140
141 # Pairs and positives
142 if (IDcol != FALSE){
143   # Calc true matches
144   truepositives <- sum(unlist(clearTextA[,..IDcol]) %in% unlist(clearTextB[,..IDcol]))
145
146   # Generate ID Col
147   IDA <- as.character(unlist(clearTextA[,..IDcol]))
148   IDB <- as.character(unlist(clearTextB[,..IDcol]))
149
150   # subset the data
151   clearTextA <- clearTextA[, ..idents]
152   clearTextB <- clearTextB[, ..idents]
153 } else {
154
155   # Generate crude TP estimate by using keys
156   clearTextA$key <- apply(clearTextA[,..idents],1,paste0, collapse="")
157   clearTextB$key = apply(clearTextB[,..idents],1,paste0, collapse="")
158
159   #Estimate TP

```

```

160 truepositives <- sum(clearTextA$key %in% clearTextB$key)
161
162 # Generate ID Col by generating same IDs for exact matches, and SHA1 random IDs for all
  ↪ other pairs
163 clearTextA$IDA <- ""
164 set.seed(42)
165 clearTextA[which(clearTextA$key %in% clearTextB$key), "IDA"] <-
  ↪ replicate(truepositives, fastdigest(sample(LETTERS, 1000, replace = TRUE)))
166 set.seed(42)
167 clearTextB$IDB <- ""
168 clearTextB[which(clearTextB$key %in% clearTextA$key), "IDB"] <-
  ↪ replicate(length(clearTextB$key %in% clearTextA$key), fastdigest(sample(LETTERS,
  ↪ 1000, replace = TRUE)))
169 clearTextA$IDA[clearTextA$IDA == ""] <-
  ↪ sha1(as.character(1:length(clearTextA$IDA[clearTextA$IDA == ""])), key = "A")
170 clearTextB$IDB[clearTextB$IDB == ""] <-
  ↪ sha1(as.character(1:length(clearTextA$IDA[clearTextA$IDA == ""])), key = "B")
171
172 # Generate ID Col
173 IDA <- as.character(clearTextA$IDA)
174 IDB <- as.character(clearTextB$IDB)
175
176 # subset the data
177 clearTextA <- clearTextA[, ..idents]
178 clearTextB <- clearTextB[, ..idents]
179 }
180
181
182 ##### Start working #####
183
184 # Get EM weights
185 ests <- fastLink(clearTextA, clearTextB, varnames = idents,
186                 stringdist.match=Stringidents, stringdist.method="jw", n.cores = cores,
  ↪ estimate.only = TRUE)
187
188
189 save(ests, file="EM_estimates.Rdata") # In case of crash
190 load("EM_estimates.Rdata")
191
192 # Weights
193 m <- ests$p.m

```

```

194 u <- ests$p.u
195
196 # Loop over parameter space
197 for (q in Q){
198
199   cat("\n\n\nLine-by-line checking...", format(Sys.time()), " (%a, %d-%m-%Y,
    ↪ %H:%M:%S)", "\n")
200
201
202   # Generate full linkage key
203   keys <- apply(clearTextA, 1, paste0, collapse = "")
204   keys <- gsub(" ", "", keys)
205
206   # Split into bigrams
207   qgrams <- (sapply(keys, function(key) substring(key, first=seq(1, nchar(key)), last=seq(q,
    ↪ nchar(key)+q))))
208   # Count uniques and save measures
209   unigramcounts <- as.numeric(unlist(lapply(qgrams, function(x) length(unique(x)))))
210   # ngram Count
211   ngramsC <- as.numeric(unlist(lapply(qgrams, function(x) length(x))))
212   qgrams <- unlist(qgrams)
213   qgramlist <- as.character(na.omit(ifelse(qgrams == "" | nchar(qgrams) != q, NA, qgrams)))
214
215   cat("\n\n\nLine-by-line checking complete, calculating measures...",
    ↪ format(Sys.time()), " (%a, %d-%m-%Y, %H:%M:%S)", "\n")
216
217   # Sum missings
218   missings <- sum(apply(clearTextA, 2, function(x) sum(is.na(x)))) +
    ↪ sum(apply(clearTextA, 2, function(x) sum(x[!is.na(x)]=="")))
219
220   # Mean entropy calculation
221   meanentropy <- mean(apply(clearTextA, 2, function(x) entropy(table(x))))
222
223   # q90/q10-ratio
224   q90 <- round(quantile(as.numeric(table(qgramlist)), probs=seq(0,1,0.1))["90%"])
225   q10 <- round(quantile(as.numeric(table(qgramlist)), probs=seq(0,1,0.1))["10%"])
226
227   # Measures
228   meanengrams <- mean(ngramsC)
229   sdgrams <- sd(ngramsC)
230

```

```

231 skew <- skewness(table(qgramlist))
232 ginicoefficientqgrams <- Gini(table(qgramlist))
233
234 ginicoefficient <- mean(apply(clearTextA, 2, function(x) Gini(table(x))))
235
236 errorestimation <- q90/q10 # amount of rare combinations in data
237 missingamount <- missings
238 meanmissing <- missings/nrow(clearTextA)
239 uniqueness <- mean(uniquengrams/ngramsC)
240 sduniqueness <- sd(uniquengrams/ngramsC)
241
242
243 for (k in K){
244
245   for (i in seq(1:bootstraps)){
246
247     cat("\n\n\nEncrypting...", format(Sys.time(), " (%a, %d-%m-%Y, %H:%M:%S)"), "\n")
248
249     ## Generate password vector for BF encoding, resample for boot-strapping
250     PWs <- replicate(length(identfs), fastdigest(sample(LETTERS, 10000, replace =
251       ↪ TRUE)))
252
253     encryptedA <- CreateCLK(ID = IDA, clearTextA[, ..identfs], k = k, padding =
254       ↪ rep(0,length(identfs)), q = rep(q,length(identfs)), l = l, password =
255       ↪ PWs)#, "13124", "124124")
256     encryptedB <- CreateCLK(ID = IDB, clearTextB[, ..identfs], k = k, padding =
257       ↪ rep(0,length(identfs)), q = rep(q,length(identfs)), l = l, password =
258       ↪ PWs)#, "13124", "124124")
259
260     # Calculate hamming weight and NO of unique patterns
261     hammingweight <- mean(c(nchar(gsub("0", "", encryptedA$CLKs)), nchar(gsub("0", "",
262       ↪ encryptedB$CLKs))))
263     uniquepatternsA <- length(unique(encryptedA$CLKs))
264     uniquepatternsB <- length(unique(encryptedB$CLKs))
265
266     #
267     # # Tree data
268     write.table(encryptedA, "TreeA_OPC.csv", sep="\t", row.names=FALSE, quote=FALSE,
269       ↪ col.names=FALSE)
270     write.table(encryptedB, "TreeB_OPC.csv", sep="\t", row.names=FALSE, quote=FALSE,
271       ↪ col.names=FALSE)

```

```

264
265 # Run MBT
266 multibitTree.load("TreeB_OPC.csv",threads=cores, leafLimit = leaflimit)
267 result <- multibitTree.searchFile("TreeA_OPC.csv", Tani)
268
269
270 # # Classify candidate pairs
271 tp <- sum((result$fingerprint) == (result$query))
272 fp <- sum((result$fingerprint) != (result$query))
273 fn <- truepositives - tp
274 tn <- pairs - (fn + tp + fp)
275 #
276 # # Calculate central measures
277 ReductionRate <- 1 - ((tp+fp)/(tn+fn))
278 precision <- tp / (tp+fp)
279 recall <- tp / (tp+fn)
280 Fmeasure = (recall + precision) / 2
281
282 # Save RAM
283 gc()
284
285 # Set counter plus one
286 counter <- counter + 1
287
288 cat("\nClassification", counter, "of", numCombinations,"Finished
  ↳ (",format((counter / numCombinations)*100, digits=3),"%)!",
  ↳ format(Sys.time()," (%a, %d-%m-%Y, %H:%M:%S)"), ": \nFilesizes:\t",
  ↳ format(filesizeA,scientific=FALSE),"\nData source and mode:\t" , inputFile, "
  ↳ ", mode,"\nBootstrap:\t\t", i, "/", bootstraps, "\nk/q:\t\t", k, "/",q
  ↳ ,"\nTanimoto:\t", tanimotoGoal, "\n\nPositives:\t", truepositives, "\nTrue
  ↳ Positives:\t", tp, "\nRecall:\t\t", recall, "\nFalse Positives:", fp,
  ↳ "\nPrecision:\t", precision, "\nFalse Negatives:", fn, "\nF-Score:\t",
  ↳ Fmeasure, "\n\n")
289
290 # Append to result
291 meta <- rbind(meta, data.frame(d="NC Voter", type, q, k, i, l,
  ↳ Tani=tanimotoGoal,Fmeasure, filesizeA, filesizeB, meanentropy, meanengrams,
  ↳ sdgrams, skew, ginicoefficient, ginicoefficientqgrams, errorestimation ,
  ↳ hammingweight, uniqueness, sduniqueness, meanmissing, missingamount,
  ↳ uniquepatternsA,uniquepatternsB,m,u))
292

```

```

293     write.table(meta, "tempResult.csv", sep="\t", row.names=FALSE, quote=FALSE,
                ↪ col.names=TRUE)
294   }
295 }
296 }
297
298
299 # Prepare resulting data to predict from it
300 newdata <- data.frame(meta, stringsAsFactors = FALSE, row.names = NULL)
301 names(newdata)[names(newdata) == "tanimotoGoal"] <- "Tani"
302 write.table(newdata, "Trainresults_NCVoter.csv", sep="\t", row.names=FALSE, quote=FALSE,
                ↪ col.names=TRUE)
303 newdata <- read.csv("Trainresults_NCVoter.csv", sep = "\t", stringsAsFactors = FALSE)
304
305 # Predict
306 newfitted <- data.frame(newdata, predicted = predict(LM, newdata, type = "response"))
307 newfittedBIC <- data.frame(newdata, predicted = predict(BIC, newdata, type = "response"))
308
309 # Get optimal prameter estimate
310 parameters <-
    ↪ newfitted[which.max((newfitted$predicted)),c("Tani", "q", "k", "l", "predicted")]
311 parameters2 <-
    ↪ newfittedBIC[which.max((newfittedBIC$predicted)),c("Tani", "q", "k", "l", "predicted")]
312
313 # Write out parameters for use in loop
314 write.table(parameters, "Optimal_Parameters.csv", sep="\t", row.names = FALSE)
315 write.table(parameters2, "Optimal_Parameters_BIC.csv", sep="\t", row.names = FALSE)
316
317 cat("\n All done!")

```

Source Code C.4: Evaluating evaluation data set Mortality/Commercial for testing OPC.

```

1 #####
2 #
3 # Estimating optimal parameter choices for Mortality data
4 #
5 #####
6
7
8 ##### User settings #####

```

```
9
10
11 # Set working dir for script
12 #setwd("E:/Priv/Dropbox/Dissertation/Programme")
13 # File to run the optimization over
14 inputFile <- "Daten/EWMA.csv"
15 # Identifiers
16 idents <- c("Vorname","Nachname","DOB")
17 # ID-column. If none, set to FALSE (WARNING: This will be detrimental to the model
  ↳ quality!)
18 IDcol <- "ID"
19 # String-Distance columns. For EM weighting
20 Stringidents <- c("Vorname","Nachname")
21
22 # Desired BF length
23 l <- 500
24 # Type: BF or CLK
25 type <- "CLK"
26 # Bootstrap repetitions
27 bootstraps <- 1
28 # Desired Tanimoto threshold
29 tanimotoGoal <- 0.9
30 # Cores of the machine for multithreading
31 cores <- 7
32 # Mode: "Train" for using a subset of the original data set as training data, "Link" if
  ↳ both files are available
33 mode <- "Link"
34 # If mode "Link": relative size of the linking subsample (for large files)
35 linkSample <- 1
36 # If mode "Train": relative size of the training subsample
37 trainSample <- 0.2
38 # If mode "Link": file name of file to link to
39 inputFileB <- "Daten/HOSPITAL.csv"
40
41
42
43 #####
44
45 # Libs used
46 library(PPRL)
47 library(fastdigest)
```

```
48 library(stringdist)
49 library(entropy)
50 library(data.table)
51 library(fastLink)
52 library(moments)
53 library(ineq)
54
55 ##### Static vars, change only if you know what you are doing #####
56
57 # Training data BF length
58 lTraining <- 500
59 # Training data file
60 trainFN <- "Complete_Results.csv"
61 # q-grams to test
62 Q <- c(1,2,3,4)
63 # Hash function test space
64 K <- seq(from = 1, to = 40, by = 1)
65 # MBT leaf limit
66 leaflimit <- 3
67 # Calc number of combinations in total
68 numCombinations <- length(K) * length(Q) * bootstraps
69 # Set counter for progress report
70 counter <- 0
71 # Result vector initialization
72 meta <- NULL
73
74 #####
75
76
77 # Read training file for model building
78 traindata <- read.csv(trainFN, sep="\t")
79
80 # Subset to fixed tanimoto threshold
81 fixed <- subset(traindata, Tani == tanimotoGoal)
82
83 # Fixed model calls
84
85 BIC <- glm(Fmeasure ~ k + q + errorestimation + skew + meanentropy +
86   log(u) + ginicoefficient + poly(k * q * meanengrams * hammingweight,
87     2) + meanmissing + uniquepatternsA,
88   data =fixed)
```

```

89 LM <- lm(
90   Fmeasure ~ k + q +
91     errorestimation*skew + uniqueness + ginicoefficient +
92     poly(k*q*meanengrams*hammingweight, 2) +
93     uniquepatternsA + m, data=fixed)
94
95 ##### Work with data
96
97 # Read input
98 clearTextA <- fread(inputFile, colClasses = "character", stringsAsFactors = FALSE)
99
100 # If second file, read it. Otherwise, sample
101 if (mode == "Link"){
102   clearTextB <- fread(inputFileB, colClasses = "character", stringsAsFactors = FALSE)
103
104   clearTextA <- clearTextA[sample(nrow(clearTextA),round(nrow(clearTextA)*linkSample)),]
105
106   clearTextB <- clearTextB[sample(nrow(clearTextB),round(nrow(clearTextB)*linkSample)),]
107
108 } else {
109   clearTextB <- clearTextA[sample(nrow(clearTextA),round(nrow(clearTextA)*trainSample)),]
110 }
111
112 ## Deduplicate by ID
113 clearTextA <- clearTextA[!duplicated(clearTextA[,..IDcol]),]
114 clearTextB <- clearTextB[!duplicated(clearTextB[,..IDcol]),]
115
116
117
118 # Check file sizes
119 filesizeA <- nrow(clearTextA)
120 filesizeB <- nrow(clearTextB)
121 # Number of pairs
122 pairs <- as.numeric(filesizeA) * as.numeric(filesizeB)
123
124 # Pairs and positives
125 if (IDcol != FALSE){
126   # Calc true matches
127   truepositives <- sum(unlist(clearTextA[,..IDcol]) %in% unlist(clearTextB[,..IDcol]))
128
129   # Generate ID Col

```

```

130 IDA <- as.character(unlist(clearTextA[,..IDcol]))
131 IDB <- as.character(unlist(clearTextB[,..IDcol]))
132
133 # subset the data
134 clearTextA <- clearTextA[, ..idents]
135 clearTextB <- clearTextB[, ..idents]
136 } else {
137
138 # Generate crude TP estimate by using keys
139 clearTextA$key <- apply(clearTextA[,..idents],1,paste0, collapse="")
140 clearTextB$key = apply(clearTextB[,..idents],1,paste0, collapse="")
141
142 #Estimate TP
143 truepositives <- sum(clearTextA$key %in% clearTextB$key)
144
145 # Generate ID Col by generating same IDs for exact matches, and SHA1 random IDs for all
  ↳ other pairs
146 clearTextA$IDA <- ""
147 set.seed(42)
148 clearTextA[which(clearTextA$key %in% clearTextB$key), "IDA"] <-
  ↳ replicate(truepositives,fastdigest(sample(LETTERS, 1000, replace = TRUE)))
149 set.seed(42)
150 clearTextB$IDB <- ""
151 clearTextB[which(clearTextB$key %in% clearTextA$key), "IDB"] <-
  ↳ replicate(length(clearTextB$key %in% clearTextA$key),fastdigest(sample(LETTERS,
  ↳ 1000, replace = TRUE)))
152 clearTextA$IDA[clearTextA$IDA == ""] <-
  ↳ sha1(as.character(1:length(clearTextA$IDA[clearTextA$IDA == ""] )), key = "A")
153 clearTextB$IDB[clearTextB$IDB == ""] <-
  ↳ sha1(as.character(1:length(clearTextA$IDA[clearTextA$IDA == ""] )), key = "B")
154
155 # Generate ID Col
156 IDA <- as.character(clearTextA$IDA)
157 IDB <- as.character(clearTextB$IDB)
158
159 # subset the data
160 clearTextA <- clearTextA[, ..idents]
161 clearTextB <- clearTextB[, ..idents]
162 }
163
164 ##### Start working #####

```

```

165
166 # Get EM weights
167 ests <- fastLink(clearTextA, clearTextB, varnames = idents,
168                 stringdist.match=Stringidents, stringdist.method="jw", n.cores = cores,
169                 ↪ estimate.only = TRUE)
170
171 save(ests, file="EM_ests.Rdata") # In case of crash
172 load("EM_ests.Rdata")
173
174 # Weights
175 m <- ests$p.m
176 u <- ests$p.u
177
178 # Loop over parameter space
179 for (q in Q){
180
181     cat("\n\n\nLine-by-line checking...", format(Sys.time()), " (%a, %d-%m-%Y,
182         ↪ %H:%M:%S)", "\n")
183
184     # Generate full linkage key
185     keys <- apply(clearTextA, 1, paste0, collapse = "")
186     keys <- gsub(" ", "", keys)
187
188     # Split into bigrams
189     qgrams <- (sapply(keys, function(key) substring(key, first=seq(1, nchar(key)), last=seq(q,
190         ↪ nchar(key)+q))))
191     # Count uniques and save measures
192     uniquengrams <- as.numeric(unlist(lapply(qgrams, function(x) length(unique(x)))))
193     # ngram Count
194     ngramsC <- as.numeric(unlist(lapply(qgrams, function(x) length(x))))
195     qgrams <- unlist(qgrams)
196     qgramlist <- as.character(na.omit(ifelse(qgrams == "" | nchar(qgrams) != q, NA, qgrams)))
197
198     cat("\n\n\nLine-by-line checking complete, calculating measures...",
199         ↪ format(Sys.time()), " (%a, %d-%m-%Y, %H:%M:%S)", "\n")
200
201     # Sum missings
202     missings <- sum(apply(clearTextA, 2, function(x) sum(is.na(x)))) +
203         ↪ sum(apply(clearTextA, 2, function(x) sum(x[!is.na(x)]=="")))

```

```

201
202 # Mean entropy calculation
203 meanentropy <- mean(apply(clearTextA, 2, function(x) entropy(table(x))))
204
205 # q90/q10-ratio
206 q90 <- round(quantile(as.numeric(table(qgramlist)), probs=seq(0,1,0.1))["90%"])
207 q10 <- round(quantile(as.numeric(table(qgramlist)), probs=seq(0,1,0.1))["10%"])
208
209 # Measures
210 meanengrams <- mean(ngramsC)
211 sdgrams <- sd(ngramsC)
212
213 skew <- skewness(table(qgramlist))
214 ginicoefficientqgrams <- Gini(table(qgramlist))
215
216 ginicoefficient <- mean(apply(clearTextA, 2, function(x) Gini(table(x))))
217
218 errorestimation <- q90/q10 # amount of rare combinations in data
219 missingamount <- missings
220 meanmissing <- missings/nrow(clearTextA)
221 uniqueness <- mean(uniquengrams/ngramsC)
222 sduniqueness <- sd(uniquengrams/ngramsC)
223
224
225
226 for (k in K){
227
228   for (i in seq(1:bootstraps)){
229
230     cat("\n\n\nEncrypting...", format(Sys.time(), "(%a, %d-%m-%Y, %H:%M:%S)"), "\n")
231
232     ## Generate password vector for BF encoding, resample for boot-strapping
233     PWs <- replicate(length(identfs), fastdigest(sample(LETTERS, 10000, replace =
234       ↪ TRUE)))
235
236     encryptedA <- CreateCLK(ID = IDA, clearTextA[, ..identfs], k = k, padding =
237       ↪ rep(0,length(identfs)), q = rep(q,length(identfs)), l = l, password =
238       ↪ PWs)#,"13124","124124")
239
240     encryptedB <- CreateCLK(ID = IDB, clearTextB[, ..identfs], k = k, padding =
241       ↪ rep(0,length(identfs)), q = rep(q,length(identfs)), l = l, password =
242       ↪ PWs)#,"13124","124124")

```

