FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Predictor Factory: Learning from Relational Data

by

Jan Motl

A dissertation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

Dissertation degree study programme: Informatics
Department of Applied Mathematics

Prague, August 2021

Supervisor:
doc. Ing. Pavel Kordík, Ph. D.
Department of Applied Mathematics
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Co-Supervisor:
Prof. Ing. Filip Železný, Ph. D.
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo nám. 13
121 35 Prague 2
Czech Republic

Abstract and Contributions

Propositionalization algorithms transform relational data into a single table with features, which can be used for classification or regression with conventional machine learning tools. However, the contemporary propositionalization algorithms were not designed to work on changing data and suffered from the production of many irrelevant and redundant features. We altered the propositionalization to work on temporal data and introduced meta-learning, which predicts, which features will be relevant and unique. To test Predictor Factory, our implementation of propositionalization, we have created a repository of relational datasets and implemented a scalable algorithm for relationship discovery between tables in the dataset. The implementations were open-sourced and applied to real-world banking, government, marketing, medicine, and telecommunication problems.

In particular, the main contributions of the dissertation thesis are as follows:

1. Relational dataset repository for benchmarking relational algorithms (83 datasets).
2. Algorithm for discovering relationships between tables in a database (scales independently of the row count in the tables).
3. Optimal algorithm for stratified partitioning based on multiple columns for cross-validation (with exact solution).
4. Predictor Factory for automatic feature extraction from relational data for classification and regression.
5. Approach for removing trend and seasonal variation from relational data (improves accuracy, P=.0005).
6. Generalization of common aggregate functions to temporal data (improves accuracy, P=.016).
7. Meta-learning for faster feature extraction (reduces runtime by 90%).
8. Classifiers for learning on a stream of features (up to 10 000 times faster than scikit-learn).

Keywords:
relational learning, propositionalization, feature extraction, data preprocessing.

Acknowledgements

First of all, I would like to express my gratitude to my dissertation thesis supervisor, Pavel Kordík.

I would like to thank to Jaroslav Bendl, Adéla Blažková, Adéla Chodounská, Aleš Fišer, Karel Klouda, Jan Kukačka, Jiří Kukačka, Manuel Muñoz, Wei Nie, Václav Ostrožlík, Oliver Schulte, Batal Thibaut, Filip Trojan, and Filip Železný for their help and support.

Special thanks go to the staff of the Department of Applied Mathematics, who maintained a pleasant and flexible environment for my research. I would like to express special thanks to the department management for providing the funding for my research. My research has also been partially supported by the Czech Science Foundation as projects No. 13-17187S, and 18-18080S; by the Grant Agency of the Czech Technical University in Prague, grants No. SGS15/117/OHK3/1T/18, SGS16/119/OHK3/1T/18, and SGS17/210/OHK3/3T/18; and by Deloitte Advisory s.r.o.

Dedication

To my parents and the curious mind that decided to read this thesis

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 History and Significance . . . 1
- 1.3 Terminology . . . 1
  - 1.3.1 Relational Databases . . . 1
  - 1.3.2 Relational Supervised Learning . . . 2
  - 1.3.3 Time Series . . . 2
- 1.4 Problem Statement . . . 2
- 1.5 Structure of the Dissertation Thesis . . . 4

2 Background and State-of-the-Art **7**
- 2.1 Theoretical Background . . . 7
  - 2.1.1 Data Representation . . . 7
  - 2.1.2 Information Propagation . . . 9
  - 2.1.3 Feature Function . . . 12
  - 2.1.4 Feature Refinement . . . 16
  - 2.1.5 Feature Selection . . . 17
  - 2.1.6 Feature Collection . . . 18
  - 2.1.7 Feature Representation . . . 19
  - 2.1.8 Evaluation . . . 20
  - 2.1.9 Conclusions . . . 20
- 2.2 Empirical Comparison . . . 22
  - 2.2.1 Introduction . . . 22
  - 2.2.2 Related Work . . . 23
  - 2.2.3 Method Description . . . 24
  - 2.2.4 Evaluation . . . 26
  - 2.2.5 Application . . . 27

3 Predictor Factory **35**

3.1 Data Representation . . . 35
3.1.1 Processing in a Database or in an Application . . . 35
3.1.2 Typology . . . 36
3.1.3 Automatic Data Type Detection . . . 37
3.2 Information Propagation . . . 38
3.2.1 Training, Testing & Scoring Data in a Single Relation . . . 38
3.2.2 Do we need Target Identifier? . . . 39
3.2.3 Identifiers . . . 41
3.2.4 Are Lookup Tables Useful? . . . 42
3.2.5 Relationship Detection . . . 43
3.2.6 Time Constraints . . . 43
3.3 Feature Function . . . 46
3.3.1 Naming Convention . . . 47
3.3.2 Supervised Features . . . 48
3.3.3 Text Attributes . . . 49
3.3.4 Dirty Text Attributes . . . 49
3.4 Feature Selection . . . 50
3.4.1 Adjusted Chi^2 . . . 50
3.4.2 Concept Drift . . . 51
3.4.3 Downsampling . . . 52
3.4.4 Duplicate Feature Detection . . . 54
3.4.5 Feature Selection Ahead of Feature Collection . . . 56

4 Implementation **59**
4.1 Technology . . . 59
4.2 Vendor-Agnostic . . . 59
4.3 Architecture . . . 61
4.3.1 Network . . . 62
4.3.2 Configuration Files . . . 63
4.3.3 Metadata . . . 63
4.3.4 SQL . . . 63
4.3.5 Main . . . 63
4.4 Testing and Validation . . . 64
4.4.1 Connection Leaks . . . 64
4.4.2 User Testing . . . 64

5 Relational Repository **69**
5.1 Goals . . . 69
5.2 Design . . . 69
5.3 Content . . . 70
5.4 Access and Contributions . . . 71
5.5 The Meta-Database . . . 71
5.6 Conclusions . . . 75

6 Empirical Evaluation **77**
6.1 Algorithms . . . 77
6.2 Datasets . . . 77
6.3 Protocol . . . 77
6.4 Results . . . 78
6.5 Discussion . . . 79

7 Foreign Key Constraint Identification **81**
7.1 Introduction . . . 81
7.2 Related Work . . . 82
7.3 Method . . . 83
7.4 Results . . . 87
7.5 Discussion . . . 89
7.6 Conclusions . . . 90

8 Stratified Cross-Validation by Multiple Columns **93**
8.1 Introduction . . . 93
8.2 Definitions . . . 94
8.3 Related Work . . . 94
8.4 Solution . . . 95
8.5 Experiments . . . 98
8.6 Results . . . 100
8.7 Discussion . . . 103
8.8 Conclusions . . . 104

9 Trend and Seasonality Elimination **105**
9.1 Introduction . . . 105
9.2 Propositionalization . . . 106
9.3 Detrending and Deseasoning . . . 106
9.4 Method . . . 107
9.5 Experiments . . . 107
9.6 Results . . . 109
9.7 Discussion . . . 110
9.8 Conclusions . . . 111

10 Generalized Aggregates **113**
10.1 Introduction . . . 113
10.2 Related Work . . . 114
10.3 Method . . . 115
10.4 Experiments . . . 116
10.5 Results . . . 118
10.6 Discussion . . . 119
10.7 Conclusions . . . 119

11 Meta-learning **121**
11.1 Introduction . . . 121
11.2 Related Work . . . 122
11.3 Method . . . 123
11.4 Experiment . . . 124
11.5 Results . . . 128
11.6 Discussion . . . 131
11.7 Conclusions . . . 134

12 Learning on Stream of Features **135**
12.1 Online Random Forest . . . 135
12.1.1 Implementation . . . 137
12.1.2 Experiments . . . 139
12.1.3 Discussion . . . 141
12.1.4 Conclusions . . . 141
12.2 Online Discriminant Analysis . . . 141
12.2.1 Introduction . . . 142
12.2.2 Related work . . . 143
12.2.3 QDA Algorithm . . . 144
12.2.4 Experiments . . . 148
12.2.5 Discussion . . . 155
12.2.6 Conclusions . . . 157

13 Discussion **159**

14 Conclusions **161**
14.1 Contributions . . . 163
14.2 Future Work . . . 164

Bibliography **165**

Reviewed Publications of the Author Relevant to the Thesis **185**

Remaining Publications of the Author Relevant to the Thesis **187**

Relevant Theses Supervised by the Author **191**

Remaining Publications of the Author **193**

A Appendix **203**
A.1 Feature Functions . . . 203
A.2 Leaking Features . . . 206
A.3 Computational Complexity . . . 206
A.4 User Manual . . . 208

List of Figures

1.1 Relation, tuple and attribute represented as table, row and column respectively. 2
1.2 Segmentation of data based on time. 3

2.1 Decomposition of relational learning algorithms into 7 blocks. Each block contains a non-exhaustive list of applicable options. 8
2.2 An example of a star schema. 10
2.3 An example of a snowflake schema. 10
2.4 Four types of feature functions. From left to right, then top to bottom: scalar, horizontal aggregate, vertical aggregate, cross. 13
2.5 Projection of algorithms from Table 2.3 into a two-dimensional space with Multiple Correspondence Analysis. 21
2.6 Graph with count of wins. 24
2.7 Directed acyclic graph. 25
2.8 Network visualization. 25
2.9 Nodes with ranking. 25
2.10 Accuracy of ranking methods based on their ability to cope with missing values. 27
2.11 Network of the relational algorithms. Algorithms with more accurate predictions are on the right. Superior algorithms are in red nodes. 30
2.12 Plot with expected ranking of relational classifiers and propositionalization tools. The higher value is better. 34

3.1 Variable types. 36
3.2 Automatic data type detection for CSV files. 38
3.3 Diagram illustrating the selection of time constraint. 46
3.4 Data split. Filled circles represent training data and hollow circles represent testing data. Grey circles represent ignored data. The image is adapted from [240]. 51

4.1 Component diagram of Predictor Factory. 62

4.2 Class diagram of Predictor Factory. 67

7.1 The algorithm decomposition. 84
7.2 Example entity diagram with a hierarchy of subclasses. 91

8.1 Example of data preprocessing for ILP. 95
8.2 When we care only about positive samples (A in the first row), we may split the data into two folds following way: $[A]$ and $[B, B]$, because each fold has approximately the same count of positive samples A. When we switch the definition of the positive class (B in the second row), we end up with a better split $[A, B]$ and $[B]$ – the quality of A distribution remained unchanged while the quality of B distribution improved. When we one-hot encode the input data into n binary columns (the third row) instead of $n - 1$ binary columns (the first two rows), we end up with the better split without the need to define the positive class. Note that the encoding into n binary columns generalizes well to multi-class problems, where it can be unclear which class(es) should be treated as the positive. 96
8.3 No matter how do we attempt to assign these three rows into two folds, there is going to be a column with the difference of the value appearance equal to two. 97
8.4 Ranking of the cross-validation algorithms based on the unsupervised measures. Smaller area is better. 102
8.5 Ranking of the cross-validation algorithms based on the average of supervised measures. Smaller area is better. 102
8.6 Ranking of the cross-validation algorithms based on the variance of supervised measures. Smaller area is better. 103

10.1 Example sigmoids, which can be used to approximate aggregates. Red: `sum`. Green: `max`. Blue: `min complement`. The midpoints and slopes of `max` and `min complement` sigmoids are less extreme than in Table 10.1 to improve legibility of the graph. We assume $\mathbf{o} = \mathbf{x}$. 117

11.1 An example of a feature generative function *min* applied on attribute *att1*, which converts the multi-instance problem into a single-instance problem solvable with a common attribute value classifier. In this trivial example, the feature space contains only a single feature vector but it may generally contain thousands of feature vectors. 122
11.2 Flowchart of meta-learning on features. 123
11.3 Performance profile. The shaded area represents the 95% prediction interval for a random curve. 128
11.4 Area under misclassification error. Smaller is better. 132

12.1 The difference between learning from a stream of samples (top) and a stream of features (bottom). In both cases, we have n samples and d features at time t. But at time $t+1$, we either have one more sample (top) or one more feature (bottom). 136
12.2 Difference between learning from a stream of samples (left) and a stream of features (right). In both cases, we have n samples and d features at time t. However, at time $t+1$, we either have one more sample (left) or one more feature (right). 142
12.3 Updating the system of equations by appending a single column into triangular matrix R. 146
12.4 Comparison of the classifiers based on ROC-AUC. Groups of classifiers that are not significantly different (at p=0.01) are connected. 152
12.5 Classifier ROC-AUC and cross-validation runtime tradeoff. Regularized discriminant analysis (RDA) and linear discriminant analysis (LDA) are on the knee-point. The runtime is for the *offline* use case (all features are available at once). If we evaluated the classifiers on a stream of features, the *average* runtime of some of the offline classifiers would be in days. 153
12.6 Inserting a new feature into QDA is faster and scales better than the calculation of QDA from scratch. 155

List of Tables

2.1 Information propagation algorithms can be categorized into 3 categories based on the direction of propagation. . . . 9
2.2 The result of application Snowball algorithm on data from Figure 2.3. The sum of rows over all tables is 93.000. . . . 11
2.3 Anatomy of propositionalization algorithms. . . . 20
2.4 In a complete block design, every treatment is run for every block exactly once. . . . 22
2.5 Hypothetical example of a situation, where imputation of missing values results in misleading conclusions. The values represent classification accuracy (tied rank). Missing values are represented with a question mark. . . . 23
2.6 List of datasets used in the meta-analysis. . . . 31
2.7 List of algorithms used in the meta-analysis. . . . 32

3.1 Illustration of different naming conventions. . . . 48
3.2 Examples of dirty text in relational repository. . . . 49

5.1 List of databases in the repository. . . . 73

6.1 List of the used relational datasets. The size is in MB including indexes. Type describes the origin of the dataset. . . . 78
6.2 The 10-fold cross-validation estimate for the accuracy with a decision tree. A hyphen means that the propositionalization algorithm crashed on the dataset. The best results for each dataset are highlighted. . . . 79

7.1 Feature importance for primary key identification for different feature sets. Higher weight means higher importance. . . . 88
7.2 Feature importance for foreign key constraint identification. Higher weight means higher importance. . . . 89

7.3 Literature review of different approaches to foreign key constraint identification on TPC-H 1GB database. "D" stands for data, "M" stands for metadata. Unknown values are represented with a question mark. 89
7.4 Empirical evaluation of metadata-based approaches to foreign key constraint identification on 72 databases together. 90

8.1 Unsupervised measures categorized by the level of label interactions. n represents the count of all the labels in the data set. 100
8.2 List of the used multi-label data sets. 101

9.1 List of the used relational datasets from relational repository [A.12]. Regression and polynomial classification problems were converted to binary problems with the logic described in Classes column. Threshold column defines the split to training and testing set. 108
9.2 The effect of temporal normalization and inclusion of slope as a feature on AUC-ROC. D&D stands for detrended and deseasoned. Bold font indicates the best result for a dataset. 109

10.1 List of parameters to approximate common aggregate functions. 116
10.2 List of the used relational datasets from relational repository [A.12]. Regression and polynomial classification problems were converted to binary problems with the logic described in Classes column. 117
10.3 AUC-ROC for different aggregation methods. Bold font indicates the best result for a dataset. 119

11.1 Used databases. The range of relevant features is estimated with forward & backward selection with a decision tree (the percentage of features when meta-learning feature selection reaches accuracy corresponding to accuracy obtained on all the features). 125
11.2 Taxonomy of feature functions (data type they work on: c-character, n-numeric, t-temporal). The horizontal axis differentiates between feature functions working on a single attribute and multiple attributes. The vertical axis differentiate between feature functions working on a single tuple and multiple tuples. 126
11.3 Expected standardized relevance (bigger is better), runtime (smaller is better) and redundancy (smaller is better) of feature functions (sorted by the feature utility). 127
11.4 Leave-one-out accuracy of individual models. 129
11.5 Meta-features for relevance prediction. 129
11.6 Meta-features for redundancy prediction. 129
11.7 Meta-features for runtime prediction. 130
11.8 POPs for all databases based on the used individual models. PI column contains the upper 95% prediction interval of POPs for random ordering of the features. The best values are in bold. 130

11.9 Contribution of models to POP. Adjusted R^2: 0.564. 131
11.10 Contribution of models to reduction of the area under misclassification error curve. Adjusted R^2: 0.486. 132
11.11 Contribution of meta-feature categories to the reduction of the count of engineered features needed to reach or surpass model accuracy obtained on a complete set of features. Adjusted R^2: 0.308. 133

12.1 List of used data sets. 140
12.2 Training and scoring times in seconds for scikit-learn QDA and the proposed updatable QDA, when we insert the last feature. Dataset metadata: count of samples, features, classes, and the maximal absolute Pearson's correlation coefficient between two different features in the dataset. Predictions from scikit-learn and the proposed QDA are identical. The datasets are ordered by the obtained speed up. 150
12.3 The used classifier hyperparameters. The classifiers are described in Section 12.2.4.2. 151
12.4 Testing ROC-AUC from cross-validation. 154
12.5 Model size per class for scoring (the top 3 lines) and model update (the bottom 3 lines). The offline implementation is in Equation (12.6), and the online version is in Equation (12.15). 156

A.1 List of implemented feature functions. 206

Glossary

attribute is a column in a table. In this text, the difference between an attribute and a feature is that an attribute is a column in an input table, while a feature is a column in an output table produced with propositionalization 1

feature function produces features from attributes 12

join table, also "associative table" or "junction table", is used in relational databases to represented many-to-many relationships between tables 2

non-target table, also "background table", is any table in a database that is not the target table 2

table is a data structure that consists of a heading and an unordered set of tuples, which share the same type 1

target is an attribute with the classes (in case of classification) or continuous values (in case of regression) that we wish to predict 2

target id is a unique identifier that identifies entities, for which we aim to perform the prediction 2

target table is a single table that contains the target, the target id, and optionally the target timestamp 2

target timestamp determines the moment when we wish to make the prediction 2

Acronyms

AV Attribute-Value 7
CFS Correlation Feature Selection 17
CSV Comma-Separated Values 37
FFT Fast Fourier Transformation 12
GUI Graphical User Interface 61
IG Information Gain 50
ILP Inductive Logic Programming 14
JDBC Java Database Connectivity 43
JRE Java Runtime Environment 209
JSON JavaScript Object Notation 36
MI Multiple-Instance 19
MV Multi-View 17
ODBC Open Database Connectivity 59
RL Relational Learning 19
SQL Structured Query Language 1
SVM Support Vector Machine 17
XML Extensible Markup Language 36
XSD XML Schema Definition 59

Chapter 1

Introduction

1.1 Background

Whenever we want to make a descriptive, predictive or prescriptive model based on relational data stored in a Structured Query Language (SQL) database, we are confronted with a problem – the source data are in the form of many tables, but many modeling algorithms require data in the form of a single table. There are two common approaches how to tackle this incompatibility: Convert the data into a single table. Or adapt a propositional algorithm (an algorithm working with a single table) to work with the multiple tables [211, p. 346].

1.2 History and Significance

Based on the research of Getoor [86], relational classifiers existed already in 1976 – the same year when SQL was released into industry [33]. Nevertheless, it is still common in industry that relational classification problems are solved with propositional classifiers and the data transformation is done manually.

1.3 Terminology

The thesis is about temporal relational learning from relational databases. Hence, we borrow terminology from three domains: relational databases, relational learning, and time series analysis.

1.3.1 Relational Databases

A relational database is a collection of tables called relations, each of which is assigned a unique name. Each relation (see Figure 1.1) consists of a set of attributes and stores a large set of tuples.

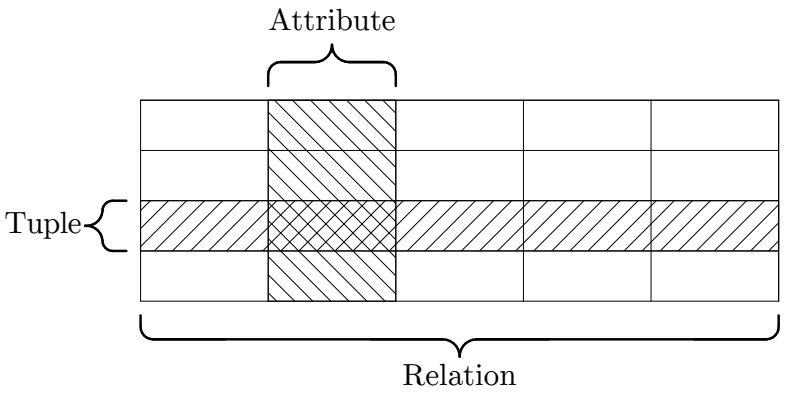

Figure 1.1: Relation, tuple and attribute represented as table, row and column respectively.

Relationships are a logical connection between two different tables. A relationship can be either one-to-one (1:1) or one-to-many (1:n). Many-to-many (n:n) relationships are modeled as a pair of one-to-many relationships with a join table (also called a junction table).

1.3.2 Relational Supervised Learning

In relational supervised learning, a relational database is composed of a single target table and n non-target tables, where $n \in \mathbb{Z}_{\geq 0}$. The target table contains m targets, where $m \in \mathbb{Z}_{>0}$, a single target id, an optional single target timestamp and l attributes, where $l \in \mathbb{Z}_{\geq 0}$. Non-target tables (also called background tables [95]) contain additional information about the target ids. Non-target tables are either directly connected to the target table with relationships or over other non-target tables.

1.3.3 Time Series

A time series is a series of numerical data points taken over time. A time series can be taken at fixed constant time steps or varying time steps. If varying steps are used, the size of the steps can be either ignored (as in "sequence learning") or accounted for.

1.4 Problem Statement

This thesis focuses on automatic conversion of relational data into a single table in a process called propositionalization [228, p. 812]. Propositionalization is a research topic at least from 1991 [157]. However, some of the propositionalization algorithms are designed to deal only with static data [190] and when they are applied to temporal data (with a time dimension), they suffer from *data leakage from the future*. For example,

if a bank wants to predict propensity to default (the probability that a customer is not going to pay off his/her loan), the bank may want to know the propensity to default before the bank accepts the customer's loan application. Algorithms designed for static datasets calculate the estimates for the time of the last record in the data. In the case of a Financial dataset [12], a popular benchmark dataset for relational classification, it means that propensity to default is estimated, when some of the outcomes are already known and written in the data.

History The problem with data leaking from the future was already recognized by Frank et al. [77] and the applied remedy was to manually remove customer's records after the date of the loan application. An algorithmic remedy was proposed by Neto et al. in CoMoVi [190].

The solution in CoMoVi defines an *observation point* – the time when we wish to have the prediction. All data before the observation point are used for model training. And all data after the observation point are used for model evaluation. Predictor Factory preserves the notion of observation point, but enhances it with a notion of *black out* and *training window* (see Figure 1.2).

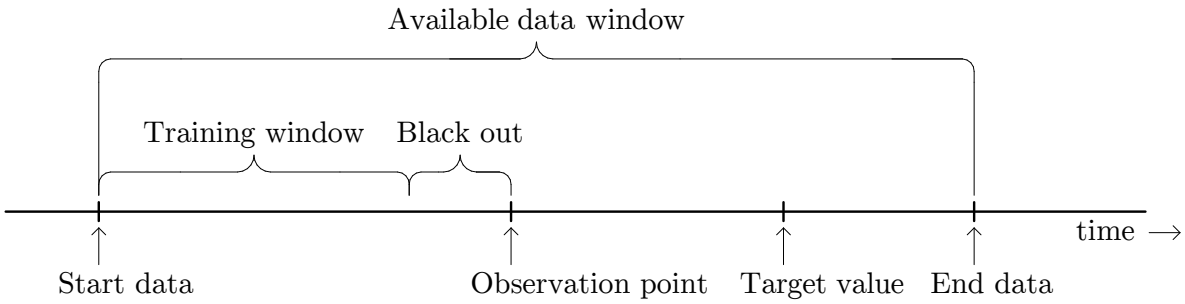

Figure 1.2: Segmentation of data based on time.

Black out Sometimes it may take time until data from all data sources are processed and loaded into a database. For example, a current industry standard for the maximum length of the delay in the bank industry for data-warehouses is 24 hours. In praxis, it means that whenever we make a prediction based on the data in the data-warehouse, the data from the last 24 hours may not be present. To simulate this delay in data availability, the newest data before the observation point are ignored. The length of this filter is given by *black out* parameter and can be set to zero to turn of the feature.

History length A model may use data from an operational database to be able to use the newest data. However, operational databases do not necessarily have to store complete transaction history. For example, the current standard in the banking industry is to preserve only the last one or two years in the operation databases. To simulate the

limited history length, all data older than *history length* from the observation point are ignored. To ignore this feature, history length can be set to infinity.

1.5 Structure of the Dissertation Thesis

The thesis is organized into 14 chapters as follows:

1. Introduction: Describes the topic and contributions of this dissertation thesis.
2. Background and State-of-the-Art: Provides the decomposition of propositionalization algorithms into 8 building blogs and identifies the state-of-the-art in relational learning.
3. Predictor Factory: Describes the design of our propositionalization tool, Predictor Factory.
4. Implementation: Describes the architecture of Predictor Factory and justifies the chosen technologies.
5. Relational Repository: Introduces relational datasets, which were collected for Predictor Factory evaluation.
6. Empirical Evaluation: Provides comparison of Predictor Factory to other propositionalization tools.
7. Foreign Key Constraint Identification: Because relational datasets sometimes miss the information about the relationships between the tables in the database, we describe an algorithm for foreign key constraint discovery. Only once we know the relationships between the tables, we can use data stored in non-target tables to predict a targets in the target table.
8. Stratified Cross-Validation by Multiple Columns: Describes how to handle multiple targets in the target table.
9. Trend and Seasonality Elimination: Describes how to detrend and deseason relational data.
10. Generalized Aggregates: Common aggregate function like like *min*, *max*, or *count* ignore the order of the values. We provide a generalization of these aggregates, which is suitable for both, static and temporal data.
11. Meta-learning: Achilles heel of propositionalization is that it produces a wast quantity of irrelevant and/or redundant features. However, since the features are generated sequentially and not all at once, we can build a meta model to predict the features' univariate relevancy, redundancy and runtime. Based on the empirical results, meta learning reduces propositionalization runtime to one-tenth of the original runtime without loss of accuracy of the downstream model.

12. Learning on Stream of Features: The disadvantage of the implemented meta learning is that it is unaware of the used downstream model. To remedy that, we modified two classifiers, random forest and discriminant analysis, to work efficiently on a stream of features, making it practical to train and use a meta model, which predicts improvement of the accuracy of the chosen downstream model, when we calculate another feature.
13. Discussion: Describes the experience with Predictor Factory.
14. Conclusions: Summarizes the results of our research, suggests possible topics for further research, and concludes the thesis.

Chapter 2

Background and State-of-the-Art

> Those who don't know history
> are doomed to repeat it.
>
> Edmund Burke

2.1 Theoretical Background

A propositionalization algorithm can be decomposed into 7 building blocks, depicted in Figure 2.1. Each block is discussed in a separate section. The sections are ordered from top to bottom.

2.1.1 Data Representation

The most common machine learning task is supervised attribute-value (AV) learning [16]. In supervised AV learning, each instance has a label and instances are represented by a fixed set of variables (also called attributes). However, it is not always possible to represent an instance with a fixed set of attributes. In these situations, relational learning often provides a solution. Different formalisms have been used in machine learning to describe instances with a variable set of attributes. The most important ones are graph, relational, and logic.

Graph Graphs consist of vertices (also called nodes or points) and edges (also called links or lines). Generally, vertices describe one type of entity (e.g., people) and edges describe the relationships between the vertices (e.g., friendship). Nevertheless, it is possible mix up different types of entities in a single graph (e.g., legal and natural people) and differentiate between them with labels. Common graph databases (e.g., Neo4j,

Apache Giraph, or GraphDB) are all schema-less and represent attributes (also called properties) of the vertices and edges with name-value pairs. The learning from graphs is called graph mining [17]. The typical supervised learning tasks in graph mining are vertex attribute value prediction (e.g., what is the age of the person?), edge attribute value prediction (e.g., how long will the friendship last?), and edge prediction (e.g., are these two people friends?).