```

237
238 # Calculate hamming weight and NO of unique patterns
239 hammingweight <- mean(c(nchar(gsub("0","", encryptedA$CLKs)),nchar(gsub("0","",
  ↪ encryptedB$CLKs))))
240 uniquepatternsA <- length(unique(encryptedA$CLKs))
241 uniquepatternsB <- length(unique(encryptedB$CLKs))
242 #
243 # # Tree data
244 write.table(encryptedA, "TreeA_OPC.csv", sep="\t", row.names=FALSE, quote=FALSE,
  ↪ col.names=FALSE)
245 write.table(encryptedB, "TreeB_OPC.csv", sep="\t", row.names=FALSE, quote=FALSE,
  ↪ col.names=FALSE)
246 #
247 # Run MBT
248 multibitTree.load("TreeB_OPC.csv",threads=cores, leafLimit = leaflimit)
249 result <- multibitTree.searchFile("TreeA_OPC.csv", Tani)
250 #
251 # # Classify candidate pairs
252 tp <- sum((result$fingerprint) == (result$query))
253 fp <- sum((result$fingerprint) != (result$query))
254 fn <- truepositives - tp
255 tn <- pairs - (fn + tp + fp)
256 #
257 # # Calculate central measures
258 ReductionRate <- 1 - ((tp+fp)/(tn+fn))
259 precision <- tp / (tp+fp)
260 recall <- tp / (tn+fn)
261 Fmeasure = (recall + precision) / 2
262
263 # Set counter plus one
264 counter <- counter + 1
265
266 cat("\nClassification", counter, "of", numCombinations,"Finished
  ↪ (",format((counter / numCombinations)*100, digits=3),"%)! ",
  ↪ format(Sys.time()," (%a, %d-%m-%Y, %H:%M:%S)"), ": \nFilesizes:\t",
  ↪ format(filesizeA,scientific=FALSE),"\nData source and mode:\t" , inputFile, "
  ↪ ", mode,"\nBootstrap:\t\t", i, "/", bootstraps, "\nk/q:\t\t", k, "/",q
  ↪ ,"\nTanimoto:\t", tanimotoGoal, "\n\nPositives:\t", truepositives, "\nTrue
  ↪ Positives:\t", tp, "\nRecall:\t\t", recall, "\nFalse Positives:", fp,
  ↪ "\nPrecision:\t", precision, "\nFalse Negatives:", fn, "\nF-Score:\t",
  ↪ Fmeasure, "\n\n")

```

```

267
268 # Append to result
269 meta <- rbind(meta, data.frame(d="Mortality Test Data", type, q, k, i, l,
  ↪ Tani=tanimotoGoal,Fmeasure, filesizeA, filesizeB, meanentropy, meanengrams,
  ↪ sdgrams, skew, ginicoefficient, ginicoefficientqgrams, errorestimation ,
  ↪ hammingweight, uniqueness, sduniqueness, meanmissing, missingamount,
  ↪ uniquepatternsA,uniquepatternsB,m,u))
270 write.table(meta, "tempResult.csv", sep="\t", row.names=FALSE, quote=FALSE,
  ↪ col.names=TRUE)
271 }
272 }
273 }
274
275
276 # Prepare resulting data to predict from it
277 newdata <- data.frame(meta, stringsAsFactors = FALSE, row.names = NULL)
278 names(newdata)[names(newdata) == "tanimotoGoal"] <- "Tani"
279 write.table(newdata, "Trainresults.csv", sep="\t", row.names=FALSE, quote=FALSE,
  ↪ col.names=TRUE)
280 newdata <- read.csv("Trainresults.csv", sep = "\t", stringsAsFactors = FALSE)
281
282 # Predict
283 newfitted <- data.frame(newdata, predicted = predict(LM, newdata, type = "response"))
284 newfittedBIC <- data.frame(newdata, predicted = predict(BIC, newdata, type = "response"))
285
286 # Get optimal prameter estimate
287 parameters <-
  ↪ newfitted[which.max((newfitted$predicted)),c("Tani","q","k","l","predicted")]
288 parameters2 <-
  ↪ newfittedBIC[which.max((newfittedBIC$predicted)),c("Tani","q","k","l","predicted")]
289
290 # Write out parameters for use in loop
291 write.table(parameters, "Optimal_Parameters.csv",sep="\t", row.names = FALSE)
292 write.table(parameters2, "Optimal_Parameters_BIC.csv",sep="\t", row.names = FALSE)
293
294 cat("\n All done!")

```

Source Code C.5: Full evaluation routine for all models against best-practice solutions. Main evaluation routine. Required for plotting results.

```
1 #####
2 #
3 # Full test routine testing Optimal parameter choices against best-practice solutions
4 #
5 #####
6
7
8
9 # Clear workspace, set working dir
10 #rm(list=ls())
11 #setwd("E:/Priv/Dropbox/Dissertation/Programme")
12
13
14 # Libs used
15 library(PPRL)
16 library(fastdigest)
17 library(stringdist)
18 library(entropy)
19 library(data.table)
20 library(grid)
21 library(gridExtra)
22 library(fastLink)
23 library(ggplot2)
24 library(sjstats)
25 library(gmodels)
26 library(rsm)
27 library(boot)
28 library(neuralnet)
29 library(randomForest)
30 library(MASS)
31 library(multibitTree)
32
33 #### File names ####
34 ewodataFN <- "Daten/Abgleich_UKE_SCHUFA_EMA.csv"
35 ncvoterAFN <- "Daten/ncvoter-20140619-temporal-balanced-ratio-1to1-a.csv"
36 ncvoterBFN <- "Daten/ncvoter-20140619-temporal-balanced-ratio-1to1-b.csv"
37 telCDAFN <- "Daten/A_10000_python_R0_C0_0100_percent.csv"
```

```
38 telCDBFN <- "Daten/B_10000_python_R20_C50_0100_percent.csv"
39 FebrlAFN <- "Daten/FEBRL_10000_20_A.csv"
40 FebrlBFN <- "Daten/FEBRL_10000_20_B.csv"
41 FebrlWAAFN <- "Daten/Adrian_Perfect_100k.csv"
42 FebrlWABFN <- "Daten/Adrian_10percent_100k.csv"
43 inputFile <- "Daten/EWMA.csv"
44 inputFileB <- "Daten/HOSPITAL.csv"
45
46
47 # Set data sets for loop
48 datasets <- c("German Mortality", "NC Voter", "German Telephone CD 20% errors", "German
↳ Telephone CD 0% errors", "FEBRL", "FEBRL WA", "Mortality Test Data")
49 variant <- c("Standard CLK k = 5", "Standard CLK k = 10", "Standard CLK k = 15", "Optimal
↳ Parameter choice (LM)", "Optimal Parameter choice (BIC)", "50%-Rule Parameter
↳ choice", "Optimal Parameter choice (RSM)", "Optimal Parameter choice (RF)", "Optimal
↳ Parameter choice (NN)")
50 cores <- 7
51 leaflimit <- 3
52 tanimotoThresholds <- c(1.0, 0.95, 0.9, 0.85, 0.8)
53 fehler <- c(0)
54 l <- 500
55 nNCV <- 50000
56
57 # For progress reports, calc number of runs needed
58 numCombinations <- length(tanimotoThresholds) * length(variant) * length(fehler) *
59   length(datasets)
60
61 # Init Counter to count progress
62 counter <- 0
63
64
65 # Init result matrix
66 meta <- NULL
67
68
69 # Read training file for model building
70 traindata <- read.csv("Complete_Results.csv", sep="\t")
71
72 # Subset to fixed tanimoto threshold
73 fixed <- subset(traindata, Tani == 0.9)
74
```

```

75 # Build new complete data set with evaluation data results for rapid model comparisons
76 trainresultsFull <- rbind(read.csv("Trainresults_FEBRLWA.csv",
  ↪ sep="\t"),read.csv("Trainresults_Mortality.csv", sep="\t"),
  ↪ read.csv("Trainresults_NCVoter.csv", sep="\t"))
77
78 # Kill unnecessary cols
79 kill <- c("tp", "fp", "fn", "tn", "recall", "precision")
80 fulldata <- fixed[,(!names(fixed) %in% kill)]
81 fulldata$variant <- NULL
82 fulldata$positives <- NULL
83 fulldata$i <- 1
84
85 # Bind training to evaluation data
86 fulldata <- rbind(fulldata,trainresultsFull)
87
88 # Train models only on training data
89 fixed <- subset(traindata, Tani == 0.9)
90
91 cat("\n\n\nClassifications start!", format(Sys.time()," (%a, %d-%m-%Y, %H:%M:%S)"),"\n")
92
93
94
95 ##### Function definitions #####
96
97 ##### Standardization #####
98 unwanted_array = list('Š'='S', 'š'='s', 'Ž'='Z', 'ž'='z', 'À'='A', 'Á'='A', 'Â'='A',
  ↪ 'Ã'='A', 'Ä'='Ae', 'Å'='A', 'Æ'='A', 'Ç'='C', 'Ð'='D', 'È'='E', 'É'='E',
99 'Ê'='E', 'Ë'='E', 'Ï'='I', 'Í'='I', 'Î'='I', 'Ï'='I', 'Ñ'='N',
  ↪ 'Ö'='Oe', 'Ò'='O', 'Ó'='O', 'Ô'='O', 'Õ'='O', 'Ö'='O', 'Ø'='O',
  ↪ 'Ù'='U',
100 'Ú'='U', 'Û'='U', 'Ü'='Ue', 'Ý'='Y', 'Þ'='B', 'ß'='Ss', 'à'='a',
  ↪ 'á'='a', 'â'='a', 'ã'='a', 'ä'='ae', 'å'='a', 'æ'='a', 'ç'='c',
101 'è'='e', 'é'='e', 'ê'='e', 'ë'='e', 'ì'='i', 'í'='i', 'î'='i',
  ↪ 'ï'='i', 'ð'='o', 'ñ'='n', 'ò'='o', 'ó'='o', 'ô'='o', 'õ'='o',
102 'ö'='oe', 'ø'='o', 'ü'='ue', 'ù'='u', 'ú'='u', 'û'='u', 'ý'='y',
  ↪ 'ÿ'='y', 'þ'='b', 'ÿ'='y' )
103
104 normalize_string=function(string){
105   string = enc2utf8(string)
106   string=gsub("[ ]$*+.[^{|(\\#%&~/<=>!,:;\\")}]@", "",string)
107   string=gsub("-", "",string)

```

```

108 string =chartr(paste(names(unwanted_array), collapse=' '),
109               paste(unwanted_array, collapse=' '),
110               string)
111 string=gsub("[\t\n\r\f\v]", "",string)
112 string=gsub("[0-9]", "",string)
113 string=toupper(string)
114 string=iconv(string, "utf8", "ASCII", sub="")
115 return(string)
116 }
117
118 normalize_dobs=function(string){
119   if(is.na(string)){
120     return("")
121   } else {
122     string = as.character(string)
123     if (nchar(string) == 1){
124       string = paste0("0",string)
125     } else if (nchar(string) == 0){
126       string = ""
127     }
128     return(string)
129   }
130 }
131
132
133 ## Bootstrap mean
134 boot.mean = function(x,B,binwidth=NULL){
135   n = length(x)
136   boot.samples = matrix( sample(x,size=n*B,replace=TRUE), B, n)
137   boot.statistics = apply(boot.samples,1,mean)
138   se = sd(boot.statistics)
139   require(ggplot2)
140   if ( is.null(binwidth) )
141     binwidth = diff(range(boot.statistics))/30
142   p = ggplot(data.frame(x=boot.statistics),aes(x=x))
143     ↪ +geom_histogram(aes(y=..density..),binwidth=binwidth) + geom_density(color="red")
144   #plot(p)
145   interval = mean(x) + c(-1,1)*2*se
146   print( interval )
147   return( list(boot.statistics = boot.statistics, interval=interval, se=se, plot=p) )
148 }

```

```

148
149
150
151
152 # Fixed model calls
153 # BIC
154
155 # Full model for BIC-selection
156 fullmod <- lm(Fmeasure ~ k + (q) + errorestimation + skew + uniqueness + meanentropy +
  ↪ log(u) + ginicoefficient + uniquepatternsA + errorestimation + k * q *
  ↪ hammingweight + meanengrams*hammingweight +
157           meanmissing + log(m) + u*m,
158           data=fixed)
159 # Minimal model
160 minmod <- lm(Fmeasure ~ k + (q), data = fixed)
161
162 # Use BIC-selection to get the model
163 stepTest <- stepAIC(fullmod, minmod, direction = "both", k = log(nrow(fixed))) # bic
164 #stepTest2 <- stepAIC(fullmod, minmod, direction = "both", k = 2) # aic
165 BIC2 <- stepTest
166
167
168 # LM
169 LM <- lm(Fmeasure ~ k + (q) + errorestimation * skew + uniquepatternsA *
170           uniqueness + ginicoefficient + k * hammingweight +
171           hammingweight * meanengrams + uniquepatternsA + log(u),
172           data=fixed)
173
174
175 # Tune mtry value
176 tune <- subset(traindata, Tani == 0.9)
177 tuneRF(tune[,c("k", "q", "errorestimation", "skew", "uniquepatternsA", "uniqueness",
  ↪ "ginicoefficient", "hammingweight", "meanengrams", "u", "m")], tune[, "Fmeasure"],
  ↪ ntreeTry=3000, stepFactor=1, plot=FALSE)
178
179 # Random forest
180 r1 <- randomForest(Fmeasure ~ k + q + errorestimation + skew + uniqueness +
  ↪ ginicoefficient + hammingweight +
181           meanengrams + u + uniquepatternsA, data=fixed,
182           mtry= 3, ntree=500, nodesize=20, #nodesize=5,
183           importance=TRUE, nPerm=5, keep.forest=TRUE)

```

```

184
185
186 # RSM: Preproc data
187 RSMdata <- subset(fixed, Tani == 0.9)[,c("k", "q", "u", "meanengrams", "hammingweight",
  ↪ "Fmeasure")]
188 RSMdata_unscaled <- subset(fixed, Tani == 0.9)[,c("k", "q", "u", "meanengrams",
  ↪ "hammingweight", "Fmeasure")]
189 # Scale min/max
190 maxi = apply(RSMdata , 2, function(x) max(as.numeric(x)))
191 mini = apply(RSMdata, 2, function(x) min(as.numeric(x)))
192 # Scaling
193 RSMdata = as.data.frame(scale(RSMdata, center = mini, scale = maxi - mini))
194 # Train model
195 rsm.train <- rsm(Fmeasure ~ SO(k, q) + TWI(q, k, hammingweight) + (u), data = RSMdata)
196
197
198 # Neural network
199 NNata <- fixed[fixed$Tani == 0.9 ,c("k", "q", "skew", "uniqueness", "errorestimation",
  ↪ "hammingweight", "ginicoefficient", "meanengrams", "uniquepatternsA", "u",
  ↪ "Fmeasure")]
200 NNata_unscaled <- NNata
201 # Scale min/max
202 maxi = apply(NNata , 2, function(x) max(as.numeric(x)))
203 mini = apply(NNata, 2, function(x) min(as.numeric(x)))
204 # Scaling
205 NNata = as.data.frame(scale(NNata, center = mini, scale = maxi - mini))
206 # fit neural network
207 set.seed(42)
208 NN <- neuralnet(Fmeasure ~ k + q + ginicoefficient + hammingweight +
209               uniquepatternsA + (u), NNata, hidden = 2, linear.output = T, rep = 1)
210
211
212 ##### Loop over datasets #####
213 for (d in datasets){
214
215
216 # RSM: Preproc data
217 newdataRSM <- subset(fulldata, Tani == 0.9 & data == d)
218 newdataRSM <- newdataRSM[,c("k", "q", "u", "meanengrams", "hammingweight", "Fmeasure")]
219
220 maxi = apply(newdataRSM , 2, function(x) max(as.numeric(x)))

```

```

221 mini = apply(newdataRSM, 2, function(x) min(as.numeric(x)))
222
223 # Scale data
224 newdataRSM = as.data.frame(scale(newdataRSM, center = mini, scale = maxi - mini))
225 newdataRSM$u <- 0.5
226 newdataRSM$meanengrams <- 0.5
227
228 # Predict RSM
229 RSM <- data.frame(newdataRSM, predicted = predict(rsm.train, newdataRSM, type =
↳ "response"))
230 # Re-Scale to get actual values
231 RSM$Fmeasure <- (RSM$Fmeasure * (max(RSMdata_unscaled$Fmeasure) -
↳ min(RSMdata_unscaled$Fmeasure))) + min(RSMdata_unscaled$Fmeasure)
232 RSM$k <- (RSM$k * (max(RSMdata_unscaled$k) - min(RSMdata_unscaled$k))) +
↳ min(RSMdata_unscaled$k)
233 RSM$q <- (RSM$q * (max(RSMdata_unscaled$q) - min(RSMdata_unscaled$q))) +
↳ min(RSMdata_unscaled$q)
234
235
236 # New data to predict from
237 newdata <- subset(fulldata, Tani == 0.9 & data == d)# & q == 2)
238
239 # Predict LM/BIC/RF
240 newfitted <- data.frame(newdata, predicted = predict(LM, newdata, type = "response"))
241 newfittedBIC <- data.frame(newdata, predicted = predict(BIC2, newdata, type =
↳ "response"))
242 newfittedRF <- data.frame(newdata, predicted = predict(r1, newdata))
243
244 # Get optimal prameter estimate
245 parameters <- newfitted[which.max((newfitted$predicted)), c("Fmeasure", "Tani", "q",
↳ "k", "l", "predicted")]
246 parameters2 <- newfittedBIC[which.max((newfittedBIC$predicted)), c("Fmeasure", "Tani",
↳ "q", "k", "l", "predicted")]
247 parameters3 <- newfittedRF[which.max((newfittedRF$predicted)), c("Fmeasure", "Tani",
↳ "q", "k", "l", "predicted")]
248 parametersRSM <- RSM[which.max((RSM$predicted)), c("Fmeasure", "q", "k", "predicted")]
249
250
251 ## Prediction using neural network
252 newdataNN <- fulldata[fulldata$Tani == 0.9 & fulldata$data == d , colnames(NN$covariate)]
253 newdataNN_unscaled <- newdataNN

```

```

254 # Scale again
255 maxi = apply(newdataNN , 2, function(x) max(as.numeric(x)))
256 mini = apply(newdataNN, 2, function(x) min(as.numeric(x)))
257 # Scaling
258 newdataNN = as.data.frame(scale(newdataNN, center = mini, scale = maxi - mini))
259 # Unscalable because constant values -> 0.5
260 newdataNN[sapply(newdataNN, is.nan)] <- 0.5
261
262 # Predict and rescale to original values
263 predict_testNN = neuralnet::compute(NN, newdataNN)
264 predict_testNN = (predict_testNN$net.result * (max(NNata_unscaled$Fmeasure) -
  ↪ min(NNata_unscaled$Fmeasure))) + min(NNata_unscaled$Fmeasure)
265 # Prediction data frame
266 newfitNN <- cbind(fullldata[fullldata$Tani == 0.9 & fullldata$data == d,], predict_testNN)
267
268 # Get NN parameters
269 kNN <- round(mean(newfitNN[which(newfitNN$predict_testNN %in%
  ↪ newfitNN$predict_testNN[newfitNN$predict_testNN >=
  ↪ (quantile(as.numeric((newfitNN$predict_testNN)),
  ↪ probs=seq(0,1,0.1))["90%"])])], "k"))
270 qNN <- round(mean(newfitNN[which(newfitNN$predict_testNN %in%
  ↪ newfitNN$predict_testNN[newfitNN$predict_testNN >=
  ↪ (quantile(as.numeric((newfitNN$predict_testNN)),
  ↪ probs=seq(0,1,0.1))["90%"])])], "q"))
271
272 # Get RSM parameters
273 kRSM <- parametersRSM$k
274 #round(mean(RSM[which(RSM$predicted %in% RSM$predicted[RSM$predicted >=
  ↪ (quantile(as.numeric((RSM$predicted)), probs=seq(0,1,0.1))["90%"])])], "k"))
275 qRSM <- parametersRSM$q
276 #round(mean(RSM[which(RSM$predicted %in% RSM$predicted[RSM$predicted >=
  ↪ (quantile(as.numeric((RSM$predicted)), probs=seq(0,1,0.1))["90%"])])], "q"))
277
278
279 # Write out parameters for use in loop
280 write.table(parameters, "Optimal_Parameters.csv", sep="\t", row.names = FALSE)
281 write.table(parameters2, "Optimal_Parameters_BIC.csv", sep="\t", row.names = FALSE)
282 write.table(parameters3, "Optimal_Parameters_RF.csv", sep="\t", row.names = FALSE)
283
284
285 ##### Preprocess depending on data #####

```