Relational In comparison to graphs, relational model represents both, vertices and edges, with a single data structure: tables. Historically, this simplified the implementation of the databases. Also, relational databases do not enforce that entity tables are linked to other entity tables only over relationship tables. Nevertheless, in comparison to typical graph databases, relational databases (e.g., Oracle, Microsoft SQL Server, or PostgreSQL) require a strict schema, making secondary processing (reporting, predictive modeling...) generally easier, as we are guaranteed, that each record in the table has the same set of attributes. The learning from relational databases is commonly referred as relational data mining or multi-relational data mining [17].

Logic Logic formalism is attractive because it can describe not only facts, but also rules (e.g., concept definitions, inference rules, or background knowledge about the domain). The disadvantage of logic-based representations is that many types of reasoning, including learning, can be computationally expensive or even intractable [17]. An example of a logic database is Datomic [130], which is a variant of Datalog. From the point of temporal relational learning, the nice property of Datomic is that each attribute in the database is, without exception, timestamped and that the database is append only (changed or deleted attribute values are marked as invalid but left in the database). This combination of properties allows reliable "time-traveling", or in our case, it allows us to reliably prevent data leakage from the future in predictive modeling.

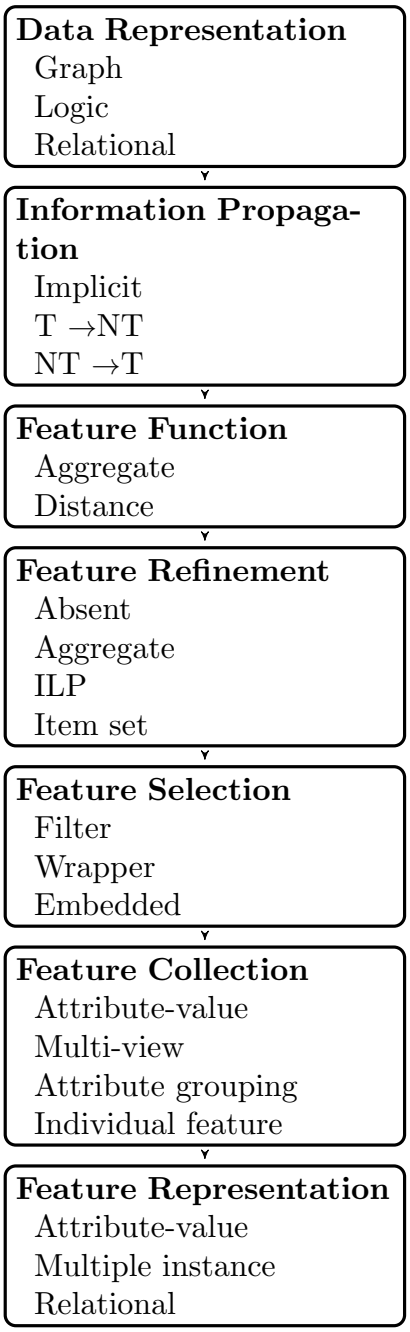

Figure 2.1: Decomposition of relational learning algorithms into 7 blocks. Each block contains a non-exhaustive list of applicable options.

Summary All these representations are, at least in theory, equally capable to represent relational data, as we can convert data from one representation into another and back, without loss of information. In the praxis, the conversion from one representation to another might be troublesome due to practical data storage limitations. For example, Datomic has a hard limit of 2^{20} attributes per entity (a little over 1 million)[4, p. 214], while Oracle has a hard limit of 1 000 attributes per table.

2.1.2 Information Propagation

During information propagation, the content of non-target tables is matched up with the content of the target table. This matching ensures that evidence in the tables is matched with the right entity (target id) in the target table.

Each data representation formalism has its own mechanism how to perform matching. However, the implicit mechanisms are, particularly in the relational and in less extent in logic formalism, the bottlenecks in the processing. Hence, multiple alternative approaches have been developed.

Besides the implicit approaches, all the information propagation algorithms can be divided into two types: you either bring the evidence from the non-target tables to the target table (NT → T) or you bring the target table to the non-target tables (T → NT) – see Table 2.1.

Name	Approach
Logic	Implicit
Graph	Implicit
Universal join	Implicit
Snowball	T → NT
Selection Graph [136]	T → NT
Tuple Id propagation [266]	T → NT
Target to non-target [88]	T → NT
Non-target to target [88]	NT → T
RollUp [137]	NT → T

Table 2.1: Information propagation algorithms can be categorized into 3 categories based on the direction of propagation.

Universal join Universal join is the implicit approach how to match evidence with the target entities in a relational database. In universal join, all non-target tables are left joined to the target table[1]. When all the relationships between the target table and the non-target tables are 1:1 or n:1, the resulting table has as many rows as the target table

[1]Left join is used instead of generally faster inner join to avoid the disappearance of entities in the presence of 1:0..n relationships between the target table and the non-target table.

and everything works nicely. However, if the relationships are of type 1:n, the resulting table grows in the count of rows.

Example 2.1.1. Let's imagine a star schema with a target table as the *fact table* and 3 non-target tables (Figure 2.2) as the *dimensions*. If the target table has 100 rows and each dimension table has exactly 10 rows associated with each row in the target table, then the resulting table will have $100 \cdot 10^3 = 100.000$ rows. With a snowflake schema, where each dimension has 3 sub-dimensions and each sub-dimension has exactly 10 rows associated with each row in the dimension (Figure 2.3), the situation is getting even worse – we get $100 \cdot (10^3)^3 = 100.000.000.000$ rows in the resulting table.

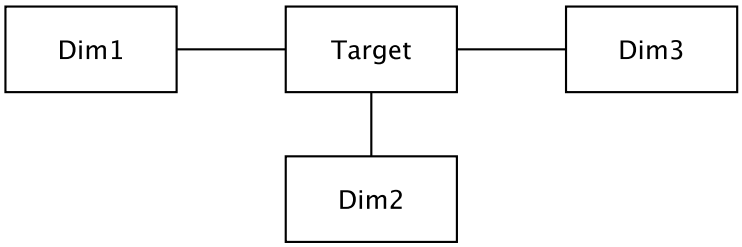

Figure 2.2: An example of a star schema.

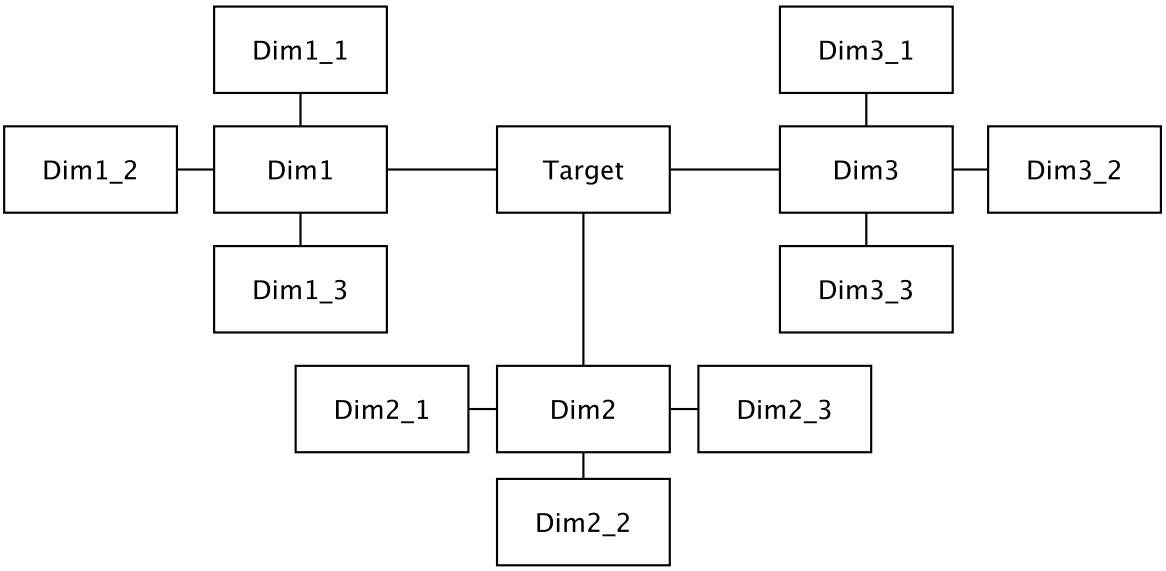

Figure 2.3: An example of a snowflake schema.

While universal join approach is attractive for its simplicity, due to possible difficulties with making, storing and processing the resulting table, universal join approach is not universally applicable [151].

Snowball The problem with the big resulting table can be somehow reduced by using independent joins – instead of making a single resulting table, which contains all the non-target tables, we can join the target table with each non-target table individually.

Example 2.1.2. Let's consider the snowflake schema from Figure 2.3. With independent joins, we get tables listed in Table 2.2.

Join function	#Rows in the resulting table
$t1 = join(targetTable, dimension_1)$	1.000
$t2 = join(targetTable, dimension_2)$	1.000
$t3 = join(targetTable, dimension_3)$	1.000
$t1.1 = join(t1, dimension_{1.1})$	10.000
$t1.2 = join(t1, dimension_{1.2})$	10.000
$t1.3 = join(t1, dimension_{1.3})$	10.000
$t2.1 = join(t2, dimension_{1.1})$	10.000
$t2.2 = join(t2, dimension_{1.2})$	10.000
$t2.3 = join(t2, dimension_{1.3})$	10.000
$t3.1 = join(t3, dimension_{1.1})$	10.000
$t3.2 = join(t3, dimension_{1.2})$	10.000
$t3.3 = join(t3, dimension_{1.3})$	10.000

Table 2.2: The result of application Snowball algorithm on data from Figure 2.3. The sum of rows over all tables is 93.000.

Since this method propagates data from the target table into non-target tables in an incremental matter, this method is further referenced as Snowball method.

Tuple ID Tuple ID propagation method [266] is one of the most efficient methods in the regard of the used space – tuple ID propagation does not create new tables. It just adds a vector type column into the non-target tables. And this new column contains references to the entities in the target table. The only complication is that not all databases support arrays, the most natural data type for storing the vector of references.

Performance comparison of snowball and Tuple ID propagation was done by Ghionna et al. [88].

RollUp The last discussed method how to propagate information is called RollUp [137]. This method is the only representative of the approach "data to the target table". With this method, the furthest non-target tables from the target table are joined with the less

distant tables. And this process is repeated until all the non-target tables are joined to the target table. What makes this method unique is that feature functions (which are discussed in Section 2.1.3, but for illustration, imagine aggregate functions like *min*, *max*, *sum*, etc.) are applied to relationships 1:n before the tables are joined, while with other methods, feature functions are, if at all, applied after information propagation. Consequently, the product of a join always has as many rows as the table closer to the target table. Hence, from the point of the row count, RollUp method is as efficient as Tuple ID propagation.

A slight disadvantage of RollUp method is that it is not always evident which feature function to use until the non-target table is propagated to the target table, which contains the target needed for feature relevancy evaluation. Knobbe [137] simply applies all possible feature functions. Hence, the count of the calculated features grows exponentially with the count of relationships in the path of propagation [120]. This problem was addressed by Gjorgjioski and Džeroski in PRORED [89] by the introduction of the stochastic sampling of the feature functions.

A theorized solution of the problem could be to use a greedy algorithm. First, the target would be propagated from the target table into all non-target tables, permitting application of feature relevance evaluation. Second, only the most relevant (predictive) features would be rolled up to the target table with RollUp method. Of course, greedy algorithm may not find the best combination of the feature functions. Also, for non-target tables that are in the closest proximity (as given by the length of the propagation path) and that are not further used for the target propagation, it can be reasonable to directly apply RollUp method, since the speed gain provided by the greedy algorithm can be smaller than the cost associated with the target propagation.

Summary An important consideration related to each information propagation algorithm is whether we want to be able to calculate multivariate features (feature calculated from several attributes – they are further discussed in Section 2.1.3). This extension is perfectly natural for RollUp algorithm since all the attributes, though possibly aggregated multiple times, in the end end up in the target table. However, all analyzed algorithms that are using RollUp (Polka [137], PRORED and DFS [120]) are exclusively using univariate feature functions that are using just one attribute as an argument. An example of a propositionalization algorithm that calculates feature functions on attributes spanning several tables is CLAMF [77].

2.1.3 Feature Function

Feature functions are applied on data in the hope that the transformed data will be easier to work with. An example of such transformation in image processing is fast Fourier transformation (FFT). After application of FFT, convolution is simpler, making it attractive to transform the image with FFT and back with inverse FFT just to perform the convolution.

2.1.3.1 Categorization

There are multiple categories of feature functions:

1. Relational (like *min*, max, *correlation*...)
2. Distance and similarity measures (like *Euclidean*, *cosine*, *Jaccard index*...)
3. Kernel functions (like *Gaussian*, *centrality*, *random walk*...)

Relational feature functions can be divided into 4 categories [77, 120] based on the count of parameters and dimensionality of the parameters that the function accepts. The horizontal dimension in Figure 2.4 says whether the feature function works on a single attribute or on multiple attributes. The vertical dimension says whether the feature function works on a single tuple or multiple tuples.

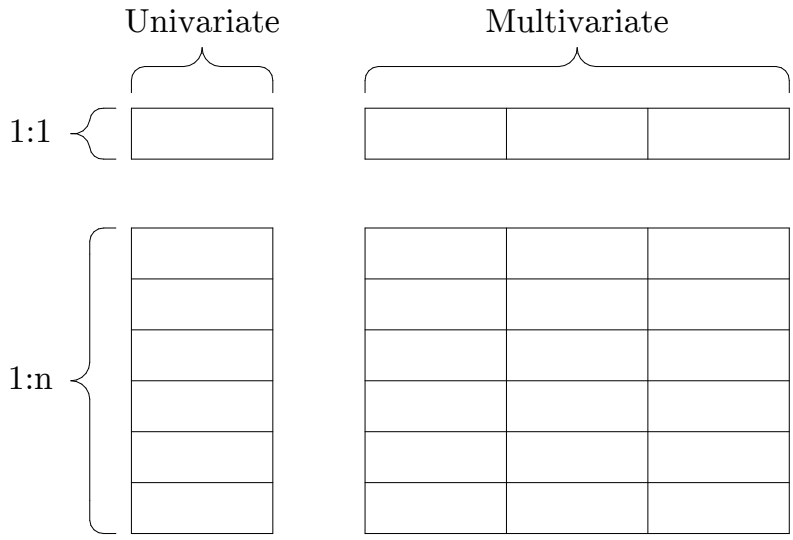

Figure 2.4: Four types of feature functions. From left to right, then top to bottom: scalar, horizontal aggregate, vertical aggregate, cross.

1:n Whenever we have a 1:n relationship between an entity in the target table and a non-target table, there are two approaches how to deal with that. Either we can use a generalized multiple instance classifier [75], like in the experiment done by Reutemann [218] and later on by Schulte [235]. Or we can transform the data into 1:1 relationship with feature functions. In the Schulte's empirical evaluation, the classification employing feature functions was both, more accurate and faster than classification employing a multiple instance classifier. However, it is noteworthy that Schulte used very simple multiple instance classifiers.

Univariate With univariate features we assume that all interactions between the attributes are modeled with a classifier. An example of a univariate feature function are *sum* or *count*.

Multivariate With multivariate feature functions, interactions between attributes can be captured. An example of a multivariate feature function is calculation of BMI (Body Mass Index), which is calculated as:

$$BMI = \frac{weight}{height^2}. \tag{2.1}$$

There are two reasons why to use multivariate features. First, many metrics in physics or econometrics can be described with a combination of a small set of variables (e.g., the metric system is based on 7 base units, which include mass and length). And if physicists and economists found the ability to combine attributes explicitly useful, maybe we should be able to do that as well. A non-exhaustive list of approaches how to construct multivariate features include: Inductive Logic Programming (ILP) based, decision tree related [159], genetic programming related [57] and annotation based [241]. Second, multivariate features can help to decrease bias of models (particularly linear).

Scalar An example of a scalar feature function is a conversion of customer's date of birth into the age of the customer. The idea behind this transformation is, that a model learned on the date of birth is likely going to get obsolete faster, than a model based on the age. This assumption can be particularly true if we predict propensity to buy (the probability that a customer is going to buy a product) of a legislatively restricted product like alcohol or social insurance, where restrictions on the customers' age apply.

Cross Cross function works with multiple attributes and multiple tuples. An example of such function is average monthly spending, calculated from a transaction table with date and amount attributes.

2.1.3.2 Aggregates

In the realm of ILP, the most common aggregate is an *existential quantifier* [187]. In the realm of relational databases, one of the most common aggregates are *mean*, *max*, *min*, *sum*, *standard deviation* and *count* [144, 137]. Nevertheless, there is infinitely many possible aggregates[2]. For example, Frank proposed multivariate aggregates like *correlation* [77]. Perlich proposed how to aggregate ids [200]. It is also possible to use graph features, like different measures of *centrality* [135]. However, note that differently named feature functions may actually represent the same function. For example, *count* aggregate in a relational database and *degree* in an undirected graph are the same feature function.

[2]For a trivial illustration of the argument, imagine a percentile function with a continuous percentile.

Existential quantifier The *existential quantifier* expresses that the statements within its scope are true for at least one instance of something. For example, RSD [143] algorithm is using only *existential quantifier* for classification.

Count The advantage of the *count* feature function, in comparison to the *existential quantifier*, is that it has a higher resolution. However, on small datasets, the *existential quantifier* is commonly preferred. Nevertheless, Perovšek showed that Wordification algorithm, which is exclusively using *count* features, delivers higher accuracy on *train* dataset than RSD [201], even though the dataset contains only 20 instances.

Continuous variables There are two common approaches how to get features from continuous attributes – discretization and aggregation. After discretization, we can use *existential quantifier* or *count*. Or we can approximate the bag of numbers with descriptive statistics like *avg* or *standard deviation*.

Distance and similarity We may measure distance or similarity of a bag of tuples to a bag of training tuples belonging to the positive class and use the resulting value as a feature. This idea was exploited in ACORA [200]. For a review of distances, see [215] and [50].

Kernel It is also possible to use kernels as features. Either we may use fixed kernels or we may learn the kernels. Kernels are used in kFOIL [155] and kLog [78].

Summary Relational data do not have to consist only of nominal and continuous attributes but might also include text, images, sounds, and other types of data. However, feature extraction from these domains (with the exception of the text) is out of the scope of this thesis due to the sheer size of the problem.

2.1.3.3 Desirable Properties

The main properties of a well-behaved feature function are:

Discrimination Features should have high predictive power.

Reusability It should be possible to reuse features in different models and applications.

Transformability Besides directly reusing a feature f, it should be easy to use a transformation of it (e.g., $\log(f)$, $max(f)$, $\sum f_t$ over a time window...).

Interpretability It should be easy to understand the meaning of features and interpret their values. This property is particularly useful for prescriptive analysis.

Reliability It should be easy to identify issues with the features.

2.1.4 Feature Refinement

The idea behind feature refinement is an incremental improvement of a preliminary feature into a better feature [27]. An example of such refinement can be optimization of a feature parameter (e.g., the used quantile), generalization or specialization.

ILP An Inductive Logic Programming (ILP) algorithm starts from an initial hypothesis and applies pre-defined operators called refinement operators (e.g., generalization or specialization) to create new hypotheses. The aim is to find such hypothesis that covers many positive examples and minimum of the negative ones [27]. For detail overview of ILP methods see [211].

Frequent item set An example of a frequent item set algorithm is Apriori [3]. Apriori algorithm proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The discovered frequent item sets are then treated as features to the classifier.

Aggregate Even simple aggregate feature functions like minimum or maximum can be optimized. In FICO Data Spiders [69], the time frame over which to calculate the aggregate can be optimized. For example, if it holds that the recent data are more predictive of the target than the old data, the aggregate calculated over the last month can be more predictive than the same aggregate calculated over the last year.

However, sometimes it is the older data that are more predictive of the target value. For example, we would expect that loss given default (the amount of funds that is lost by a financial institution when a borrower defaults on a loan) is more sensitive to the ability of the borrower to pay back his/her first installment than his/her second installment.

But even if the dataset does not contain time dimension, we may want to optimize the aggregate functions to better deal with noise in the data. For example, in the presence of noise, it may turn out that the 94^{th} percentile is more predictive of the target value than the plain maximum. Or that the n-th biggest value, where n is a constant that does not change based on the count of samples, is a better predictor of the target value than the plain maximum.

2.1.4.1 Summary

The refinement procedures can be computationally demanding. ILP and frequent item set methods often use mechanisms to prune unpromising settings quickly. It is also possible to perform pruning, when aggregates are used [256]. Nevertheless, some authors prefer to use heuristics like genetic algorithms [69].

2.1.5 Feature Selection

Although the choice of a feature selection algorithm should depend not only on the data but also on the used model [100], some propositionalization algorithm contain an embedded feature selection, either because the design greatly benefits from it, as is the case of NT →T propagation [137] or multi-view (MV) classification [94], or because the propositionalization algorithm requires an optimization criterium as it performs feature refinement.

Feature selection methods can be divided into filters, wrappers and embedded methods [101]. Filters select subsets of variables as a pre-processing step, independently of the chosen model. Wrappers use the model as a black box to score subsets of variable according to their predictive power. Embedded methods perform variable selection in the process of training and are usually specific to given learning machines.

The individual methods can be further divided. For example, filter methods can be divided based on the knowledge whether they treat features individually (univariate) or together (multivariate). Following paragraphs give a non-exhaustive list of methods used in relational classifiers.

Univariate filter Propositionalization algorithms are critiqued by de Raedt for the production of many irrelevant features [209]. The simplest posthumous treatment of the issue is to apply a univariate filter. For example, a specific problem of NT →T propagation is that the count of the produced features grows exponentially with the count of 1:n relationships in the path between the non-target table and the target table [120]. To combat this exponential growth of the features, Gjorgjioski successfully used stochastic sampling of the feature functions [89].

Multivariate filter Another critique of propositionalization algorithms by de Raedt is the creation of the many highly correlated features [209]. Correlated features can cause troubles in algorithms like Linear Discriminant Analysis or Naive Bayes. To combat this problem, Guo [30] opts for Correlation Feature Selection (CFS) method, which is based on the following hypothesis: "Good feature subsets contain features highly correlated with the classification, yet uncorrelated with each other" [102][3].

Wrapper Wrapper approach is taken by kFOIL [155], which evaluates the quality of a feature refinement with Support Vector Machine (SVM).

Embedded Dataconda [229] is using Lasso regression to select the top 20 features.

[3]Note that "correlation does not imply redundancy" [25].

Summary A feature selection should generally optimize following criteria: Maximize relevancy, minimize redundancy, maximize stability[4], and minimize time consumption [34].

2.1.6 Feature Collection

When features are calculated, they can be either collected into a single table, multiple tables or each feature can be left alone.

Attribute-value Propositional tools traditionally generate a single "feature table" that contains all generated features [144]. The advantage of the attribute-value (AV) approach is that the produced table can be directly processed with common propositional tools. The disadvantage of this approach is that the output table may have to store more features than is it the limit of the database on the maximal count of columns in a single table (e.g., 1 000 in Oracle).

Multi-view In a multi-view (MV) approach, an algorithm produces a feature table for each table in the database. A learner is then executed on each individual feature table. And the predictions of the learners are then ensembled together with another learner [96].

There are two significant advantages of such approach. First, if the information present in each feature table is independent of the information in other feature tables, the learning is greatly simplified without loss of accuracy. Hence, the MV approach is efficient in term of running time. Second, the MV approach partially alleviates the AV's limit on the maximal count of columns in a single table because the generated features are spread over multiple tables. Still, if the count of the generated features is bigger than the count of attributes, they may not fit into the limit. The disadvantage of the MV approach is that if the assumption of the table independence does not hold, the accuracy is hampered.

Attribute grouping The MV approach can be taken a level further – instead of joining features belonging to a single table together, features from the same attribute can be joined together. To our best knowledge, this approach has not been discussed in the literature.

The advantage of attribute grouping is that it allows easy parallelization in the case that each attribute is stored in a different computation node – the attributes do not have to be transmitted between the nodes.

Individual feature In the end, it is also possible to train a model on each feature individually and merge the produced estimates with another model. This approach was taken by Schulte in [235].

[4] If we want to perform ensembling, we may want to, on the other hand, promote diversity.

2.1.6.1 Hierarchy

It holds that what can be learned from the individual features, can be learned with the attribute grouping. What can be learned with the attribute grouping, can also be learned with the MV approach. And what can be learned with the MV approach can be learned with the AV approach:

$$AV \subset MV \subset Attribute\ grouping \subset Individual\ features \tag{2.2}$$

2.1.7 Feature Representation

We can partition learning algorithms based on the data structure that is used for supervised learning.

Attribute-value learning In attribute-value (AV) learning all the data resides in a single table and each record has its own label.

Multiple-instance learning In multiple-instance (MI) learning, each input object or event is represented by a set of instances, named a bag, and it is the bag that carries a label.

In early MI research, a strong assumption was made regarding the relationship between instances inside the bags and the label of the bag. This assumption is called the standard MI assumption. Under this assumption, each instance has a hidden class label which identifies it as either a positive or a negative instance, and a bag is considered to be positive if and only if it contains at least one positive instance. This is generally believed to be true for the musk drug activity prediction problem, where a molecule will have the desired drug effect if and only if one or more of its conformations binds to the target binding site [54]. However, in other problem domains, this assumption may not apply. Consequently, different or more general assumptions were developed. For a review of MI learning assumptions see Foulds's and Frank's work [75].

Relational learning In relational learning (RL), the data can be spread over multiple tables. And one of the tables, which is called target table, contains the target.

2.1.7.1 Hierarchy

It holds that problems solvable with an AV learner can be learned with a MI learner because AV learning is just a subset of MI learning, where each bag contains only one instance.

It also holds that problems solvable with a MI learner can be learned with a relational learner, because a MI problem can be represented in the form of two tables, where target

table contains labels of bags and the second table contains instances with a foreign key to the target table:

$$AVL \subset MIL \subset RL \tag{2.3}$$

De Raedt suggested that the MI paradigm could be the sweet spot between the AV and relational representations, being more expressive than the former, and much more easy to learn than the latter [209].

2.1.8 Evaluation

The proposed decomposition of the problem was tested on propositionalization algorithms listed in Table 2.3.

Algorithm	Representation	Propagation	Function	Refinement	Collection	Learning	Selection
ACORA [200]	Relational	T→N	Distance	No	1 table	AV	–
Aggregated Predictions [235]	Relational	Implicit	–	No	1 table	MI	–
CLAMF [77]	Relational	T→N	Aggregate, ILP	Yes	1 table	AV	–
CrossMine [266]	Logic	N→T	ILP	Yes	1 table	AV	–
Dataconda [229]	Relational	T→N	Aggregate	Yes	1 table	AV	Lasso
Deep Feature Synthesis [120]	Relational	N→T	Aggregate	No	1 table	AV	–
FICO Data Spiders [69]	Relational	T→N	Aggregate	Yes	1 table	AV	–
KXEN Event Log [232]	Relational	T→N	Aggregate	No	1 table	AV	–
LBP [55]	Logic	Implicit	Aggregate	No	1 table	AV	–
Linus [157]	Logic	Implicit	ILP	Yes	1 table	AV	–
Lynx-RSM [173]	Logic	Implicit	ILP	Yes	1 table	AV	SLS
MAFIA [122]	Graph	Implicit	Item set	Yes	1 table	AV	–
MILK [218]	Relational, Logic	Implicit	–	No	1 table	MI	–
MRC [98]	Relational	T→N	Aggregate	No	n tables	AV	CFS
RELAGGS [144]	Relational	T→N	Aggregate	No	1 table	AV	–
Polka [137]	Relational	N→T	Aggregate	No	1 table	AV	–
PRORED [89]	Relational	N→T	Aggregate	No	1 table	AV	Stochastic
REPART [279]	Relational	Implicit	ILP	Yes	1 table	MI	–
RSD [271]	Logic	Implicit	ILP	Yes	1 table	AV	–
SMFI [123]	Graph	Implicit	Item set	Yes	1 table	AV	–
Wordification [201]	Relational	T→N	Aggregate	No	1 table	AV	TF-IDF

Table 2.3: Anatomy of propositionalization algorithms.