```
286 if (d == "NC Voter"){
287
288   # Read
289   clearTextB <- fread(ncvoterBFN, sep=",")
290   clearTextA <- fread(ncvoterAFN, sep=",")
291
292   # Subset identifiers
293   clearTextA <- clearTextA[,c(1,2,4,5)]
294   clearTextB <- clearTextB[,c(1,2,4,5)]
295
296   # Subset overlapping records
297   clearTextA <- clearTextA[(clearTextA$voter_id %in% clearTextB$voter_id),]
298   clearTextB <- clearTextB[(clearTextB$voter_id %in% clearTextA$voter_id),]
299
300   # Deduplicate
301   clearTextA <- clearTextA[!duplicated(clearTextA$voter_id),]
302   clearTextB <- clearTextB[!duplicated(clearTextB$voter_id),]
303
304   # Draw sample
305   set.seed(21)
306   clearTextA <- clearTextA[sample(1:nrow(clearTextA), nNCV, replace=FALSE),]
307   set.seed(21)
308   clearTextB <- clearTextB[sample(1:nrow(clearTextB), nNCV, replace=FALSE),]
309
310   clearTextA$Day <- ""
311   clearTextA$Month <- ""
312   clearTextA$Year <- clearTextA$age
313
314   clearTextB$Day <- ""
315   clearTextB$Month <- ""
316   clearTextB$Year <- clearTextB$age
317   clearTextA$age <- NULL
318   clearTextB$age <- NULL
319
320   ### names anpassen für alle datasets
321
322 } else if (d == "German Telephone CD 20% errors"){
323
324   clearTextB <- fread(telCDBFN, sep="\t")
325   clearTextA <- fread(telCDAFN, sep="\t")
326
```

```
327 # Reorder
328 clearTextA <- clearTextA[,c(1,7,8,5,4,3)]
329 clearTextB <- clearTextB[,c(1,7,8,5,4,3)]
330
331
332 } else if (d == "German Telephone CD 0% errors"){
333
334   clearTextB <- fread(telCDAFN, sep="\t")
335   clearTextA <- fread(telCDAFN, sep="\t")
336
337 # Reorder
338 clearTextA <- clearTextA[,c(1,7,8,5,4,3)]
339 clearTextB <- clearTextB[,c(1,7,8,5,4,3)]
340
341
342 } else if (d == "German Mortality"){
343   clearText <- fread(ewodataFN, sep=",")
344
345   clearTextB <- subset(clearText, quelle=="uke")
346   clearTextA <- subset(clearText, quelle=="einwo")
347   clearTextA <- clearTextA[,c(14,2,3,5,6,7)]
348   clearTextB <- clearTextB[,c(14,2,3,5,6,7)]
349
350 } else if (d == "Mortality Test Data"){
351   clearText <- fread(ewodataFN, sep=",")
352
353   clearTextB <- subset(clearText, quelle=="schufa")
354   clearTextA <- subset(clearText, quelle=="einwo")
355   clearTextA <- clearTextA[,c(14,2,3,5,6,7)]
356   clearTextB <- clearTextB[,c(14,2,3,5,6,7)]
357
358 } else if (d == "FEBRL"){
359
360   clearTextB <- fread(Febr1BFN, sep="\t")
361   clearTextA <- fread(Febr1AFN, sep="\t")
362
363   clearTextA <- clearTextA[,c(1,2,3,10)]
364   clearTextB <- clearTextB[,c(1,2,3,10)]
365   clearTextA$Day <- (substr(clearTextA$V10,7,8))
366   clearTextA$Month <- (substr(clearTextA$V10,5,6))
367   clearTextA$Year <- (substr(clearTextA$V10,1,4))
```

```
368 clearTextA$V10 <- NULL
369
370 clearTextB$Day <- (substr(clearTextB$V10,7,8))
371 clearTextB$Month <- (substr(clearTextB$V10,5,6))
372 clearTextB$Year <- (substr(clearTextB$V10,1,4))
373 clearTextB$V10 <- NULL
374
375
376 } else {
377   clearTextA <- fread(FebrlWAAFN, colClasses = "character", stringsAsFactors = FALSE)
378   clearTextB <- fread(FebrlWABFN, colClasses = "character", stringsAsFactors = FALSE)
379
380   clearTextA$Day <- (substr(clearTextA$`Date Of Birth`,7,8))
381   clearTextA$Month <- (substr(clearTextA$`Date Of Birth`,5,6))
382   clearTextA$Year <- (substr(clearTextA$`Date Of Birth`,1,4))
383   clearTextA$`Date Of Birth` <- NULL
384   clearTextB$Day <- (substr(clearTextB$`Date Of Birth`,7,8))
385   clearTextB$Month <- (substr(clearTextB$`Date Of Birth`,5,6))
386   clearTextB$Year <- (substr(clearTextB$`Date Of Birth`,1,4))
387   clearTextB$`Date Of Birth` <- NULL
388
389
390   clearTextA <- clearTextA[,c(2,4,6,12,13,14)]
391   clearTextB <- clearTextB[,c(2,4,6,12,13,14)]
392
393
394
395
396 }
397
398 # Read error-free file
399 names(clearTextA) <- c("ID", "Vorname", "Nachname", "Day", "Month", "Year")
400 names(clearTextB) <- c("ID", "Vorname", "Nachname", "Day", "Month", "Year")
401
402
403
404 ##### General preproc #####
405 clearTextA$ID <- as.character(clearTextA$ID)
406 clearTextB$ID <- as.character(clearTextB$ID)
407
408
```

```
409 # Preprocess A
410
411 ## Standardize data
412 clearTextA$Vorname <- normalize_string(clearTextA$Vorname)
413 clearTextA$Nachname <- normalize_string(clearTextA$Nachname)
414
415 clearTextA$Day <- sapply(clearTextA$Day, normalize_dobs)
416 clearTextA$Month <- sapply(clearTextA$Month, normalize_dobs)
417 clearTextA$Year <- as.character(iffelse(is.na(clearTextA$Year), "", clearTextA$Year))
418
419 # Generate full DOB
420 clearTextA$DOB <- paste(clearTextA$Day,clearTextA$Month,clearTextA$Year, sep="")
421 #clearTextA$DOB <- substr(clearTextA$DOB, 3,6)
422
423
424 # Preprocess B
425
426 ## Standardize data
427 clearTextB$Vorname <- normalize_string(clearTextB$Vorname)
428 clearTextB$Nachname <- normalize_string(clearTextB$Nachname)
429
430 clearTextB$Day <- sapply(clearTextB$Day, normalize_dobs)
431 clearTextB$Month <- sapply(clearTextB$Month, normalize_dobs)
432 clearTextB$Year <- as.character(iffelse(is.na(clearTextB$Year), "", clearTextB$Year))
433
434 # Generate full DOB for salting
435 clearTextB$DOB <- paste(clearTextB$Day,clearTextB$Month,clearTextB$Year, sep="")
436 #clearTextB$DOB <- substr(clearTextB$DOB, 3,6)
437
438 ## Deduplicate by ID
439 clearTextA <- clearTextA[!duplicated(clearTextA$ID),]
440 clearTextB <- clearTextB[!duplicated(clearTextB$ID),]
441
442
443 ## File sizes, overlap
444 filesizeA <- (nrow(clearTextA))
445 filesizeB <- (nrow(clearTextB))
446
447 pairs <- as.numeric(filesizeA) * as.numeric(filesizeB)
448 truepositives <- sum(clearTextA$ID %in% clearTextB$ID)
449 positives <- truepositives
```

```
450 clearTextA$LinkKey <- paste(clearTextA$Vorname, clearTextA$Nachname, clearTextA$DOB,
451   ↪ sep="")
452 clearTextB$LinkKey <- paste(clearTextB$Vorname, clearTextB$Nachname, clearTextB$DOB,
453   ↪ sep="")
454 cat("\n\n\nClear-Text files ready, analyzing files", format(Sys.time()), " (%a, %d-%m-%Y,
455   ↪ %H:%M:%S)", "\n")
456
457
458 # Loop over every combination of parameters
459 for (v in variant){
460
461   ### Encryption phase ###
462   cat("\n\n\nEncryption start!", format(Sys.time()), " (%a, %d-%m-%Y, %H:%M:%S)", "\n")
463
464   if (v == "Standard CLK k = 20"){
465
466     # Parameters
467     k <- 20
468     q <- 2
469     RMSE <- NA
470
471   } else if (v == "Standard CLK k = 10"){
472
473     # Parameters
474     k <- 10
475     q <- 2
476     RMSE <- NA
477
478   } else if (v == "Standard CLK k = 5"){
479
480     # Parameters
481     k <- 5
482     q <- 2
483     RMSE <- NA
484
485   } else if (v == "Standard CLK k = 15"){
486
487     # Parameters
```

```

488     k <- 15
489     q <- 2
490
491     RMSE <- NA
492
493   } else if (v == "Optimal Parameter choice (LM)") {
494
495     # Calculate Root Mean Square Error (RMSE)
496     RMSE = (sum((newfitted$Fmeasure - newfitted$predicted)^2) / nrow(newfitted)) ^ 0.5
497
498     # Parameters
499     k <- read.csv("Optimal_Parameters.csv", sep="\t")$k
500     q <- read.csv("Optimal_Parameters.csv", sep="\t")$q
501
502   } else if (v == "Optimal Parameter choice (RF)") {
503
504     # Calculate Root Mean Square Error (RMSE)
505     RMSE = (sum((newfittedRF$Fmeasure - newfittedRF$predicted)^2) / nrow(newfittedRF)) ^
506     ↪ 0.5
507
508     # Parameters
509     k <- read.csv("Optimal_Parameters_RF.csv", sep="\t")$k
510     q <- read.csv("Optimal_Parameters_RF.csv", sep="\t")$q
511
512   } else if (v == "Optimal Parameter choice (BIC)") {
513
514     # Parameters
515     k <- read.csv("Optimal_Parameters_BIC.csv", sep="\t")$k
516     q <- read.csv("Optimal_Parameters_BIC.csv", sep="\t")$q
517
518     # Calculate Root Mean Square Error (RMSE)
519     RMSE = (sum((newfittedBIC$Fmeasure - newfittedBIC$predicted)^2) /
520     ↪ nrow(newfittedBIC)) ^ 0.5
521
522   } else if (v == "Optimal Parameter choice (RSM)") {
523
524     # Parameters
525     k <- kRSM
526     q <- qRSM

```

```

527     # Calculate Root Mean Square Error (RMSE)
528     RMSE = (sum((RSM$Fmeasure - RSM$predicted)^2) / nrow(RSM)) ^ 0.5
529
530
531 } else if (v == "Optimal Parameter choice (NN){
532
533     # Parameters
534     k <- kNN
535     q <- qNN
536
537     # Calculate Root Mean Square Error (RMSE)
538     RMSE = (sum((newfitNN$Fmeasure - newfitNN$predict_testNN)^2) / nrow(newfitNN)) ^ 0.5
539
540 } else if (v == "50%-Rule Parameter choice"){
541
542     # Parameters
543     q <- 2
544     kA <- round((1 / round(mean(nchar(clearTextA$LinkKey)-(q-1)))) * log(2))
545     kB <- round((1 / round(mean(nchar(clearTextB$LinkKey)-(q-1)))) * log(2))
546     k <- max(c(kA,kB))
547
548 } else {
549     stop("Illegal variant: ", v)
550 }
551
552 # Encrypt to CLKs
553 encryptedA <- CreateCLK(ID = clearTextA$ID, clearTextA[, c(2,3,7)], k = k, padding =
554   ↪ rep(0,3), q = c(q,q,q), l = l, password = c("12341231", "41412412", "415212512"))
555 encryptedB <- CreateCLK(ID = clearTextB$ID, clearTextB[, c(2,3,7)], k = k, padding =
556   ↪ rep(0,3), q = c(q,q,q), l = l, password = c("12341231", "41412412", "415212512"))
557
558 cat("\n\n\nEncryption finished!", format(Sys.time(), "(%a, %d-%m-%Y, %H:%M:%S)"), "\n")
559 # Encrypt
560
561 # Write Tree data
562 write.table(encryptedA, "TreeA.csv", sep="\t", row.names=FALSE, quote=FALSE,
563   ↪ col.names=FALSE)
564 write.table(encryptedB, "TreeB.csv", sep="\t", row.names=FALSE, quote=FALSE,
565   ↪ col.names=FALSE)

```

```

564
565
566 ## Loop over thresholds
567 for (Tani in tanimotoThresholds){
568
569     # Run MBT
570     multibitTree.load("TreeB.csv",threads=cores, leafLimit = leaflimit)
571     result <- multibitTree.searchFile("TreeA.csv", Tani)
572
573
574     # Classify candidate pairs
575     tp <- sum((result$fingerprint) == (result$query))
576     fp <- sum((result$fingerprint) != (result$query))
577     fn <- positives - tp
578     tn <- pairs - (fn + tp + fp)
579
580     # Calculate central measures
581     ReductionRate <- 1 - ((tp+fp)/(tn+fn))
582     precision <- tp / (tp+fp)
583     recall <- tp / (tp+fn)
584     Fmeasure = (recall + precision) / 2
585
586     rm(result)
587     #gc()
588
589     # Set counter plus one
590     counter <- counter + 1
591
592     cat("\nClassification", counter, "of", numCombinations,"Finished (", format((counter
593     ↪ / numCombinations)*100, digits=3),"%)! ", format(Sys.time(), " (%a, %d-%m-%Y,
594     ↪ %H:%M:%S)"), ": \nFilesizes:\t", format(filesizeA,scientific=FALSE), "\nData
595     ↪ source:\t", d, "\nMethod:\t\t", v, "\nk/q:\t\t", k, "/", q, "\nTanimoto:\t",
596     ↪ Tani, "\n\nPositives:\t", truepositives, "\nTrue Positives:\t", tp,
597     ↪ "\nRecall:\t\t", recall, "\nFalse Positives:", fp, "\nPrecision:\t", precision,
598     ↪ "\nFalse Negatives:", fn, "\nF-Score:\t", Fmeasure, "\n\n")
599
600
601     # Append to result
602     meta <- rbind(meta, cbind(d, v, q, k, l, Tani, filesizeA,filesizeB,positives, tp,
603     ↪ fp, fn, tn, recall, precision,Fmeasure, RMSE))
604     write.table(meta, "tempResult.csv", sep="\t", row.names=FALSE, quote=FALSE,
605     ↪ col.names=TRUE)

```

```
597
598     }
599   }
600 }
601
602
603 # Format Output column names and write final result file
604 colnames(meta) <- c("data", "variant", "q", "k", "l", "Tani", "filesizea", "filesizeb",
  ↪ "realpositives", "tp", "fp", "tn", "fn", "recall", "precision", "Fmeasure", "RMSE")
605 write.table(meta, "./results/R_Metainfo_Testbett_CompareOPC_vs_Train.csv", sep="\t",
  ↪ row.names=FALSE, col.names=TRUE)
606
607 cat("\n\n\nAll Classifications Finished!", format(Sys.time(), "%a, %d-%m-%Y,
  ↪ %H:%M:%S"), "\n")
```

## Source code used for all result plots

Source Code D.1: Source code for the bigram frequency plot.

```
1 library(tidyverse)
2 library(data.table)
3
4 #setwd("E:/Priv/Dropbox/Dissertation/Programme")
5
6 # Read files
7 DE <- fread("Daten/NN_Bigram_Frequencies_DE.csv")
8 PL <- fread("Daten/NN_Bigram_Frequencies_PO.csv")
9
10 # Get top 50 DE bigrams
11 names <- head(DE[order(DE$freq, decreasing=TRUE), ],50)
12
13 # Make factors for plotting, reorder by bigram count
14 names$bigram <- factor(names$bigram)
15 names$bigram <- reorder(names$bigram, names$freq, mean)
16
17 # Scientific notation disabled
18 options(scipen=10000)
19
20 # counts for DE
21 p1 <- ggplot(data=names, aes(x=bigram, y=freq)) + geom_bar(stat="identity",
  ↳ fill="#004c93") + coord_flip() + theme_bw() +
22   scale_y_continuous(expand = c(0,0), limits = c(0, (max(names$freq) +
  ↳ round(max(names$freq) * 0.05 ) ))) + xlab("Bigram") + ylab("\nCount")
23 p1 <- p1 + ggtitle("German Nationals\n")
24 p1 + theme_bw() + theme(axis.text.x = element_text(angle = 0, vjust = 0.1), axis.text.y
  ↳ = element_text(size=20, hjust=0.5),
```

```

25     panel.spacing = unit(0.85, "lines"), text =
        ↪ element_text(size=35),
26     legend.justification=c(0.9,0.05), legend.key =
        ↪ element_blank(), plot.title = element_text(size=22),
27     legend.key.size = unit(5,"char"), strip.background =
        ↪ element_rect(colour='white',fill = 'white')) # look
28 # Save plot
29 ggsave("results/Bigram_Counts_DE.pdf",
30       height=16, width=10, units="in", device=cairo_pdf()) # Fig 4.3a
31
32
33
34 # Get top 50 for PL
35 names <- head(PL[order(PL$freq, decreasing=TRUE), ],50)
36
37
38 # Make factors for plotting, reorder by bigram count
39 names$bigram <- factor(names$bigram)
40 names$bigram <- reorder(names$bigram, names$freq, mean)
41
42 # Scientific notation disabled
43 options(scipen=10000)
44
45 # counts for pL
46 p1 <- ggplot(data=names, aes(x=bigram, y=freq)) + geom_bar(stat="identity",
        ↪ fill="#004c93") + coord_flip() + theme_bw() +
47   scale_y_continuous(expand = c(0,0), limits = c(0, (max(names$freq) +
        ↪ round(max(names$freq) * 0.05 ) ))) + xlab("Bigram") + ylab("\nCount")
48 p1 <- p1 + ggtitle("Polish Nationals\n")
49 p1 + theme_bw() + theme(axis.text.x = element_text(angle = 0, vjust = 0.1), axis.text.y
        ↪ = element_text(size=20, hjust=0.5),
50     panel.spacing = unit(0.85, "lines"), text = element_text(size=35),
51     legend.justification=c(0.9,0.05), legend.key = element_blank(),
        ↪ plot.title = element_text(size=22),
52     legend.key.size = unit(5,"char"), strip.background =
        ↪ element_rect(colour='white',fill = 'white')) # look
53 #Save plot
54 ggsave("results/Bigram_Counts_PL.pdf",
55       height=16, width=10, units="in", device=cairo_pdf()) # Fig 4.3b

```

Source Code D.2: Source code for time/RAM plot by tree type and *l*.