If Multiple Correspondence Analysis is applied to the data in Table 2.3, we get a two-dimensional projection depicted in Figure 2.5. The grouping of the algorithms into 5 clusters was done manually. The labels of the clusters were determined by the shared properties of algorithms in a cluster. Note that some of the algorithms overlap in the projection, like MAFIA [122] and SMFI [123], because they have identical functional decomposition.

2.1.9 Conclusions

This section followed "divide and conquer" strategy, in which a propositionalization algorithm was decomposed into individual building blocks.

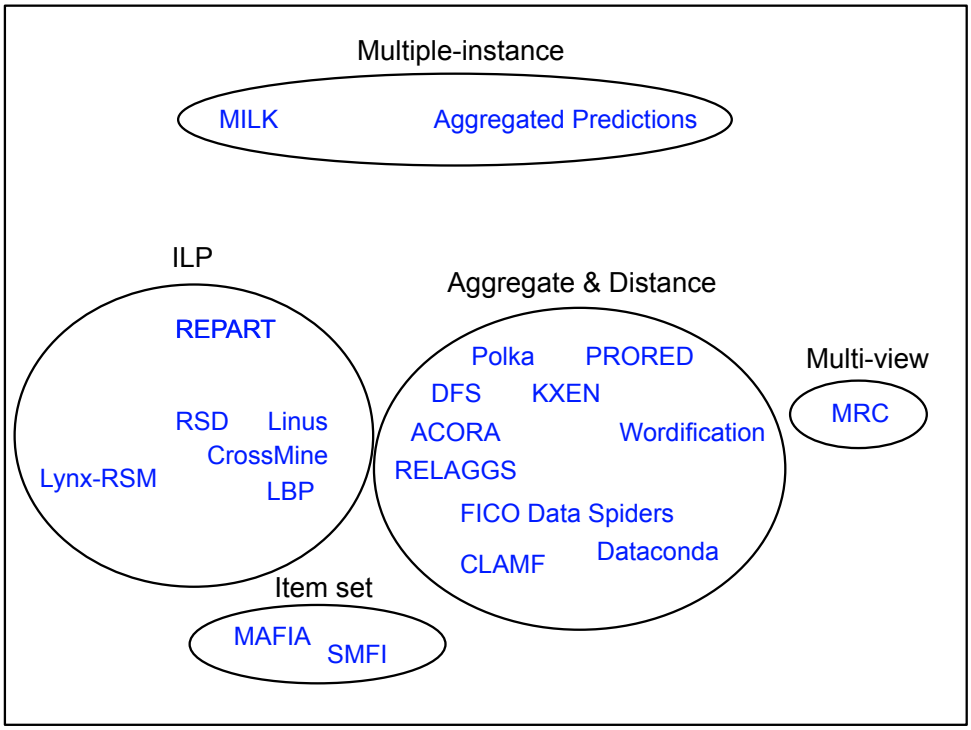

Figure 2.5: Projection of algorithms from Table 2.3 into a two-dimensional space with Multiple Correspondence Analysis.

Framework The described decomposition of a propositionalization algorithm allows construction of a framework in which many current and new relational algorithms can be implemented. Such framework would allow easy empirical comparison of algorithms, development of meta-learning in relational learning, and it would provide a testbed for algorithms that improve just a part of a relational learner.

Model The described decomposition also provides a conceptual model of how to look at propositionalization.

This is an important thing as there is not a literature dedicated to propositionalization algorithms – there are only publications about individual implementations of propositionalization algorithms[5].

New instances New instances of building blocks were identified. Namely, new ways of feature collection (attribute grouping and individual feature) were identified. Furthermore, it was identified that the MI classifiers tested in the literature on relational data were subpar. Hence, it is desirable to do new experiments in this field.

[5]There are great books about relational learning. However, their scope is much wider than mere propositionalization.

New combinations The decomposition of the problem allows a creation of new combinations, giving rise to new algorithms.

2.2 Empirical Comparison

This section contains a meta-study of different approaches to supervised relational learning. The comparison of the algorithms was done with an algorithm described in the following paragraphs.

2.2.1 Introduction

Sometimes, we may want to compare different treatments on different blocks (see Table 2.4). For example, we may want to compare a set of algorithms on a set of datasets and evaluate algorithms' performance. If we can do all the measurements, the evaluation is quite easy – we can just follow the recommendation of Demšar [48], apply an omni-bus test like Friedman test followed with a post-hoc analysis, like Bergmann–Hommel's or Li's procedures [250], and we are done.

	Block 1	Block 2	$\cdots$	Block b
Treatment 1	X_{11}	X_{12}	$\cdots$	X_{1b}
Treatment 2	X_{21}	X_{22}	$\cdots$	X_{2b}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
Treatment t	X_{t1}	X_{t2}	$\cdots$	X_{tb}

Table 2.4: In a complete block design, every treatment is run for every block exactly once.

But sometimes, we may have difficulties to collect measures for all combinations of treatments and blocks. For example, when two or more features are collinear, the covariance matrix of the entire dataset is singular, and inevitably, invertible, which causes problems in algorithms like Fisher's LDA [68]. Or the algorithms may not terminate on the dataset in a reasonable time [14]. In such scenarios, we may be inclined to perform missing value imputation [68]; however, such imputation does not only result in overly optimistic confidence intervals [110], but may also result in misleading conclusions as illustrated in Table 2.5.

The hypothetical scenario in Table 2.5 depicts evaluation of 4 algorithms A-D on 1001 datasets based on classification accuracy. However, all 4 algorithms are measured on the same dataset just once. In the rest of the cases, only two algorithms, C and D, are compared. If we apply a method based on arithmetic mean, average ranking (like in Friedman's test) or count of wins (like in sign test) on the complete dataset, we get that algorithm C is the best (the best values in the table are shown in bold), even thought

Dataset	Alg A	Alg B	Alg C	Alg D
#0	0.9 (1)	0.8 (2)	0.1 (3)	0 (4)
#1	? (2.5)	? (2.5)	1 (1)	0 (4)
#2	? (2.5)	? (2.5)	1 (1)	0 (4)
#3	? (2.5)	? (2.5)	1 (1)	0 (4)
⋮	⋮	⋮	⋮	⋮
#999	? (2.5)	? (2.5)	1 (1)	0 (4)
#1000	? (2.5)	? (2.5)	1 (1)	0 (4)
Nan-avg acc	0.9	0.8	$\approx \mathbf{1}$	0
Imp-avg acc	≈ 0.5	≈ 0.5	$\approx \mathbf{1}$	1
Avg ranking	≈ 2.5	≈ 2.5	$\approx \mathbf{1}$	4
# Wins	1	0	**1000**	0
Proposed	**1**	2	3	4

Table 2.5: Hypothetical example of a situation, where imputation of missing values results in misleading conclusions. The values represent classification accuracy (tied rank). Missing values are represented with a question mark.

the first record, the only one that is complete, suggests that algorithm A is superior to algorithm C.

The remedy in the scenario depicted in Table 2.5 is easy – we can delete all incomplete records and apply a test of our choice on the rest of the records. However, if neither record is complete, we have to either perform an imputation (and in the case of imputation with an average become susceptible to the described attack) or use an algorithm that can deal with missing values.

2.2.2 Related Work

Friedman's test [82] was developed to analyze data with one response per cell and no missingness. Subsequent modifications of Friedman's test generally relax one of the two requirements, although some permit both. The first relaxation permits missing data with one response per cell at most. The second relaxation permits multiple responses per cell [110].

Missingness Durbin proposed a Friedman-type test for a balanced incomplete block design [59], which permits missing data by design. Skillings and Mack [239] proposed a more general Friedman-type test for an unbalanced incomplete block design, which permits missing data that are missing by design or missing completely at random.

Missingnes & Multiple response Bernard and van Elteren [13] then adapted Durbin's model for scenarios with an arbitrary number of responses per cell. Nevertheless, data

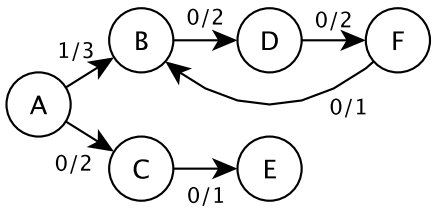

Figure 2.6: Graph with count of wins.

still have to be missing by design.

Network meta-analysis Network meta-learning algorithms [191] are better suited for a meta-analysis than Friedman-test-based algorithms as they permit multiple responses per cell and missing data do not have to be missing by design.

Still, application of network meta-learning algorithms can be troublesome because they require confidence intervals (or other parameters from which confidence intervals can be calculated) for each response. However, only around one-third (14/45) of the studied articles about relational classifiers contained information about the accuracy of reported values (the data are further discussed in Section 2.2.5.1).

Proposed Compared to methods described above, the proposed method can deal with multiple responses per cell, missing values do not have to be missing be design and confidence intervals are not required.

2.2.3 Method Description

The proposed method (called GraphRank) is motivated by graph theory.Figure 2.6 depicts an example network of 6 treatments A-F. The weights of the directed edges represent count of wins of the ancestor over the successor/of the ancestor over the successor, where the direction of the edge is from the weaker treatment to the stronger treatment.

From these partial orderings, we can estimate the complete ordering with object ranking [118] – a special application of learning to rank algorithms. However, object ranking in the most general form is NP-hard.

If we had a directed acyclic graph, we could enumerate all topological orderings that satisfy all partial constraints [255] and take the average rank of a treatment over all plausible orderings as the estimated rank [259, 260].

However, cycles in meta-analyses do appear [191]. The cycles in a graph can be identified [172] and broke at their weakest point as estimated with a sign test [48]. If the weakest point in a cycle is not unique, multiple treatments are estimated and averaged. The result of the application of the algorithm on the network fromFigure 2.6 is in Figure 2.7.

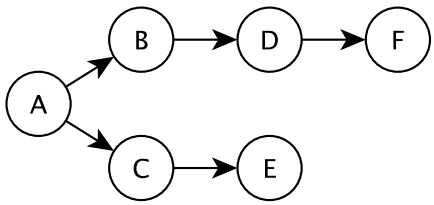

Figure 2.7: Directed acyclic graph.

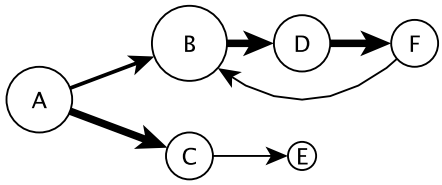

Figure 2.8: Network visualization.

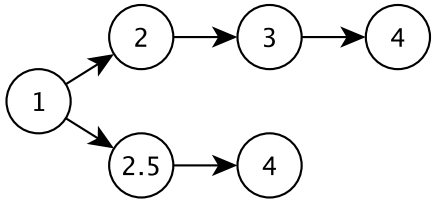

Figure 2.9: Nodes with ranking.

A BPMN layout algorithm, an alternative to hierarchical layout algorithms that produces easier to read graphs [134], is used to place the treatments from left to right based on their order. To depict the confidence in the ordering of two treatments connected with an edge, a sign test can be used. The width of the edge is then inversely proportional to the calculated p-value. Additionally, the size of a node (treatment) can be proportional to the count of trials, in which the treatment is present. The resulting graph is in Figure 2.8.

A graph with the ranking in place of the treatment labels is in Figure 2.9.

2.2.3.1 Assumptions

GraphRank assumes that the partial orderings are transitive, i.e., that $(A \prec B) \wedge (B \prec C) \rightarrow (A \prec C)$, and that the responses are independent. GraphRank does not assume any commensurability of responses or differences nor does it assume normal distributions.

2.2.4 Evaluation

The ability of GraphRank to rank treatments was compared to the ranking method used in Friedman's and Wittkowski's tests [259]. Wittkowski's test is a Friedman-type test that can deal with missing values. A nice property of Wittkowski's test, in comparison to Skillings's test, is that on a complete dataset it returns the same conclusion as Friedman's test.

2.2.4.1 Data

The algorithms were tested on a recent empirical evaluation of 179 classifiers on 121 datasets by Fernández-Delgado[68]. This dataset was selected because it is thematically close to the desired application of the developed algorithm – evaluation of relational classifiers.

2.2.4.2 Data Preprocessing

Since some classifiers in Fernández-Delgado's evaluation were not able to cope with each dataset, only a subset of 109 classifiers without any missing value was used.

2.2.4.3 Evaluation Criterium

The performance of the treatment ordering is evaluated with Spearman's correlation coefficient. The ideal ordering is obtained on the complete dataset of 109 classifiers with Friedman's algorithm and compared with the ordering obtained on a dataset with missing values, introduced into the dataset with uniform distribution. Since Friedman's algorithm does not work with missing values, the missing values were imputed with dataset's average accuracy. This treatment of missing values was chosen because it was used in the Fernández-Delgado's work.

2.2.4.4 Result

Figure 2.10 depicts the average results after 100 repeats.

GraphRank algorithm provides comparable accuracy with Wittkowski's approach. The difference between the rank by Friedman's algorithm and GraphRank on the complete dataset is caused by the different ranking of boosting and bagging. More often than not, boosting gives better accuracy than bagging. Also, boosting on average gives better results than bagging. However, by Friedman's algorithm, bagging is better than boosting.

The reason for Friedman's conclusion is the fact that a boosted weak classifier sometimes performs worse than the weak classifier by itself (a detail discussion of the problem is in [53]). In these instances, the rank of boosting is much worse than the rank of bagging, which reliably provides good accuracy. On the other hand, when boosting outperforms bagging (frequently and substantially), the rank of boosting is just a bit better

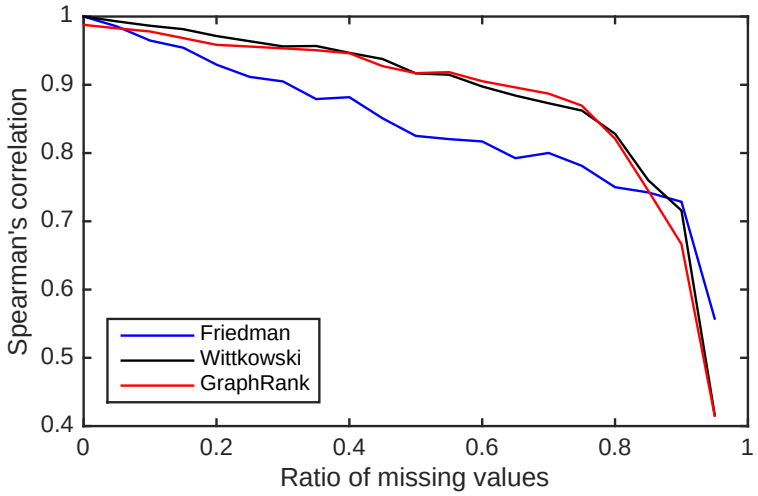

Figure 2.10: Accuracy of ranking methods based on their ability to cope with missing values.

than the rank of bagging because both these algorithms are already at the top of the rank.

This example illustrates the fact that the decision about which treatment of two is better is in Friedman's algorithm influenced by presence or absence of other treatments – if boosting and bagging are compared alone with Friedman's test, boosting comes out as the winner! Note that the simple average on a dataset with exactly one response per cell is not influenceable by the presence or absence of other treatments.

2.2.5 Application

The developed algorithm is applied on a meta-analysis of algorithms for relational classification, which aims to compare algorithms based on their discriminatory ability. In the following paragraphs, it is explained why an experimental evaluation of relational classifiers is a difficult task and what are the challenges in a meta-analysis of relational classifiers.

A direct comparison of relational classifiers is a demanding task, because different relational classifiers demand data in different formats. The data formats for relational classifiers can be divided into three categories [85]:

- Logical based
- Graph based
- Relational

Nevertheless, even formats in the same category may not be compatible. For example, in logical based representation, Alchemy requires a combination of predicates and C++

code [140], TreeLiker uses "pseudo Prolog" format[6] and FOIL uses a (space separated) binary valued data set[7]. Consequently, the observed count of experimentally evaluated relational classifiers in a single study ranges from 1 to 8 with the average of 3 algorithms (the data are further described in Section 2.2.5.1). These numbers are in stark contrast with Fernández-Delgado's 179 propositional classifiers in a single study [68].

Neither a direct comparison of results from literature is easy, because of the different variants of the benchmarking datasets and different protocols (data preprocessing steps). The possibly most popular relational benchmark dataset, Mutagenesis dataset, can be accompanied with 4 types of background knowledge [164] and is available in two versions – a regression friendly version with 188 samples and a regression unfriendly version with 42 samples [47]. Alternatively, both versions can be merged into a single dataset with 230 samples [73]. This makes for $5 \times 3 = 15$ versions of Mutagenesis dataset.

In Alzheimer dataset, at least 6 different target columns can be predicted: acetyl, amine, memory, toxic, choline and scopolamine [14, 76]. In Financial dataset, some authors apply time based filters [190], while other do not [143]. Unless all these details (and likely many other we are not aware of) are known and accounted for, a direct comparison of algorithms on these datasets is troublesome, because the values are not commensurable [48].

2.2.5.1 Data

To collect the data, Google Scholar was searched for keyword:

```
relational classification accuracy OR precision OR recall OR f-
   measure OR f1 OR auc OR roc OR gini OR lift
```

Out of 315 000 returned results, only the first 100 results were processed. Out of the 100 articles, only experimental studies were preserved, because surveys do not generally guarantee that each measure in the survey was obtained with the exactly some protocol (with the same data preprocessing steps and parameter setting). Consequently, the measured values in the surveys may not be always commensurable. To avoid an easy mistake, all surveys were preventively excluded from the meta-analysis. After exclusion of irrelevant and survey articles, 45 were left and examined.

Following data were collected from the articles:

Algorithm name	Common abbreviated name, if available.
Dataset name	Common abbreviated name, if available.
Target name	The attribute in the target table.
Measure name	Like accuracy, F-measure, AUC-ROC or AUC-PR.
Measure value	A decimal value in range 0..1.
Publication name	The title of the publication.

[6] See http://ida.felk.cvut.cz/treeliker/Data.html

[7] See http://cgi.csc.liv.ac.uk/~frans/KDD/Software/FOIL_PRM_CPAR/foil.html

Publication url The link to the publication.

Target name was collected, because some datasets, like Alzheimer [133] or CORA [174], have multiple target attributes. Measure name was collected, because algorithms can be evaluated based on different criteria. If multiple protocols were evaluated in a single publication, only the results from the protocol with the highest average measure values were recorded. The collected data were checked for duplicities and eventual duplicates were removed. The collected data contains 575 records about 48 relational classifiers (see Table 2.7) on 17 datasets (see Table 2.6). The data are available for download at http://motl.us/benchmarking/.

2.2.5.2 Study Design

Treatment, block and response were defined following way:

Treatment Algorithm name.

Block Concatenate(Dataset, Target, Measure, Publication name).

Response Measure value.

The used design blocks for differences between the datasets, targets, measures and protocols. The design assumes that a higher measure value is always better. This assumption holds for all the used measures (classification accuracy, F-measure, AUC-ROC and AUC-PR).

Furthermore, to maximize the amount of information passed to the proposed method, the used design deliberately uses responses from all targets of the datasets, although there is not a reason to believe that the responses are independent.

2.2.5.3 Result

The network of algorithm comparisons is depicted in Figure 2.11. If algorithm A is compared with algorithm B in a literature, there is an edge between algorithm A and algorithm B. If algorithm A has more wins over algorithm B than algorithm B has wins over algorithm A, the edge is directed from algorithm B to algorithm A.

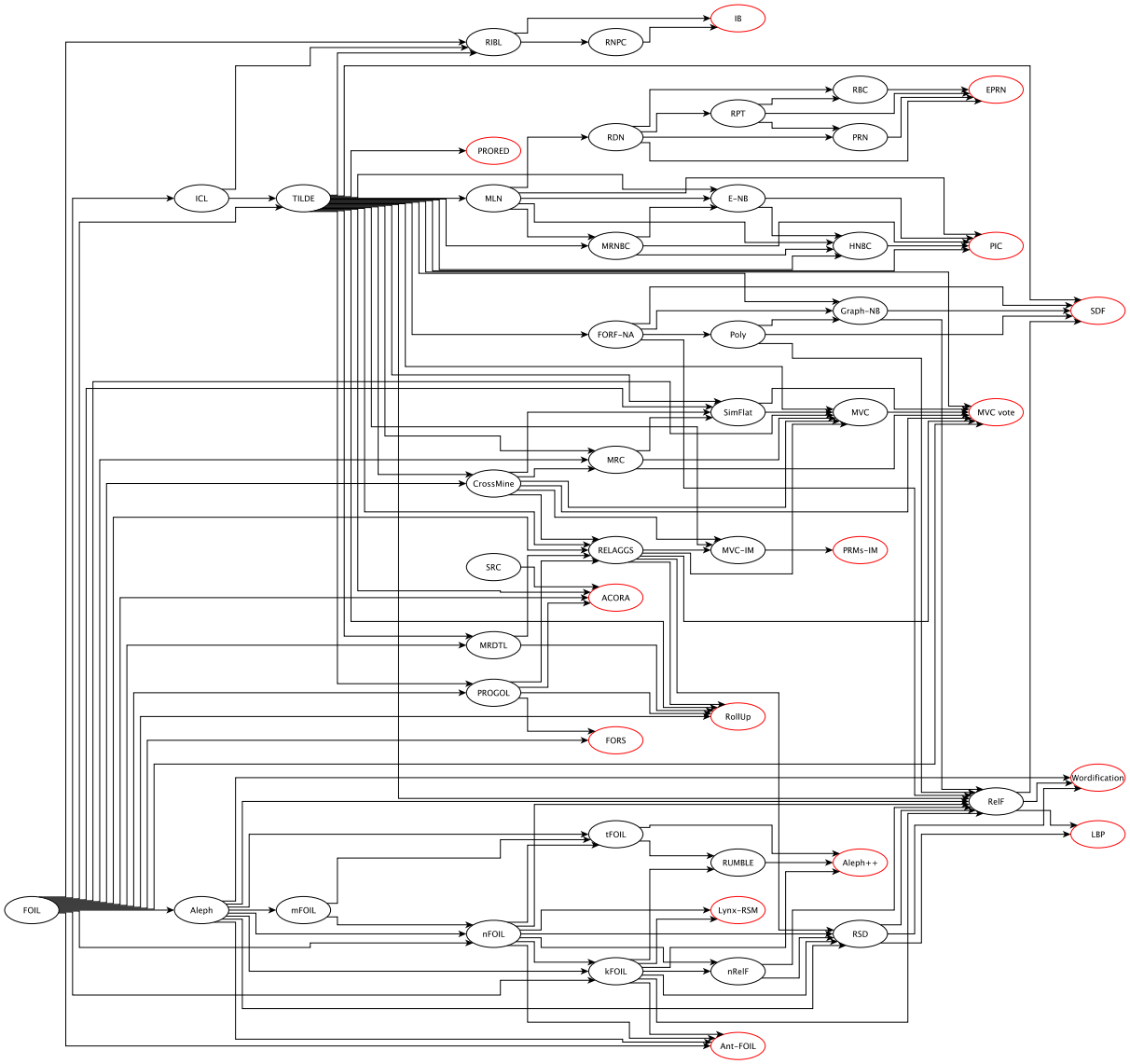

Figure 2.11: Network of the relational algorithms. Algorithms with more accurate predictions are on the right. Superior algorithms are in red nodes.

Database	Domain	Reference
Alzheimer	Medicine	[133]
Biodegradability	Medicine	[18]
CAD	Industry	[149]
Carcinogenesis	Medicine	[244]
CiteSeer	Education	[235]
CompuScience	Education	[205]
CORA	Education	[174]
CPM	Medicine	[149]
Diterpenes	Medicine	[61]
Drug-pyrimidines	Medicine	[107]
DSSTox	Medicine	[220]
ECML	Finance	[146]
Epinions	Retail	[64]
Financial	Finance	[12]
Fingerprints	Medicine	[84]
Genes	Medicine	[38]
Hepatitis	Medicine	[234]
IMDb	Entertainment	[234]
IPO	Finance	[200]
KRK	Entertainment	[186]
LIMAS	Language	[261]
Mesh	Industry	[56]
Mondial	Geography	[234]
MovieLens	Entertainment	[234]
Musk	Medicine	[54]
Mutagenesis	Medicine	[47]
NCI786	Medicine	[246]
NWE	Geography	[29]
OMOP	Medicine	[189]
PremiereLeague	Sport	[234]
PTC	Medicine	[106]
Thrombosis	Medicine	[42]
Trains	Logistic	[178]
University	Education	[234]
UW-CSE	Education	[234]

Table 2.6: List of datasets used in the meta-analysis.

Algorithm	Type	Ref
ACORA	Distance	[200]
Aleph	ILP	[201]
Aleph++	Probabilistic	[114]
Ant-FOIL	ILP	[264]
CrossMine	ILP	[266]
E-NB	Probabilistic	[233]
EPRN	Probabilistic	[205]
FOIL	ILP	[155]
FORF-NA	Decision Tree	[257]
FORS	ILP	[121]
Graph-NB	Probabilistic	[162]
HNBC	Probabilistic	[233]
IB	Distance	[84]
ICL	ILP	[212]
kFOIL	Kernel	[155]
LBP	Propositional	[55]
Lynx-RSM	Propositional	[173]
mFOIL	ILP	[156]
MLN	Probabilistic	[221]
MRC	Multi-View	[98]
MRDTL	Decision Tree	[6]
MRNBC	Probabilistic	[233]
MVC	Multi-View	[95]
MVC vote	Multi-View	[180]
MVC-IM	Multi-View	[97]
nFOIL	Probabilistic	[153]
nRelF	ILP	[150]
PIC	Probabilistic	[233]
Poly	ILP	[150]
PRMs-IM	Probabilistic	[87]
PRN	Probabilistic	[208]
PROGOL	ILP	[185]
PRORED	Propositional	[89]
RBC	Probabilistic	[194]
RDN	Probabilistic	[192]
RELAGGS	Propositional	[144]
RelF	ILP	[149]
RIBL	Distance	[63]
RNPC	Distance	[84]
RollUp	Propositional	[137]
RPT	Probabilistic	[193]
RSD	Propositional	[143]
RUMBLE	Kernel	[226]
SDF	Decision Tree	[14]
SimFlat	Propositional	[97]
tFOIL	Probabilistic	[154]
TILDE	Decision Tree	[233]
Wordification	Propositional	[201]

Table 2.7: List of algorithms used in the meta-analysis.

Plot in Figure 2.12 depicts estimated ranking of relational classifiers after 1 000 bootstraps. The sampling in the bootstrapping was done at the level of individual articles, not at the level of individual measurements, to assess the impact of literature selection.

2.2.5.4 Limitation

In the following paragraphs, different sources of bias in the conducted meta-analysis are discussed.

Publication bias Authors of the methods may prefer to exclude the datasets, where their algorithms do not perform well. The risk of bias is higher if an algorithm is evaluated only by the algorithm authors on a small set of datasets.

Parameter tuning Authors of the algorithms may also spend a disproportionally more time on parameter tuning of their own algorithm than of the others. Hence, algorithms that require careful setting, data pre-processing or post-processing may perform significantly better, if they are set by their authors [A.9].

Protocol setting Also, a protocol may systematically favor specific algorithms. For example, if an author proposes an algorithm for imbalanced datasets, it is natural that the proposed algorithm is tested on imbalanced datasets and that the proposed algorithm fares favorably against algorithms that were not designed to work on imbalanced datasets. But such comparison does not give any evidence about the performance of the proposed algorithm on balanced datasets, where the proposed algorithm may fare miserably.

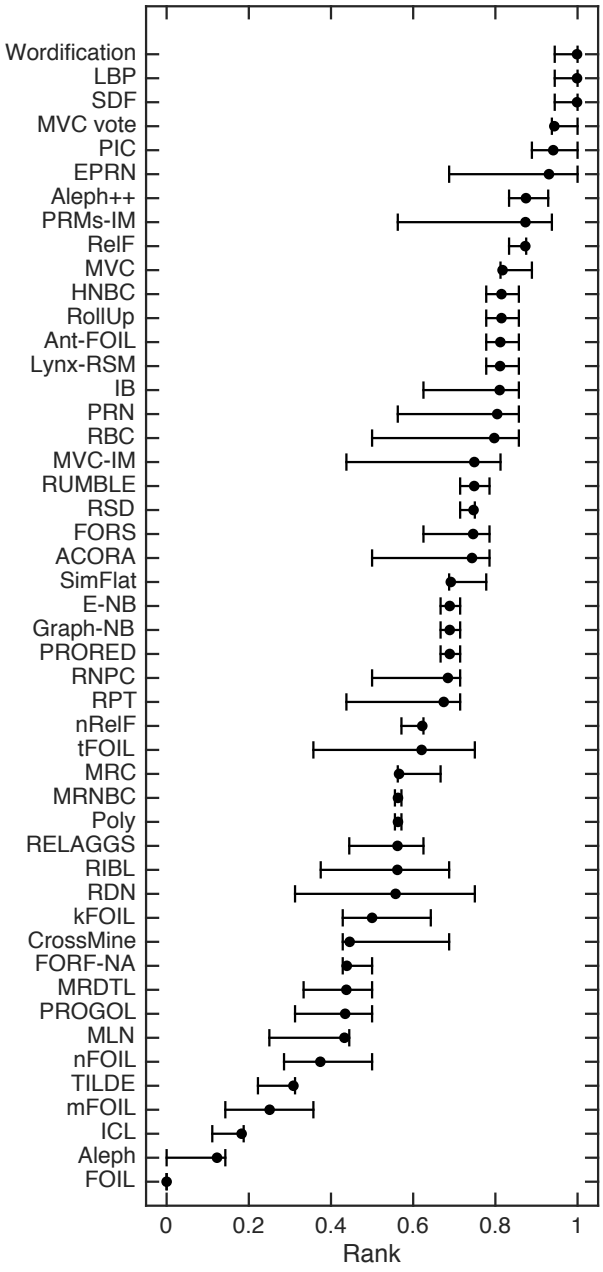

Figure 2.12: Plot with expected ranking of relational classifiers and propositionalization tools. The higher value is better.

Chapter 3

Predictor Factory

All data is suffering

Andreas Kollegger

Based on the problem decomposition in Section 2.1, Predictor Factory is an algorithm for propositionalization of relational data stored in an SQL database. The information in the database is propagated from the target table to other tables in the database with joins. In total, over 50 SQL patterns defining a feature function were tested (the list is in Appendix A.1). Predictor Factory does not refine features. For feature selection, Predictor Factory is using Chi^2 [111]. Any advanced feature selection is left to other tools. The created features are then returned to the user in a single table for attribute-value supervised learning. The following sections, which follow the same order used in Chapter 2, describe algorithm properties beyond this baseline.

3.1 Data Representation

This section justifies the data processing in the database and describes the used statistical and data types.

3.1.1 Processing in a Database or in an Application

The world of relation classifiers can be divided into two groups based on where the classifier is implemented. The classification can be either implemented in a database or in a standalone application.

The advantage of in-database processing is that the data are not transferred. This saves transmission cost and encoding/decoding cost [67]. The disadvantage is the fact, that coding of complex feature functions in Java, R, or another language of choice can be faster than in plain SQL. Another thing to consider is the fact, that a database takes

care of parallelization or distributed computation. However, a person pays a toll in the form of lost control over the implementation details.

Predictor Factory performs calculations in the database.

3.1.2 Typology

Variables can be classified based on different criteria. For example, psychologist Stanley Smith Stevens developed a classification with four levels (also called "scales") of measurement: nominal, ordinal, interval, and ratio [245]. Other classifications include those by Chrisman [39] and by Mosteller & Tukey [184].

In statistics, groups of individual data points may be classified as belonging to any of various statistical data types, e.g., categorical ("red", "blue", "green"), real number (1.68, -5, 1.7e+6), etc. The data type is a fundamental component of the semantic content of the variable, and controls which sorts of probability distributions can logically be used to describe the variable, the permissible operations on the variable, the type of regression analysis used to predict the variable, etc. The concept of data type is similar to the concept of level of measurement but more specific: For example, count data require a different distribution (e.g., a Poisson distribution or binomial distribution) than non-negative real-valued data require, but both fall under the same level of measurement (a ratio scale).

In Predictor Factory, five data types, that are mutually exclusive and exhaustive, are recognized: time, numerical, character, boolean and other. The other category includes all data types that Predictor Factory does not know how to process. Examples of data types, that Predictor Factory does not know how to process are binary data (like images), Extensible Markup Language (XML)/JavaScript Object Notation (JSON) data, geometric data (like points or polygons), objects, arrays, and user-defined data types. The statistical type layer further differentiates between numerical values and nominal values that are just stored as numbers (see Figure 3.1).

Statistical type:	time	numerical	nominal	character	boolean	--
Data type:	time	numerical		character	boolean	other

Figure 3.1: Variable types.

Sometimes, nominal values are stored as numbers, for example, to save data storage or as a kind of obfuscation in the published database. This causes problems, because, for example, the arithmetic mean of a nominal attribute is not generally considered meaningful [245], although exceptions exist [167].

The assignment of appropriate statistical types to the attributes in the data source is time-consuming and error-prone because of the user's fatigue. This problem can be

alleviated by automatic detection of the statistical types. However, this detection may not be reliable[1].

Hence, Predictor Factory ignores statistical types and instead produces the most discriminative features. An example, where aggregates like *median* and *mode* are not discriminative on an ordinal attribute while *arithmetic mean* is:

$$\begin{aligned} A &: [1, 1, 1, 1, 5] \\ B &: [1, 1, 1, 5, 5] \end{aligned} \tag{3.1}$$

Median and *mode* are the same for both instances. But *averages* differ. Another inconvenience is that ordinality is a property of a relationship of the attribute and the target, not just of the attribute alone. Hence, the statistical types would have to be set for each combination of attributes and targets.

3.1.3 Automatic Data Type Detection

One of the less joyful tasks of data mining is the import of data into a SQL database from comma-separated values (CSV) files. CSV files do not preserve data type information. If metadata about the files are not available, the data types have to be estimated directly from the data. The suggested activity diagram for loading and staging CSV files into a SQL database for analytical purposes[2] is in Figure 3.2.

Load In the first phase, all attributes are loaded into the database with text data type. The text data type is used because it preserves all information present in the CSV files, which by itself contain text. The advantage of the text datatype, in comparison to varchar, is that they generally provide a generous limit on both, the attribute size and row size[3].

Numeric In the second phase, each attribute is individually cast in place to the numeric data type with a big enough precision limit. If the conversion fails, the attribute is not likely numeric and is preserved as text. Note that numeric data type is used instead of double to avoid possible rounding errors and overflows/underflows.

Integer Numeric type is often overkill, particularly for keys. Hence, it can be practical to cast numeric attributes to integers in place. Unfortunately, not all databases emit a warning when loss of precision happens. Hence, it is necessary to check that neither

[1] In the author's experiments, the best reached accuracy in classification of attributes into Steven's typology was 87%.

[2] We assume that all the data are available at the moment of loading. This assumption commonly does not hold in primary systems.

[3] Text attributes are, in comparison to varchar attributes, generally stored outside of the table and the table merely contains links to the content. Nevertheless, the implementation details are not mandated by the SQL standards and may differ from vendor to vendor.

record in the attribute contains decimal values and that the attribute's maximum and the minimum are going to fit into the range of the integer before the cast to integer data type.

Numeric trimming The initial numeric data type is commonly impractically big. Hence, it is practical to cast the numeric attributes to numeric types with smaller precision. Once again, it is necessary to first identify the minimum necessary size that preserves all original information before the cast.

Char Text attributes that contain records of constant length can be cast to char data type with a constant length.

Varchar Text attributes that are short enough can be cast to varchar to make processing of the table faster.

Other data types The rest of the data types, like timestamp, date, or time are left up to the user to set them manually, even though there are libraries in Python that promise automatic detection of different formats of temporal attributes[4].

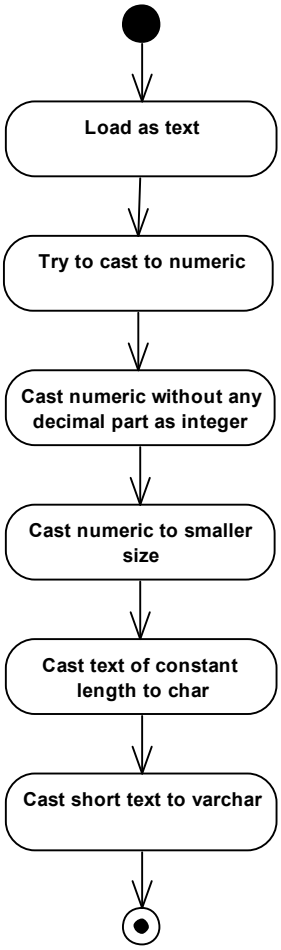

Figure 3.2: Automatic data type detection for CSV files.

3.1.3.1 Third-Party Solution

Another possibility how to import troublesome datasets is to import them with RapidMiner[5] or another tool and once imported into the third-party tool, they can be pushed into the database.

3.2 Information Propagation

This section justifies the used structure of the target table and the decision to use all available tables and attributes in the database.

3.2.1 Training, Testing & Scoring Data in a Single Relation

In challenges like kaggle or KDD Cup, it is customary that the target table is split into two files – one file is with labeled data, which are intended for model training, another

[4]See: http://pythonhosted.org/feedparser or http://dateparser.readthedocs.io.
[5]See: http://rapidminer.com.

file is with unlabeled data that have to be scored. There are at least two ways how to process the target table that is split into several tables. Either we can process each table separately, or we can append the tables into a single table.

Separate tables The advantage of having separate tables is that it is less likely that testing or scoring data inadvertently leak and influence the model[6].

A single table The disadvantage of keeping the tables separate is that the tables may not be fully compatible. We are not guaranteed anymore that the tables are set identically across all the target tables (e.g., the charset encoding), that the attributes are typed identically across the tables, that the attributes are named identically across the tables, that some attributes are not missing, or that the effectively same constraints (foreign, unique...) are applied across the target tables.

Conclusion Since it is difficult to enforce compatibility between the target tables without enforcing total identity between the definition of the tables (including unimportant details like order of attributes) while still covering all the edge scenarios (specific to the database vendor, database version, type, and setting of the hosting OS, setting of the database and possibly other variables) that can make the tables incompatible, the responsibility of providing compatible tables would have to be left to the user. On the other end, it is solely a responsibility of a developer to make sure that training, testing & scoring data do not get unintentionally mixed during the propositionalization. Thus, if we want to decrease the count of user errors, a single target table approach is favorable. If we want to decrease developer errors, a multiple target table approach is better. Predictor Factory implements the single table approach.

3.2.2 Do we need Target Identifier?

The target id is an attribute in the target table, which together with the optional target timestamp uniquely identifies tuples in the target table. While some algorithms, like Aleph [243], do not require the existence of the target identifier, other algorithms, like Deep Feature Synthesis [119], require the existence of the target identifier.

Predictor Factory requires the existence of the target identifier, which can be either a single attribute or a set of attributes. Following paragraphs enlist reasons behind this requirement.

3.2.2.1 Cross-validation

Let us imagine a database with two tables, `TargetTable(IdTransaction, BuysMilk)` and `CustomerOrder(IdCustomer, IdTransaction)`. Then if we want to predict `BuysMilk` for a new customer (without any transaction in `TargetTable`), we have a problem. It

[6] In semi-supervised learning, "leaks" from the scoring data set are intentional.

is natural to perform splitting to training and testing samples at `TargetTable`, but if we do it randomly, we assign some of the customer's transactions into the training split and some of the transactions (of the same customer) into the testing split. The problem is, that a very plastic classification algorithm (like k-NN) may very well learn how to assign the `CustomerIDs` to the likelihood of buying milk (hence making reliable predictions for the current customers), but perform miserably on prediction on new customers with unseen `CustomerID`. And if we perform random (or even stratified) splitting at `TransactionID` level, we get an overly optimistic estimate of prediction accuracy on new customers.

Of course, it could be argued that we should remove ids (like `IdCustomer`) from the prediction table. But that is not going to solve the issue completely. Let's consider `CustomOrder` with thousands of binary columns describing the customers. The dimensionality is so high that regardless of minor changes of customer's vector between the transactions we can uniquely identify the customers (for a proof of concept, see fingerprinting on the internet without cookies).

Hence, the only thorough way how to avoid overly positive estimates is to perform the split for validation at `IdCustomer` level. And that's the reason why we want to have target identifier (in this case `IdCustomer`) in `TargetTable`.

3.2.2.2 Sets

Predictor Factory reads data from SQL databases. SQL databases are inspired by relational algebra. Relational algebra itself is based on set theory. And sets do not preserve order. Consequently, not all data stores, which Predictor Factory supports, preserve the order of tuples in relations. And if the tuples in the target table are not uniquely identifiable and the database does not preserve the order of tuples, we are not able to match the predictions to the subjects, even if we internally assign unique ids to the records, since we are not guaranteed that the created ids will be assigned in any specific order.

Explanatory analysis Of course, it is not always necessary to be able to match up predictions with real-world entities. For example, if we were performing an explanatory analysis, it could be sufficient, if we "just" created a well-performing model. And we are actually able to measure the performance of a model even if we are not able to exactly match predictions with the samples. For example, if we were performing binary classification, we could store positive samples in one relation and negative samples in another relation. And with this setting, we may learn that x positive and y negative samples were misclassified, giving us information about the quality of the model. A similar approach is taken by Aleph [243]. Nevertheless, Predictor Factory was developed with predictions in mind. Prediction is even in the name of Predictor Factory. Hence, it is required that tuples in the target table are uniquely identifiable. And one way how to enforce the uniqueness is to require target id.

3.2.2.3 Duplicate Detection

Sometimes, duplicate records get into a target table. This is, for example, a case of *PTE* or *Lahman* dataset in Relational Repository [A.12]. Duplications of a record increase the weight of the record in the model, resulting in biased predictions. An efficient, although not sufficient[7] way, how to detect duplicates, is to validate the uniqueness of target id and target timestamp tuples in the target table.

3.2.2.4 Conclusion

The target identifier in the target table is a convenient way how to control what a model should learn and how to evaluate the model's performance.

3.2.3 Identifiers

Should we extract features from identifiers (ids)? It depends. If artificial ids are used thoroughly in the database, it is best to not apply feature functions on the ids, because they, in theory, do not contain any information that would not be present in other columns. This approach is used, for example, in MVC [95].

Natural keys However, if natural keys are used in the database, we had better include the natural keys in feature extraction because in the worst-case scenario all the columns in the database are part of some natural key. An example of an algorithm, where ids are used for predictions, is ACORA [200].

Degenerate dimensions In key-value databases or logic databases, it is common to observe degenerate dimensions. A degenerate dimension is a table, which contains at most one non-key attribute. Degenerate dimensions have some nice properties. For example, the database does not have to have a *null* value to represent the concept of a missing value, because if some value is not known, the record for the value is not simply written into the database [103, p. 290][8].

On the other end, if the closed-world assumption [187] holds and the non-key attribute is binary, the degenerate table does not have to contain the non-key attribute, because the mere presence (or absence) of the key in the dimension table provides all the necessary information. An example of such degenerate dimension without any non-key attribute is in `zmatecne` table in `geneea` dataset. If we were ignoring ids, we would miss the information present in `zmatecne` table.

[7] Duplicate records can still appear in the non-target tables.

[8] Another advantage of degenerate dimensions is their natural ability to model temporal databases in relational databases since each record in the database can have metadata about its validity in time in the key. The application of degenerate dimensions to model changing data is further discussed at http://www.anchormodeling.com.

Leaking data Another point of view is, that identifiers can be a source of leaking data (the common sources of leaks in relational data are summarized in Appendix A.2). Hence, in the deployment, we may want to play it safe and completely avoid identifiers in the models. On the other end, in challenges, it can be necessary to exploit such leakages to score well in a challenge[9].

Ordinal id If an artificial id is set to auto-increment, the artificial id gives us information about the succession of records. If feature functions were not applied on ordinal ids, the tuples did not contain any other attribute that would determine the sequence of the records, and the database not preserve record ordering, potentially useful information about the succession of the records would be lost.

Discussion The advantage of not processing ids is that we avoid the generation of potentially non-informative features. On the other end, the advantage of processing the ids is that we do not have to make any assumption about the nature of the ids or the design of the schema.

Because of the mentioned reasons, Predictor Factory by default uses identifiers in feature extraction. Nevertheless, usage of identifiers can be optionally turned off. Furthermore, all predictors that are calculated from identifiers are marked as potentially leaking and it is left up to the user to decide, whether their inclusion in the models is desirable.

Summary There are three theoretical justifications for processing ids: composite foreign keys, degenerate dimensions, and ordinality of ids. And there is one practical consideration: exploitation of leaking data in challenges.

3.2.4 Are Lookup Tables Useful?

Based on *entity-relationship* (ER) model [171], a table either models an entity or a relationship. However, lookup tables model an attribute. Lookup tables, as used in databases, can serve multiple purposes: they may contain pre-computed values, they may contain a textual description of an identifier, or they may constrain values in the main table via a foreign key. The following paragraphs discuss their utility for modeling.

Statistical type inference The presence of a foreign key constraint in the lookup table tells us that we may treat the foreign key (possibly an integer attribute) as a nominal attribute. This knowledge itself helps with the selection of appropriate feature functions.

[9] See: http://blog.kaggle.com/2015/09/22/caterpillar-winners-interview-1st-place-gilberto-josef-leustagos-mario/.

Cardinality & interestingness The presence of a lookup table informs us that the nominal attribute has a finite cardinality. And nominal attributes with a finite (and possibly moderate) cardinality are more suitable for feature functions like *Weight of Evidence* than nominal attributes with extremely high cardinality (like `row_id`). Admittedly, we can calculate the cardinality of any attribute even without the lookup tables. However, the presence of the lookup table suggests that the database architect considered the attribute to be important and small enough to create the lookup table.

Connection between attributes The biggest benefit of a lookup table is that it may create a loop. For example, a lookup table with the airport names can be used twice in a flight table – once as the departure airport and once as the destination airport. These multiple uses of the same lookup table create the loop. The presence of a loop informs us that it could be interesting to calculate the statistics of the airports. In our example, we could calculate the average count of flights from the airports and use these averages as features (once for the departure and once for the destination airport). While we can calculate these statistics without the lookup tables, since we have airport ids in the flight table, the presence of the loop allows us to assign an average count of departing flights and an average count of arriving flights[10] for both, the departure and the destination airports of each flight record. While this information is still computable even without the lookup table, it would be computationally expensive to test each combination of nominal attributes for a match (and usefulness).

Conclusion Predictor Factory, by default, uses all available tables.

3.2.5 Relationship Detection

An independent application, described in Chapter 7, is used to reconstruct the foreign key constraints from the database. The result is provided in a XML file with the structure compatible with $information\_schema.key\_column\_usage$ and $getImportedKeys$ from Java Database Connectivity (JDBC). The exported file can be then used to either guide Predictor Factory or to set the foreign key constraints directly in the database. Of course, the XML can be also generated manually or generated automatically and reviewed by a person.

3.2.6 Time Constraints

Time constraint is a temporal attribute in a table that defines, whether the row is in the time frame applicable for a prediction, or not. A row in a table is suitable for feature calculation iff:

$$timeConstraint_i \leq (observationPoint_i - blackout) \tag{3.2}$$

[10] We may expect these two numbers to be close to each other. Nevertheless, information about the "creation" and the "death" of airplanes can be of interest.

and

$$timeConstraint_i \geq (observationPoint_i - historyLength), \tag{3.3}$$

where *timeConstraint* is the value of the time constraint attribute at row i. $observationPoint_i$ is the time when we wish to have the prediction. And *blackout* and *historyLength* are parameters as depicted in Figure 1.2.

3.2.6.1 Identification of Time Constraints

A practical problem with the time constraints is their identification. For example, if we have a table with ten temporal attributes, which attribute should we use as the time constraint? If we had to select the time constraint for a single table, it would be perfectly feasible to do it manually. However, databases commonly contain multiple tables and setting the time constraint for each table can soon become not only tiring but also error-prone.

3.2.6.2 Temporal Databases

The identification of a time constraint attribute would be easy if the data were stored in degenerate tables that have exactly one key, one attribute, and one temporal attribute (informing us about the time of record entry) and it was known, which attribute is which (for example, because of defined order of attributes in the tables). This paradigm is further extended in some of the implementations of temporal databases [60].

Why exactly one attribute? One attribute in a table is desirable because if there is an update of a record, it is immediately evident, which attribute was changed. Further apologetic of these degenerate tables is given by authors of LogicBlox, an implementation of Datalog [5].

Reality Unfortunately, in the realm of relational databases, metadata about the data are traditionally stored together with the data in a form of additional attributes[11]. And these metadata are not constrained to follow any single standard exactly[12]. Some of the more common archetypes of how to model changing data in a database are discussed by Kimball [132, Chapter Slowly Changing Dimension].

3.2.6.3 Algorithm

A time constraint must fulfill the following conditions:

- Is of a date/datetime/timestamp datatype.
- Does not contain bigger values than the current time.

[11] This follows the relational philosophy that everything is a relation.
[12] SQL:2011 defines temporal support. Nevertheless, not all datasets follow the standard [126].

- Does not contain any missing value.
- Some records must remain after application of the time constraint.

The actual algorithm that selects up to one temporal attribute in a table as a time constraint is depicted in Figure 3.3. The logic of individual blocks in the figure is described in the following paragraphs.

Data type A natural choice of a data type for a time of record entry is *timestamp*, because it generally provides finer granularity than *year*, *date* or *datetime* data type and it defines a point in time uniquely, not like *time* data type.

Note that temporal attributes that are stored as characters (e.g., common in SQLite) or numbers (e.g., as Unix time) are ignored as detection of attribute's real data type is another nontrivial task. Also, temporal attributes that are decomposed into several attributes in a database (e.g., into six attributes named as `year`, `month`, `day`, `hour`, `minute`, and `second`) are not correctly treated as a fully autonomous and robust reconstruction of the complete timestamp is non-trivial.

Because of the listed limitations, only attributes with *date*, *datetime* or *timestamp* data type are considered for a role of a time constraint.

Future value Attributes marking a data entry should never point to future[13]. Hence, if an attribute contains values that are in the future, the attribute cannot be considered as a valid time constraint.

Null It is hard to think of a good reason why a date of record entry would be missing. Hence, a time constraint is required not to contain any null record.

If out of all candidates for a time constraint exactly one temporal attribute has a not-null constraint in the database and is set as "always generated", the attribute is immediately selected as the time constraint.

3.2.6.4 When the Time Constraint is Ignored

Time constraints are ignored/not detected in the following scenarios:

1. The relationship between the target table and a non-target table is n:1.
2. The relationship between the target table and a non-target table is 1:1.
3. Target time is not set up.

[13]To make sure that a record is not entered with a wrong timestamp, it is possible to set the timestamp attribute as "generated always". This setting means that the value is automatically generated by the database and that the value cannot be overwritten by a user. See temporal features in SQL:2011 [148] for details.

n:1 relationship When there is n:1 relationship between the target table and a non-target table, the non-target table is commonly just a lookup table (a table with a description of codes that is used to constrain possible values in an attribute in another table via a foreign key constraint). Lookup tables do not generally contain temporary attributes. Consequently, time constraints are not set for n:1 tables.

1:1 relationship Whenever there is 1:1 relationship between the target table and a non-target table, the non-target table commonly contains static content. For example, it is customary to divide information about customers into two tables – a static table, which contains hardly ever-changing information like date of birth, and a dynamic table, which contains information about the customer's surname, count of dependents (children), and so on. Since the content of static tables is not versioned, time constraints cannot be applied.

Target time is not set up If target timestamp is not set up, it is assumed that the problem is static. In that case, time constraints are not used.

3.2.6.5 Manual Setting

Since automatic detection of time constraints may fail, Predictor Factory can read an XML file with externally defined time constraints.

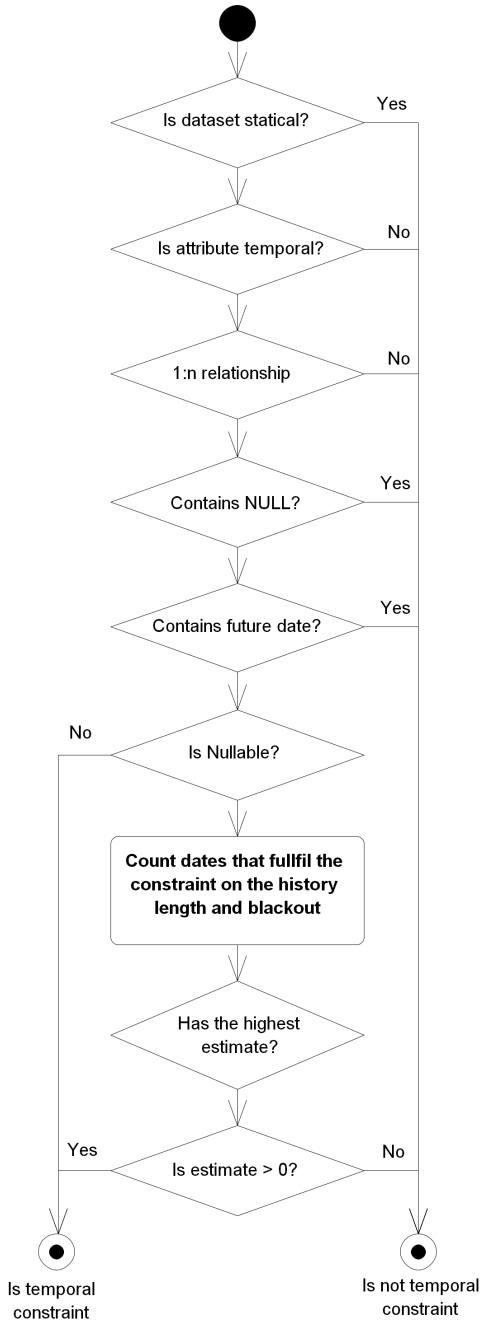

Figure 3.3: Diagram illustrating the selection of time constraint.

3.3 Feature Function

This section describes the naming convention and less common feature functions.

3.3.1 Naming Convention

> There are only two hard things in Computer Science: cache invalidation and naming things.
>
> Phil Karlton

The created features are named with the following naming convention, which was adapted from Wordification [201]:

```
table__column__featureFunction__featureParameter,
```

where *table* is a table name, *column* is a column name, *featureFunction* is a feature function name in camelCase notation[14].

3.3.1.1 Feature Naming

The following paragraphs discuss two questions. First, if the feature name consists of the feature function name and attribute name, should the feature function be a prefix or suffix of the attribute name? Second, should the feature function name be of a constant length or a variable length?

Aesthetics Prefix convention is aesthetically pleasing if the prefix is always of the same length because the root of the name always starts at the same position. On the other hand, if the feature function name is of a variable length, suffix convention is more aesthetically pleasing because it at least preserves visual grouping of the features by attribute names (see Table 3.1 for the illustration).

Debugging The prefix notation is more convenient, if an error in a feature is more likely related to the feature function than to the input attributes because if the features are sorted by the name, the features coming from the same flawed feature function are grouped together. Reversely, if an error in a feature is more likely related to an error in the input attributes than in the feature functions, suffix convention is more practical.