```

1 library(PPRL)
2 library(fastdigest)
3 library(pryr)
4 library(ggplot2)
5 library(data.table)
6 library(tidyverse)
7
8 # Time results in days for 5-20 mio records (Mail Adrian Brown 28/01/16)
9 dat <- data.frame(size = c(5, 6.834999, 10, 15, 20), time = c(0.9326388889, 1.6979166667,
10   ↪ 4.0888888889, 19.0833333333, 31.7104166667))
11
12 # Plot time in hours
13 p1 <- ggplot(dat, aes(x=size, y=time/60, color="white")) + geom_line(size=1,
14   ↪ color="black") + geom_point(size=8, color="black")
15 p1 <- p1 + xlab("Filesize (Mio)") + ylab("Time in Hours\n") + guides(colour=FALSE)
16 p1 + theme_bw() + theme(panel.margin = unit(0.7, "lines"), text = element_text(size=36),
17   ↪ legend.justification=c(0.9,0.05), legend.key = element_blank(), legend.key.size =
18   ↪ unit(3,"char"))
19 #p1 + theme_bw() + scale_colour_solarized("blue")
20 # theme_solarized(light = TRUE) + scale_colour_solarized("blue")
21 ggsave("AUS_Times_Q2016.pdf", height=12, width=14, units="in", device=cairo_pdf)
22
23 ### Times 1 mio vs 1 mio
24
25 # Data: Curtin university evaluation of MBT with different thresholds
26 mbtTimes <- tribble(
27   ~threshold, ~time,
28   1,          0.1616666666666664,
29   0.95,       0.6045000000000007,
30   0.9,         1.900333333333333,
31   0.85,        4.354999999999999,
32   0.8,         20.8125
33 )
34
35 # Plot
36 p1 <- ggplot(data = mbtTimes, aes(x = threshold, y = (time)))
37 p1 <- p1 + geom_point(size=8) # points
38 p1 <- p1 + geom_line(size=1) # lines

```

```

36 p1 <- p1 + xlab("Tanimoto threshold") + ylab("Time in minutes") # labels
37 p1 <- p1 + scale_y_continuous(breaks = seq(0,22,2)) # set year breaks
38 p1 <- p1 + scale_x_continuous(breaks = seq(0.8,1.0,0.05)) # set year breaks
39
40 p1 + theme_bw() + theme(axis.text.x = element_text(angle = 0, vjust = 0.1),
41                          panel.spacing = unit(0.85, "lines"), text = element_text(size=35),
42                          legend.justification=c(0.9,0.05), legend.key = element_blank(),
43                          legend.key.size = unit(5,"char"),
44                          strip.background = element_rect(colour='white',fill = 'white'))
45                          ↪ # look
46 ggsave("../Time_Threshold2.pdf",
47         height=12, width=12, units="in", device=cairo_pdf()) # Fig 4.4
48
49 ### Test: MBT variant and l and threshold vs Time and RAM
50
51
52
53 # Variants to test
54 Variants <- c("mtan",
55              "utan",
56              "mtanp",
57              "utanp")
58 # Set number of test cases and BF lengths and thresholds
59 N <- c(50000, 100000, 200000, 300000, 400000, 500000)
60 L <- c(250,500,1000)
61 tanimotoThresholds <- c(0.8,0.85,0.9,0.95)
62 # Base parameters
63 q <- 2
64 baseK <- 20
65 basel <- 1000
66 cores <- 6
67 leaflimit <- 3
68 set.seed(69)
69
70 # Counter for progress
71 passes <- length(N) * length(L) * length(Variants) * length(tanimotoThresholds)
72 i <- 0
73
74
75 # Init result

```

```

76 res <- matrix(NA, nrow=passes, ncol=6)
77
78
79 # Loop over sizes
80 for (n in N){
81
82
83   dat <- sapply(rep(NA, n), function(x) fastdigest(sample(LETTERS, 512, replace = TRUE)))
84
85
86   # 20% errors
87   datB <- dat
88   datB[(n - round(0.2 * n)):n] <- gsub("2","y", datB[(n - round(0.2 * n)):n])
89   #datB[(n - round(0.2 * n)):n] <- gsub("1","x", datB[(n - round(0.2 * n)):n])
90
91   for (l in L){
92
93     # Make equivalent k dependent on length
94     k <- round((baseK/baseL) * l)
95
96
97     # Gen BF
98     testDataBF <- CreateBF(ID = as.character(1:n), dat,
99                           k = k, padding = 0, q = 2, l = l, password = "(H)$6Uh04q")
100
101
102
103     testDataBFB <- CreateBF(ID = as.character(1:n), datB,
104                             k = k, padding = 0, q = 2, l = l, password = "(H)$6Uh04q")
105
106
107
108     # Write Tree data
109     write.table(testDataBF, "mbtSearch_2.0.1/TreeA.csv", sep="\t", row.names=FALSE,
110               ↪ quote=FALSE, col.names=FALSE)
111     write.table(testDataBFB, "mbtSearch_2.0.1/TreeB.csv", sep="\t", row.names=FALSE,
112               ↪ quote=FALSE, col.names=FALSE)
113
114   for (variant in Variants){
115
116     ## Loop over thresholds

```

```

115   for (Tani in tanimotoThresholds){
116
117       # Start timer
118       starttime <- proc.time()
119
120       #memuse <- memory.size(TRUE)
121
122       if (variant == "mtanp" | variant == "utanp"){
123           # Kill the "p"
124           variant <- substr(variant,1,4)
125           # Run MBT
126           system(paste("cd mbtSearch_2.0.1/ ;./mbtSearch -i TreeA.csv -m ", Tani, " -o
127             ↳ MBT_Outfile3.csv -t ",cores," -a ",variant," -p -l ",leaflimit," TreeB.csv",
128             ↳ sep=""))
129           # Put "p" back in
130           variant <- paste0(variant,"p")
131       } else {
132           # Just run it
133           system(paste("cd mbtSearch_2.0.1/ ;./mbtSearch -i TreeA.csv -m ", Tani, " -o
134             ↳ MBT_Outfile3.csv -t ",cores," -a ",variant," -l ",leaflimit," TreeB.csv",
135             ↳ sep=""))
136       }
137
138       # Used memory
139       memuse <- as.numeric(mem_used())/1000/1000
140
141       # Time taken in minutes
142       time <- proc.time() - starttime
143       time <- as.numeric(time[3])/60
144
145       # Read result file
146       #result <- fread("mbtSearch_2.0.1/MBT_Outfile3.csv", header = TRUE, sep = ",",
147         ↳ stringsAsFactors = FALSE)
148
149       i <- i + 1
150
151       cat("\nPass ", i, "of ",passes, "finished (",round(i/passes, digits = 2),"%",
152         ↳ time/RAM:",round(time),"/", round(memuse)," min/MB)", format(Sys.time())," (%a,
153         ↳ %d-%m-%Y, %H:%M:%S)")
154
155       res[i,] <- c(variant, n, l, Tani, time, memuse)

```

```

149     }
150   }
151 }
152 }
153
154
155 # Make df
156 res <- as.data.frame(res, stringsAsFactors = FALSE)
157 # Names
158 names(res) <- c("Variant", "n", "l", "Threshold", "time", "memuse")
159 # Make numeric
160 res[,2:6] <- sapply(res[,2:6],as.numeric)
161
162 # Write outfile
163 write.table(res, "Timing_l_Tani.csv", sep="\t", row.names=FALSE, quote=FALSE,
164   ↪ col.names=TRUE)
165
166 ##### Plotting
167
168 res <- fread("Timing_l_Tani.csv")
169
170 p5 <- ggplot(res, aes(x=as.numeric(n), y=time, color=Variant, group=Variant,
171   ↪ shape=Variant)) +
172   geom_point(size=5) + geom_line(size=1) +
173   scale_x_continuous(breaks = seq(0,500000,50000)) +
174   scale_shape_manual(name = "Tree type", labels= c("Multibit tree w/o Symdex", "Union bit
175   ↪ tree w/o Symdex", "Multibit tree + Symdex", "Union bit tree + Symdex"),
176   ↪ values=c(15:18)) +
177   scale_colour_manual(name = "Tree type", labels= c("Multibit tree w/o Symdex", "Union bit
178   ↪ tree w/o Symdex", "Multibit tree + Symdex", "Union bit tree + Symdex"),
179   ↪ values=c("#004c93", "#56B1F7", "#000000", "#c43f4b")) +
180   xlab("File Size") + ylab("Time in Minutes") +
181   theme_bw() + facet_grid(l ~ Threshold, labeller = label_both) +
182   theme(panel.spacing = unit(1, "lines"), text = element_text(size=27), axis.text.x =
183   ↪ element_text(size = 21, angle = 90, vjust = 0.5), axis.title=element_text(size=30),
184   ↪ legend.justification=c(0.9,0.05), legend.key = element_blank(), legend.key.size =
185   ↪ unit(5,"char"), strip.background = element_rect(colour='white',fill = 'white'))
186
187 show(p5)
188 ggsave("Timing_l_Tani.pdf",height=13, width=21, units="in", device=cairo_pdf)
189
190

```

```

181
182
183 # Time: l vs RAM and l vs Time
184 subs <- subset(res, Threshold == 0.8 & Variant == "mtan")
185 subs$l <- factor(subs$l, levels = c("1000", "500", "250"))
186
187 p5 <- ggplot(subs, aes(x = as.numeric(n), y=memuse, color=l, group=l)) +
188   geom_point(size=5) + geom_line(size=1) +
189   scale_x_continuous(breaks = seq(0,500000,50000)) +
190   scale_colour_manual(name = "Length l", values=c("#004c93", "#56B1F7",
191     ↪ "#000000", "#c43f4b")) +
192   ylab("Memory used in MB") + xlab("File Size") +
193   theme_bw() +
194   theme(panel.spacing = unit(1, "lines"), text = element_text(size=27), axis.text.x =
195     ↪ element_text(size = 21, angle = 90, vjust = 0.5), axis.title=element_text(size=30),
196     ↪ legend.justification=c(0.9,0.05), legend.key = element_blank(), legend.key.size =
197     ↪ unit(5,"char"), strip.background = element_rect(colour='white',fill = 'white'))
198 show(p5)
199 ggsave("RAM_1500.pdf",height=13, width=21, units="in", device=cairo_pdf) # Fig 4.1
200
201
202 p5 <- ggplot(subs, aes(x = as.numeric(n), y=time, color=l, group=l)) +
203   geom_point(size=5) + geom_line(size=1) +
204   scale_x_continuous(breaks = seq(0,500000,50000)) +
205   scale_colour_manual(name = "Length l", values=c("#004c93", "#56B1F7",
206     ↪ "#000000", "#c43f4b")) +
207   xlab("File Size") + ylab("Time in Minutes") +
208   theme_bw() +
209   theme(panel.spacing = unit(1, "lines"), text = element_text(size=27), axis.text.x =
210     ↪ element_text(size = 21, angle = 90, vjust = 0.5), axis.title=element_text(size=30),
211     ↪ legend.justification=c(0.9,0.05), legend.key = element_blank(), legend.key.size =
212     ↪ unit(5,"char"), strip.background = element_rect(colour='white',fill = 'white'))
213 show(p5)
214 ggsave("Time_1500.pdf",height=13, width=21, units="in", device=cairo_pdf) # Fig 4.2
215
216
217 # Threshold vs Time vs RAM
218
219 subs <- subset(res, l == 500 & Variant == "mtan")

```

```

214 subs$Threshold <- factor(subs$Threshold, levels = c("0.8", "0.85", "0.9", "0.95"),
  ↪ labels=c("0.80", "0.85", "0.90", "0.95"))
215
216 p5 <- ggplot(subs, aes(x=as.numeric(n), y=time, color=Threshold, group=Threshold)) +
217   geom_point(size=5) + geom_line(size=1) +
218   scale_x_continuous(breaks = seq(0,500000,50000)) +
219   scale_colour_manual(name = "Tanimoto threshold", values=c("#004c93", "#56B1F7",
  ↪ "#000000", "#c43f4b")) +
220   xlab("File Size") + ylab("Time in Minutes") +
221   theme_bw() +
222   theme(panel.spacing = unit(1, "lines"), text = element_text(size=27), axis.text.x =
  ↪ element_text(size = 21, angle = 90, vjust = 0.5), axis.title=element_text(size=30),
  ↪ legend.justification=c(0.9,0.05), legend.key = element_blank(), legend.key.size =
  ↪ unit(5,"char"), strip.background = element_rect(colour='white',fill = 'white'))
223 show(p5)
224 ggsave("Time_Threshold.pdf",height=13, width=21, units="in", device=cairo_pdf) # Fig 4.5
225
226
227
228 # Tree type vs runtime
229 subs <- subset(res, l == 1000 & Threshold == 0.9)
230
231 p5 <- ggplot(subs, aes(x=as.numeric(n), y=time, color=Variant, group=Variant,
  ↪ shape=Variant)) +
232   geom_point(size=5) + geom_line(size=1) +
233   scale_x_continuous(breaks = seq(0,500000,50000)) +
234   scale_shape_manual(name = "Tree type", labels= c("Multibit tree w/o Symdex", "Union bit
  ↪ tree w/o Symdex", "Multibit tree + Symdex", "Union bit tree + Symdex"),
  ↪ values=c(15:18)) +
235   scale_colour_manual(name = "Tree type", labels= c("Multibit tree w/o Symdex", "Union bit
  ↪ tree w/o Symdex", "Multibit tree + Symdex", "Union bit tree + Symdex"),
  ↪ values=c("#004c93", "#56B1F7", "#000000", "#c43f4b")) +
236   xlab("File Size") + ylab("Time in Minutes") +
237   theme_bw() + facet_grid(l ~ Threshold, labeller = label_both) +
238   theme(panel.spacing = unit(1, "lines"), text = element_text(size=27), axis.text.x =
  ↪ element_text(size = 21, angle = 90, vjust = 0.5), axis.title=element_text(size=30),
  ↪ legend.justification=c(0.9,0.05), legend.key = element_blank(), legend.key.size =
  ↪ unit(5,"char"), strip.background = element_rect(colour='white',fill = 'white'))
239 show(p5)
240 ggsave("Times_MBT_type.pdf",height=13, width=21, units="in", device=cairo_pdf)
241

```

```

242 # Times vs Tree type for l = 500/l=1000
243 subs <- subset(res, l == 500 | l == 1000)
244
245 p5 <- ggplot(subs, aes(x=as.numeric(n), y=time, color=Variant, group=Variant,
↳ shape=Variant)) +
246   geom_point(size=5) + geom_line(size=1) +
247   scale_x_continuous(breaks = seq(0,500000,50000)) +
248   scale_shape_manual(name = "Tree type", labels= c("Multibit tree w/o Symdex","Union bit
↳ tree w/o Symdex","Multibit tree + Symdex","Union bit tree + Symdex"),
↳ values=c(15:18)) +
249   scale_colour_manual(name = "Tree type", labels= c("Multibit tree w/o Symdex","Union bit
↳ tree w/o Symdex","Multibit tree + Symdex","Union bit tree + Symdex"),
↳ values=c("#004c93", "#56B1F7", "#000000", "#c43f4b")) +
250   xlab("File Size") + ylab("Time in Minutes") +
251   theme_bw() + facet_grid(l ~ Threshold, labeller = label_both) +
252   theme(panel.spacing = unit(1, "lines"), text = element_text(size=27), axis.text.x =
↳ element_text(size = 21, angle = 90, vjust = 0.5), axis.title=element_text(size=30),
↳ legend.justification=c(0.9,0.05), legend.key = element_blank(), legend.key.size =
↳ unit(5,"char"), strip.background = element_rect(colour='white',fill = 'white'))
253 show(p5)
254 ggsave("Times_MBT_type.pdf",height=13, width=21, units="in", device=cairo_pdf) # Fig 3.10

```

Source Code D.3: Source code for all result plots.

```

1 ##### PLOTTING #####
2
3
4 # Load libs
5 library(ggplot2)
6 library(grid)
7 library(gridExtra)
8 library(colorspace)
9 library(RColorBrewer)
10 library(rsm)
11 library(boot)
12 library(randomForest)
13 library(stargazer)
14 library(xtable)
15 library(dplyr)
16 library(effects)

```

```
17 library(cvTools)
18 library(car)
19 library(gmodels)
20 library(MASS)
21 library(devtools)
22 install_github("markwestcott34/stargazer-booktabs")
23
24 #setwd("E:/Priv/Dropbox/Dissertation/Programme")
25
26
27 ## Bootstrap mean function
28 boot.mean = function(x,B,confidence = 0.95,binwidth = NULL){
29   require(boot)
30   n = length(x)
31   boot.samples = matrix( sample(x,size = n*B,replace = TRUE), B, n)
32   boot.statistics = apply(boot.samples,1,mean)
33   se = sd(boot.statistics)
34   require(ggplot2)
35   if ( is.null(binwidth) )
36     binwidth = diff(range(boot.statistics))/30
37   p = ggplot(data.frame(x = boot.statistics),aes(x = x))
38     ↪ +geom_histogram(aes(y=..density..),binwidth = binwidth) + geom_density(color =
39     ↪ "red")
40   #plot(p)
41   ciLow = mean(x) + -1*qnorm(1-(confidence/2))*se
42   ciHigh = mean(x) + 1*qnorm(1-(confidence/2))*se
43   # print( ciLow, ciHigh )
44   return(data.frame(ciLow = ciLow, ciHigh = ciHigh, se = se) )
45 }
46
47 # Datasets list
48 datasets <- c("German Mortality", "NC Voter", "German Telephone CD 20% errors", "German
49 ↪ Telephone CD 0% errors", "FEBRL", "FEBRL WA", "Mortality Test Data")
50
51 # # Read data
52 plotdata <-
53   ↪ as.data.frame(read.csv("../Results/R_Metainfo_Testbett_CompareOPC_vs_Train.csv", sep =
54   ↪ "\t"), stringsAsFactors = FALSE)
```

```

53
54 # Make sure central measures are numeric
55 plotdata$recall <- as.numeric(plotdata$recall)
56 plotdata$precision <- as.numeric(plotdata$precision)
57 plotdata$Fmeasure <- as.numeric(plotdata$Fmeasure)
58
59
60 # Read training file for model building
61 traindata <- read.csv("Complete_Results.csv", sep = "\t")
62
63 # Subset to fixed tanimoto threshold
64 fixed <- subset(traindata, Tani == 0.9)
65
66
67 # Build new complete data set with evaluation data results for rapid model comparisons
68 trainresultsFull <- rbind(read.csv("Trainresults_FEBRLWA.csv", sep =
  ↪ "\t"),read.csv("Trainresults_Mortality.csv", sep = "\t"))
69 trainresultsFull$l <- 500
70 trainresultsFull$missingamount.1 <- NULL
71
72 trainresultsFull <- rbind(trainresultsFull, read.csv("Trainresults_NCVoter.csv", sep =
  ↪ "\t"))
73 trainresultsFull$type <- NULL
74
75 kill <- c("tp", "fp", "fn", "tn", "recall", "precision")
76 fulldata <- fixed[(!names(fixed) %in% kill)]
77 fulldata$variant <- NULL
78 fulldata$positives <- NULL
79 fulldata$i <- 1
80
81 fulldata <- rbind(fulldata,trainresultsFull)
82 fixed <- subset(fulldata, Tani == 0.9)
83
84
85 # RSM Plot
86 # RSM: Preproc data
87 RSMdata <- subset(fulldata, Tani == 0.9)[,c("k", "q", "u", "meanengrams", "hammingweight",
  ↪ "Fmeasure")]
88 RSMdata_unscaled <- subset(fulldata, Tani == 0.9)[,c("k", "q", "u", "meanengrams",
  ↪ "hammingweight", "Fmeasure")]
89

```

```

90 # Scale min/max
91 maxi = apply(RSMdata , 2, function(x) max(as.numeric(x)))
92 mini = apply(RSMdata, 2, function(x) min(as.numeric(x)))
93 # Scaling
94 RSMdata = as.data.frame(scale(RSMdata, center = mini, scale = maxi - mini))
95 # Train model
96 rsm.train <- rsm(Fmeasure ~ SO(k, q) + TWI(q, k, hammingweight) + u, data = RSMdata)
97
98
99
100 # Fixed model calls
101 fullmod <- lm(Fmeasure ~ k + (q) + errorestimation + skew + uniqueness + meanentropy +
  ↪ log(u) + ginicoefficient + uniquepatternsA + errorestimation + k * q *
  ↪ hammingweight + meanengrams*hammingweight +
102             meanmissing + log(m) + u*m,
103             data = fixed)
104
105 minmod <- lm(Fmeasure ~ k + (q), data = fixed)
106
107
108
109 stepTest <- stepAIC(fullmod, minmod, direction = "both", k = log(nrow(fixed))) # bic
110 #stepTest2 <- stepAIC(fullmod, minmod, direction = "both", k = 2) # aic
111 BIC2 <- stepTest
112
113
114 # LM
115 LM <- lm(Fmeasure ~ k + (q) + errorestimation * skew + uniquepatternsA *
116         uniqueness + ginicoefficient + k * hammingweight +
117         hammingweight * meanengrams + uniquepatternsA + log(u),
118         data = fixed)
119
120
121 # Random forest
122 r1 <- randomForest(Fmeasure ~ k + q + errorestimation + skew + uniqueness +
  ↪ ginicoefficient + hammingweight +
123             meanengrams + u + uniquepatternsA, data = fixed,
124             mtry= 3, ntree = 500, nodesize = 20,
125             importance = TRUE, nPerm = 5, keep.forest = TRUE)
126
127

```