Standard ISO-11179 data element-naming conventions defines `<attribute>_<property>` format. If applied to names of features, it corresponds to suffix notation with the variable feature function name length.

[14] Generally, it is advisable to avoid a mixture of cases in named entities in relational databases because it may lead to the necessity to quote the name entities. However, camelCase notation is shorter than snake case notation. And that is particularly important in legacy databases with strict limits on the length of named entities. For example, the limit on attribute names in SAS is just 32 chars. This limitation of legacy databases is also the reason why we do not adhere to ISO/IEC 11179-5:2005, an industry standard for naming conventions for meta-data, since it promotes overly long names.

	Prefix	Suffix
Constant feature function name length	avg_amount avg_balance avg_type min_amount min_balance min_type std_amount std_balance std_type	amount_avg amount_min amount_std balance_avg balance_min balance_std type_avg type_min type_std
Variable feature function name length	average_amount average_balance average_type min_amount min_balance min_type stddev_amount stddev_balance stddev_type	amount_average amount_min amount_stddev balance_average balance_min balance_stddev type_average type_min type_stddev

Table 3.1: Illustration of different naming conventions.

Conclusion Since it is difficult to come out with meaningful and helpful names of the same length for many feature functions, Predictor Factory demeans itself to the convention of giving names to feature functions of variable lengths. Since users of Predictor Factory were frequently obtaining leaking features in the output table with the features, Predictor Factory is using suffix notation to make dealing with the leaking features easier.

3.3.2 Supervised Features

Supervised features [177, 276, 183, 70, 115, 265] are features that utilize target column. These features are particularly useful for encoding high-cardinality nominal features into numbers. Examples of high-cardinality features include IP-addresses, product ids, and phone numbers.

An example of a supervised feature is *Weight of Evidence* (WoE) [181], which is defined as:

$$WoE = ln\frac{p(a_i|c^+)}{p(a_i|c^-)}, \tag{3.4}$$

where $p(a_i|c^+)$ is a likelihood of observing nominal attribute a with value i given positive class c^+. WoE is particularly interesting, if used together with logistic regression, because

a sum of logarithms is the same as a product or the ratios. And the ratios in WoE are merely a normalized count of positive classes divided by the count of negative classes.

When the target is in a 1:1 relationship to the attribute, we directly return WoE. When the target is in a 1:n relationship to the attribute, we return the average of WoE.

3.3.3 Text Attributes

Text attributes are internally converted to bag-of-words representation and further processed with supervised features defined in Section 3.3.2. The implementation uses Apache Lucene[15] library for text processing (e.g., tokenization, lowercase conversion, language-specific stemming, and N-grams).

3.3.4 Dirty Text Attributes

One of the practical issues in relational learning is how to handle low-quality text attributes. For example, FNHK database violates the first normal form (1. NF) by storing all case's secondary diagnoses in a varchar attribute (instead of storing `case_id-diagnose_id` pairs in a separate table). In this specific case, the secondary diagnoses are separated with a space. Unfortunately, possibly due to copy-pasting, individual diagnoses sometimes got concatenated (or separated with multiple spaces, as illustrated in Table 3.2). Hence, if we wanted to properly model the data as relational, not only that we would have to normalize the data, but we would also have to fix the typographical errors.

Database	Complication	Example
FNHK	special characters, typos	I480 Z921 M5422I252 E780
Grants	abbreviations	Partnrshp vs. PARTNRSHIPS
Stats	typos	zero-inflated vs. zero-inflation
TalkingData	special characters, hierarchy	game-Action, game-Puzzle
Walmart	special characters	SN FG BR

Table 3.2: Examples of dirty text in relational repository.

Both these steps, normalization, and typographical error correction, can generally require human intervention and be time-consuming. And they are not the only issue that we might encounter in a database. Based on Kim [131], dirty text data can arise from a variety of mechanisms:

1. Typographical errors (e.g., proffesor instead of professor)
2. Extraneous data (e.g., name and title, instead of just the name)

[15] See: https://lucene.apache.org/.

3. Abbreviations (e.g., Dr. for doctor)
4. Aliases (e.g., Ringo Starr instead Richard Starkey)
5. Encoding formats (e.g., ASCII, EBCDIC, etc.)
6. Uses of special characters (space, colon, dash, parenthesis, etc.)
7. Concatenated hierarchical data (e.g., state-county-city vs. state-city)

3.3.4.1 Conclusion

Inspired by [201, 31], we extract character trigrams from text and varchar attributes and use them as features. The detail data flow is following: lower-case transformation, bag of character trigrams per target id, and TF-IDF normalization.

We lower-case the strings, because it generally improves the accuracy of the downstream models[16]. The character trigrams are used, because they were the most predictive based on empirical evaluations [31]. TF-IDF normalization is used as it is known to work well not only on text data but even on relational data [201]. Other unsupervised [251] and supervised [152] normalization methods were not tested.

3.4 Feature Selection

This section describes how feature relevancy and redundancy is calculated.

3.4.1 Adjusted Chi^2

Chi^2 shares the same disadvantage with Information Gain (IG) – it prefers attributes with high cardinality, like ids. This bias is not desirable because models trained on attributes with high cardinality do not generalize well on unseen data and are prone to overfitting.

An example, where an attribute with high cardinality has high Chi^2 on training data but zero Chi^2 on testing data is a categorical id attribute. The id perfectly classifies the training data. But when a model has to classify a new instance with a new id, the best that the model can do is to return the majority class (i.e., the model will always predict the majority class).

To fix the undesirable bias of Chi^2, we can use the same remedy that was used to fix Information Gain in Information Gain Ratio – penalize the value by the cardinality of the attribute.

The used implementation assumes uniform distribution of the observed counts and is calculated with the following equation:

$$Chi^2_{adj} = \frac{Chi^2}{n}, \tag{3.5}$$

[16]The case information is extracted with dedicated feature generative functions like *title case*.

where n is the cardinality of the attribute.

3.4.2 Concept Drift

To evaluate, how Chi^2 generalizes to new samples, training data are split by time and universe (target id) into training and testing part, as depicted in Figure 3.4.

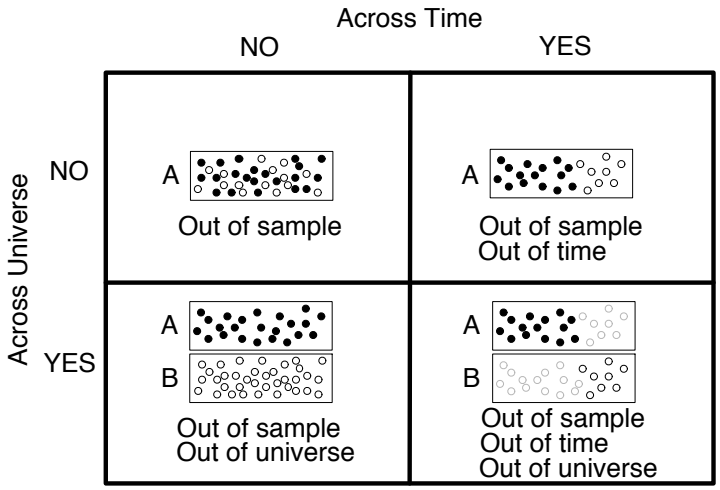

Figure 3.4: Data split. Filled circles represent training data and hollow circles represent testing data. Grey circles represent ignored data. The image is adapted from [240].

The training data are used to calculate Chi^2. And the difference in distributions between $p(\vec{y}|\vec{x})$ in the training and testing data are used to evaluate a *real concept drift*. The *real concept drift* refers to changes in the conditional distribution of target variable given the feature(s), while the distribution of the feature(s) may stay unchanged [83][17].

Since out-of-universe split splits the data to n folds, the estimated concept drift is an average over all folds. There are multiple options how to measure differences between two distributions/histograms, to name a few, Moving Earth Distance and KL-divergence [32], [225]. Nevertheless, generalized Jaccard similarity (in literature also known as Ružička index), which is a generalization of Jaccard index to vectors of real non-negative variables, is utilized to compare histograms of the training and testing histograms[18]. Generalized Jaccard similarity is given by:

$$J(\vec{x}, \vec{y}) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)} \tag{3.6}$$

[17]While concept drift in the features (a subset of *virtual drift*) may not necessarily influence the accuracy of the trained model, it can still be beneficiary to detect concept drift of the used features in the deployment as the format, in which the data are produced or stored, can change in time. And we want to be informed about such changes.

[18]Ružička index has also the advantage, in comparison to Jaccard index, that we do not have to evaluate `union all` clause, which is rather expensive in SQL, but only *min*, *max* and *sum* [214, p. 77].

where $\vec{x}$ is a nominal attribute, $\vec{y}$ is the target and i is an instance (tuple).

Generalized Jaccard similarity was used because of the following properties:

- Is easy to calculate in SQL-92 (as it does not require iterative calculation as Moving Earth Distance)
- Does not have a problem with zero probabilities (as KL divergence)
- Is a similarity (no need to convert distance to similarity)
- Returns values in the range from 0–1 (no need to normalize)
- Works with both, discretized numerical attributes and nominal attributes (does not assume ordinality as Moving Earth Distance)

The adjusted Chi^2 is then calculated as: $Chi^2_{adj} = Chi^2 \cdot sim$, where sim is the average of the similarity on the testing sets (there are two testing sets because we split the data by the universe into two approximately equally big sets). We may look at Chi^2 as training relevance and to Chi^2_{adj} as testing relevance.

3.4.3 Downsampling

To accelerate the evaluation of features, the features can be calculated and evaluated only on a sample of the available data. If the feature appears to be promising, the feature can be calculated and evaluated on additional data. If the feature does not appear to be promising, further evaluation of the feature can be postponed.

The search for the top n most predictive features can be directed with branch and bound (B&B) algorithm. B&B algorithm requires an optimistic ("admissible") heuristic. This heuristic can be based on Chi^2 statistic. To obtain a monotone ("consistent") heuristic from Chi^2, we can calculate:

$$Chi^2(x_1, y_1) + \max_{x_2}(Chi^2(x_2, y_2)), \tag{3.7}$$

where x_1 is the calculated part of the feature, y_1 is the corresponding target and $\max_{x_2}()$ is the maximal Chi^2 that we can obtain on the rest of the predictor.

The maximum possible Chi^2 is given by:

$$\max_x(Chi^2(x, y)) = (|class| - 1) \sum_{i \in class} |x_i|, \tag{3.8}$$

where $class$ is a class in the target, $|class|$ is the count of unique classes and $|X_i|$ is the count of samples in class i.

Random Sampling The essential idea is that a small uniform random sample of tuples t of the relation r often well represents the entire relation [45]. However, not all samples have to provide the same value for model learning, as discussed in the following paragraphs.

Unbalanced data Sometimes, the class labels are unbalanced. And arguably, instances belonging to the rare class are more valuable than instances of the majority class. Hence, we may first evaluate features on a downsampled subset with the equal class ratio, and only once the promising features are identified, finish the calculation of the promising features on the rest of the data.

But is it the best to downsample to 1:1 ratio? Based on the empirical evidence it is better to have slightly more rare classes than of majority classes [207].

Unlabeled data Arguably, labeled data are more valuable for supervised feature selection than unlabeled data. Hence, we may prefer to start the search for the best features with labeled data. And only once the promising features are identified, finish their calculation on the unlabeled data.

Practical considerations In conventional databases, is faster to execute a *single large* query instead of executing *many small* queries on mutually exclusive and exhaustive subsamples [45, p. 8].

There are different reasons for this behavior. Among others, it can be caused by memory locality [49] and query processing overhead.

Hence, if we want to deploy branch and bound algorithm to conventional databases, the steps performed by branch and bound algorithm should be as big as possible. In the simplest case, we can divide the data into just two groups. The first group may contain only labeled downsampled instances. The second group would then contain the rest of the instances. The initial screening for the best features can be done on the first group of instances. And only the most promising features would be then calculated on the second group of instances.

3.4.3.1 Sample Size

Propositionalization can produce a high number of irrelevant and redundant features. To reduce the time spent on generating these "useless" features, we can first propositionalize a subset of *target id*s, evaluate the features, and continue with the propositionalization of the remaining *target id*s only with the features, that were evaluated to be relevant and unique.

But intuitively, there is an overhead associated with the sampling. Hence, there might be situations when sampling increases the runtime, instead of decreasing the runtime. The following paragraph analyses this dilemma.

Let's consider a naive model describing the runtime of Predictor Factory:

$$t = m \cdot f + n \cdot v \cdot m, \tag{3.9}$$

where m is the count of generated features, n is the count of data samples (tuples) in the target table, f is the fixed time associated with the calculation of a single feature (e.g., the network lag between the client with Predictor Factory and the server) and v is

a variable time associated with the calculation of a single feature value for a single data sample (e.g., the processing time of the database).

Now, let's modify the equation for a situation where we first calculate the features for p samples ("probes") with the subsequent calculation of the rest of the features for top s features ("subset"):

$$t = m \cdot f + p \cdot v \cdot m + s \cdot f + (n - p)v \cdot s; \quad p < n, s < m. \tag{3.10}$$

If we combine equations Equations (3.9) and (3.10) together an assume following constants:

$$\begin{aligned} p &= 2000 \quad \text{(commonly in range from } 10^2 \text{ to } 10^4\text{)}, \\ s &= 20 \quad \text{(commonly in range from 10 to } 10^2\text{)}, \\ m &= 2000 \quad \text{(commonly in range from } 10^2 \text{ to } 10^5\text{)}, \\ f &= 0.1 \quad \text{(commonly in range from } 10^{-2} \text{ to } 10^{-1}\text{) seconds}, \\ v &= 10^{-5} \quad \text{(commonly in range from } 10^{-4} \text{ to } 10^{-6}\text{) seconds}, \end{aligned} \tag{3.11}$$

we get that if the target table contains more than n=2101 samples, we save time by employing the sampling. If we vary the variables between the extremes, we get estimates in the range from 2 001 to 53 000.

Implementation The default setting of the probe count p in Predictor Factory is 2 000 samples per class, in the case of a classification problem, respectively 4 000 samples in the case of a regression problem. These numbers were selected to be able to get decently reliable estimates of the features' relevancy [92, 138, 26].

The default setting in Predictor Factory is to use the sampling if:

1. the target table contains more than 10 000 samples,
2. and the sample count n is at least twice the size of the probe count p.

Discussion Branch and bound (B&B) algorithm can be used to prune away unpromising features. B&B is attractive as it guarantees that the returned top n features are optimal in respect to the used feature weighting (in our case Chi^2). However, experiments with naive bounds on Chi^2, where the lower bound is 0 and the upper bound is equal to the count of samples, showed that B&B does not provide any acceleration in comparison to complete evaluation of all features on real-world datasets.

3.4.4 Duplicate Feature Detection

> Entities should not be
> multiplied beyond necessity.
>
> William Ockham

Propositionalization produces a high number of irrelevant and redundant features. But is the generation of redundant features inevitable? Yes, it is:

Example 3.4.1. If the database contains product costs in multiple currencies, then these costs are likely highly positively correlated. Hence, unless we test input attributes for the correlation, the generated features will be positively correlated and potentially redundant[19]

Duplicate features can also be generated from categorical attributes if bijection between two categorical attributes can be defined:

Example 3.4.2. When we apply *distinct count* feature function on `product identifier` and `product description`, we might get the same results.

Problem description Duplicate features are inconvenient. They increase runtime, take space, and sometimes even result in failure of an algorithm (e.g., in a textbook implementation of Linear Discriminant Analysis).

Detection before feature construction It can be tempting to detect bijections directly on the attributes in the database and work only on a unique set of attributes to save runtime and to get a unique set of features. However, if we stick with the example on the `product identifier` and `product description`, while *distinct count* may not find any difference between these two attributes, *text length* feature function may find differences. For example, we may find out that products with long descriptions are sold more frequently than products with short descriptions. But we may not be able to identify this from the `product identifier`, because they can be of a constant length. On the other end, `product identifier` may contain information about the order, in which products were added into the database, and this order may not be recoverable from the `product description`. Hence, we may safely remove duplicate attributes in the database without any loss of information only if the attributes are exact duplicates (copies).

Detection after feature construction We may either detect *hard-duplicates* or *soft-duplicates*. The difference is that *hard-duplicates* are identical attributes, while *soft-duplicates* are attributes that may not be identical but provide similar information as measured, for example, with correlation coefficient.

CFS An example of an algorithm for feature selection with *soft-duplicate* penalization is Correlation Feature Selection (CFS) [102]. The disadvantage of CFS is that we have to calculate the correlation of each attribute with another attribute. Although the runtime of the calculation of the correlation matrix is greatly reduced by the initial binning of the continuous features, the algorithm still has $\mathcal{O}(n^2)$ time complexity, where n is the count of the attributes.

[19] High absolute correlation does not imply redundancy. For an illustration, see [99, page 10, figure 2 (e)].

FCBF The complexity of CFS was improved, for example, in Fast Correlation-Based Filter (FCBF), which has $\mathcal{O}(n \log n)$ complexity [270]. The speedup in FCBF was obtained by using heuristics.

3.4.4.1 Monotonic Transformation Invariant Hashing

For detection of *hard-duplicates*, we can hash each feature and store the hashes in a hash map, where the key is the hash and the value is the feature id for a potential duplicity confirmation. If a new feature happens to have a hash that is already in the hash map, the new feature is a potential duplicate.

For the hashing, we want a monotonic transformation invariant hash. Why? Because decision trees are invariant to the monotonic transformation of features – if we apply a monotonic transformation on the features, the decision thresholds will change, but the structure of the learned tree and the accuracy of the learned model remains the same (bar the changes due to approximations in the practical implementations of decision trees, rounding errors...). And the current state-of-the-art classifier [258], extreme gradient boosting, is based on decision trees.

A monotonic transformation invariant hash can be obtained from a monotonic transformation varying hash h with the following algorithm:

1. Bin continuous attributes with the equal-height algorithm.
2. Calculate hash h from the vector of counts of samples in each bin.

Experiment Coincidentally, Chi^2 is similar to the described algorithm and we already calculate Chi^2 for each feature. Couldn't we use Chi^2 as the hash? When tested on relational repository, Chi^2 had 0 false positives on 19 320 features.

Computational complexity Chi^2 of the calculated features can be stored in a hash map for $\mathcal{O}(1)$ search and insert on average.

3.4.5 Feature Selection Ahead of Feature Collection

Since we propagate the target and the target id into all relations, we have all the necessary information to perform univariate feature selection locally. Thus, we do not have to put all features into a single relation to evaluate them [238]. The advantage of pushing feature selection ahead of feature collection is two-fold – it saves computation time and storage.

Computation time For example, if millions of features are constructed and only one thousand of features, which are considered to be the most desirable, are required at the output, only the selected thousand features have to be joined together.

Storage The calculated features may contain *null* values, even though the dataset does not contain any missing value. An example of a category that generates *null* values are aggregate features, which return *null*, when they are passed an empty vector (e.g., *min*, *max*, *std*...). Since relational databases do not have to have support for sparse storage, a *null* value may take the same amount of space as a *non-null*. To alleviate this issue, Predictor Factory stores only *non-null* values in the generated features. The *nulls* are then added into the feature only once it is clear that the feature is going to be returned. The *nulls* are added into the features during the collection phase with left join.

Chapter 4

Implementation

4.1 Technology

Predictor Factory is an ensemble of SQL, XML, and Java. The justification of the individual technologies follows.

SQL justification. Based on O'Reilly survey [169] 71% of data scientists use SQL, making SQL the most commonly used tool by data scientists.

XML justification. While not everyone likes XML, XML is well known and provides XML Schema Definition (XSD) for XML validation. And validation is important because users are allowed to define their own feature functions.

Java justification. If we want to be able to process data in different databases, then it is useful to use a unified interface. The two most common database interfaces are Open Database Connectivity (ODBC) and JDBC. Since JDBC is slightly more modern than ODBC, it was decided to use JDBC. Since JDBC is an interface developed for Java, the code is also written in Java[1].

4.2 Vendor-Agnostic

Predictor Factory requires all the (input) data to be stored in a single database. But this database can be one of: Microsoft SQL Server, MySQL, Oracle, PostgreSQL, or SAS. Each of these databases is using a different dialect of SQL and SAS is not even relational (SAS predates Codd's concept of a relational database by 4 years). To bridge

[1] An alternative to Java could have been Groovy, which is compatible with Java but provides features that make work with relational databases more comfortable. Namely, Groovy provides multi-line strings and named parameters in prepared statements. Nevertheless, at the time, Groovy did not provide statical typing, which is desirable in large projects.

the differences, JDBC, which provides a unified interface to databases from Java, was utilized. The following paragraphs list functionality of JDBC, which while not essential for building a vendor-agnostic application, makes it easier.

Metadata During the initial phase, Predictor Factory has to collect metadata about tables, attributes, and foreign key constraints. These metadata can be obtained at least three ways:

1. by using vendor's specific query;
2. by utilizing *information_schema*;
3. by using *DatabaseMetaData* interface of JDBC.

Out of these choices, the most versatile method is JDBC's *DatabaseMetaData* interface. Note, however, that JDBC *merely* defines an interface, which is implemented in a database driver. And not all drivers are bug-free and feature complete. To deal with the deficiencies in the individual drivers, a bridge design pattern was employed.

SQL-JAVA type conversion Predictor Factory should know which feature functions are applicable to each data type in a database. But each database can support a different set of data types. There are at least three ways how to obtain a mapping between the database data types and the application's data types:

1. by hand-curated mapping for each database vendor;
2. by querying *information_schema*;
3. by mapping JDBC's *DATA_TYPE* to application's data types.

Out of these options, the utilization of JDBC's mapping of SQL's data types to JDBC's *DATA_TYPE* may look like an unnecessary additional step. However, there is just a single set of JDBC's *DATA_TYPE* (for a given version of JDBC) that have to be mapped, making this choice more scalable than a hand-curated mapping of database data types to application's data types. Note, however, that the mapping is implemented in the database drivers and is not, once again, always trouble-free.

Escape syntax While JDBC provides a uniform connection interface to databases, the SQL queries broadcasted over the JDBC interface may differ from vendor to vendor. And they do. For example, each database may name functions differently (even essential functions like *log* or *sqrt*), making it difficult to write vendor-agnostic queries. There are at least five ways how to deal with these differences:

1. use a common subset of functions;
2. have a set of "personalized" queries for each database vendor;
3. use a library which transcribes the queries to the appropriate dialect;

4. write your own transcription library;
5. use JDBC escape sequences.

Utilization of a ready-to-use library like LINQ (for C#) or JOOQ (for Java) is the most attractive approach. Unfortunately, many libraries support only common OLTP databases. But the first two deployments of Predictor Factory were for SAS and Netezza. Furthermore, not many libraries have support for both, data manipulation and data creation language, which is a requirement for calculation and storage of features directly in the database.

Fortunately, ODBC introduced so-called "escape sequences", which provide fixed names to elementary functions that are not in the SQL standard. These escape sequences are then transparently transcribed to the vendor's dialect in the database driver[2]. Since JDBC inherited these escape sequences from ODBC, the escape sequences are a safe choice for vendor-agnostic queries.

4.3 Architecture

> Programming is breaking of one big impossible task into several very small possible tasks.
>
> Jazzwant

The architecture of Predictor Factory was inspired with Relational Classification – Tree Approach [7], Proper [218], JOOQ[3] and SQuirelL SQL Client[4].

Component diagram Predictor Factory has 4 interfaces: a connection to a database via JDBC, a Graphical User Interface (GUI) defined in FXML, a setting passed in an XML file, and SQL definitions of feature functions wrapped in XML together with metadata like feature function description or author name (see: Figure 4.1).

A class diagram of Predictor Factory, depicted in Figure 4.2, copies the functional decomposition of a propositionalization diagram in Figure 2.1. Namely, *Data representation, Propagation, Feature creation, and Feature collection* are each implemented in a separate package. The connection to a database is realized in two layers: a SQL layer, which converts ANSI SQL to vendor's dialect of SQL, and Network layer, which relays the SQL to the database. The setting and feature functions, that are in XML, are read and written in XML package. GUI is implemented in a separate package as well.

[2] In some databases, like SAS, the transcription is performed in the database rather than in the driver.
[3] See `http://www.jooq.org`.
[4] See `http://squirrel-sql.sourceforge.net`.

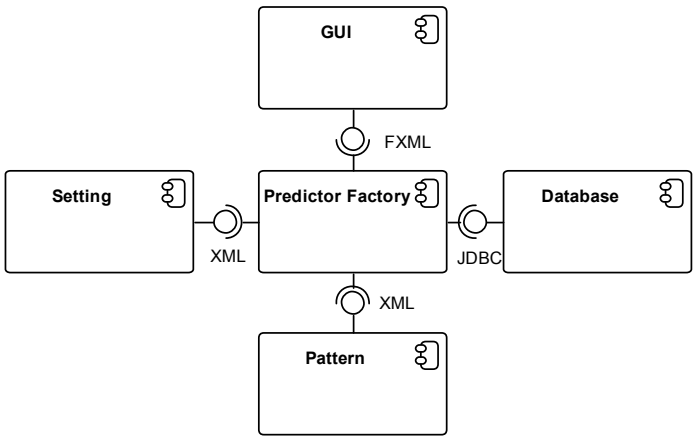

Figure 4.1: Component diagram of Predictor Factory.

4.3.1 Network

Network package provides a connection to a database.

Pooling Connection pooling is not strictly required as Predictor Factory never uses more than one connection in a single moment. However, Predictor Factory may run for hours, if not days, and with increasing runtime, the likelihood of a connection failure increases. To facilitate recovery from such dropped connections, a pool of connections is maintained. When a command has to be executed on the database, a working connection is retrieved from the pool and after finishing the command, the connection is immediately returned to the pool for reuse. If a connection is dropped during the execution of the command, the results of the command are lost to Predictor Factory. Nevertheless, subsequent commands will automatically use a new connection instead of attempting to use the dropped connection.

Database class *Database* class defines properties of the connection. The properties can be either defined as a set of server name, port number, database name or with a URL string, which permits the setting of advanced connection parameters.

Driver class *Driver* class contains information about the JDBC driver like className, URL prefix, and database separator. The *Driver* class is separate from *Database* or *Dialect* class because multiple drivers for a single database may exist. And reversely, a single driver may work with multiple databases.

Network class Parameters are passed to the driver as URL, which is either constructed from *Driver* template and individual properties of the *Database* or directly relayed

from the *Database*. Username and password are always passed to the driver as separate parameters.

4.3.2 Configuration Files

In Java world, there are two common ways how to implement configuration files – XML files or property files[5]. Both approaches are perfectly suitable. The advantage of XML files is that they support tree structure and lists. However, this advantage was utilized only to represent a list of individual settings. Arguably, the same outcome could have been achieved by having a directory with n property files. The advantage of such an approach is that it would be easier to migrate individual configuration files from one computer to another.

The XML files are read immediately at the start of Predictor Factory and checked for validity with XSD to fail fast in the case of user's error.

4.3.3 Metadata

Metadata about the database are collected with JDBC DatabaseMetaData interface. If a driver does not implement the interface, the functionality is implemented in the database vendor's dialect of SQL.

4.3.4 SQL

All SQL/DatabaseMetaData queries to the database go thru *Dialect* class, which assembles queries in the appropriate dialect for the given database. *Dialect* class also takes care of converting the database responses into the expected format (e.g., SAS DatabaseMetaData class in JDBC driver pads all entity names with spaces to a constant length. If these additional spaces were not removed, queries using quoted entity names could fail because quoted entity names are looked up including the spaces).

4.3.5 Main

The main function can be described with the following pseudocode:

```
connectionPool = getConnectionPool;
meta = getMetaData(connectionPool);
meta = propagate(connectionPool, meta);
features = featurify(connectionPool, meta);
collect(connectionPool, features);
```

[5]Other methods include preference and JNDI API.