```

128 fileConn<-file("Models.tex")
129
130 # Model regression tables
131 writeLines(stargazer(LM, BIC2, title = "Regression models",align = TRUE, omit.stat =
  ↪ c("f"), no.space = TRUE, type = "latex",
132     column.labels = c("Simple linear model", "BIC-selected linear model"), digits =
  ↪ 3, digits.extra = 0,
133     ci = TRUE,
134     ci.separator = ";",
135     star.char = c("+","*"),
136     star.cutoffs = c(0.05,0.001),
137     notes = c("$^{+}$p$<$0.05", "$^{*}$p$<$0.001"),
138     notes.append = FALSE,
139 covariate.labels = c("k", "q", "errorestimation",
140     "skewness", "uniquepattA", "uniqueness", "meanentropy", "gini",
141     "hammingweight", "meanngr", "meanmissing", "log(m)", "k*q", "log(u)",
142     "errorest*skewness", "uniquepattA*uniqueness", "k*hammingweight",
143     "hammingweight*meanngr", "q*hammingweight", "k*q*hammingweight"),
144 dep.var.caption = "Mean of precision and recall",
145 dep.var.labels = "MPR")[17:63], fileConn)
146 close(fileConn)
147
148
149 # RSM Surfaces
150 cairo_pdf(file = "SO_RSM.pdf", width = 20, height = 10) # Fig 4.8
151 par(mfrow = c(1,2))
152 contour(rsm.train, ~ hammingweight + k, image = TRUE, main = "Second-order RS contour
  ↪ plot", xlabs = c("k Hash functions", "q-grams"), cex.lab = 1.3, cex.axis = 1.2,
  ↪ labcex = 1.4)
153 persp(rsm.train, hammingweight ~ k, xlabs = c("\nStandard. k Hash functions", "\nStandard.
  ↪ q-grams"),
154     zlab = "\nPredicted Standard. MPR", main = "Second-order response surface", theta =
  ↪ 150, phi = 30, cex.lab = 1.6, cex.axis = 1.2, contours = TRUE)
155 dev.off()
156
157
158 cairo_pdf(file = "testResult_BIC_Multidata.pdf", width=20, height=10) # Fig 5.2
159 par( mfrow=c(1, 2) )
160 fullmod <- lm(Fmeasure ~ k + (q) + Tani + errorestimation + skew + uniqueness +
  ↪ meanentropy + log(u) + ginicoefficient + uniquepatternsA + errorestimation + k * q *
  ↪ hammingweight + meanengrams*hammingweight +

```

```

161         meanmissing + log(m) + u*m,
162         data = subset(traindata, data=="FEBRL"))
163
164 persp(fullmod, k ~ Tani, zlab = "\n\nMPR", theta = -120, phi = 25,
165        col="gray40", border="gray10", main = "FEBRL",
166        xlabs =rev(c("\nTanimoto threshold","\nHash functions")))
167
168 fullmod <- lm(Fmeasure ~ k + (q) + Tani + errorestimation + skew + uniqueness +
169 ↪ meanentropy + log(u) + ginicoefficient + uniquepatternsA + errorestimation + k * q *
170 ↪ hammingweight + meanengrams*hammingweight +
169         meanmissing + log(m) + u*m,
170         data = subset(traindata, data=="German Mortality"))
171 persp(fullmod, k ~ Tani , zlab = "\n\nMPR", theta = -120, phi = 25,
172        col="gray40", border="gray10", main = "Mortality & Hospital data",
173        xlabs =rev(c("\nTanimoto threshold","\nHash functions")))
174 dev.off()
175
176
177
178 # Fit plot BIC
179 newdata <- subset(fulldata, Tani == 0.9 & data == "NC Voter" & q == 1)
180 newfittedBIC <- data.frame(newdata, predicted = predict(BIC2, newdata, type = "response"))
181
182 cairo_pdf(file = "Predicted_vs_F.pdf", width = 18, height = 15) # Fig 5.7
183 par(mgp = c(3,1,5), mar = c(4.5,4.5,1,1))
184 plot(newfittedBIC$Fmeasure ~ newfittedBIC$predicted, ylab = "\nF-Measure", xlab =
185 ↪ "Predicted F-measure", cex = 2, xlim = c(0.84,0.92), ylim = c(0.84,0.92),
186 ↪ cex.axis = 2,cex.lab = 2,cex.main = 2.2) # xlim = c(0.84,0.95), ylim = c(0.84,0.95)
187 segments(0,0,1,1, lty = "dotted")
188 dev.off()
189
190 # Fit plot RF
191 newdata <- subset(fulldata, Tani == 0.9 & data == "NC Voter")
192 newdata$fitForest <- predict(r1, newdata)
193 ggplot(newdata, aes(y = Fmeasure, x = fitForest, color = factor(q), group = q)) +
194 ↪ xlab("RF-predicted mean prec./rec.") + ylab("Real mean prec./rec") +
195 ↪ scale_colour_manual(name = "q", labels= as.character(1:4), values = c("#004c93",
196 ↪ "#56B1F7", "#c43f4b", "#000000")) +
197 ↪ geom_point(size = 6) + geom_line(size = 1) +
198 ↪ # coord_fixed(xlim = c(0.82, 0.97), ylim = c(0.82,0.97)) +

```

```

198 theme_bw() + theme(panel.spacing = unit(0.25, 'lines'),
199     text = element_text(size = 22),
200     axis.title = element_text(size = 28),
201     legend.justification = c(0.9,0.05),
202     legend.key = element_blank(),
203     legend.key.size = unit(5,"char"),
204     strip.background = element_rect(colour='white',fill = 'white')) +
205 ggtitle(paste("Data:", unique(newdata$data)))
206 ggsave("RF_predict_vs_fit.pdf", width = 15, height = 12, unit = "in", device=cairo_pdf) #
    ↪ Fig 5.8
207
208
209
210
211
212
213 # Crossvalidate
214 crossvalids <- NULL
215 for (d in datasets){
216   fixedCV <- subset(fixed, data == d)
217   fixedCV$kx <- fixedCV$k
218   fixedCV$hw <- fixedCV$hammingweight
219   cvBICs <- cvFit (BIC2, data = fixedCV, y = fixedCV$Fmeasure,
220     cost = mape, K = 5, R = 30, includeSE = TRUE)
221   cvLMs <- cvFit (LM, data = fixedCV, y = fixedCV$Fmeasure,
222     cost = mape, K = 5, R = 30, includeSE = TRUE)
223   cvRSMs <- cvFit (rsm.train, data = fixedCV, y = fixedCV$Fmeasure,
224     cost = mape, K = 5, R = 30, includeSE = TRUE)
225   cvRFs <- cvFit (r1, data = fixedCV, y = fixedCV$Fmeasure,
226     cost = mape, K = 5, R = 30, includeSE = TRUE)
227   crossvalids <- rbind(crossvalids, data.frame(data = d,
228     cvBIC = cvBICs[["cv"]][["CV"]], seBIC = cvBICs[["se"]][["CV"]],
229     cvLM = cvLMs[["cv"]][["CV"]], seLM = cvLMs[["se"]][["CV"]],
230     cvRSM = cvRSMs[["cv"]][["CV"]], seRSM = cvRSMs[["se"]][["CV"]],
231     cvRF = cvRFs[["cv"]][["CV"]], seRF = cvRFs[["se"]][["CV"]]))
232 }
233
234 # Reorder
235 crossvalids <- crossvalids[c(4,3,5,1,2,7,6),]
236
237 # Write all diagnostics to tex file

```

```

238 fileConn<-file("Diagnostics.tex")
239
240 # LM
241 sink(fileConn)
242
243 print(xtable(
244   data.frame(data = aggregate(sub$Fmeasure[sub$variant == "Simple linear model"], by =
245     ↪ list(sub$data[sub$variant == "Simple linear model"]),
246       mean, na.action = na.exclude)$Group.1, meanF =
247     ↪ aggregate(sub$Fmeasure[sub$variant == "Simple linear model"],
248       by = list(sub$data[sub$variant == "Simple linear model"]),
249       mean, na.action= na.exclude)$x, cv = crossvalids$cvLM, se = crossvalids$seLM,
250       RMSE = aggregate(sub$RMSE[sub$variant == "Simple linear model"],
251       by = list(sub$data[sub$variant == "Simple linear model"]),
252       mean, na.action= na.exclude)$x),
253   digits = 3, display = c("s", "s", "f", "e", "e", "f")),math.style.exponents = TRUE,
254   ↪ booktabs = TRUE, include.rownames = FALSE)
255
256 sink(fileConn, append = TRUE)
257
258 # BIC
259 print(xtable(
260   data.frame(data = aggregate(sub$Fmeasure[sub$variant == "BIC-selected linear model"],
261     by = list(sub$data[sub$variant == "BIC-selected linear model"]), mean,
262     ↪ na.action= na.exclude)$Group.1,
263   meanF = aggregate(sub$Fmeasure[sub$variant == "BIC-selected linear model"],
264     by = list(sub$data[sub$variant == "BIC-selected linear model"]), mean,
265     ↪ na.action= na.exclude)$x,
266   cv = crossvalids$cvBIC, se = crossvalids$seBIC, RMSE =
267     ↪ aggregate(sub$RMSE[sub$variant == "BIC-selected linear model"],
268     by = list(sub$data[sub$variant == "BIC-selected linear model"]),
269     mean, na.action= na.exclude)$x),
270   digits = 3, display = c("s", "s", "f", "e", "e", "f")),math.style.exponents = TRUE,
271   ↪ booktabs = TRUE, include.rownames = FALSE)
272
273 sink(fileConn, append = TRUE)
274
275 # RSM
276 print(xtable(
277   data.frame(data = aggregate(sub$Fmeasure[sub$variant == "Optimal Parameter choice
278     ↪ (RSM)"],

```

```

271     by = list(sub$data[sub$variant == "Optimal Parameter choice (RSM)"]), mean,
      ↪ na.action= na.exclude)$Group.1,
272     meanF = aggregate(sub$Fmeasure[sub$variant == "Optimal Parameter choice
      ↪ (RSM)"]),
273     by = list(sub$data[sub$variant == "Optimal Parameter choice (RSM)"]), mean,
      ↪ na.action= na.exclude)$x,
274     cv = crossvalids$cvRSM, se = crossvalids$seRSM,
275     RMSE = aggregate(sub$RMSE[sub$variant == "Optimal Parameter choice (RSM)"]),
276     by = list(sub$data[sub$variant == "Optimal Parameter choice (RSM)"]), mean,
      ↪ na.action= na.exclude)$x),
277     digits = 3, display = c("s", "s", "f", "e", "e", "f")),math.style.exponents = TRUE,
      ↪ booktabs = TRUE, include.rownames = FALSE)
278
279
280 sink(fileConn, append = TRUE)
281
282
283
284
285
286 # RF
287 print(xtable(
288     data.frame(data = aggregate(sub$Fmeasure[sub$variant == "Optimal Parameter choice
      ↪ (Forests)"]),
289             by = list(sub$data[sub$variant == "Optimal Parameter choice (Forests)"]),
      mean, na.action= na.exclude)$Group.1,meanF =
      ↪ aggregate(sub$Fmeasure[sub$variant == "Optimal Parameter choice
      ↪ (Forests)"]),
291     by = list(sub$data[sub$variant == "Optimal Parameter choice (Forests)"]),
      mean, na.action= na.exclude)$x, cv = crossvalids$cvRF, se = crossvalids$seRF,
292     RMSE = aggregate(sub$RMSE[sub$variant == "Optimal Parameter choice
      ↪ (Forests)"]),
293     by = list(sub$data[sub$variant == "Optimal Parameter choice (Forests)"]),
      ↪ mean, na.action= na.exclude)$x),
294     digits = 3, display = c("s", "s", "f", "e", "e", "f")),math.style.exponents = TRUE,
      ↪ booktabs = TRUE, include.rownames = FALSE)
295
296
297 sink()
298 close(fileConn)
299
300

```

```

301
302 # Plots by measure
303
304 # Set colors, preprocess data
305 set.seed(19)
306 cbbPalette <- (c("#000000", "#2643dc",
307                 "#5ca03f",
308                 "#344494",
309                 "#c43f4b",
310                 "#ce5d2c", "#555555"))
311
312
313 set.seed(19)
314 shapes <- sample(21:25, length(unique(plotdata$variant)), replace = TRUE)
315
316 plotdata$data <- factor(plotdata$data,
317                         levels = c("FEBRL WA", "Mortality Test Data", "FEBRL", "German
↳ Mortality", "German Telephone CD 0% errors", "German
↳ Telephone CD 20% errors", "NC Voter"),
318                         labels = c("FEBRL WA", "Mortality & Commercial", "FEBRL",
↳ "Mortality & Hospital", "Telephone CD 0% errors", "Telephone
↳ CD 20% errors", "NC Voter"))
319
320 plotdata$variant <- factor(plotdata$variant,
321                           levels = c("Optimal Parameter choice (RSM)", "Optimal
↳ Parameter choice (RF)", "Optimal Parameter choice (NN)",
↳ "Optimal Parameter choice (BIC)",
322                                       "Optimal Parameter choice (LM)", "50%-Rule Parameter
↳ choice", "Standard CLK k = 15" , "Standard
↳ CLK k = 10" , "Standard CLK k = 5"),
323                           labels = c("Optimal Parameter choice (RSM)", "Optimal Parameter choice (Forests)",
↳ "Optimal Parameter choice (NN)", "BIC-selected linear model",
324                                       "Simple linear model", "50%-Rule Parameter choice", "Standard CLK k = 15" ,
↳ "Standard CLK k = 10" , "Standard CLK k = 5"))
325
326 # No NN, no k = 15
327 plotdata <- subset(plotdata, plotdata$variant != "Optimal Parameter choice (NN)" &
↳ plotdata$variant != "Standard CLK k = 15")
328 plotdata$variant <- droplevels(plotdata$variant)
329
330

```

```

331 # Main Plot: F-Measure of all variants by Tanimoto-Threshold and errors
332 p1 <- ggplot(subset(plotdata, data == "Mortality & Commercial" | data == "NC Voter" | data
  ↪ == "FEBRL WA" ))
333 p1 <- p1 + geom_line(aes(y = Fmeasure, x = Tani, group = variant, colour = variant), size =
  ↪ 1)
334 p1 <- p1 + geom_point(aes(y = Fmeasure, x = Tani, group = variant, shape = variant, colour =
  ↪ variant, fill = variant), size = 7)
335 p1 <- p1 + xlab("\nTanimoto threshold") + ylab("Mean Prec./Rec.\n")
336 p1 <- p1 + scale_colour_manual(name = "Variant", labels= levels(plotdata$variant), values
  ↪ = cbbPalette)
337 p1 <- p1 + scale_fill_manual(name = "Variant", labels= levels(plotdata$variant), values =
  ↪ cbbPalette)
338 p1 <- p1 + scale_shape_manual(name = "Variant", labels= levels(plotdata$variant), values =
  ↪ shapes)
339 p1 <- p1 + ggtitle("Test data")
340
341 p1 + facet_grid(. ~ data) + theme_bw() + theme(panel.spacing = unit(0.85, "lines"),
342       text = element_text(size = 27), axis.text.x = element_text(size = 23, angle = 90,
  ↪ vjust = 0.5),
343       axis.title = element_text(size = 30), legend.justification = c(0.9,0.05),
344       legend.key = element_blank(), legend.key.size = unit(5,"char"),
345       strip.background = element_rect(colour='white',fill = 'white'))
346 # Save Plot
347 ggsave("./results/Testbett_EncryptionResults_F-Score_Testdata_experimental.pdf", height =
  ↪ 11, width = 21, units = "in", device=cairo_pdf) # Fig 5.14
348
349
350 # Main Plot: F-Measure of all variants by Tanimoto-Threshold and errors
351 p1 <- ggplot(subset(plotdata, data != "Mortality & Commercial" & data != "NC Voter" & data
  ↪ != "FEBRL WA" ))
352 p1 <- p1 + geom_line(aes(y = Fmeasure, x = Tani, group = variant, colour = variant), size =
  ↪ 1)
353 p1 <- p1 + geom_point(aes(y = Fmeasure, x = Tani, group = variant, shape = variant, colour =
  ↪ variant, fill = variant), size = 7)
354 p1 <- p1 + xlab("\nTanimoto threshold") + ylab("Mean Prec./Rec.\n")
355 p1 <- p1 + scale_colour_manual(name = "Variant", labels= levels(plotdata$variant), values
  ↪ = cbbPalette)
356 p1 <- p1 + scale_fill_manual(name = "Variant", labels= levels(plotdata$variant), values =
  ↪ cbbPalette)
357 p1 <- p1 + scale_shape_manual(name = "Variant", labels= levels(plotdata$variant), values =
  ↪ shapes)

```

```

358 p1 <- p1 + ggtitle("Training data")
359 p1 + facet_grid(. ~ data) + theme_bw() + theme(panel.spacing = unit(0.85, "lines"), text
  ↪ = element_text(size = 27), axis.text.x = element_text(size = 23, angle = 90, vjust =
  ↪ 0.5),
360     axis.title = element_text(size = 30), legend.justification = c(0.9,0.05),
361     legend.key = element_blank(), legend.key.size = unit(5,"char"), strip.background =
  ↪ element_rect(colour='white',fill = 'white'))
362 # Save Plot
363 ggsave("./results/Testbett_EncryptionResults_F-Score_experimental.pdf", height = 11,
  ↪ width = 21, units = "in", device=cairo_pdf) # Fig 5.13
364
365
366 # Sub Plot: Recall of all variants by Tanimoto-Threshold and errors
367 p2 <- ggplot(subset(plotdata, data != "Mortality & Commercial" & data != "NC Voter" & data
  ↪ != "FEBRL WA" ))
368 p2 <- p2 + geom_line(aes(y = recall, x = Tani,group = variant, colour = variant), size = 1)
369 p2 <- p2 + geom_point(aes(y = recall,x = Tani,group = variant, shape = variant, colour =
  ↪ variant, fill = variant), size = 7)
370 p2 <- p2 + xlab("\nTanimoto threshold") + ylab("Recall\n")
371 p2 <- p2 + scale_colour_manual(name = "Variant", labels= levels(plotdata$variant), values
  ↪ = cbbPalette)
372 p2 <- p2 + scale_fill_manual(name = "Variant", labels= levels(plotdata$variant), values =
  ↪ cbbPalette)
373 p2 <- p2 + scale_shape_manual(name = "Variant", labels= levels(plotdata$variant), values =
  ↪ shapes)
374 p2 <- p2 + ggtitle("Training data")
375 p2 + facet_grid(. ~ data) + theme_bw() + theme(panel.spacing = unit(0.85, "lines"), text
  ↪ = element_text(size = 27),
376     axis.text.x = element_text(size = 23, angle = 90, vjust = 0.5), axis.title =
  ↪ element_text(size = 30),
377     legend.justification = c(0.9,0.05), legend.key = element_blank(), legend.key.size
  ↪ = unit(5,"char"),
378     strip.background = element_rect(colour='white',fill = 'white'))
379 # Save Plot
380 ggsave("./results/Testbett_EncryptionResults_Recall_experimental.pdf", height = 11, width
  ↪ = 21, units = "in", device=cairo_pdf) # Fig 5.9
381
382
383 # Sub Plot: Recall of all variants by Tanimoto-Threshold and errors
384 p2 <- ggplot(subset(plotdata, data == "Mortality & Commercial" | data == "NC Voter" |
  ↪ data == "FEBRL WA" ))

```

```

385 p2 <- p2 + geom_line(aes(y = recall, x = Tani,group = variant, colour = variant), size = 1)
386 p2 <- p2 + geom_point(aes(y = recall,x = Tani,group = variant, shape = variant, colour =
  ↪ variant, fill = variant), size = 7)
387 p2 <- p2 + xlab("\nTanimoto threshold") + ylab("Recall\n")
388 p2 <- p2 + scale_colour_manual(name = "Variant", labels= levels(plotdata$variant), values
  ↪ = cbbPalette)
389 p2 <- p2 + scale_fill_manual(name = "Variant", labels= levels(plotdata$variant), values =
  ↪ cbbPalette)
390 p2 <- p2 + scale_shape_manual(name = "Variant", labels= levels(plotdata$variant), values =
  ↪ shapes)
391 p2 <- p2 + ggtitle("Test data")
392 p2 + facet_grid(. ~ data) + theme_bw() + theme(panel.spacing = unit(0.85, "lines"), text
  ↪ = element_text(size = 27),
393     axis.text.x = element_text(size = 23, angle = 90, vjust = 0.5), axis.title =
  ↪ element_text(size = 30),
394     legend.justification = c(0.9,0.05), legend.key = element_blank(), legend.key.size
  ↪ = unit(5,"char"),
395     strip.background = element_rect(colour='white',fill = 'white'))
396 # Save Plot
397 ggsave("./results/Testbett_EncryptionResults_Recall_experimental_Testdata.pdf", height =
  ↪ 11, width = 21, units = "in", device=cairo_pdf) # Fig 5.10
398
399
400 # Sub Plot: Precision of all variants by Tanimoto-Threshold and errors
401 p3 <- ggplot(subset(plotdata, data != "Mortality & Commercial" & data != "NC Voter" & data
  ↪ != "FEBRL WA" ))
402 p3 <- p3 + geom_line(aes(y = precision, x = Tani,group = variant, colour = variant), size
  ↪ = 1)
403 p3 <- p3 + geom_point(aes(y = precision,x = Tani,group = variant, shape = variant, colour
  ↪ = variant, fill = variant), size = 7)
404 p3 <- p3 + xlab("\nTanimoto threshold") + ylab("Precision\n")
405 p3 <- p3 + scale_colour_manual(name = "Variant", labels= levels(plotdata$variant), values
  ↪ = cbbPalette)
406 p3 <- p3 + scale_fill_manual(name = "Variant", labels= levels(plotdata$variant), values =
  ↪ cbbPalette)
407 p3 <- p3 + scale_shape_manual(name = "Variant", labels= levels(plotdata$variant), values =
  ↪ shapes)
408 p3 <- p3 + ggtitle("Training data")
409 p3 + facet_grid(. ~ data) + theme_bw() + theme(panel.spacing = unit(0.85, "lines"),
410     text = element_text(size = 27), axis.text.x = element_text(size = 23, angle = 90,
  ↪ vjust = 0.5),

```

```

411   axis.title = element_text(size = 30), legend.justification = c(0.9,0.05),
412   legend.key = element_blank(), legend.key.size = unit(5,"char"),
413   strip.background = element_rect(colour='white',fill = 'white'))
414 # Save Plot
415 ggsave("./results/Testbett_EncryptionResults_Precision_experimental.pdf", height = 11,
416   ↪ width = 21, units = "in", device=cairo_pdf) # Fig 5.11
417
418 # Sub Plot: Precision of all variants by Tanimoto-Threshold and errors
419 p3 <- ggplot(subset(plotdata, data == "Mortality & Commercial" | data == "NC Voter" | data
420   ↪ == "FEBRL WA" ))
421 p3 <- p3 + geom_line(aes(y = precision, x = Tani,group = variant, colour = variant), size
422   ↪ = 1)
423 p3 <- p3 + geom_point(aes(y = precision,x = Tani,group = variant, shape = variant, colour
424   ↪ = variant, fill = variant), size = 7)
425 p3 <- p3 + xlab("\nTanimoto threshold") + ylab("Precision\n")
426 p3 <- p3 + scale_colour_manual(name = "Variant", labels= levels(plotdata$variant), values
427   ↪ = cbbPalette)
428 p3 <- p3 + scale_fill_manual(name = "Variant", labels= levels(plotdata$variant), values =
429   ↪ cbbPalette)
430 p3 <- p3 + scale_shape_manual(name = "Variant", labels= levels(plotdata$variant), values =
431   ↪ shapes)
432 p3 <- p3 + ggtitle("Test data")
433 p3 + facet_grid(. ~ data) + theme_bw() + theme(panel.spacing = unit(0.85, "lines"), text
434   ↪ = element_text(size = 27),
435   axis.text.x = element_text(size = 23, angle = 90, vjust = 0.5), axis.title =
436   ↪ element_text(size = 30),
437   legend.justification = c(0.9,0.05), legend.key = element_blank(), legend.key.size
438   ↪ = unit(5,"char"),
439   strip.background = element_rect(colour='white',fill = 'white'))
440 # Save Plot
441 ggsave("./results/Testbett_EncryptionResults_Precision_experimental_Testdata.pdf", height
442   ↪ = 11, width = 21, units = "in", device=cairo_pdf) # Fig 5.12
443
444
445
446
447
448
449 ## Central dot chart: MPR by method at T = 0.9
450
451 sub <- subset(plotdata, Tani == 0.9) # & data != "German Telephone CD 0% errors")
452 sub$data <- droplevels(sub$data)

```

```

441 sub$variant <- reorder(sub$variant, sub$Fmeasure, mean)
442 sub$data <- reorder(sub$data, -sub$Fmeasure, min)
443
444 sub <- sub[order(sub$Fmeasure),]
445
446 # Central Dotchart with ggplot
447 p4 <- ggplot(sub, aes(x = variant, y = Fmeasure, colour = data))
448 p4 <- p4 + geom_point(shape = 16, size = 7) + facet_wrap(~ data, ncol = 1) + coord_flip()
449 p4 <- p4 + xlab("") + ylab("\nMean Prec./Rec.") + guides(colour = FALSE) +
450   labs(caption = "All values at a Tanimoto threshold of T = 0.9")
451 p4 <- p4 + theme_bw() + theme(panel.spacing = unit(0.25, 'lines'),
452   text = element_text(size = 22),
453   axis.title = element_text(size = 28),
454   legend.justification = c(0.9,0.05),
455   legend.key = element_blank(),
456   legend.key.size = unit(5,"char"),
457   strip.background = element_rect(colour='white',fill = 'white'),
458   plot.caption = element_text(size = 16, hjust = -.7, vjust = 0.1),
459   panel.grid.major.x = element_blank() ,
460   panel.grid.minor.x = element_blank() ,
461   axis.text.x = element_text(colour = "#111111"),
462   axis.text.y = element_text(size = 22, colour = "#111111"),
463   ## explicitly set the horizontal lines (or they will disappear too)
464   panel.grid.major.y = element_line(color = "#444444", linetype = 3))
465 show(p4)
466 #Save the plot
467 ggsave("./results/Dotchart_F-Score_OptTrain_experimental.pdf", height = 19, width = 13,
468   ↪ units = "in", device=cairo_pdf) # Fig 5.15
469
470
471
472 ## Diagnostic plots
473
474
475 # Random forest diagnostics plot
476
477 modelname <- r1
478 fitted <- r1$predicted
479 truescore <- fixed$Fmeasure
480 resid <- truescore - fitted

```

```

481 stdresid <- resid / sd(resid)
482
483
484 cooksd <- rep(0, length(fitted))
485 for (i in 1:length(fitted)){
486
487   cooksd[i] <- sum((fitted[-i] - predict(r1, newdata = fixed[-i,]))^2) / var(r1$mse) * 6
488
489 }
490
491
492 model <- data.frame(fitted, resid, stdresid, truescore, cooksd, data = fixed$data, q =
  ↪ factor(fixed$q))
493
494 p1 <- ggplot(model, aes(fitted, resid))+geom_point(size = 2, aes(color = q))
495 p1 <- p1 + stat_smooth(method = "loess")+geom_hline(yintercept = 0, col = "red", linetype
  ↪ = "dashed")
496 p1 <- p1 + xlab("Fitted values") + ylab("Residuals")
497 p1 <- p1 + ggtitle("Residual vs Fitted Plot") + theme_bw() + facet_grid(data ~ .) +
498   theme(panel.spacing = unit(0.25, 'lines'),
499     text = element_text(size = 22),
500     axis.title = element_text(size = 28),
501     legend.justification = c(0.9,0.05),
502     legend.key = element_blank(),
503     legend.key.size = unit(5,"char"),
504     strip.text = element_text(size = 8),
505     strip.background = element_rect(colour='white',fill = 'white'))
506
507
508 p2 <- ggplot(model, aes(sample = stdresid)) +
509   stat_qq(size = 2) +
510   stat_qq_line(size = 1) + xlab("Theoretical Quantiles")+ylab("Standardized Residuals")
511 p2 <- p2 + ggtitle("Normal Q-Q")+theme_bw() + theme(panel.spacing = unit(0.25, 'lines'),
512   text = element_text(size = 22),
513   axis.title = element_text(size = 28),
514   legend.justification = c(0.9,0.05),
515   legend.key = element_blank(),
516   legend.key.size = unit(5,"char"),
517   strip.background =
  ↪ element_rect(colour='white',fill =
  ↪ 'white'))

```

```

518
519 p3 <- ggplot(model, aes(fitted, sqrt(abs(stdresid))))+geom_point(aes(color = q),size = 2,
  ↪ na.rm = TRUE)
520 p3 <- p3 + stat_smooth(method = "loess", na.rm = TRUE)+xlab("Fitted Value")
521 p3 <- p3 + ylab(expression(sqrt("|Standardized residuals|")))
522 p3 <- p3 + ggtitle("Scale-Location")+theme_bw() + facet_grid(data ~ .) +
523   theme(panel.spacing = unit(0.25, 'lines'),
524     text = element_text(size = 22),
525     axis.title = element_text(size = 28),
526     legend.justification = c(0.9,0.05),
527     legend.key = element_blank(),
528     legend.key.size = unit(5,"char"),
529     strip.text = element_text(size = 8),
530     strip.background = element_rect(colour='white',fill = 'white'))
531
532 subs <- data.frame(variable = names(importance(modelname)[,2]), importance =
  ↪ importance(modelname)[,2])
533 subs$variable <- reorder(subs$variable, subs$importance, mean)
534
535 # Dotchart with ggplot
536 p4 <- ggplot(subs, aes(x = variable, y = importance)) +ggtitle("Variable Importance")+
537   geom_point(shape = 16, size = 4) + coord_flip() + xlab("") + ylab("\nImportance") +
538   theme_bw() + theme(panel.spacing = unit(0.25, 'lines'),
539     text = element_text(size = 22),
540     axis.title = element_text(size = 28),
541     legend.justification = c(0.9,0.05),
542     legend.key = element_blank(),
543     legend.key.size = unit(5,"char"),
544     strip.background = element_rect(colour='white',fill = 'white'),
545     plot.caption = element_text(size = 16, hjust = -.7, vjust = 0.1),
546     panel.grid.major.x = element_blank() ,
547     panel.grid.minor.x = element_blank() ,
548     axis.text.x = element_text(colour = "#111111"),
549     axis.text.y = element_text(size = 22, colour = "#111111"),
550     ## explicitly set the horizontal lines (or they will disappear too)
551     panel.grid.major.y = element_line(color = "#444444", linetype = 3))
552
553 # Save
554 g1 <- arrangeGrob(p1,p3, layout_matrix = rbind(c(1),c(2)))
555 ggsave("./results/Diagnostic_RF.pdf", height = 19, width = 10, units = "in", g1,
  ↪ device=cairo_pdf) # Fig 5.6

```

```

556
557 # Variable importance
558 ggsave("./results/RF_VarImp.pdf", height = 11, width = 11, units = "in", p4,
  ↪ device=cairo_pdf) # Fig 4.12
559
560
561 # Function for all other diagnostic plots
562 diagPlot<-function(model){
563   p1 <- ggplot(model, aes(.fitted, .resid))+geom_point()
564   p1 <- p1 + stat_smooth(method = "loess")+geom_hline(yintercept = 0, col = "red",
  ↪ linetype = "dashed")
565   p1 <- p1 + xlab("Fitted values")+ylab("Residuals")
566   p1 <- p1 + ggtitle("Systematic Residuals vs Fitted?") + theme_bw() +
  ↪ theme(panel.spacing = unit(0.25, 'lines'),
567         text = element_text(size = 18),
568         axis.title = element_text(size = 28),
569         legend.justification = c(0.9,0.05),
570         legend.key = element_blank() )
571
572
573   p2 <- ggplot(model, aes(sample=.stdresid)) + stat_qq() + stat_qq_line() +
574     xlab("Theoretical Quantiles")+ylab("Standardized Residuals")
575   p2 <- p2 + ggtitle("Q-Q Plot: Normally distributed?") + theme_bw() +theme(panel.spacing
  ↪ = unit(0.25, 'lines'),
576     text = element_text(size = 18),
577     axis.title = element_text(size = 28),
578     legend.justification = c(0.9,0.05),
579     legend.key = element_blank() )
580
581   p3 <- ggplot(model, aes(.fitted, sqrt(abs(.stdresid)))) + geom_point(na.rm = TRUE)
582   p3 <- p3 + stat_smooth(method = "loess", na.rm = TRUE) + xlab("Fitted Value")
583   p3 <- p3 + ylab(expression(sqrt("|Standardized residuals|")))
584   p3 <- p3 + ggtitle("Visual test for Homoscedacity") + theme_bw() + theme(panel.spacing
  ↪ = unit(0.25, 'lines'),
585     text = element_text(size = 18),
586     axis.title = element_text(size = 28),
587     legend.justification = c(0.9,0.05),
588     legend.key = element_blank() )
589
590   p4 <- ggplot(model, aes(seq_along(.cooks), .cooks)) + geom_bar(stat = "identity",
  ↪ position = "identity")

```

```

591 p4 <- p4 + xlab("Obs. Number") + ylab("Cook's distance")
592 p4 <- p4 + ggtitle("Cook's distance") + theme_bw() + theme(panel.spacing = unit(0.25,
↪ 'lines'),
593   text = element_text(size = 18),
594   axis.title = element_text(size = 28),
595   legend.justification = c(0.9,0.05),
596   legend.key = element_blank() )
597
598 p5 <- ggplot(model, aes(.hat, .stdresid))+geom_point(aes(size=.cooks), na.rm = TRUE)
599 p5 <- p5 + stat_smooth(method = "loess", na.rm = TRUE)
600 p5 <- p5 + xlab("Leverage")+ylab("Standardized Residuals")
601 p5 <- p5 + ggtitle("Leverage plot + Cooks distance")
602 p5 <- p5 + scale_size_continuous("Cook's Distance", range = c(1,5))
603 p5 <- p5 + theme_bw() + theme(legend.position = "bottom")+ theme(panel.spacing =
↪ unit(0.25, 'lines'),
604   text = element_text(size = 18),
605   axis.title = element_text(size = 28),
606   legend.justification = c(0.9,0.05),
607   legend.key = element_blank() )
608
609 p6 <- ggplot(model, aes(.hat, .cooks))+geom_point(na.rm = TRUE)+stat_smooth(method =
↪ "loess", na.rm = TRUE)
610 p6 <- p6 + xlab("Leverage hii")+ylab("Cook's Distance")
611 p6 <- p6 + ggtitle("Cook's dist vs Leverage hii/(1-hii)")
612 p6 <- p6 + geom_abline(slope = seq(0,3,0.5), color = "gray", linetype = "dashed")
613 p6 <- p6 + theme_bw() + theme(panel.spacing = unit(0.25, 'lines'),
614   text = element_text(size = 18),
615   axis.title = element_text(size = 28),
616   legend.justification = c(0.9,0.05),
617   legend.key = element_blank() )
618
619 return(list(rvfPlot = p1, qqPlot = p2, sclLocPlot = p3, cdPlot = p4, rvlevPlot = p5,
↪ cvlPlot = p6))
620 }
621
622 diagPlts <- diagPlot(BIC2)
623 g1 <- arrangeGrob(diagPlts$rvfPlot, diagPlts$qqPlot, diagPlts$sclLocPlot,
↪ diagPlts$rvlevPlot, layout_matrix = rbind(c(1,2),c(3,4)))
624
625 ggsave("./results/Diagnostic_BIC.pdf", height = 18, width = 15, units = "in", g1,
↪ device=cairo_pdf) # Fig 5.4

```

```

626
627
628
629 diagPlts<-diagPlot(LM)
630 g1 <- arrangeGrob(diagPlts$rvfPlot, diagPlts$qqPlot, diagPlts$sclLocPlot,
631   ↪ diagPlts$rvlevPlot, layout_matrix = rbind(c(1,2),c(3,4)))
632
633 ggsave("./results/Diagnostic_LM.pdf", height = 18, width = 15, units = "in", g1,
634   ↪ device=cairo_pdf) # Fig 4.6/5.3
635
636 diagPlts<-diagPlot(rsm.train)
637 g1 <- arrangeGrob(diagPlts$rvfPlot, diagPlts$qqPlot, diagPlts$sclLocPlot,
638   ↪ diagPlts$rvlevPlot, layout_matrix = rbind(c(1,2),c(3,4)))
639
640 ggsave("./results/Diagnostic_RSM.pdf", height = 18, width = 15, units = "in", g1,
641   ↪ device=cairo_pdf) # Fig 5.5
642
643 # VIFs
644 (vif(LM))
645 (vif(BIC2))
646 (vif(r1))
647 (vif(rsm.train))
648
649
650
651 # Preprocess for T = 0.9 and right ordering of factors
652 sub <- subset(plotdata, Tani == 0.9) # & data != "German Telephone CD 0% errors")
653 sub$data <- droplevels(sub$data)
654 sub$variant <- reorder(sub$variant, sub$Fmeasure, mean)
655 sub$data <- reorder(sub$data, -sub$Fmeasure, min)
656
657 sub <- sub[order(sub$Fmeasure),]
658
659 cairo_pdf("./results/Dotchart_F-Score_OptTrain.pdf", height = 18, width = 15)
660 dotchart(sub$Fmeasure, groups = sub$data, labels = sub$variant, cex = 2.2,
661   xlab = "Mean Prec./Rec.", main = "Mean Prec./Rec. at T = 0.9", gcolor = "#004c93",
662   ↪ pt.cex = 3, pch = 19, lheight = 22)

```

```

662 dev.off()
663
664
665 # Print measures
666 print(aggregate(sub$Fmeasure, by = list(sub$data), mean))
667 print(aggregate(as.numeric(sub$tp), by = list(sub$data,sub$variant), mean))
668 print(aggregate(as.numeric(sub$fp), by = list(sub$data,sub$variant), mean))
669
670
671 # Table q and k by method
672 sub$data <- reorder(sub$data, -sub$Fmeasure, mean)
673 tabs <- subset(sub, !variant %in% c("Optimal Parameter choice (NN)", "Standard CLK k = 5",
674   ↪ "Standard CLK k = 10" ))
675
676 tabs$variant <- droplevels(tabs$variant)
677
678 fileConn<-file("Parameter_Choices.tex")
679 writeLines(print(xtable(cbind(xtabs(formula = k~data + variant, data = tabs),
680   ↪ xtabs(formula = q ~ data + variant, data = tabs))), booktabs = TRUE), fileConn)
681 close(fileConn)
682
683
684 # Table F-Measure + bootstrapped CIs by method
685 tabs <- subset(sub, !variant %in% c("Optimal Parameter choice (NN)"))
686 tab3 <- data.frame(data = aggregate(tabs$Fmeasure, by = list(tabs$variant), mean)$Group.1,
687   meanF = aggregate(tabs$Fmeasure, by = list(tabs$variant), mean)$x,ciLow =
688     ↪ aggregate(tabs$Fmeasure,
689     by = list(tabs$variant), function(x) (boot.mean(x, 1000)$ciLow))$x,ciHigh =
690     ↪ aggregate(tabs$Fmeasure,
691     by = list(tabs$variant), function(x) (boot.mean(x, 1000)$ciHigh))$x)
692 tab3 <- tab3[order(tab3$meanF, decreasing = TRUE),]
693
694
695 fileConn<-file("MPR_by_Variant.tex")
696 writeLines(print(xtable(tab3, digits = 4), booktabs = TRUE, include.rownames = FALSE),
697   ↪ fileConn)
698 close(fileConn)
699
700
701 # Table meta data by data set
702 tabs <- subset(fulldata, q==2 & k ==1 & Tani == 0.9)
703 tabs <- tabs[c(2,7,4,3,1,5,6),]
704 posits <- subset(plotdata, q==2 & Tani == 0.9 & variant == "Standard CLK k = 5")

```

```
698 posits <- posits[c(7,2,3,4,5,6,1),]
699 tabs <- cbind(tabs, posits$realpositives)
700 tabs <- tabs[,c("data", "filesizeA", "filesizeB", "posits$realpositives", "meanentropy",
  ↪ "meanengrams", "skew", "ginicoefficient", "errorestimation", "uniqueness", "m", "u")]
701 tabs <- tabs[c(7,3,4,5,2,6,1),]
702
703 fileConn<-file("Metadata_datasets.tex")
704 writeLines(print(xtable(tabs, digits = 3, booktabs = TRUE, display = c("s", "d", "d",
  ↪ "d", "d", "f", "f", "f", "f", "f", "f", "f", "g", "g"), math.style.exponents = TRUE),
  ↪ include.rownames = FALSE), fileConn)
705 close(fileConn)
```

## Stand-alone Source code

Source Code E.1: Standalone Source code for optimal identifier choice estimation.