4.4 Testing and Validation

4.4.1 Connection Leaks

Many databases have a limit on the count of open connections at a single moment. These limits are by default quite small (e.g., 100 in MySQL) and cannot be stretched too much (e.g., Oracle has a hard limit of 2048 open connections). Hence, it is important to properly close all open connections to the database, once the connections are needed. Otherwise, the database, sooner or later, starts to reject new connections.

Stress testing To detect unclosed connections, Predictor Factory was executed in a loop many times against a database. If at least one connection was left open after the application's termination, the loop would prematurely end on connection rejection. All supported databases (Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SAS) were stress-tested on the connection leaks because each JDBC driver may behave slightly differently (e.g., Oracle driver is sensitive to closing JDBC resources in proper order while MySQL driver is forgiving in this respect). No connection leaks were observed.

Driver Some JDBC drivers, like pgjdbc-ng – an alternative driver for PostgreSQL, also provide a setting for the detection of unclosed connections. Unfortunately, pgjdbc-ng was sometimes reporting false positives (some of the queries were taking more time than was the setup connection timeout).

Proxy Communication between Predictor Factory and a MySQL database was also observed with Neor Profile SQL proxy[6]. Predictor Factory always opened two connections at the beginning and used one of these two connections until the end. This observation is in the agreement with the configuration of the pooling library in Predictor Factory – Predictor Factory processes all queries to the database serially. And the pooling library is set to open two connections at the beginning.

4.4.2 User Testing

The usability of Predictor Factory was tested on 9 international students. The following paragraphs describe the population of the testers, the measured objective, the outcome, found issues, taken remedies, and the limitations of the performed user testing.

Population Predictor Factory was tested on international students in age 21–25 that attended a one-hour-long presentation about relational learning during Summer vacation 2016. Nine students attended the presentation. Eight of them found the topic interesting enough to spend another hour testing Predictor Factory. A more detailed demographic is not available as these data were not collected.

[6] See http://www.profilesql.com.

Metric The measured metric was whether the tested subjects can build a predictive model on *financial* dataset. This objective required propositionalization of relational data with Predictor Factory. The data itself were already loaded into a relational database. For success, a subject had to generate a decision tree in RapidMiner with 10-fold cross-validation AUC higher 0.9 in less than 60 minutes.

Outcome Out of 9 subjects, 7 subjects successfully delivered a predictive model. One of the subjects dropped out before the start of the testing because of a lack of interest. Another subject was not able to install Java Runtime Machine on his laptop.

Java Out of all issues, the biggest issue was with Java prerequisite, as Predictor Factory is written in Java. Out of 8 subjects, 5 subjects were not able to immediately run Predictor Factory. It was found out that many Linux distributions by default install OpenJDK (and not Oracle JDK). And OpenJDK does not include support for JavaFX, in which the GUI is written. Another common issue was outdated versions of Java. One subject was not able to install Java within the 60-minute limit at all. We were not able to identify the root cause of the failure.

Users It was found out that users are able to select a continuous attribute as the target for classification. Or that they are able to pick a nominal attribute as the target timestamp. Misconfiguration of Predictor Factory appeared in 3 cases.

Remedies As a remedy to Java problem, documentation for Predictor Factory was updated with the requirement for Oracle JDK 1.8 or higher. Alternatively, OpenJDK 1.8 is permissible, if it is accompanied with an additional OpenFX package. To deal with the evident misconfiguration of Predictor Factory, user inputs are now checked for plausibility. If the user input is invalid, an error message with the problem description and a recommended action is returned.

Limitations The performed user testing suffers from two significant problems. First, the target audience for Predictor Factory are data mining practitioners in financial institutions and retail. But Predictor Factory was tested on students. Second, if a subject got stuck for more than 5 minutes, the subject was told how to get over the obstacle. If subjects were left to solve the task individually and unaided, the success rate would be most likely lower than the measured. The interventions can be justified with two aims: to not let the subjects leave with bitterness about relational learning in their hearts. And to identify more troublesome spots in Predictor Factory beyond the issues with Java.

Conclusion The performed user testing identified deficiencies in Predictor Factory, that would be otherwise difficult to identify. Namely, Predictor Factory was tested on a diverse set of operating systems. This facilitated a realization that OpenJDK lacks

JavaFX support, which can be, nevertheless, installed with a separate package. Additionally, users tried scenarios in Predictor Factory, that were vastly different from the scenarios taken by the developer. That helped to identify deficiencies in the user input checks.

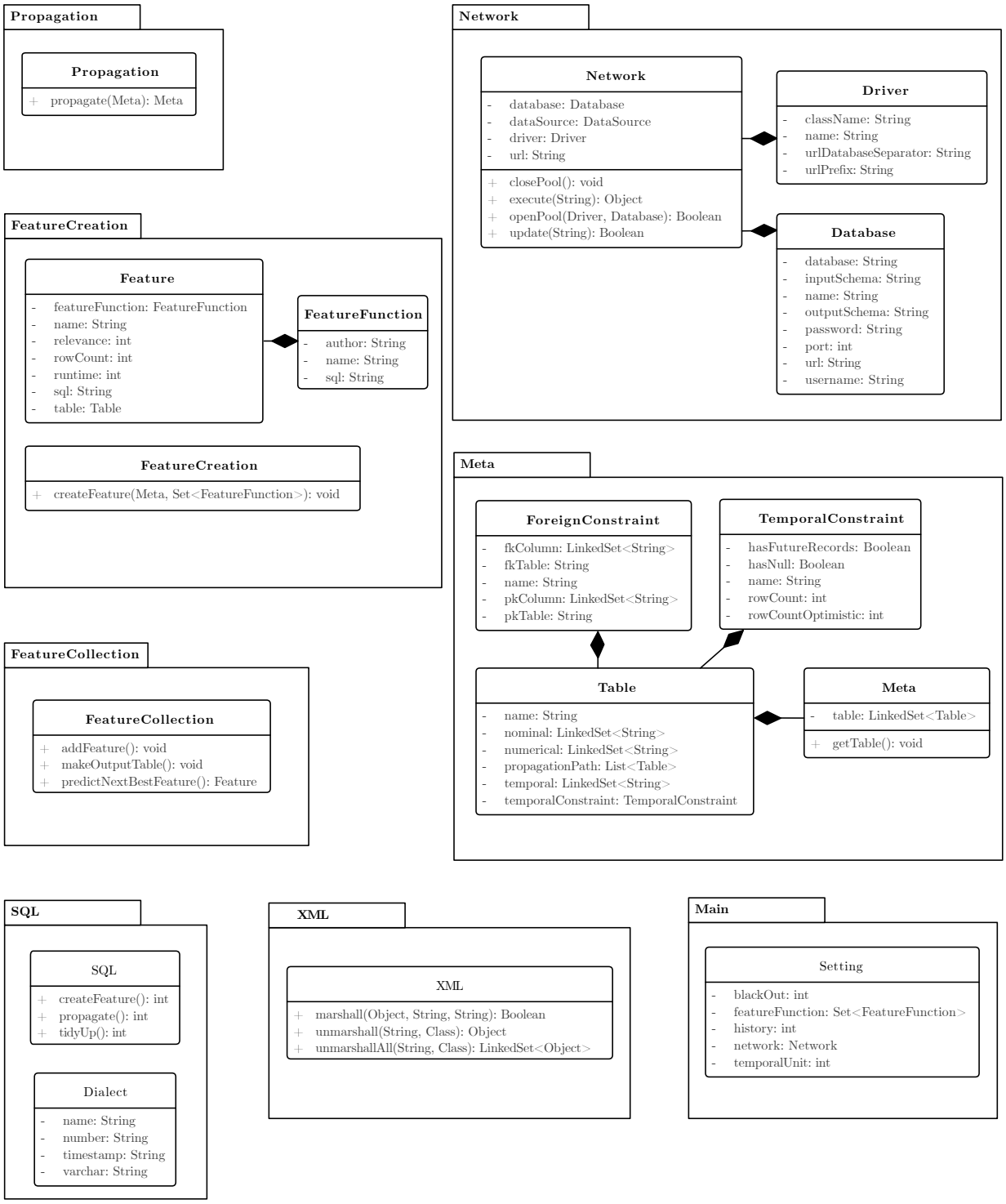

Figure 4.2: Class diagram of Predictor Factory.

Chapter 5

Relational Repository

The aim of Relational Repository is to support machine learning research with multi-relational data. The repository currently contains 83 SQL databases hosted on a public MySQL server located at relational.fit.cvut.cz. A searchable meta-database provides metadata (e.g., the number of tables in the database, the number of rows and columns in the tables, the number of self-relationships).

5.1 Goals

Many organizations maintain their data in relational databases, which support complex structured data. Extending machine learning from traditional single-table methods to multi-relational data is an important direction for practical applications. The statistical and algorithmic challenges that arise from multi-relational data have been addressed in a number of research communities, such as Statistical-Relational Learning, Multi-Relational Data Mining, and Inductive Logic Programming. Experience with the UCI Machine Learning Repository[1] has shown that a shared repository of benchmark datasets facilitates research progress [9]. The UCI Machine Learning Repository contains mainly datasets stored in a single data table. Our goal is to provide a similar service for the relational learning community for relational datasets that contain multiple interrelated tables.

5.2 Design

The repository is maintained in a public MySQL server hosted by Czech Technical University in Prague. Each dataset is stored as a MySQL database on the server. Different formats have been introduced for storing multi-relational data. The advantages of using the SQL (SQL stands for "Structured Query Language") format include the following.

[1] http://archive.ics.uci.edu/ml/

- The SQL format is a based on a standard widely used in industry. Using SQL databases in machine learning facilitates cross-community knowledge transfer and collaborations between machine learning and database researchers.
- Because SQL is a common standard, many programming environments support accessing and processing SQL data. This includes machine learning and statistical platforms such as R, Clowdflows [141], RapidMiner, and Weka. All general application languages provide SQL database connectivity, including Python, Java, and C++.
- The data description facilities of SQL provide a standard for defining *metadata* about the structure of the dataset. For example, information about the entities linked by a relationship is specified using primary and foreign keys. This metadata is recorded in the system catalog, and can be queried by machine learning applications.

To facilitate using tools developed for other relational data formats, we have provided scripts for converting MySQL data to other common data formats used in relational learning http://www2.cs.sfu.ca/~oschulte/jbn/DataConversion/MLN.html. This includes the Wisconsin Logic Learning format (WILL) and the .db format used in the Alchemy system. The ClowdFlows system also provides data format conversion, for example from MySQL to the Aleph Inductive Logic Programming Format[2].

5.3 Content

The repository currently contains 83 databases. This includes common benchmark datasets used in relational learning, like eastbound/westbound train dataset [178] or biodegradability dataset [18]. We have also aimed at providing a diversity of databases, for instance in terms of the number of records and in terms of the complexity of the relational schema. Hence, also synthetic datasets from different database vendors are included, as they are designed to show off capabilities of their database software. An example of such a synthetic dataset is Adventure Works, which is interesting not only because of its complexity, but also because:

- it uses both, simple and composite keys;
- it contains a diverse set of data types, including datetime, blob (images) and geometry;
- it contains missing values.

[2] http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html

5.4 Access and Contributions

Read-only access can be obtained via a database connection with the following parameters.

Hostname relational.fit.cvut.cz

Port 3306

Username guest

Password relational

To contribute a database, please contact the repository administrators; a web contact form is available https://relational.fit.cvut.cz/contact. One possibility is to provide us with a MySQL dump of your database. Another option is to provide us with read access to your database on your server, so we can migrate the database to the public server. A web form for contributing is available https://relational.fit.cvut.cz/contribute.

5.5 The Meta-Database

Table 5.1 shows selected metadata from the meta-database. The meaning of the columns is as follows.

#Relations The number of tables in the database.

#Instances Count of rows in the target table.

Size Size including indexes.

Type The dataset is either a measurement or synthetically generated.

Domain The original domain.

Task Classification or regression.

Database	#Relations	#Instances	Size	Type	Domain	Task
Accidents	3	503362	234.5 MB	Real	Government	Class
AdventureWorks	71	31223	233.8 MB	Synth	Retail	Regr
Airline	19	413046	455.5 MB	Real	Retail	Class
Atherosclerosis	4	389	3.2 MB	Real	Medicine	Class
BasketballMen	9	1536	18.3 MB	Real	Sport	Regr
BasketballWomen	8	234	2.3 MB	Real	Sport	Class
Biodegradability	5	328	3.3 MB	Real	Medicine	Regr
Bupa	9	345	300 KB	Real	Medicine	Class
CCS	6	1000	15.9 MB	Real	Financial	Regr
CDESchools	3	2269	12.3 MB	Real	Government	Regr

CORA	3	2708	4.5 MB	Real	Education	Class
CS	8	100	300 KB	Synth	Financial	Class
Carcinogenesis	6	329	21 MB	Real	Medicine	Class
Chess	2	295	300 KB	Real	Sport	Class
CiteSeer	3	3312	5.9 MB	Real	Education	Class
ClassicModels	8	273	500 KB	Synth	Retail	Regr
ConsumerExp	3	2047961	337.6 MB	Real	Retail	Class
Countries	4	247	9.8 MB	Real	Geography	Regr
CraftBeer	2	558	300 KB	Real	Entertainment	Class
Credit	9	9861	317.9 MB	Synth	Retail	Class
DCG	2	1130	300 KB	Synth	Education	Class
Dallas	3	219	400 KB	Real	Government	Class
Dunur	17	276	800 KB	Real	Kinship	Class
Elti	11	1081	600 KB	Real	Kinship	Class
Employee	6	2838426	197.4 MB	Synth	Retail	Regr
ErgastF1	14	31313	60.4 MB	Real	Sport	Class
FNHK	3	41392	130.8 MB	Real	Medicine	Regr
FTP	2	29555	7.5 MB	Synth	Retail	Class
Facebook	2	511	2 MB	Real	Social	Class
Financial	8	682	78.8 MB	Real	Financial	Class
GOSales	5	1000	22.4 MB	Synth	Retail	Regr
Geneea	19	3556	61.4 MB	Real	Government	Class
Genes	3	862	1.8 MB	Real	Medicine	Class
Grants	12	385882	890.8 MB	Real	Education	Regr
Hepatitis	7	500	2.2 MB	Real	Medicine	Class
Hockey	22	7579	15.6 MB	Real	Sport	Class
IMDb	7	738576	477.1 MB	Real	Entertainment	Class
KRK	1	1000	100 KB	Synth	Sport	Class
Lahman	25	23111	74.1 MB	Real	Sport	Regr
LegalActs	5	583672	240.2 MB	Real	Government	Class
Mesh	29	223	1.1 MB	Real	Industry	Regr
Mondial	40	204	3.2 MB	Real	Geography	Class
MooneyFamily	68	92	3.2 MB	Synth	Kinship	Class
MovieLens	7	6039	154.9 MB	Real	Entertainment	Class
Musk	2	92	400 KB	Real	Medicine	Class
Mutagenesis	3	188	900 KB	Real	Medicine	Class
NBA	5	30	300 KB	Real	Sport	Class
NCAA	9	268	35.8 MB	Real	Sport	Class
Nations	3	14	1.2 MB	Real	Geography	Class
Northwind	29	830	1.1 MB	Synth	Retail	Regr
PTC	4	343	8.1 MB	Real	Medicine	Class
PTE	38	299	4.4 MB	Real	Medicine	Class
Pima	9	768	700 KB	Real	Medicine	Class
PremiereLeague	4	380	11.3 MB	Real	Sport	Class
PubMed_Diabetes	3	20055	44.1 MB	Real	Education	Class

Pubs	11	18	400 KB	Synth	Retail	Regr
Pyrimidine	2	74	1 KB	Real	Medicine	Regr
Restbase	3	9524	3 MB	Real	Retail	Regr
SAP	5	60024	246.9 MB	Synth	Retail	Class
SAT	36	76	2.7 MB	Real	Industry	Class
SFScores	3	23833	10.3 MB	Real	Government	Regr
Sakila	16	15991	6.4 MB	Synth	Retail	Regr
SalesDB	4	6698788	584.3 MB	Synth	Retail	Regr
SameGen	4	1081	300 KB	Real	Kinship	Class
Seznam	4	1458233	146.8 MB	Real	Retail	Regr
Shakespeare	4	32980	8.8 MB	Real	Entertainment	Class
Stats	8	41793	658.4 MB	Real	Education	Regr
StudentLoan	10	1000	900 KB	Real	Education	Class
TPCC	9	28600	165.8 MB	Synth	Retail	Class
TPCD	8	147600	2.5 GB	Synth	Retail	Class
TPCDS	24	97006	4.8 GB	Synth	Retail	Class
TPCH	8	148534	2 GB	Synth	Retail	Regr
Thrombosis	3	806	1.9 MB	Real	Medicine	Class
Trains	2	20	100 KB	Synth	Education	Class
Triazine	2	186	200 KB	Real	Medicine	Regr
UTube	2	547	200 KB	Real	Industry	Class
UW-CSE	4	278	200 KB	Real	Education	Class
University	5	38	300 KB	Synth	Education	Class
VOC	8	8073	2.7 MB	Real	Retail	Class
VisualGenome	6	1350342	322.4 MB	Real	Education	Class
Walmart	4	4607680	167.3 MB	Real	Retail	Regr
WebKP	3	877	12.8 MB	Real	Education	Class
World	3	239	700 KB	Real	Geography	Class

Table 5.1: List of databases in the repository.

The name of the meta-database schema is `meta`. This schema contains a number of tables with information about the databases, as well as the performance of different learning algorithms on the databases. The name of the table that contains information about the databases is `meta.information`. Some of this metadata is automatically exported in HTML format for display on the webpage relational.fit.cvut.cz. In the following paragraphs, we list the names of the main column and their meaning. When we refer to "all columns" or "all rows", we mean all columns/rows of all tables in a database. The metadata contain the following main groups of information: basic database statistics, information about columns or fields, foreign key structure, and classification information.

Basic Database Statistics Various basic properties, such as record count and missing values.

row_count The total number of rows, or records.

row_max The maximum number of rows, or records, in a single table.

column_count The total number of columns, or fields.

download_url A URL containing further information about the dataset, such as provenance.

null_count The number of table entries with null values; typically this is the number of table entries with missing values.

Column Information These columns contain metainformation about the types of columns/fields/attributes in the database tables. The list is mutually exclusive and collectively exhaustive as it holds: $column\_count = geo\_count + date\_count + lob\_count + string\_count + numeric\_count$.

geo_count The number of columns that represent spatial attributes. (These are called "geographic" features in MySQL.)

date_count The number of columns that represent temporal attributes (date, time, or year).

lob_count The number of columns that store large objects (e.g., images).

string_count The number of columns that store string values. This typically includes discrete attributes.

numeric_count The number of numeric columns.

Foreign Key Structure A foreign key points from one table to another. Chen *et al.* propose visualizing the foreign key relationships in a semantic relationship graph [35]: The graph contains a directed edge from table T to table T' if table T references T' in a foreign key constraint. These columns represent information about the structure of the semantic relationship graph.

primary_key_count The number of primary keys.

composite_key_count The number of primary keys that comprise more than one column.

foreign_key_count The number of foreign keys.

self_referencing_table_count The number of tables such that the table contains a foreign key pointer to one of its own columns. This occurs for example when a relational schema represents a class hierarchy or taxonomy.

has_loop Whether there exists a loop of foreign key pointers over several tables. An example of a loop is when between a person table and a university table exists two foreign keys – the first foreign key signifies that a person is studying at a university, while the second foreign key signifies that the person is teaching at the university.

Classification Many of the databases in the repository have been used to study classification in relational data. There is often a standard class label for such studies; we refer to this as the *target attribute*. These columns contain information relevant to the target attribute where it exists.

target_column The target attribute most often used in relational classification studies.

target_table The table that contains the target column.

target_id The primary key field of the target table.

instance_count The number of rows in the target table.

class_count The number of class labels.

majority_class_ratio The proportion of the majority class label on instance count.

5.6 Conclusions

In this chapter, we presented Relational Repository (RR), an easily accessible collection of datasets for relational learning. The RR was designed with supervised learning in mind. To this end, the RR contains 83 ready to download datasets. One of the important features of the RR is that it provides meta-data about the datasets. The RR meta-data can be accessed at https://relational.fit.cvut.cz/.

Chapter 6

Empirical Evaluation

> What gets measured gets improved.
>
> Peter Ferdinand Drucker

6.1 Algorithms

Predictor Factory is empirically compared to:

1. Aleph
2. RelF
3. RSD
4. Wordification

The listed algorithms were selected because their implementations are freely available and work with relational formalism, mitigating the necessity of time-consuming and error-prone conversion of the data from one formalism to another[1].

6.2 Datasets

The algorithms were executed on 16 datasets from relational repository. The list of the used datasets is in Table 6.1.

6.3 Protocol

Following paragraphs describe the experiment protocol.

[1] The used implementations are from ClowdFlow.

Database	#Relations	#Instances	Size	Type	Domain
Carcinogenesis	6	329	26.3	Real	Medicine
CS	8	100	0.3	Synth	Finance
Financial	8	682	94.1	Real	Finance
Genes	3	862	1.9	Real	Medicine
Hepatitis	7	500	2.2	Real	Medicine
Mondial	33	454	3.3	Real	Geography
MovieLens	19	3832	47.9	Real	Entertainment
Mutagenesis	3	188	0.9	Real	Medicine
NBA	4	30	0.3	Real	Sport
NCAA	10	268	40.6	Real	Sport
PremiereLeague	4	363	11.3	Real	Sport
Trains	2	20	0.1	Synth	Logistic
University	5	38	0.3	Synth	Education
UW-CSE	4	278	0.2	Real	Education

Table 6.1: List of the used relational datasets. The size is in MB including indexes. Type describes the origin of the dataset.

Metrics The used metrics are 10-fold cross-validated: classification accuracy, ROC-AUC and F-measure.

Propositionalization All propositionalization algorithms were executed with their default setting.

Models The classification was performed with J48 decision tree with the default setting. The decision tree was used because of its ability to handle nominal attributes, missing values and collinear features.

Workflows Aleph, ReLF, RSD and Wordification were executed with ClowdFlows' workflow available at http://clowdflows.com/workflow/4018/, which is using operators from Weka. The results from Predictor Factory were evaluated in a standalone Weka with a comparable workflow. Weka was chosen as the tool of choice because it permits application of the identical implementation of the decision tree and identical evaluation of the models [227].

6.4 Results

The measured classification accuracies are in Table 6.2. Notably, propositionalization tools generally do not work with composite foreign keys, like in the original Mondial

dataset. Sometimes the algorithms crash from unknown reasons. The results on other measures were comparable to results obtained with classification accuracy.

Dataset	Aleph	Predictor Factory	RelF	RSD	Wordification
Carcinogenesis	0.55	**0.82±0.07**	0.60	0.60	0.80
CS	–	**0.96±0.01**	–	–	–
Financial	0.87	0.87±0.06	**0.98**	0.95	0.90
Genes	–	0.62±0.02	–	**0.84**	–
Hepatitis	**0.78**	0.74±0.08	0.69	0.59	0.65
Mondial	–	**0.79±0.05**	–	–	–
MovieLens	–	0.78±0.03	0.61	–	**0.83**
Mutagenesis	0.81	**0.93±0.08**	0.87	0.90	0.82
NBA	–	**0.60±0.02**	–	–	–
NCAA	–	**0.70±0.02**	–	–	–
PremierLeague	–	**0.67±0.03**	–	–	0.35
Trains	0.70	**0.95±0.15**	0.75	0.80	**0.95**
University	–	**0.89±0.04**	–	0.37	0.84
UW-CSE	0.85	0.88±0.15	0.81	**0.89**	**0.89**

Table 6.2: The 10-fold cross-validation estimate for the accuracy with a decision tree. A hyphen means that the propositionalization algorithm crashed on the dataset. The best results for each dataset are highlighted.

When it comes to the comparison of the accuracies, Predictor Factory is in par with the other propositionalization algorithms. The bad accuracy on the Financial dataset is caused by taking the time constraints in the account. If the time constraints are lifted, Predictor Factory reaches the accuracy of 0.99±0.01.

6.5 Discussion

Possibly the biggest single limitation of Predictor Factory is the fact that you have to assemble the target table by yourself. And if you make a mistake during the creation of the target table, everything is wrong – you then optimize something different from what do you want to optimize or/and you do it on a wrong population.

Chapter 7

Foreign Key Constraint Identification

For relational learning, it is important to know the relationships between the tables. In relational databases, the relationships can be described with foreign key constraints. However, the foreign keys may not be explicitly specified. In this chapter, we present how to automatically and quickly identify primary & foreign key constraints from metadata about the data. Our method was evaluated on 72 databases and has F-measure of 0.87 for foreign key constraint identification. The proposed method significantly outperforms in runtime related methods reported in the literature and is database vendor agnostic.

7.1 Introduction

Whenever we want to build a predictive model on relational data, we have to be able to connect individual tables together [37]. In *Structured Query Language* (SQL) databases, the *relationships* (connections) between the tables can be defined with *foreign key* (FK) constraints. However, FK constraints are not always available. This can happen, for example, whenever we work with legacy databases or data sources, like *comma separated value* (CSV) files.

Identification of relationships from database belongs to *reverse engineering* from databases [198] and can be done manually or by means of handcrafted rules [10, 37, 147, 273]. Manual FK constraint discovery is very time-consuming for complex databases [176]. And handcrafted systems may overfit to small collections of databases, used for the training. Therefore we use machine learning techniques for this task and evaluate them on a collection of 72 databases.

Unfortunately, FK constraint identification is difficult. If we have n columns in a database, then there can be n^2 FK constraints, as each column can reference any column in the database, including itself[1]. Hence, there is n^2 candidate FK constraints.

[1]We have not observed any instance of a column referencing itself. Nevertheless, SQL standard does not forbid it.

Example 7.1.1. If we have a medium-sized database with 100 tables, each with 100 columns, then we have to consider 10^8 candidate FK constraints.

We can evaluate probability p that a single candidate FK constraint is a FK constraint with a classifier (e.g., logistic regression) in a constant time. Hence, if we assumed that the probability $p_{i,j}$, which denotes a probability that a column i references column j, is independent of all other candidate FK constraints in the database, the computational complexity of FK constraint identification would be $O(n^2)$. However, the probabilities do not appear to be independent.

Example 7.1.2. If we had two columns A, B and we had known that $p_{A,B} = 0.9$ and $p_{B,A} = 0.8$ then under assumption of independence it would be reasonable to predict that column A references column B and also that column B references column A. However, directed cyclic references[2] do not generally appear in the databases as it would make updates inconveniently difficult [170]. Hence, our example database most likely contains only one FK constraint with A referencing B.

If we accepted that the FK constraints are not independent of each other, we could generate each possible combination of FK constraints and calculate the probability that the candidate combination of FK constraints is the true combination of FK constraints. The computational complexity of such algorithm is $O(2^{n^2})$. Clearly, a practical algorithm must take simplifying assumptions to scale to complex databases.

The applications of the FK constraint discovery, besides relational learning, include *data quality* assessment [2] and *database refactoring* [176].

The chapter is structured as follows: first, we describe related work, then we describe our method, then we describe our experiments and their results, discuss the results and provide a conclusion.

7.2 Related Work

Li et al. [161] formulated a related problem, attribute correspondence identification, as a classification problem.

Rostin et al. [224] formulate FK constraint identification as a classification problem.

Meurice et al. [176] compared different data sources for the FK constraint identification: database schema, Hibernate XML files, JPA Java code and SQL code. Based on their analysis, the database schema data source has four times higher recall than any other data source. In this chapter, we focus solely on the database schema data source. Furthermore, they introduce 4 rules for filtering the candidate FK constraints: the "likeliness" of the candidate FK constraint must be above a threshold, the FK constraints cannot be bidirectional, the column(s) of the selected FK constraints can be used only once and there can be only a single (undirected) path from FK constraints between any two tables.