```
1 #####
2 #
3 # Estimating optimal identifier choices for input files specified
4 #
5 # Input: At least one readily preprocessed input file. If two files, both must have the
6   ↪ same column names.
7 # Recommended: One or two files with IDs.
8 # Packages required: PPRL, fastdigest, stringdist, entropy, data.table, fastLink, moments,
9   ↪ ineq
10 # User-defined settings in lines 21 to 45.
11 #
12 # This script reads the file(s), generates all measures required for the estimation,
13 # and outputs the optimal identifier choice.
14 #
15 # Output:
16 #   IdentifierChoice.csv
17 #
18 # Christian Borgs, 2019
19 #
20 #####
21 ##### User settings #####
22
23 # Set working dir for script
24 setwd("E:/Priv/Dropbox/Dissertation/Programme")
25 #setwd("~/Dropbox/Dissertation/Programme")
```

```
26
27 # File to run the optimization over
28 inputFile <- "Daten/A_10000_python_R0_C0_0100_percent.csv"
29 # file name of file to link to
30 inputFileB <- "Daten/B_10000_python_R20_C50_0100_percent.csv"
31
32 # Identifiers: Smalles set to test
33 identsStandard <- c("V7","V8")
34 # Identifiers: All possible
35 identsMax <- c("V7","V8","V3","V4","V5","V6","V2")
36 #identsMax <- c("V7","V8","V5","V4","V3","V2","V6","V9")
37 # ID-column. If none, set to FALSE (WARNING: This will be detrimental to the model
  ↪ quality!)
38 IDcol <- "V1"
39 # String-Dinstance columns. For EM weighting
40 Stringidents <- c("V7","V8")
41
42 # Percent of file to sample fromm if file large (1 = no sampling)
43 trainfraction <- 1
44 # Cores of the machine for multithreading
45 cores <- 6
46
47
48
49 #####
50
51 # Libs used
52 library(PPRL)
53 library(data.table)
54 library(fastLink)
55 library(rlist)
56
57 #####
58
59 ## Automate list of identifiers
60 xyc <- list(identsStandard)
61 lenMin <- length(identsStandard)
62 for (j in lenMin:length(identsMax)) {
63   id <- c(identsStandard, identsMax[(lenMin + 1): (j + 1)])
64   id <- c(na.omit(id))
65   xyc <- list.append(xyc, id)
```

```
66 }
67 identsets <- unique(xyc)
68 #
69 # # Temporary
70 # identsets <- list(
71 #   c("Vorname", "Nachname"),
72 #   c("Vorname", "Nachname", "Day"),
73 #   c("Vorname", "Nachname", "Day", "Month"),
74 #   c("Vorname", "Nachname", "Day", "Month", "Year"),
75 #   c("Vorname", "Nachname", "Day", "Month", "Year", "DOB"),
76 #   c("Vorname", "Nachname", "Day", "Month", "Year", "RAND"),
77 #   c("Vorname", "Nachname", "Day", "Month", "Year", "RAND2")
78 # )
79
80 ##### Work with data
81
82 # Read input
83 clearTextA <- fread(inputFile, colClasses = "character", stringsAsFactors = FALSE)
84 clearTextB <- fread(inputFileB, colClasses = "character", stringsAsFactors = FALSE)
85
86
87 clearTextA <- clearTextA[sample(nrow(clearTextA), round(trainfraction*nrow(clearTextA)),
88   ↪ replace = FALSE),]
89 clearTextB <- clearTextB[sample(nrow(clearTextB), round(trainfraction*nrow(clearTextB)),
90   ↪ replace = FALSE),]
91
92 # Check file sizes
93 filesizeA <- as.numeric(nrow(clearTextA))
94 filesizeB <- as.numeric(nrow(clearTextB))
95
96 # Generate ID Col
97 IDA <- as.character(unlist(clearTextA[,..IDcol]))
98 IDB <- as.character(unlist(clearTextB[,..IDcol]))
99
100
101 clearTextAbak <- clearTextA
102 clearTextBbak <- clearTextB
103
104 bestIdentifier <- 0
```

```

105 #bestIdentifierF <- 0
106 #bestIdentifierFName <- ""
107 bestIdentifierName <- ""
108
109 tests <- NULL
110
111 for (idents in identsets){
112
113   cat("\n\n")
114   print(idents)
115
116   # subset the data
117   clearTextA <- clearTextAbak[, ..idents]
118   clearTextB <- clearTextBbak[, ..idents]
119
120   cat("\nActual overlap:", sum(IDA %in% IDB),"\n")
121
122   ##### Start working #####
123
124   # Get EM weights (silently)
125   capture.output(est$ <- fastLink(clearTextA, clearTextB, varnames = idents,
126     stringdist.match=Stringidents[Stringidents %in% idents],
127     ↪ stringdist.method="jw", n.cores = cores, estimate.only = FALSE),
128     ↪ file='NULL')
129
130   # Weights
131   #m <- est$p.m
132   #u <- est$p.u
133
134   # Weights
135   m <- est$EM$p.m
136   u <- est$EM$p.u
137
138   meanAW <- log2(m/u)
139   meanDAW <- log2((1-m)/(1-u))
140
141   print("Agree/Disagree-Weights")
142   print(cbind(meanAW,meanDAW))
143
144   print("Global m and u")

```

```

144 print(cbind(m,u))
145
146 print("Matrix for m")
147 #print(matrix(unlist(ests$p.gamma.k.m), ncol=2, nrow=length(idents), byrow=TRUE,
↪ dimnames=list(idents)))
148 print(matrix(unlist(ests$EM$p.gamma.k.m), ncol=2, nrow=length(idents), byrow=TRUE,
↪ dimnames=list(idents)))
149
150 print("Matrix for u")
151 #print(matrix(unlist(ests$p.gamma.k.u), ncol=2, nrow=length(idents), byrow=TRUE,
↪ dimnames=list(idents)))
152 print(matrix(unlist(ests$EM$p.gamma.k.u), ncol=2, nrow=length(idents), byrow=TRUE,
↪ dimnames=list(idents)))
153
154 #
155 positives <- sum(IDA %in% IDB)
156 pairs <- filesizeA * filesizeB
157 #
158 # # Calc measures
159 tp <- sum(IDA[ests$matches[,1]] == IDB[ests$matches[,2]])
160 fp <- sum(IDA[ests$matches[,1]] != IDB[ests$matches[,2]])
161 fn <- positives - tp
162 tn <- pairs - positives + fp
163 precision <- tp / (tp + fp)
164 sens <- recall <- tp / (tp + fn)
165 spec <- tn / (tn + fp)
166 Fmeasure <- mean(precision,recall)
167 #
168 print("F-Measure, TP, FP")
169 print(cbind(Fmeasure, tp, fp))
170
171 #if(Fmeasure > bestIdentifierF){
172 # bestIdentifierF <- Fmeasure
173 # bestIdentifierNameF <- idents
174 #}
175 if(abs(meanAW) > bestIdentifier){
176 bestIdentifier <- abs(meanAW)
177 bestIdentifierName <- idents
178 }
179 tests <- rbind(tests, data.frame(idents = paste0(idents, collapse = ","),Fmeasure,
↪ recall, precision, m,u, meanAW, meanDAW))

```

```
180 #tests <- rbind(tests, data.frame(identfs = paste0(identfs, collapse = ","), m,u, meanAW,
    ↪ meanDAW))
181 write.table(tests,"IdentifierChoice.csv",sep="\t",row.names=FALSE)
182 }
183
184
185 #print("Final Matrix for m")
186 #print(matrix(unlist(estsf$p.gamma.k.m), ncol=2, nrow=length(identfs), byrow=TRUE,
    ↪ dimnames=list(identfs)))
187
188 #print("Final Matrix for u")
189 #print(matrix(unlist(estsf$p.gamma.k.u), ncol=2, nrow=length(identfs), byrow=TRUE,
    ↪ dimnames=list(identfs)))
190
191 cbind(aggregate(abs(testsf$meanAW), by=list(testsf$identfs),
    ↪ mean),aggregate(abs(testsf$Fmeasure), by=list(testsf$identfs), mean)$x)
192
193
194
195 cat("\n Best identifier combination:", bestIdentifierName, "\tAgreement weight:
    ↪ ",bestIdentifier)
```

## Source Code E.2: Standalone Source code for optimal parameter choice estimation.

```

1 #####
2 #
3 # Estimating optimal parameter choices for an input file specified
4 #
5 # Input: At least one readily preprocessed input file. If two files, both must have the
6   ↪ same column names.
7 # Recommended: One or two files with IDs.
8 # Packages required: PPRL, fastdigest, stringdist, entropy, data.table, fastLink, moments,
9   ↪ ineq, multibitTree
10 # User-defined settings in lines 28 to 58.
11 #
12 # This script reads the file(s), generates all measures required for the estimation,
13 # fits a model and predicts the optimal parameters.
14 # Optionally, includes possibility to generate multiple iterations, e.g. for bootstrapped
15   ↪ estimates.
16 # Gives optimal estimates for a simple linear model, a model generated by BIC- or AIC-based
17   ↪ model selection
18 # and a response surface model estimation (RSM).
19 #
20 # Output:
21 #   Optimal_Parameters_LM.csv
22 #   Optimal_Parameters_BIC.csv
23 #   Optimal_Parameters_RSM.csv
24 #   Optimal_Parameters_RF.csv
25 #
26 #
27 # Christian Borgs, 2019
28 #
29 #####
30
31 ##### User settings #####
32
33 # Set working dir for script
34 setwd("E:/Priv/Dropbox/Dissertation/Programme")
35 # File to run the optimization over
36 inputFile <- "Daten/Adrian_Perfect_100k.csv"
37 # Identifiers

```

```
35 idents <- c("Given Name", "Family Name", "Date Of Birth")
36 # ID-column. If none, set to FALSE (WARNING: This will be detrimental to the model
   ↪ quality!)
37 IDcol <- "Group Id"
38 # String-Distance columns. For EM weighting
39 Stringidents <- c("Given Name", "Family Name")
40
41 # Desired BF length
42 l <- 500
43 # Type: BF or CLK
44 type <- "CLK"
45 # Bootstrap repetitions
46 bootstraps <- 1
47 # Desired Tanimoto threshold
48 tanimotoGoal <- 0.9
49 # Cores of the machine for multithreading
50 cores <- 7
51 # Mode: "Train" for using a subset of the original data set as training data, "Link" if
   ↪ both files are available
52 mode <- "Link"
53 # If mode "Link": relative size of the linking subsample (for large files)
54 linkSample <- 1
55 # If mode "Train": relative size of the training subsample
56 trainSample <- 0.2
57 # If mode "Link": file name of file to link to
58 inputFileB <- "Daten/Adrian_10percent_100k.csv"
59
60 #####
61
62 # Libs used
63 library(PPRL)
64 library(MASS)
65 library(randomForest)
66 library(rsm)
67 library(fastdigest)
68 library(stringdist)
69 library(entropy)
70 library(data.table)
71 library(fastLink)
72 library(moments)
73 library(ineq)
```

```
74 library(multibitTree)
75
76 ##### Static vars, change only if you know what you are doing #####
77
78 # Training data BF length
79 lTraining <- 500
80 # Training data file
81 trainFN <- "Complete_Results.csv"
82 # q-grams to test
83 Q <- c(1,2,3,4)
84 # Hash function test space
85 K <- seq(from = 1, to = 40, by = 1)
86 # MBT leaf limit
87 leaflimit <- 3
88 # Calc number of combinations in total
89 numCombinations <- length(K) * length(Q) * bootstraps
90 # Set counter for progress report
91 counter <- 0
92 # Result vector initialization
93 meta <- NULL
94
95 #####
96
97
98 # Read training file for model building
99 traindata <- read.csv(trainFN, sep="\t")
100
101 # Subset to fixed tanimoto threshold
102 fixed <- subset(traindata, Tani == tanimotoGoal)
103
104 ### Model calls
105
106
107 # LM
108 LM <- lm(Fmeasure ~ k + (q) + errorestimation * skew + uniquepatternsA *
109         uniqueness + ginicoefficient + k * hammingweight +
110         hammingweight * meanengrams + uniquepatternsA + log(u),
111         data=fixed)
112
113
114 # Full model for BIC-selection
```

```

115 fullmod <- lm(Fmeasure ~ k + (q) + errorestimation + skew + uniqueness + meanentropy +
  ↪ log(u) + ginicoefficient + uniquepatternsA + errorestimation + k * q *
  ↪ hammingweight + meanengrams*hammingweight +
116         meanmissing + log(m) + u*m,
117         data=fixed)
118 # Minimal model
119 minmod <- lm(Fmeasure ~ k + (q), data = fixed)
120
121 # Use BIC-selection to get the model
122 BIC <- stepAIC(fullmod, minmod, direction = "both", k = log(nrow(fixed))) # bic
123
124
125 # Random forest
126 r1 <- randomForest(Fmeasure ~ k + q + errorestimation + skew + uniqueness +
  ↪ ginicoefficient + hammingweight +
127         meanengrams + u + uniquepatternsA, data=fixed,
128         mtry= 3, ntree=500, nodesize=20, #nodesize=5,
129         importance=TRUE, nPerm=5, keep.forest=TRUE)
130
131
132 # RSM: Preproc data
133 RSMdata <- fixed[,c("k", "q", "u", "meanengrams", "hammingweight", "Fmeasure")]
134 RSMdata_unscaled <- fixed[,c("k", "q", "u", "meanengrams", "hammingweight", "Fmeasure")]
135 # Scale min/max
136 maxi = apply(RSMdata , 2, function(x) max(as.numeric(x)))
137 mini = apply(RSMdata, 2, function(x) min(as.numeric(x)))
138 # Scaling
139 RSMdata = as.data.frame(scale(RSMdata, center = mini, scale = maxi - mini))
140 # Train model
141 rsm.train <- rsm(Fmeasure ~ SO(k, q) + TWI(q, k, hammingweight) + (u), data = RSMdata)
142
143
144 ##### Work with data
145
146 # Read input
147 clearTextA <- fread(inputFile, colClasses = "character", stringsAsFactors = FALSE)
148
149 # If second file, read it. Otherwise, sample
150 if (mode == "Link"){
151   clearTextB <- fread(inputFileB, colClasses = "character", stringsAsFactors = FALSE)
152

```

```

153 clearTextA <- clearTextA[sample(nrow(clearTextA),round(nrow(clearTextA)*linkSample)),]
154
155 clearTextB <- clearTextB[sample(nrow(clearTextB),round(nrow(clearTextB)*linkSample)),]
156
157 } else {
158   clearTextB <- clearTextA[sample(nrow(clearTextA),round(nrow(clearTextA)*trainSample)),]
159 }
160
161 ## Deduplicate by ID
162 clearTextA <- clearTextA[!duplicated(clearTextA$`Group Id`),]
163 clearTextB <- clearTextB[!duplicated(clearTextB$`Group Id`),]
164
165
166
167 # Check file sizes
168 filesizeA <- nrow(clearTextA)
169 filesizeB <- nrow(clearTextB)
170 # Number of pairs
171 pairs <- as.numeric(filesizeA) * as.numeric(filesizeB)
172
173 # Pairs and positives
174 if (IDcol != FALSE){
175   # Calc true matches
176   truepositives <- sum(unlist(clearTextA[,..IDcol]) %in% unlist(clearTextB[,..IDcol]))
177
178   # Generate ID Col
179   IDA <- as.character(unlist(clearTextA[,..IDcol]))
180   IDB <- as.character(unlist(clearTextB[,..IDcol]))
181
182   # subset the data
183   clearTextA <- clearTextA[, ..idents]
184   clearTextB <- clearTextB[, ..idents]
185 } else {
186
187   # Generate crude TP estimate by using keys
188   clearTextA$key <- apply(clearTextA[,..idents],1,paste0, collapse="")
189   clearTextB$key = apply(clearTextB[,..idents],1,paste0, collapse="")
190
191   #Estimate TP
192   truepositives <- sum(clearTextA$key %in% clearTextB$key)
193

```

```

194 # Generate ID Col by generating same IDs for exact matches, and SHA1 random IDs for all
    ↪ other pairs
195 clearTextA$IDA <- ""
196 set.seed(42)
197 clearTextA[which(clearTextA$key %in% clearTextB$key), "IDA"] <-
    ↪ replicate(truepositives, fastdigest(sample(LETTERS, 1000, replace = TRUE)))
198 set.seed(42)
199 clearTextB$IDB <- ""
200 clearTextB[which(clearTextB$key %in% clearTextA$key), "IDB"] <-
    ↪ replicate(length(clearTextB$key %in% clearTextA$key), fastdigest(sample(LETTERS,
    ↪ 1000, replace = TRUE)))
201 clearTextA$IDA[clearTextA$IDA == ""] <-
    ↪ sha1(as.character(1:length(clearTextA$IDA[clearTextA$IDA == ""])), key = "A")
202 clearTextB$IDB[clearTextB$IDB == ""] <-
    ↪ sha1(as.character(1:length(clearTextA$IDA[clearTextA$IDA == ""])), key = "B")
203
204 # Generate ID Col
205 IDA <- as.character(clearTextA$IDA)
206 IDB <- as.character(clearTextB$IDB)
207
208 # subset the data
209 clearTextA <- clearTextA[, ..idents]
210 clearTextB <- clearTextB[, ..idents]
211 }
212
213
214 ##### Start working #####
215
216 # Get EM weights
217 ests <- fastLink(clearTextA, clearTextB, varnames = idents,
218                 stringdist.match=Stringidents, stringdist.method="jw", n.cores = cores,
219                 ↪ estimate.only = TRUE)
220
221 save(ests, file="EM_estimates.Rdata") # In case of or if pre-estimated file exists
222 load("EM_estimates.Rdata")
223
224 # Weights
225 m <- ests$p.m
226 u <- ests$p.u
227

```

```

228 # Loop over parameter space
229 for (q in Q){
230
231   cat("\n\n\nLine-by-line checking...", format(Sys.time(), " (%a, %d-%m-%Y,
   ↪   %H:%M:%S)"), "\n")
232
233
234   # Generate full linkage key
235   keys <- apply(clearTextA, 1, paste0, collapse = "")
236   keys <- gsub(" ", "", keys)
237
238   # Split into bigrams
239   qgrams <- (sapply(keys, function(key) substring(key, first=seq(1, nchar(key)), last=seq(q,
   ↪   nchar(key)+q))))
240   # Count uniques and save measures
241   uniquengrams <- as.numeric(unlist(lapply(qgrams, function(x) length(unique(x)))))
242   # ngram Count
243   ngramsC <- as.numeric(unlist(lapply(qgrams, function(x) length(x))))
244   qgrams <- unlist(qgrams)
245   qgramlist <- as.character(na.omit(ifelse(qgrams == "" | nchar(qgrams) != q, NA, qgrams)))
246
247   cat("\n\n\nLine-by-line checking complete, calculating measures...",
   ↪   format(Sys.time(), " (%a, %d-%m-%Y, %H:%M:%S)"), "\n")
248
249   # Sum missings
250   missings <- sum(apply(clearTextA, 2, function(x) sum(is.na(x)))) +
   ↪   sum(apply(clearTextA, 2, function(x) sum(x[!is.na(x)]=="")))
251
252   # Mean entropy calculation
253   meanentropy <- mean(apply(clearTextA, 2, function(x) entropy(table(x))))
254
255   # q90/q10-ratio
256   q90 <- round(quantile(as.numeric(table(qgramlist)), probs=seq(0,1,0.1))["90%"])
257   q10 <- round(quantile(as.numeric(table(qgramlist)), probs=seq(0,1,0.1))["10%"])
258
259   # Measures
260   meanengrams <- mean(ngramsC)
261   sdgrams <- sd(ngramsC)
262
263   skew <- skewness(table(qgramlist))
264   ginicoefficientqgrams <- Gini(table(qgramlist))

```

```

265
266 ginicoefficient <- mean(apply(clearTextA, 2, function(x) Gini(table(x))))
267
268 errorestimation <- q90/q10 # amount of rare combinations in data
269 missingamount <- missings
270 meanmissing <- missings/nrow(clearTextA)
271 uniqueness <- mean(uniquengrams/ngramsC)
272 sduniqueness <- sd(uniquengrams/ngramsC)
273
274
275
276 for (k in K){
277
278   for (i in seq(1:bootstraps)){
279
280     cat("\n\n\nEncrypting...", format(Sys.time(), "%a, %d-%m-%Y, %H:%M:%S"), "\n")
281
282     ## Generate password vector for BF encoding, resample for boot-strapping
283     PWs <- replicate(length(idents), fastdigest(sample(LETTERS, 10000, replace =
284       ↪ TRUE)))
285
286     encryptedA <- CreateCLK(ID = IDA, clearTextA[, ..idents], k = k, padding =
287       ↪ rep(0, length(idents)), q = rep(q, length(idents)), l = l, password =
288       ↪ PWs)#, "13124", "124124"))
289
290     encryptedB <- CreateCLK(ID = IDB, clearTextB[, ..idents], k = k, padding =
291       ↪ rep(0, length(idents)), q = rep(q, length(idents)), l = l, password =
292       ↪ PWs)#, "13124", "124124"))
293
294
295     # Calculate hamming weight and NO of unique patterns
296     hammingweight <- mean(c(nchar(gsub("0", "", encryptedA$CLKs)), nchar(gsub("0", "",
297       ↪ encryptedB$CLKs))))
298     uniquepatternsA <- length(unique(encryptedA$CLKs))
299     uniquepatternsB <- length(unique(encryptedB$CLKs))
300
301     #
302     # # Tree data
303     write.table(encryptedA, "TreeA_OPC.csv", sep="\t", row.names=FALSE, quote=FALSE,
304       ↪ col.names=FALSE)
305     write.table(encryptedB, "TreeB_OPC.csv", sep="\t", row.names=FALSE, quote=FALSE,
306       ↪ col.names=FALSE)

```

```

298
299 # Run tree
300 multibitTree.load("TreeB_OPC.csv",threads=cores, leafLimit = leaflimit)
301 result <- multibitTree.searchFile("TreeA_OPC.csv", Tani)
302
303 #
304 #
305 # # Classify candidate pairs
306 tp <- sum((result$fingerprint) == (result$query))
307 fp <- sum((result$fingerprint) != (result$query))
308 fn <- truepositives - tp
309 tn <- pairs - (fn + tp + fp)
310 #
311 # # Calculate central measures
312 ReductionRate <- 1 - ((tp+fp)/(tn+fn))
313 precision <- tp / (tp+fp)
314 recall <- tp / (tp+fn)
315 Fmeasure = (recall + precision) / 2
316
317 #tp <- fp <- fn <- tn <- precision <- recall <- Fmeasure <- 0
318
319 # Save RAM
320 #rm(result)
321 gc()
322
323 # Set counter plus one
324 counter <- counter + 1
325
326
327 cat("\nClassification", counter, "of", numCombinations,"Finished
  ↳ (",format((counter / numCombinations)*100, digits=3),"%)!",
  ↳ format(Sys.time()," (%a, %d-%m-%Y, %H:%M:%S)"), ": \nFilesizes:\t",
  ↳ format(filesizeA,scientific=FALSE),"\nData source and mode:\t" , inputFile, "
  ↳ ", mode,"\nBootstrap:\t\t", i, "/", bootstraps, "\nk/q:\t\t", k, "/",q
  ↳ ,"\nTanimoto:\t", tanimotoGoal, "\n\nPositives:\t", truepositives, "\nTrue
  ↳ Positives:\t", tp, "\nRecall:\t\t", recall, "\nFalse Positives:", fp,
  ↳ "\nPrecision:\t", precision, "\nFalse Negatives:", fn, "\nF-Score:\t",
  ↳ Fmeasure, "\n\n")
328
329
330 # Append to result

```

```

331 meta <- rbind(meta, data.frame(d="New Data", type, q, k, i, l,
    ↪ Tani=tanimotoGoal,Fmeasure, filesizeA, filesizeB, meanentropy, meanengrams,
    ↪ sdgrams, skew, ginicoefficient, ginicoefficientqgrams, errorestimation ,
    ↪ missingamount,hammingweight, uniqueness, sduniqueness, meanmissing,
    ↪ missingamount, uniquepatternsA,uniquepatternsB,m,u))
332 write.table(meta, "tempResult.csv", sep="\t", row.names=FALSE, quote=FALSE,
    ↪ col.names=TRUE)
333 }
334 }
335 }
336
337
338 # Prepare resulting data to predict from it
339 newdata <- data.frame(meta, stringsAsFactors = FALSE, row.names = NULL)
340 names(newdata)[names(newdata) == "tanimotoGoal"] <- "Tani"
341 write.table(newdata, "Trainresults.csv", sep="\t", row.names=FALSE, quote=FALSE,
    ↪ col.names=TRUE)
342 newdata <- read.csv("Trainresults.csv", sep = "\t", stringsAsFactors = FALSE)
343
344
345
346
347 # RSM
348
349 # RSM: Preproc data
350 newdataRSM <- newdata
351 newdataRSM <- newdataRSM[,c("k", "q", "u", "meanengrams", "hammingweight", "Fmeasure")]
352 maxi = apply(newdataRSM , 2, function(x) max(as.numeric(x)))
353 mini = apply(newdataRSM, 2, function(x) min(as.numeric(x)))
354
355 # Scale data
356 newdataRSM = as.data.frame(scale(newdataRSM, center = mini, scale = maxi - mini))
357 newdataRSM$u <- 0.5
358 newdataRSM$meanengrams <- 0.5
359
360 # Predict RSM
361 RSM <- data.frame(newdataRSM, predicted = predict(rsm.train, newdataRSM, type =
    ↪ "response"))
362 # Re-Scale to get actual values
363 RSM$Fmeasure <- (RSM$Fmeasure * (max(RSMdata_unscaled$Fmeasure) -
    ↪ min(RSMdata_unscaled$Fmeasure))) + min(RSMdata_unscaled$Fmeasure)

```

```

364 RSM$k <- (RSM$k * (max(RSMdata_unscaled$k) - min(RSMdata_unscaled$k))) +
  ↪ min(RSMdata_unscaled$k)
365 RSM$q <- (RSM$q * (max(RSMdata_unscaled$q) - min(RSMdata_unscaled$q))) +
  ↪ min(RSMdata_unscaled$q)
366
367
368 # Predict LM/BIC/RF
369 newfitted <- data.frame(newdata, predicted = predict(LM, newdata, type = "response"))
370 newfittedBIC <- data.frame(newdata, predicted = predict(BIC2, newdata, type = "response"))
371 newfittedRF <- data.frame(newdata, predicted = predict(r1, newdata))
372
373 # Get optimal parameter estimate
374 parameters <- newfitted[which.max((newfitted$predicted)), c("Fmeasure", "Tani", "q", "k",
  ↪ "l", "predicted")]
375 parameters2 <- newfittedBIC[which.max((newfittedBIC$predicted)), c("Fmeasure", "Tani",
  ↪ "q", "k", "l", "predicted")]
376 parameters3 <- newfittedRF[which.max((newfittedRF$predicted)), c("Fmeasure", "Tani", "q",
  ↪ "k", "l", "predicted")]
377 parametersRSM <- RSM[which.max((RSM$predicted)), c("Fmeasure", "q", "k", "predicted")]
378
379
380 # Write out parameter estimates
381 write.table(parameters, "Optimal_Parameters_LM.csv", sep="\t", row.names = FALSE)
382 write.table(parameters2, "Optimal_Parameters_BIC.csv", sep="\t", row.names = FALSE)
383 write.table(parameters3, "Optimal_Parameters_RF.csv", sep="\t", row.names = FALSE)
384 write.table(parametersRSM, "Optimal_Parameters_RSM.csv", sep="\t", row.names = FALSE)
385
386 cat("\n All done!")
387
388 cat("\n Best parameters (k/q):\n\n Simple linear model:", parameters$k, "/", parameters$q,
  ↪ "\n BIC-selected linear model:",
389   parameters2$k, "/", parameters2$q, "\n Random Forest model:", parameters3$k, "/",
  ↪ parameters3$q, "\n Response Surface Method:", parametersRSM$k, "/",
  ↪ parametersRSM$q, "\n")

```

## Other Code

### F.1 Google Scholar results for ‘Big Data’

#### *Search strategy*

Searching using “Big Data” (including quotes) in Google Scholar. For each year since 2000, the range start and range end was set to the same year.<sup>1</sup> The resulting number of hits was noted in the R-File. The code below requires the R-Package ggplot2.

Source Code F.1: Source code for Google Scholar results for Big Data.

```
1 library(ggplot2)
2
3 bigdat <- data.frame(years = 2000:2018, count = rev(c(75800, 95700, 87700, 79600, 52600,
  ↪ 31500, 12400, 4700, 2770, 2380, 2070, 1900, 1930, 1210, 1160, 1010, 859, 703, 578)))
4
5 # Plot
6 p1 <- ggplot(data = bigdat, aes(y = count, x = (years)))
7 p1 <- p1 + geom_point(size=2,color="#004c93") # Dots
8 p1 <- p1 + geom_line(size=1,color="#004c93") # Line
9 p1 <- p1 + xlab("Year") + ylab("No. of Google Scholar Hits\n for 'Big Data'")
10 p1 <- p1 + scale_y_continuous(breaks = seq(0,75000,10000)) # Breaks values
11 p1 <- p1 + scale_x_continuous(breaks = seq(2000,2018,2)) # Breaks years
12 p1 + theme_bw() + theme(axis.text.x = element_text(angle = 90, vjust = 0.5),
13 panel.spacing = unit(0.85, "lines"), text = element_text(size=35),
14 legend.justification=c(0.9,0.05), legend.key = element_blank(),
15 legend.key.size = unit(5,"char"),
16 strip.background = element_rect(colour='white',fill = 'white')) # Look
17 ggsave("./results/Scholar_Bigdata.pdf", height=11, width=15, units="in",
  ↪ device=cairo_pdf())
```

<sup>1</sup>This corresponds to this example URL for 2018: [https://scholar.google.de/scholar?q=%22big+data%22&hl=de&as\\_sdt=0%2C5&as\\_ylo=2018&as\\_yhi=2018](https://scholar.google.de/scholar?q=%22big+data%22&hl=de&as_sdt=0%2C5&as_ylo=2018&as_yhi=2018).

## F.2 Database results for ‘Record Linkage’ in Medline and Sociological Abstracts

### *Search strategy*

*Sociological Abstracts:* Searching for “Record Linkage” in the title (`ti("record linkage")`). Only journals are displayed using the respective filter. The year range was set from 1980 to 2018. All publications were selected and exported as an XLS-File. The XLS was saved as a CSV and all years were extracted using Notepad++. The years were inserted into a vector in R and counted using `table()`.

*Medline:* Searching for “Record Linkage” in the title (`"record linkage".m_titl.`). Only articles in journals are displayed using the respective filter. For each year since 1980, the range start and range end was set to the same year. The resulting number of hits for each year was noted in the R-File along with the year.

The code below requires the R-Packages `tidyverse`, `reshape2` and `ggplot2`.

Source Code F.2: Source code for Record Linkage in Medline and Sociological Abstracts.

```
1 library(tidyverse)
2 library(reshape2)
3 library(ggplot2)
4
5 # Sociological abstracts:
6 # ti("record linkage")
7 # Only journals
8 # 1980-2019
9 # Export als XLS, Extraktion der Jahre mit Notepad++
10 socabstracts <- c(1980,
11 1986,
12 1987,
13 1988,
14 1991,
15 1996,
16 2000,
17 2000,
18 2000,
19 2005,
20 2005,
21 2006,
22 2006,
23 2006,
24 2007,
```

```
25 2007,
26 2011,
27 2011,
28 2013,
29 2013,
30 2013,
31 2014,
32 2014,
33 2016,
34 2017,
35 2017,
36 2018)
37 # Create table with counts
38 soc1 <- data.frame(years= names(table(socabstracts)),
39 count = as.numeric(table(socabstracts)))
40 # Create table with zeroes for all years
41 soc2 <- data.frame(years= factor(1980:2018),
42 count = rep(0,length(factor(1980:2018))))
43 # Fill zeroes with the years that have counts in table soc1
44 soc2[soc2$years %in% soc1$years,2] <- soc1[soc1$years %in% soc2$years,2]
45 rm(soc1) # remove temporary table
46 # Add source for plotting and make year numeric
47 soc2$source = "Sociological Abstracts"
48 soc2$years = as.numeric(as.character(soc2$years))
49
50
51 # Medline:
52 # "record linkage".m_titl.
53 # Only journal articles
54 # 1980-2018 einzeln abgefragt
55 # Zahl je Jahr herausgeschrieben
56 medline <- tribble(
57 ~years, ~count,
58 2019,4,
59 2018,63,
60 2017,78,
61 2016,59,
62 2015,65,
63 2014,52,
64 2013,35,
65 2012,47,
```

```
66 2011,46,
67 2010,31,
68 2009,30,
69 2008,24,
70 2007,14,
71 2006,22,
72 2005,24,
73 2004,12,
74 2003,8,
75 2002,12,
76 2001,13,
77 2000,13,
78 1999,13,
79 1998,20,
80 1997,15,
81 1996,7,
82 1995,17,
83 1994,10,
84 1993,6,
85 1992,5,
86 1991,9,
87 1990,11,
88 1989,17,
89 1988,5,
90 1987,7,
91 1986,11,
92 1985,11,
93 1984,6,
94 1983,4,
95 1982,1,
96 1981,6,
97 1980,5
98 )
99 # Add source for plotting
100 medline$source = "Medline"
101
102 # Make plottable data frame, delete 2018
103 plotdat <- subset(rbind(medline,soc2), as.numeric(years) < 2019)
104 plotdat <- melt(plotdat, id.vars = c("years","source"))
105
106 # Plot
```

```

107 p1 <- ggplot(data = plotdat, aes(y = value, x = (years), group=source, color=source))
108 #p1 <- p1 + geom_smooth(se=F, method="loess", color="#004c93") # optional smoother
109 p1 <- p1 + geom_point(size=3) # points
110 p1 <- p1 + geom_line(size=1) # lines
111 p1 <- p1 + xlab("Year") + ylab("No. of Results for 'Record Linkage'\n") # labels
112 p1 <- p1 + scale_color_manual(name = "Database", labels=
113 c("Medline","Sociological Abstracts"),
114 values=c("#004c93","#56B1F7")) # Set colors
115 p1 <- p1 + scale_x_continuous(breaks = seq(1980,2018,2)) # set year breaks
116 p1 + theme_bw() + theme(axis.text.x = element_text(angle = 90, vjust = 0.5),
117 panel.spacing = unit(0.85, "lines"), text = element_text(size=35),
118 legend.justification=c(0.9,0.05), legend.key = element_blank(),
119 legend.key.size = unit(5,"char"),
120 strip.background = element_rect(colour='white',fill = 'white'))
121
122 ggsave("./results/RL_years.pdf", height=11, width=18, units="in", device=cairo_pdf())

```

## F.3 Full R Version Info

### Windows 7

Source Code F.3: R version and package info for Windows.

```

1 sessionInfo()
2 R version 3.4.4 (2018-03-15)
3 Platform: x86_64-w64-mingw32/x64 (64-bit)
4 Running under: Windows 7 x64 (build 7601) Service Pack 1
5
6 Matrix products: default
7
8 locale:
9 [1] LC_COLLATE=German_Germany.1252 LC_CTYPE=German_Germany.1252
   ↪ LC_MONETARY=German_Germany.1252 LC_NUMERIC=C
   ↪ LC_TIME=German_Germany.1252
10
11 attached base packages:
12 [1] grid      stats    graphics grDevices utils    datasets methods  base
13
14 other attached packages:

```

```

15 [1] ineq_0.2-13      moments_0.14      usethis_1.4.0     devtools_2.0.2
   ↪ car_3.0-2       cvTools_0.3.2     robustbase_0.93-3 lattice_0.20-35
   ↪ effects_4.0-0   carData_3.0-0
16 [11] dplyr_0.8.0.1     xtable_1.8-2      stargazer_5.2     RColorBrewer_1.1-2
   ↪ colorspace_1.3-2 multibitTree_1.7  packrat_0.5.0     MASS_7.3-49
   ↪ randomForest_4.6-14 neuralnet_1.44.2
17 [21] boot_1.3-20      rsm_2.10          gmodels_2.18.1    sjstats_0.14.2-2
   ↪ ggplot2_3.1.0   fastLink_0.3.1    gridExtra_2.3     data.table_1.10.4-3
   ↪ entropy_1.2.1   stringdist_0.9.4.7
18 [31] fastdigest_0.6-3 PPRL_0.3.4
19
20 loaded via a namespace (and not attached):
21 [1] TH.data_1.0-8     minqa_1.2.4       rio_0.5.16         modeltools_0.2-21
   ↪ sjlabelled_1.0.8 rprojroot_1.3-2   estimability_1.3   snakecase_0.9.1
   ↪ fs_1.2.6
22 [10] rstudioapi_0.7    glmmTMB_0.2.0     remotes_2.0.2     ggrepel_0.7.0
   ↪ mvtnorm_1.0-7   coin_1.2-2        codetools_0.2-15  splines_3.4.4
   ↪ mnormt_1.5-5
23 [19] doParallel_1.0.11 sjmisc_2.7.1      pkgload_1.0.2     bayesplot_1.4.0
   ↪ ade4_1.7-10     nloptr_1.0.4      broom_0.4.3        FactoClass_1.2.4
   ↪ settings_0.2.4
24 [28] httr_1.3.1        compiler_3.4.4    emmeans_1.1.2     backports_1.1.2
   ↪ assertthat_0.2.0 Matrix_1.2-12     lazyeval_0.2.1    survey_3.34
   ↪ cli_1.0.1
25 [37] prettyunits_1.0.2 tools_3.4.4        coda_0.19-1       gtable_0.2.0
   ↪ glue_1.3.1      reshape2_1.4.3    Rcpp_1.0.0         cellranger_1.1.0
   ↪ gdata_2.18.0
26 [46] debugme_1.1.0     nlme_3.1-137     iterators_1.0.9    psych_1.7.8
   ↪ lmtest_0.9-35   stringr_1.3.0     openxlsx_4.1.0     testthat_2.0.0
   ↪ lme4_1.1-15
27 [55] gtools_3.5.0      DEoptimR_1.0-8    zoo_1.8-1          scales_0.5.0
   ↪ parallel_3.4.4 sandwich_2.4-0    pwr_1.2-2          TMB_1.7.13
   ↪ curl_3.1
28 [64] yaml_2.1.18       memoise_1.1.0     stringi_1.1.7     desc_1.2.0
   ↪ foreach_1.4.4   plotrix_3.7-4     zip_1.0.0          pkgbuild_1.0.2
   ↪ rlang_0.3.1
29 [73] pkgconfig_2.0.2   purrr_0.3.2       prediction_0.2.0   tidyselect_0.2.5
   ↪ plyr_1.8.4       magrittr_1.5      R6_2.2.2           multcomp_1.4-8
   ↪ pillar_1.3.1
30
31

```

```

32 [82] haven_1.1.1          foreign_0.8-69      withr_2.1.2        abind_1.4-5
   ↪ nnet_7.3-12         survival_2.41-3    scatterplot3d_0.3-41 tibble_2.1.1
   ↪ modelr_0.1.1
33 [91] crayon_1.3.4         KernSmooth_2.23-15  adagio_0.6.5       readxl_1.0.0
   ↪ callr_2.0.2        forcats_0.3.0      digest_0.6.15      tidyr_0.8.0
   ↪ stats4_3.4.4
34 [100] munsell_0.4.3       sessioninfo_1.1.1

```

## Ubuntu

Source Code F.4: R version and package info for Ubuntu.

```

1 R version 3.5.1 (2018-07-02)
2 Platform: x86_64-pc-linux-gnu (64-bit)
3 Running under: Ubuntu 18.10
4
5 Matrix products: default
6 BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.8.0
7 LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.8.0
8
9 locale:
10 [1] LC_CTYPE=en_US.UTF-8      LC_NUMERIC=C           LC_TIME=en_GB.UTF-8
   ↪ LC_COLLATE=en_US.UTF-8  LC_MONETARY=en_GB.UTF-8 LC_MESSAGES=en_US.UTF-8
   ↪ LC_PAPER=en_GB.UTF-8    LC_NAME=C
11 [9] LC_ADDRESS=C             LC_TELEPHONE=C        LC_MEASUREMENT=en_GB.UTF-8
   ↪ LC_IDENTIFICATION=C
12
13 attached base packages:
14 [1] stats      graphics  grDevices  utils      datasets  methods   base
15
16 other attached packages:
17 [1] fastdigest_0.6-3  ggplot2_3.1.0      fastLink_0.5.0      data.table_1.12.0
   ↪ entropy_1.2.1   stringdist_0.9.5.1 PPRL_0.3.5.2
18
19 loaded via a namespace (and not attached):
20 [1] Rcpp_1.0.0        pillar_1.3.1       compiler_3.5.1      plyr_1.8.4
   ↪ bindr_0.1.1     iterators_1.0.10   FactoClass_1.2.7    tools_3.5.1
   ↪ tibble_2.0.1     gtable_0.2.0
21 [11] lattice_0.20-35   pkgconfig_2.0.2    rlang_0.3.1         foreach_1.4.4
   ↪ Matrix_1.2-14   ggrepel_0.8.0      parallel_3.5.1      bindrcpp_0.2.2
   ↪ withr_2.1.2     stringr_1.3.1

```

```
22 [21] dplyr_0.7.8          gtools_3.8.1          ade4_1.7-13           grid_3.5.1
   ↪ tidyselect_0.2.5    scatterplot3d_0.3-41 glue_1.3.0            R6_2.3.0
   ↪ plotrix_3.7-4       adagio_0.7.1
23 [31] purrr_0.3.0          magrittr_1.5          codetools_0.2-15     scales_1.0.0
   ↪ MASS_7.3-51.1      assertthat_0.2.0     colorspace_1.4-0     xtable_1.8-3
   ↪ KernSmooth_2.23-15 stringi_1.2.4
24 [41] lazyeval_0.2.1       munsell_0.5.0         doParallel_1.0.14    crayon_1.3.4
```

# DuEPublico

Duisburg-Essen Publications online

UNIVERSITÄT  
DUISBURG  
ESSEN

*Offen im Denken*

ub | universitäts  
bibliothek

Diese Dissertation wird über DuEPublico, dem Dokumenten- und Publikationsserver der Universität Duisburg-Essen, zur Verfügung gestellt und liegt auch als Print-Version vor.

**DOI:** 10.17185/duepublico/70274

**URN:** urn:nbn:de:hbz:464-20190807-104513-3



Dieses Werk kann unter einer Creative Commons Namensnennung  
- Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0  
Lizenz (CC BY-NC-SA 4.0) genutzt werden.