[2] However, undirected cyclic references are commonly used, for example, to model hierarchies.

Chen et al. [37] describe how to significantly accelerate FK constraint identification by pruning unpromising candidates at multiple levels. We inspire from them and use multi-level architecture as well. Furthermore, they introduce 4 rules for filtering the candidate FK constraints: explore FK constraints only between the tables selected by the user, only a single FK constraint can exist between two tables, directed cycles from FK constraints are forbidden and there can be only a single (undirected) path from FK constraints between any two tables. We inspire from Meurice's and Chen's articles and reformulate their rules as *integer linear programming* (ILP) problem.

7.3 Method

To make the relationship identification fast, a predictive model was trained only on the metadata about the data, which are accessible with *Java Database Connectivity* (JDBC) API. This approach has the following properties:

1. It is fast and scalable.
2. It is database vendor agnostic.
3. It is not affected by the data quality.

The problem of relationship identification was decomposed into two subproblems: identification of *primary keys* (PKs) and identification of FK constraints (Figure 7.1). The reasoning behind this decomposition is that identification of PKs is a relatively easy task. And knowledge of PKs simplifies identification of FK constraints because FK constraints frequently reference PKs[3].

The identification of the PKs is performed in two stages: scoring and optimization. During the scoring phase, a probability that an attribute is a part of a PK (a PK can be *compound* – composed of multiple attributes) is predicted with a classifier. The probability estimates are then passed into the optimization stage, which delivers a binary prediction.

The same approach is taken for FK constraint identification. During the scoring phase, a probability that a candidate FK constraint is a FK constraint is estimated with a classifier. The probabilities are then passed into an optimizer, which returns the most likely FK constraints.

Primary Key Scoring All metadata that are exposed by JDBC[4] about attributes (as obtained with *getColumns* method) and tables (as obtained with *getTables*) were collected and considered as features for classification. For brevity, we describe and justify only features used by the final model.

[3] A FK may reference any attribute that is unique, not only PKs. Nevertheless, all FKs in the analyzed databases reference PKs.

[4] See docs.oracle.com for the documentation.

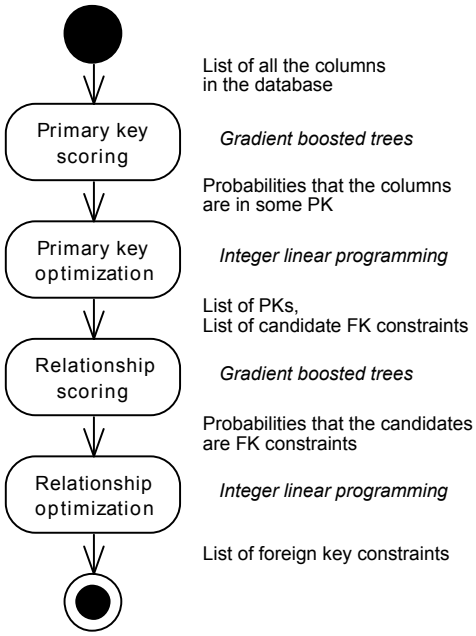

Figure 7.1: The algorithm decomposition.

Data types like *integer* or *char* are generally more likely to be PKs than, for example, *double* or *text*. To promote portability of the trained model, we do not use database data types but JDBC data types, which have the advantage that they are the same regardless of the database vendor.

Since some databases offer only a single data type for numerical attributes, we also note whether numerical attributes can contain **decimals**, as PKs are unlikely to contain decimal numbers.

Doppelgänger is an attribute, which has a name similar to another attribute in the same table. For example, *atom_ id1* is a doppelgänger to *atom_ id2*. Doppelgängers frequently share properties, i.e., either both of them are in the PK or neither of them is in the PK.

Ordinal position defines the position of the attribute in the table. PKs are frequently at the beginning of the tables.

String distance between the column and table names are helpful for identification of PKs and FKs. Opinions on the best measure for PK and FK constraint identification vary. For example, [224] uses exact match while [37] uses Jaro-Winkler distance. After testing all string measures available in *stringdist* package [165], we found that Levenshtein distance delivers the best discriminative power on the tested databases.

Keywords like *id* or *pk* frequently mark PKs. The presence of the keywords is analyzed after the attribute/table name tokenization, which works with camel case and snake case notation.

JDBC also provides attributes that leak information about the presence of the PK,

like *isNullable*, *isAutoIncrement* and *isGeneratedColumn*. Since it is unreasonable to assume that these metadata would be set correctly after importing data from CSV files, they were excluded from the model.

For comparison to features extracted from the data (and not metadata), two additional features were extracted: whether the attribute contains nulls (*containsNull*) and whether the attribute contains unique values (*isUnique*). These features are generally expensive to calculate [37]. Nevertheless, some databases, like PostgreSQL, automatically generate these statistics for each attribute in the database and provide a vendor-specific access to these statistics.

Primary Key Optimization Since each table in a well-designed database should contain a PK, a single most likely PK is identified for each table in the database. If the single most likely PK attribute in a table is a doppelgänger, all its doppelgängers in the table are declared to be part of the PK as well, creating a compound key. The described optimization can be solved with an ILP solver, which we use mostly because foreign key optimization (subsection 7.3) is using ILP formulation as well.

Foreign Key Scoring Features for FK constraints are a combination of features calculated for individual attributes from subsection 7.3 (prefixed with *fk* and *pk* respectively) with features unique for the FK constraints. The description of the unique features follows.

Data types between FK and PK attributes should match. Nevertheless, SQL permits FK constraints between *char* and *varchar* data types.

Data lengths between FK and PK attributes should match. Nevertheless, SQL explicitly permits FK constraints between attributes of different character lengths as defined in the SQL-92 specification, subsection 8.2.

String distance between FK column name and PK table name should be small because FK column names frequently embed a part of the PK table name. Similarly, FK column name should be similar to PK column name because FK column names frequently embed a part of the PK column name. On the other end, FK column name should generally differ from FK table name as they are not directly related.

Furthermore, to be able to compare metadata-based features to data-based features, we tested whether all non-null values in the FK are present in the PK (*satisfiesFKConstraint*). This is generally an expensive feature to calculate [37]. Nevertheless, some databases, like PostgreSQL, automatically calculate histograms for each attribute in the background and offer a vendor specific interface to access the histograms. And based on the range of the histograms many candidate FK constraints can be pruned. More advanced data-based features (e.g., similarity of the FK and PK distributions) were not explored as the focus of the chapter is on the metadata-based features.

Foreign Key Optimization The optimization can be formulated as an integer linear optimization problem on a directed graph $G = (V, E)$, where V is the set of attributes

in the database and E is the set of candidate FK constraints. The p_{ij} is the estimated probability that the candidate FK constraint referencing FK i to PK j is a FK constraint. The probabilities are estimated with a classification model trained on features described in subsection 7.3. Compound FKs are modeled as multiple FK constraints (one FK constraint for each attribute). We define variable x_{ij}:

$$x_{ij} = \begin{cases} 1 & \text{if the candidate FK constraint is a FK constraint} \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

The optimization problem is then:

$$\max_{x} \quad \sum_{[i,j]\in E} x_{ij} - 2 \sum_{[i,j]\in E} x_{ij}(1 - p_{ij}) \tag{7.2a}$$

s.t.

$$\sum_{j\in V} x_{ij} \leq 1, \qquad \forall i, \tag{7.2b}$$

$$\sum_{i\in S, j\in S, [i,j]\in E} x_{ij} \leq |S| - 1, \quad \forall S \subseteq V, |S| \geq 1, \tag{7.2c}$$

$$x_{i_1 j_1} - x_{i_2 j_2} = 0, \qquad \forall P, F \subseteq V, i \in F, j \in P, |P| \geq 2, \tag{7.2d}$$

$$x_{i_1 j} - x_{i_2 j} = 0, \qquad \forall D \subseteq V, i \in D, j \in V, \tag{7.2e}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in V, j \in V. \tag{7.2f}$$

The objective function defines all FK constraint candidates x_{ij} with $p_{ij} > 0.5$ as FK constraints if it does not violate any of the following constraints.

Unity constraint 7.2b enforces that a FK can reference only a single PK. While a single FK can in theory reference multiple different PKs, no such occurrence appeared in the analyzed databases.

Acyclicity constraint 7.2c ensures that the graph is (directionally) acyclic. However, this formulation of acyclicity requires an exponential number of constraints. To deal with that, we generate acyclicity constraints lazily [203]. Acyclicity constraint is desirable because if p_{ij} is high, p_{ji} is generally high as well (particularly for $i = j$). But directed cycles (even over intermediate tables) do not appear in the analyzed databases.

Completeness constraint 7.2d says that if a PK is compound, then either all or neither attribute of the PK P is referenced from the FK table by attributes F. Completeness constraint ensures that compound FKs are syntactically correct.

Doppelgänger constraint 7.2e says that if attributes are doppelgängers to each other, then either all or neither attribute from the doppelgänger set D reference the (same) PK attribute.

Constraint 7.2f defines the problem as an integer programming problem.

It should be noted that if constraints 7.2d and 7.2e are removed, we get an optimization problem similar to the identification of minimum spanning tree in a graph [108]. Hence, the FKs can be efficiently optimized with Dijkstra algorithm with a modified

termination condition (the algorithm terminates once the objective function starts to increase).

7.4 Results

Following paragraphs describe an empirical comparison of 5 classifies on 3 different sets of features from 72 databases.

Data We used 72 relational databases from relational repository [A.12]. The databases range from classical relational benchmarking databases (like *TPC-C* or *TPC-H*) to real-world databases used in challenges (e.g., from PKDD in 1999 or from Predictive Toxicology Challenge in 2000). The collection of the databases contains in total 1343 PKs, 1283 FK constraints, 6129 attributes and 788 tables. That means that on average approximately 1 of 5 attributes is part of a PK. The count of all *possible relationships* is 1 232 392 (in theory, a FK can reference any attribute in the database, including itself). That means that on average approximately 1 of 960 tested relationships are FK constraints.

Algorithm Following classification algorithms were tested on the problem: decision tree, gradient boosted trees, naive Bayes, neural network and logistic regression as implemented in RapidMiner 7.5. Since the best results were obtained with gradient boosted trees, the reported results are for gradient boosted trees.

Measure For evaluation of the classification models, AUC and F-measure [66] were used. Classification accuracy was omitted due to a significant class imbalance in FK identification task. AUC evaluates the ability of the model to rank. Hence, AUC is used to evaluate the quality of scoring. On the other end, F-measure is suitable for the evaluation of the quality of thresholding. Hence, F-measure is used to evaluate the quality of the optimization.

Validation To measure the generalization of the models to new unobserved databases, *batch cross-validation* over databases [224, section 4.3] was performed. Since 72 databases were analyzed, it means that each model was trained and evaluated 72 times. The batch cross-validation has the advantage, in comparison to 10-fold cross-validation, that the samples from a single database are either all in the training set or all in the testing set. Hence, if the samples from a single database are more similar to each other than to samples from other databases, we may expect that batch cross-validation will deliver a less optimistically biased estimate of the model accuracy on new unobserved databases than 10-fold cross-validation.

Feature Importance Generally, it is desirable to minimize the count of utilized features to make the model easier to understand and deploy. Table 7.1 depicts feature importance for PK identification as reported by gradient boosted trees for features that remained after backward selection.

Feature	*All*	*Meta*	*Ordinal*	*Data*
ordinalPosition	3279	1970	2581	-
isDoppelgänger	142	99	-	-
isDecimal	80	81	-	-
containsNull	18	-	-	239
levenshteinToTable	15	124	-	-
dataType	14	71	-	-
isUnique	9	-	-	780
containsKeyword	7	21	-	-
AUC	**0.985**	0.970	0.934	0.784

Table 7.1: Feature importance for primary key identification for different feature sets. Higher weight means higher importance.

The single most important feature for PK identification is the position of the attribute in the table. This is not so surprising because all non-compound PKs in the analyzed databases (with the exception of *Hepatitis* database) were the first attribute in the table. Indeed, if we always predicted that the first attribute in a table is a PK, we get F-measure equal to 0.934±0.007.

Table 7.2 lists feature importance for FK constraint identification. Interestingly, the knowledge whether the FK constraint is satisfiable is the least important feature from the selected features.

Optimization Contribution The PK optimization improves F-measure of PK identification from 0.845±0.069 to 0.875±0.057. While FK optimization improves F-measure of FK constraint identification from 0.743±0.020 to 0.870±0.022.

Runtime & Scalability The time required to score all 72 databases is 55 seconds in total, where 95% of the runtime is due to the fact that JDBC collects metadata about the attributes for each table individually, causing many round trips between the algorithm and the database server. When we replaced JDBC calls with a single query to *information_schema*, which provides all the data at the database level, the total runtime decreased to 5 seconds.

Furthermore, the algorithm was tested on our university database with 909 tables. The runtime was 18 minutes, due to the quadratic growth of candidate FK constraints with the count of attributes in the database [37]. To keep the memory requirements

Feature	*All*	*Meta*	*Data*
levenshteinFkColToPkTab	298.1	301.2	-
levenshteinFkColToFkTab	245.8	246.0	-
fk_isDoppelgänger	210.6	210.4	-
levenshteinFkColToPkCol	182.2	182.2	-
fk_containsKeyword	160.2	160.3	-
dataLengthAgree	92.2	92.2	-
pk_isDoppelgänger	60.3	60.3	-
fk_isPrimaryKey	57.8	57.8	-
fk_ordinalPosition	48.4	48.2	-
dataTypeAgree	10.3	10.2	-
satisfiesFKConstraint	1.6	-	382
AUC	**0.990**	0.988	0.934

Table 7.2: Feature importance for foreign key constraint identification. Higher weight means higher importance.

manageable, FK candidates were scored on the fly and only the top n FK candidates with the highest probability were kept in a heap for FK optimization.

7.5 Discussion

Table 7.3 depicts a comparison of our approach to different approaches in the literature. Since the implementations of the referenced approaches are not available, we take and report the measurements for the biggest common denominator of the evaluated databases – the TPC-H database. The approaches differ in the utilized features (e.g., Kruse et al. utilize SQL scripts, while our approach does not) and objectives (e.g., Chen et al. aim to maximize precision at the expense of recall). The results of our method for all 72 databases are available for download at `https://github.com/janmotl/linkifier`.

Reference	Features	Precision	Recall	F-measure	Runtime [s]
Zhang et al. [273]	D	**1.00**	**1.00**	**1.00**	501
Chen et al. [37]	D, M	**1.00**	**1.00**	**1.00**	14
Rostin et al. [224]	D, M	?	?	0.95	450
Our	M	0.77	0.77	0.77	1

Table 7.3: Literature review of different approaches to foreign key constraint identification on TPC-H 1GB database. "D" stands for data, "M" stands for metadata. Unknown values are represented with a question mark.

Empirical comparison of our metadata-based approach to other metadata-based approaches is in Table 7.4. Oracle Data Modeler [196] estimates FK constraints based on the knowledge of PKs (it is assumed that a FK must reference a PK), equality of column names between the FK and the PK and equality of the data types between the FK and the PK. SchemaCrawler [65] is using an extended version of these three filters. SchemaCrawler assumes that a FK must reference either a PK or a column with a unique constraint. The column names must equal but differences in the presence/absence of *id* keyword and differences between singular and plural forms are ignored, improving the recall. And datatypes must equal including their length (except of *varchar* datatype), improving the precission.

Implementation	F-measure	Runtime [s]
Oracle Data Modeler	0.06	**2.07**
SchemaCrawler	0.17	4.65
Our	**0.87**	5.14

Table 7.4: Empirical evaluation of metadata-based approaches to foreign key constraint identification on 72 databases together.

Limitations The metadata-based identification of PK and FK constraints is limited by the quality of the metadata. For example, if all the columns in the database had non-informative names and all the columns were typed as *text*, the accuracy of the predictions would suffer.

But even if the metadata are of hight quality, our metadata-base approach is not able to reliably reconstruct a hierarchy of subclasses. The problem is illustrated in Figure 7.2. Based on the table and PK names, we can correctly infer that *Person* and *Vendor* are subclasses of *BusinessEntity*. However, our metadata-based method has no means how to infer that *Customer* and *Employee* are subclasses of *Person* and not directly of *BusinessEntity*.

Both these limitations can be addressed by extending the metadata-based approach by appropriate data-based features. For example, whenever a subclass can reference multiple superclasses, the superclass with the lowest row count, which still satisfies the FK constraint, should be selected.

7.6 Conclusions

We described a method for foreign key constraint identification, which does not put any assumptions on the schema normalization, data quality, availability of vendor specific metadata or human interaction. The code for primary & foreign key constraint identification was designed to be database vendor agnostic and was successfully tested against

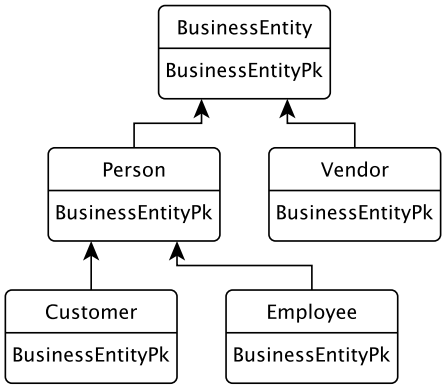

Figure 7.2: Example entity diagram with a hierarchy of subclasses.

Microsoft SQL Server, MySQL, PostgreSQL and Oracle. The code with a graphical user interface is published at GitHub (`https://github.com/janmotl/linkifier`) under BSD license. The runtime is dominated by the connection lag to the server and if the requirement on the code portability is lifted, we are able to predict primary & foreign key constraints for all 72 tested databases in 5 seconds.

Chapter 8

Stratified Cross-Validation by Multiple Columns

Stratified cross-validation is one of the standard methods of how to evaluate classifier's generalization accuracy. However, conventional implementations of cross-validation can stratify only by a single column. In this chapter, we propose to use Integer Linear Programming for stratification by multiple columns. Our experiments using an extensive set of multi-label data sets shows that the proposed method significantly outperforms non-stratified cross-validation.

8.1 Introduction

It is difficult to improve things unless we can measure them. In machine learning, one of the established methods of how to empirically evaluate supervised algorithms is cross-validation (CV).

There are many variants of CV [222, 217], but one of the more common ones for the evaluation of classifiers is a stratified CV, which assigns samples into folds such a way that each fold has (an approximately) equal distribution of the classes.

However, common implementations of stratified CV support stratification only based on a single column. But what if we want to stratify based on multiple columns, for example, because we are doing multi-label classification?

Why stratification The traditional reason why to use stratified sampling is to reduce the variance of the estimate in comparison to the variance obtained with random sampling [41, Section 5.6].

Another reason why a person could be interested into stratified CV is to increase the estimated testing accuracy in comparison to random CV [139, 93].

But most importantly, stratified CV helps us to avoid edge scenarios whenever we are working with small data sets that include a rare class in the label. If we used a

non-stratified CV on such data set, it would be fairly probable [236, Section 2] that at least one of the folds would be missing the rare class. And such absence may prevent us from evaluating the model's accuracy [74].

In extreme scenarios (e.g., when we are performing 2-fold CV), we may easily encounter a complementary nuisance: the absence of the rare class in the *training set* (while in the previous case, we were dealing with the absence of the rare class in the *testing set*). In such situations, a shelf implementation of the classifier may refuse to provide a prediction for an unseen class label. And once the count of classes in the training set decreases to one, the classifier may (rightfully) refuse to get trained because the problem degenerated into a one-class classification problem. Once again, stratified CV helps us to avoid such nuisance (till the count of the folds is equal of less than the count of samples in the rarest class).

8.2 Definitions

Split-validation is a basic method of estimating the generalization accuracy of a model on unseen data. It works by partitioning the data set into *training* and *testing* sets. The model is then trained on the training set and evaluated on the testing set.

In k-fold cross-validation, the above partitioning is repeated k times with the constraint that each sample in the data set is used exactly once. The k estimates are then averaged to provide a single estimate.

The partitioning can be random, but that can result in unequal prior probabilities across the partitions (*folds*) and consequently negatively biased estimates. *Stratified* k-fold cross-validation alleviates this issue by using the same distribution of the classes in each fold.

8.3 Related Work

Diamantidis et al. [52] extended k-fold cross-validation method with *clustering* in order to construct more representative folds. The key idea was that each fold should have an approximately equal count of samples from each cluster. This procedure should help to get folds with as equal distribution of the training samples as possible. Based on their reported experimental results, the proposed method mainly reduces the bias of the measure (i.e., it increases testing classification accuracy) on small data sets. What is important about this reference is that they "stratify" based on the features of the data set and not based on the label as it is done by conventional stratified CV. This suggests, that our proposed CV, which works on multiple columns, can be useful not only on multi-label data sets, but also on single-label or even completely unsupervised data sets.

Zeng and Martinez [272] introduced *distribution-balanced stratified cross-validation* (DBSCV), which also aims to minimize the differences in the distribution of the folds. Their algorithm is based on single-linkage from hierarchical clustering. On highly un-

balanced datasets, a slight modification of DBSCV called DOB-SCV improved testing AUC on C4.5 on average by 2 percent point in comparison to stratified cross-validation [182, 166] .

Contrary to the previously cited works, Sechidis et al. [236] did not focus on stratification in feature space but on stratification on label space, where we have multiple binary labels (multi-label classification data sets). They introduced a *greedy* algorithm, which directly equalizes the distribution of positive samples of each label across the folds. Szymański and Kajdanowicz [247] then modified this greedy algorithm to equalize the distribution of 2-way interactions between labels across the folds (i.e.: "1-way interactions" were replaced with 2-way interactions). We combine 1-way and 2-way label interactions in a single optimization formulation and complement it with n-way label interactions, a competitive baseline stratification method from [236], where n is the count of columns, over which we stratify. Furthermore, we guarantee that each fold has approximately the same count of samples (the greedy algorithm did not guarantee that) and we replace the greedy optimization with an *exact* optimization.

8.4 Solution

We solve the problem with *Integer Linear Programming* (ILP) on grouped and one-hot encoded data (see an example in Figure 8.1).

$$\underset{Data}{\begin{bmatrix}1 & 1\\ 2 & 2\\ 2 & 3\\ 1 & 1\\ 1 & 2\end{bmatrix}} \rightarrow \underset{Grouped}{\begin{bmatrix}1 & 1\\ 2 & 2\\ 2 & 3\\ 1 & 2\end{bmatrix}} \underset{Count}{\begin{bmatrix}2\\ 1\\ 1\\ 1\end{bmatrix}} \rightarrow \underset{Dummies}{\left[\begin{array}{cc|ccc}0 & 1 & 0 & 0 & 1\\ 1 & 0 & 0 & 1 & 0\\ 1 & 0 & 1 & 0 & 0\\ 0 & 1 & 0 & 1 & 0\end{array}\right]} \underset{Count}{\begin{bmatrix}2\\ 1\\ 1\\ 1\end{bmatrix}}$$

Figure 8.1: Example of data preprocessing for ILP.

Whenever the cardinalities (the counts of the unique values) of the columns are low and the row count is high, the grouping reduces the size of the ILP problem. One-hot encoding into dummies (where a column with cardinality n is encoded into n binary columns) then simplifies the formulation of the ILP problem. We encode labels with n unique values into n binary columns instead of more space-saving $n-1$ binary columns, because contrary to the previous works ([236, 247]), we support *multi-class labels* (labels with more than 2 unique values) and we want to avoid the need to define the reference class (the contrast) to exclude from the $n-1$ encoding. For the illustration of how the split can change based on the definition of which class is the "positive", see Figure 8.2.

Integer Linear Programming Ideally, we would like the stratified sampling to satisfy following requirements:

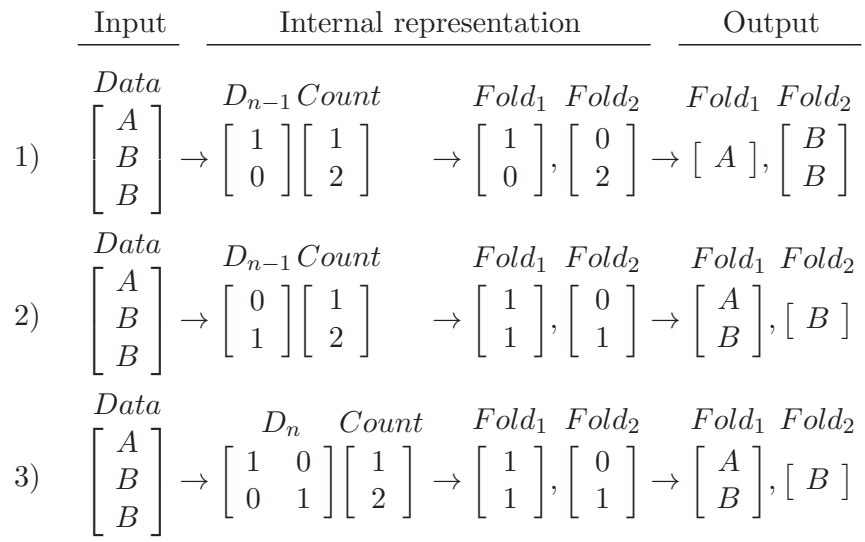

Figure 8.2: When we care only about positive samples (A in the first row), we may split the data into two folds following way: $[A]$ and $[B, B]$, because each fold has approximately the same count of positive samples A. When we switch the definition of the positive class (B in the second row), we end up with a better split $[A, B]$ and $[B]$ – the quality of A distribution remained unchanged while the quality of B distribution improved. When we one-hot encode the input data into n binary columns (the third row) instead of $n-1$ binary columns (the first two rows), we end up with the better split without the need to define the positive class. Note that the encoding into n binary columns generalizes well to multi-class problems, where it can be unclear which class(es) should be treated as the positive.

1. Each sample is assigned to exactly one fold.
2. The count of each 1-way interaction is the same in each fold.
3. The count of each 2-way interaction is the same in each fold.
4. The count of each n-way interaction is the same in each fold.
5. The count of samples is the same in each fold.

Unfortunately, requirements 2–5 are not always fulfillable since we can only assign whole samples to folds (and not fractions of the samples). For example, when we have 3 samples and 2 folds, we end up with the unequal size of the folds. Hence, we have to relax the constraints to the nearest whole numbers of the ideal continuous value. For example, when we have 3 samples and 2 folds, we constraint fold size to be ≥ 1 and ≤ 2 because the ideal fold size is 1.5.

But we can still find examples that evade these relaxed constraints (see Figure 8.3).

We could have further relaxed the constraints, but then we could have got a suboptimal result when an optimal solution exists. For example, we could have got one fold

$$\begin{matrix} \textit{Dummies} & & & \textit{Count} \\ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} & & & \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \end{matrix}$$

Figure 8.3: No matter how do we attempt to assign these three rows into two folds, there is going to be a column with the difference of the value appearance equal to two.

with 2 samples and another fold with 4 samples, when a perfect solution with 3 samples in each fold exists.

Hence, what do we really want to do is to minimize the error from the ideal (but potentially continuous) state. In ILP, we can solve it by introducing slack variables. The whole ILP formulation of multi-column stratified cross-validation is in Section 8.4, where vector S are the slack variables, matrix D and vector C are the dummies, respectively the counts from Figure 8.1 and matrix X represents the assignment of the dummies into the folds (the output of the optimization). The formulation minimizes the sum of the slack variables 8.4a subject to constraints 8.4b-l.

$$\min_{S_r} \quad \sum_r S_r \tag{1a}$$

$$\text{s.t.} \quad \sum_f X_{r,f} = C_r, \qquad \forall r, \tag{1b}$$

$$X_{r,f} \leq \left\lceil \frac{C_r}{folds} \right\rceil, \qquad \forall r, \tag{1c}$$

$$X_{r,f} \geq \left\lfloor \frac{C_r}{folds} \right\rfloor, \qquad \forall r, \tag{1d}$$

$$\sum_r D_{r,c} X_{r,f} \leq \left\lceil \frac{\sum_r D_{r,c} C_r}{folds} \right\rceil, \qquad \forall c, \forall f, \tag{1e}$$

$$\sum_r D_{r,c} X_{r,f} \geq \left\lfloor \frac{\sum_r D_{r,c} C_r}{folds} \right\rfloor, \qquad \forall c, \forall f, \tag{1f}$$

$$\sum_r D_{r,c_1} D_{r,c_2} X_{r,f} \leq \left\lceil \frac{\sum_r D_{r,c_1} D_{r,c_2} C_r}{folds} \right\rceil, \forall c, c_1 < c_2, \forall f, \tag{1g}$$

$$\sum_r D_{r,c_1} D_{r,c_2} X_{r,f} \geq \left\lfloor \frac{\sum_r D_{r,c_1} D_{r,c_2} C_r}{folds} \right\rfloor, \forall c, c_1 < c_2, \forall f, \tag{1h}$$

$$\sum_r (X_{r,f} - S_r) \leq \left\lceil \frac{\sum_r X_{r,f}}{folds} \right\rceil, \qquad \forall f, \tag{1i}$$

$$\sum_r (X_{r,f} + S_r) \geq \left\lfloor \frac{\sum_r X_{r,f}}{folds} \right\rfloor, \qquad \forall f, \tag{1j}$$

$$X_{r,f} \in \mathbb{N}, \tag{1k}$$

$$S_r \in \{0, 1\}. \tag{1l}$$

Constraint set 8.4b ensures that each unique row r is assigned exactly C_r times (requirement 1). Constraint sets 8.4c and 8.4d ensure that unique rows r are assigned to folds f approximately equally (n-way interaction requirement 4). Constraint sets 8.4e and 8.4f ensure that column values (as represented by matrix D and vector C) in each column c are assigned to folds approximately equally (1-way interaction requirement 2). Constraint sets 8.4g and 8.4h ensure that 2-way interactions are assigned to folds approximately equally (requirement 3). Constraint sets 8.4i, 8.4j and objective 8.4a ensure that folds are of equal size as far as it is possible (requirement 5). Constraint set 8.4k ensures that only whole rows are assigned (and not only fractions). Also, it forbids solutions with negative counts. Constraint set 8.4l defines a sufficient domain of the slack variables S.

Solution extraction If we assume that the samples within the same label set are i.i.d., we can just randomly assign the whole data rows from the data set following the assignment of the strata groups into the folds as given by matrix X. On the other end, if we have a reason to believe that the data within the same label set are not i.i.d., we may follow up a stratification in feature space per label set.

If we need to generate multiple different CV assignments (e.g., for nested CV), there are two sources of randomness. First, we can instruct the ILP solver to return the top n solutions[1]. Second, we can change the seed in the pseudorandom number generator before randomly assigning the whole data set rows into the folds based on the matrix X.

If we need to stratify by a continuous column, we can discretize the column (for the purpose of stratification, for all other purposes, we may still use the continuous values) and continue as if the column was discrete from the beginning.

8.5 Experiments

In this section, we present an experimental evaluation for stratified cross-validation proposed in this chapter. The goal of our experiments is to answer the following questions:

Q1. Do the generated folds have the desired properties?

Q2. Does it decrease the variance of the classification measures?

Q3. Does it increase the mean of the classification measures?

Algorithms We compare our proposed algorithm for sample-fold assignment to two other algorithms: one baseline and one state-of-the-art implementation of stratified cross-validation for multi-label classification.

As a simple baseline, we use **random** (unstratified) cross-validation. As the state-of-the-art reference for multi-label stratification, we use greedy **iterative** algorithm from [247], which takes into account second-order relationships between labels[2]. We refer our

[1] In Gurobi solver 9.0, the argument is called *PoolSolutions*

[2] We used `iterative_stratification` 0.2.0 implementation from http://scikit.ml

algorithm as **ILP**, because it uses Integer Linear Programming solver.

Measures Following the convention established in [236, 247], we use the following *unsupervised measures* to describe the quality of the generated folds:

Examples Distribution (ED) describes how much the actual size of the folds S_j deviate from the desired size of the folds c_j:

$$ED = \frac{1}{k}\sum_{j=1}^{k} ||S_j| - c_j|^k. \tag{8.1}$$

In our case, we desire 10 folds of the same size. Hence, $k = 10$ and $c_j = \frac{|D|}{10}$, where $|D|$ is the count of samples in the whole data set. This measure evaluates conformance with the requirement 5.

Label Distribution (LD) describes how the proportion of positive evidence for a label to the negative evidence for a label deviates from the same proportion in the entire data set, averaged over all folds and labels:

$$LD = \frac{1}{|L|}\sum_{i=1}^{|L|}\left(\frac{1}{k}\sum_{j=1}^{k}\left|\frac{|S_j^i|}{|S_j| - |S_j^i|} - \frac{|D^i|}{|D| - |D^i|}\right|\right). \tag{8.2}$$

This measure evaluates relaxed conformance with the requirement 2 (requirement 2 applies to each class while this measure, as defined in [236], applies only on positive class).

Fold Zeros (FZ) is the count of folds that contain at least one label without any positive example.

Fold-Label Zeros (FLZ) is the count of fold-label pairs without any positive example.

Each of these measures should be minimized. Furthermore, we use extensions of the LD, FZ, and FLPZ measures [247] that work at the level of 2-way an n-way label interaction. To keep the count of the abbreviations low, we denote the order of the interaction with a subscript (see Table 8.1).

From *supervised measures*, we use *Hamming loss, subset accuracy score, coverage error, label ranking loss, macro-averaged precision*, and *micro-averaged ROC-AUC*, once again, following the convention established by [236, 247]. For the definition and discussion of these measures, we refer the keen reader to [263].

Classifiers Since we optimize the folds at the three levels of label interaction (1-way, 2-way, and n-way), we use three different classifiers. **Classifier-Chains** (CC), which should be mostly sensitive to the one-way interactions but fairly insensitive to the two and

	Interaction		
Measure	1-way	2-way	n-way
LD	LD_1	LD_2	LD_n
FZ	FZ_1	FZ_2	FZ_n
FLZ	FLZ_1	FLZ_2	FLZ_n

Table 8.1: Unsupervised measures categorized by the level of label interactions. n represents the count of all the labels in the data set.

n-way interactions. **Fast-Greedy** (FG), which in comparison to CC explicitly models two-way interactions (do not confuse the greedy classifier with the greedy stratification algorithm). And **Label Powerset** (LP), which explicitly models the n-way interactions. As a base classifier, we use random forest with 30 trees.

Data We used 29 multi-label data sets from *Multi-Label Classification Dataset Repository*[3].

Data sets, which differ only in the feature-space, were included only once, to be able to assume sample independence in the statistical tests discussed in Section 8.5 (E.g., we include only *GpositiveGO* and exclude *GpositivePseAAC* because they are identical in the label-space).

Furthermore, we excluded data sets, where ILP was not able to prove the optimality of the found solution in less than 8 hours[4].

The list of the used data sets and their properties are in Table 8.2.

Statistical tests We evaluated the measurements with two-tailed Nemenyi test following Demšar's recommendations [48]. For the *Critical Difference* (CD) in the presented text and plots, we use significance level $\alpha = 0.05$.

8.6 Results

The following paragraphs depict ranking of the cross-validation algorithms, averaged over all the data sets. Rank 1 is always the best and rank 3 is always the worst.

Figure 8.4 shows the ranking for unsupervised measures defined in Section 8.5. The proposed algorithm delivers folds that are on average equal or better than the folds obtained with other algorithms. Noteworthy, ILP delivers folds of equal size (when possible). Hence, it matches random solution (which is not concerned with *anything else* but the fold sizes) and beats iterative solution (which is *not* concerned with the fold sizes).

[3] http://www.uco.es/kdis/mllresources/

[4] We used Gurobi 9.0 solver on a 10-core computer with 64GB RAM

Data set	Domain	Instances	Attributes	Labels	Labelsets
Arts	Text	7 484	23 150	26	599
BBC1000	Text	352	1 000	6	15
Birds	Audio	645	260	19	133
Business	Text	11 210	21 920	30	233
CHD49	Medicine	555	49	6	34
Computers	Text	12 440	34 100	33	428
Education	Text	12 030	27 530	33	511
Emotions	Music	593	72	6	27
Entertainment	Text	12 730	32 000	21	337
EukaryoteGO	Biology	7 766	12 690	22	112
Flags	Image	194	19	7	54
GnegativeGO	Biology	1 392	1 717	8	19
GpositiveGO	Biology	519	912	4	7
Guardian1000	Text	302	1 000	6	14
Health	Text	9 205	30 610	32	335
Image	Image	2 000	294	5	20
Inter3000	Text	169	3 000	6	11
Recreation	Text	12 830	30 320	22	530
Reference	Text	8 027	39 680	33	275
Reuters1000	Text	294	1 000	6	14
Scene	Image	2 407	294	6	15
Science	Text	6 428	37 190	40	457
Slashdot	Text	3 782	1 079	22	156
Social	Text	12 110	52 350	39	361
Society	Text	14 510	31 800	27	1 054
TMC2007-500	Text	28 600	500	22	1 172
VirusGO	Biology	207	749	6	17
Yeast	Biology	2 417	103	14	198
Yelp	Text	10 810	671	5	32

Table 8.2: List of the used multi-label data sets.

Figure 8.5 shows ranking for the average of supervised measures as obtained from cross-validation with 3 different classifiers. ILP gives statistically significantly higher micro-averaged ROC-AUC than iterative algorithm when evaluated with fast greedy classifier, but it also statistically non-significantly losses to iterative algorithm on example-based evaluation metrics [248] (subset accuracy and Hamming loss) when evaluated with fast greedy classifier.

Figure 8.5 shows ranking for the variance of supervised measures as obtained from cross-validation with 3 different classifiers. Both stratified approaches seem to statistically significantly decrease the variance of all measures except of macro-averaged precision

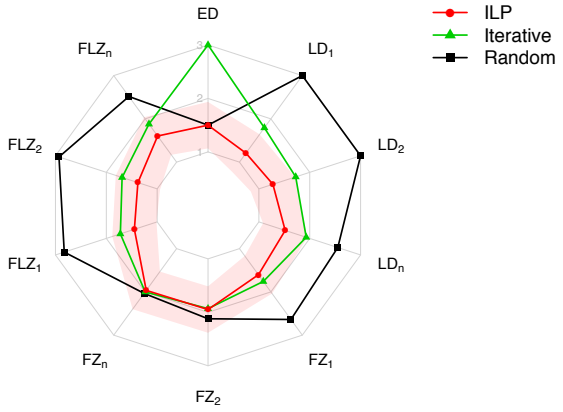

Figure 8.4: Ranking of the cross-validation algorithms based on the unsupervised measures. Smaller area is better.

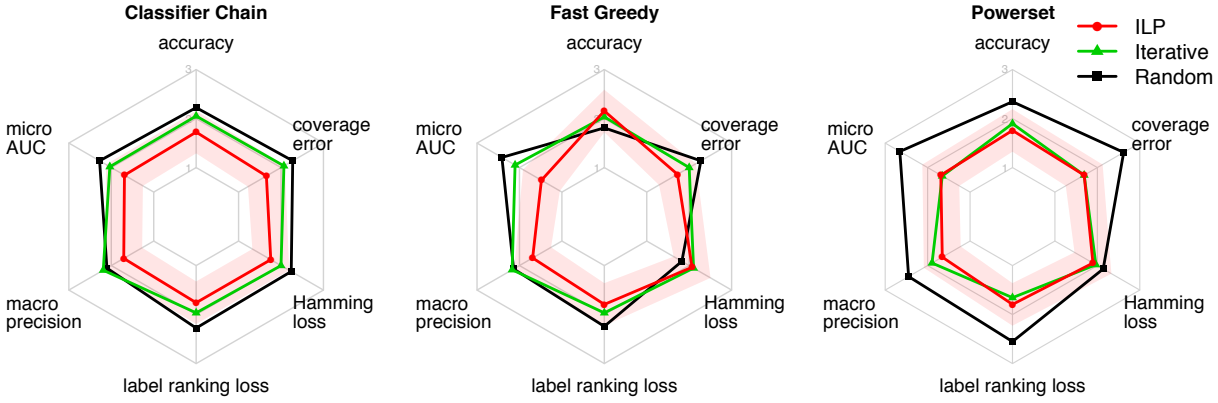

Figure 8.5: Ranking of the cross-validation algorithms based on the average of supervised measures. Smaller area is better.

and accuracy subset.

Answers The following lines answer questions formulated at the beginning of the experimental subsection.

Q1 Do the generated folds have the desired properties? On average, ILP folds have by 45% fewer folds that contain at least one label without any positive example (the FZ measure) than random folds and by 9% fewer folds than iterative folds. Hence, the probability that you will not be able to estimate the classifier's performance is substantially lower with ILP folds than with random or iterative folds.

Q2 Does it decrease the variance of the classification measures? On average, ILP folds have by 1.2% lower variance than random folds and by 0.1% lower variance than iterative folds.

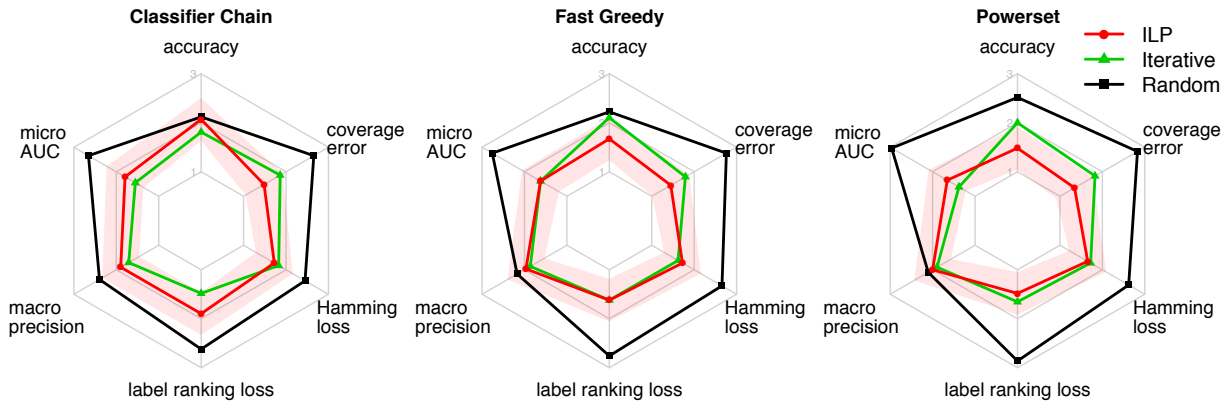

Figure 8.6: Ranking of the cross-validation algorithms based on the variance of supervised measures. Smaller area is better.

Q3 Does it improve the mean of the classification measures? On average, ILP folds have by 1.1% higher accuracy/lower loss than random folds and by 0.4% higher accuracy/lower loss than iterative folds.

The raw measurements and the code necessary for the replication of the experiment is available from https://github.com/janmotl/cv.

8.7 Discussion

Solver There are many ways how the problem could have been solved. To name just a few alternatives, we could have used Constraint Programming, SAT solver, or plain old exhaustive search with backtracking. We choose ILP because of our familiarity with the method.

I.I.D. Real-world data are frequently non-i.i.d. [240, 11] and we may desire to assign all related samples (e.g., purchases from the same customer) into the same fold. But this requirement can interfere with our constraints. E.g., the maximal count of the purchases from a single customer can be bigger than the desired fold size. An elegant way of how to avoid the clash of the constraints is to directly minimize the error from the ideal state as defined in Section 8.4 instead of defining the hard constraints 8.4c-h. However, this proposal would require weighting of the individual error terms or usage of multi-objective optimization.

Interactions Whenever we stratify by n columns with the proposed algorithm, we look only at 1-way, 2-way, and n-way interactions between the labels while ignoring all the interactions between the 2-way and n-way interactions. Hence, whenever we stratify by more than two labels, our solution does not guarantee the "optimality" of the fold assignment as it does when we stratify by one or two labels. While we could add constraints for any i-way interaction, $i \in \{1, 2, ..., n\}$, into the ILP formulation, the count of the constraints grows following Pascal's triangle. I.e.: To model all the interactions

between n labels, we would need about $c(2^n)$ constraints, where c is a constant dependent on the count of folds and the exact ILP formulation. Hence, we model only three types of interactions, that we encountered in the literature about multi-label classification: 1-way, 2-way, and n-way interactions.

8.8 Conclusions

In this chapter, we introduced an algorithm for exact stratified cross-validation, when we want to stratify by multiple columns (targets). The proposed algorithm replaces a greedy solution of the former algorithms [236, 247] with the exact solution, adds balanced distribution in the feature space and adds support for polynomial and continuous targets. The proposed algorithm statistically significantly matches or surpasses non-stratified and greedily optimized solutions in all evaluated metrics. Among others, the created folds increase testing accuracy of the trained models and decrease the incidence rate of the edge situations, when the evaluation is not possible at all. The created folds were published at http://www.uco.es/kdis/mllresources.

Chapter 9

Trend and Seasonality Elimination

Detrending and deseasoning is a common preprocessing step in time-series analysis. We argue, that the same preprocessing step should be considered on relational data, whenever the observations are time-dependent. We applied Hierarchical Generalized Additive Models (HGAMs) to detrend and deseason (D&D) 18 real-world relational datasets. The observed positive effect of D&D on the predictive accuracy is statistically significant. The proposed method of D&D might be used to improve the predictive accuracy of churn, default, or propensity models, among others.

9.1 Introduction

This chapter is concerned with feature extraction from relational data for supervised learning, like classification or regression. The studied problem is not new and is called propositionalization [137]. But one aspect of propositionalization was frequently ignored in the early propositionalization literature: the data are frequently not just relational, but also temporal. The ignorance of the temporal parts in relational data is quite understandable – neither processing of relational or temporal data is trivial when we want to do it right. This chapter attempts to slightly narrow the gap between these two worlds by introducing one basic concept from time-series modeling to the realm of propositionalization: *the trend and season removal from relational data*.

To illustrate the importance of detrending, consider a predictive model built on financial data. If the model was built on raw data, the accuracy of the model would likely degrade faster than if we build the model on inflation adjusted ("deflated") data.

To illustrate the importance of deseasoning, consider observing a person for a month. If the person went swimming in a river once during that month, we might want to make different conclusions about the person when that month is June than if the month was January.

To describe the proposed temporal normalization, we have to introduce the basics of propositionalization in Section 9.2 and the relevant parts from time-series analysis in Section 9.3. We then follow with the description of the proposed temporal decomposition

on relational data in Section 9.4, performed empirical evaluation in Section 9.5, and the results in Section 9.6. The results are discussed in Section 9.7 and concluded in Section 9.8.

9.2 Propositionalization

Propositionalization is a process of conversion of relational data into attribute-value format, which can be understood by conventional machine-learning algorithms (see [137]).

There are many propositionalization methods. In this chapter we use PredictorFactory[1]. But there are many other, which differ in the direction of data propagation (in [144], the target, which we wish to predict, is propagated to the data, while in [137] the data are propagated toward the target) or the count of the used models (in [144] and [137], only one model is used, while Guo [98] uses as many models, as there are tables in the database).

In this chapter, when we are talking about propositionalization, we assume that the `target`, which we wish to predict, resides in `target table`. The moment, when we wish to make the prediction, is defined by `target timestamp`. And the entity, for which we wish to make the prediction, is defined by `target id`.

To predict the target from other tables than the `target table`, we have to link the tables together. This is done with joins over foreign key constraints [144]. The end state of this *target propagation* is that each table in the database contains the `target column`, `timestamp`, and `id` from the `target table`.

The relationship between the target table and other tables does not always have to be 1:1. When the relationship is 1:n, the content of the tables is aggregated to the level of the data in the target table. Examples of common aggregate functions for numerical attributes include `min`, `max`, `avg`, `count`, and `stddev`.

The final step then consists of joining all the features from the propagated and potentially aggregated tables into a single table in attribute-value format.

9.3 Detrending and Deseasoning

The trend and seasonality can be removed with one of the classical methods from time-series analysis like Seasonal and Trend decomposition using Loess (STL) [40], Holt-Winters [109] or SARIMA [21].

However, these methods are not directly applicable to our data, because common implementations of these methods cannot deal with:

1. missing values (e.g., caused by holidays),
2. multiple values per sample period (e.g., multiple transactions per day),

[1] www.predictorfactory.com

3. hierarchical dependencies between the data (e.g., multiple transactions from multiple different customers).

The listed deficiencies can be handled by Hierarchical Generalized Additive Models (HGAMs) [197]. HGAMs are a combination of Generalized Additive Models (GAMs) [104] and Hierarchical Generalized Linear Models (HGLMs) [24]. GAM part takes care of modeling seasonality with cyclic cubic regression splines, while HGLM part takes care of non-independence in the data that arises from hierarchical structures. Note that HGAMs, just like GAMs or HGLMs, can handle non-uniformly spaced samples and multiple values per sample period.

9.4 Method

Assuming additive decomposition, a time-series y at time t can be written as $y_t = S_t + T_t + R_t$, where S_t is the seasonal component, T_t is the trend component, and R_t is the remainder component.

However, in this chapter, we assume that the data have a two-level hierarchical structure. The first level of data clustering is defined by `target id`. The second level is the whole population. This hierarchical structure is used, because it allows us to model both, 1:1 and 1:n relationships between `target id` and data in a relational dataset.

A two-level hierarchical time-series can be written as $y_{t,i} = S_t + T_t + T_{t,i} + B_i + R_{t,i}$, where i is a specific `target id`, S_t is the population seasonal component, T_t is the population trend component, $T_{t,i}$ is the trend of i (random slope in HGLM literature), B_i is the bias of i (random intercept in HGLM literature), and $R_{t,i}$ is the remainder component.

Furthermore, we want to support exogenous variables in the decomposition, because the data in the tables do not have to be always commensurable. For example, `transaction` table in Financial dataset [12] contains all possible types of transactions, differentiated by columns `type` and `operation` (the used datasets are discussed in Section 9.5). To block the effect of the exogenous variables, we write the decomposition as:

$$y_{t,i} = \beta X_{t,i} + S_t + T_t + T_{t,i} + B_i + R_{t,i}, \tag{9.1}$$

where $X_{t,i}$ is a vector of exogenous variables (fixed effects in HGLM literature) and β is a vector of regression coefficients.

9.5 Experiments

In the introduction, we argued that the trend and seasonality of data is important. But does it really impact the quality of the prediction? And if yes, how much?

To answer these questions, we empirically compare accuracies of models build on raw data vs. temporarily normalized data on 18 real (non-artificial) temporal datasets from relational repository [A.12] (see the list in Table 9.1).

Dataset	Target table	Target column	Target id	Target timestamp	Classes	Threshold
Accidents	nesreca	klas_nesreca	id_nesreca	cas_nesreca	B, not B	2001-06-30
Airline	On_Time	ArrDel15	rownum	FlightDate	0, 1	2016-01-16
BasketballMen	teams	rank	tmID, year	year	≤ 4, >4	1974
BasketballWomen	teams	playoff	tmID, year	year	N, Y	2003
CCS	transactions_1k	Price	TransactionID	Date	≤ 500, >500	2012-08-24
Financial	loan	status	account_id	date	A, B	1997-02-05
FNHK	pripady	Delka_hospitalizace	Identifikace_pripadu	Datum_prijeti	≤ 7, >7	2014-06-16
Geneea	hl_hlasovani	vysledek	id_hlasovani	datum	A, R	2014-12-02
Lahman	salaries	salary	teamID, playerID, lgID	yearID	≤ 500000, >500000	1998
LegalActs	legalacts	ActKind	id	update	Решение, not Решение	2012-07-24
NBA	Game	ResultOfTeam1	GameId	Date	-1, 1	2014-04-02
NCAA	target	team_id1_wins	id	season	0, 1	2012
PremierLeague	Matches	ResultOfTeamHome	MatchID	MatchDate	-1, 1	2014-01-01
Seznam	probehnuto	kc_proklikano	client_id, sluzba	datum	≤ 1000, >1000	2014-08-01
Stats	users	Reputation	Id	LastAccessDate	≤ 10, >10	2014-03-29
VOC	voyages	arrival_harbour	number, number_sup	arrival_date	Batavia, not Batavia	1723-07-20
Walmart	train	units	store_nbr, item_nbr	date	≤ 10, >10	2013-04-15
Yelp	Reviews	stars	review_id	review_date	≤ 3, >3	2011-04-25

Table 9.1: List of the used relational datasets from relational repository [A.12]. Regression and polynomial classification problems were converted to binary problems with the logic described in Classes column. Threshold column defines the split to training and testing set.

Each of the datasets was processed independently from the other datasets with the following data flow:

1. Split the dataset to training and testing set with the median of `target timestamp` (e.g.: `median(loan.date)` in Financial dataset. The used thresholds are in Table 9.1. The older set is the training set, while the newer set is the testing set. This schema allows us to model a common scenario, where a model is trained on all historical data and then used for an extended period of time without any retraining.
2. Perform target propagation, as described in Section 9.2.
3. Fork the flow into the challenger and the baseline group. In the challenger group, we perform temporal normalization. In the baseline group, we keep data as they are. Temporal normalization consists of:

   a) HGAM training on the training data.
   b) Global trend and seasonality removal from both, training and testing data.

   To see, whether it is actually beneficial to perform both, detrending and deseasoning, we also evaluate flows, where we only perform detrending, respectively deseasoning.
4. Propositionalize numerical attributes with the following aggregate functions: `min`, `max`, `avg`, `count`, `sum`, and `stddev` as described in Section 9.2. We also test adding of `trend` $T_{t,i}$ from Equation (9.1) to the set of aggregates in separate flows.

5. Train Random Forest[23] on the propositionalized training data. We could have chosen any other algorithm, but we chose Random Forest, because it delivers good results even without meta-parameter tuning[68].

6. Evaluate area under receiver operating characteristics curve (AUC-ROC) on the testing set. Once again, we could have chosen any other measure, but we chose AUC-ROC because it is generally more sensitive to the changes in the prediction than thresholding measures as AUC-ROC takes into account all possible thresholds while thresholding measures just one.

9.6 Results

Empirical results in Table 9.2 suggest that detrending, deseasoning, and inclusion of `slope` among the set of the utilized aggregates is better than the baseline, which works on raw data and does not use `slope` aggregate.

Dataset	Baseline	Detrended	Deseasoned	D&D	D&D with slope
Accidents	0.73	0.73	**0.74**	**0.74**	**0.74**
Airline	0.80	0.81	**0.85**	**0.85**	**0.85**
BasketballMen	0.65	0.64	0.65	0.65	**0.66**
BasketballWomen	0.63	0.63	0.63	0.63	**0.64**
CCS	0.69	**0.70**	**0.70**	**0.70**	**0.70**
Financial	0.87	0.89	0.89	**0.90**	**0.90**
FNHK	0.70	0.72	**0.74**	0.73	0.73
Geneea	0.77	0.77	**0.78**	**0.78**	**0.78**
Lahman	**0.58**	**0.58**	**0.58**	**0.58**	**0.58**
LegalActs	0.93	0.93	**0.94**	0.93	0.93
NBA	0.60	**0.62**	0.60	**0.62**	**0.62**
NCAA	**0.70**	**0.70**	0.69	**0.70**	**0.70**
PremierLeague	**0.67**	**0.67**	**0.67**	**0.67**	**0.67**
Seznam	0.85	**0.87**	**0.87**	**0.87**	**0.87**
Stats	0.75	0.75	0.75	0.75	**0.76**
VOC	0.93	0.93	0.93	0.93	**0.94**
Walmart	0.56	**0.59**	0.57	**0.59**	**0.59**
Yelp	0.84	**0.85**	0.84	**0.85**	**0.85**
Average	0.74	0.74	0.75	0.75	0.75
Wins	3	8	9	12	16

Table 9.2: The effect of temporal normalization and inclusion of slope as a feature on AUC-ROC. D&D stands for detrended and deseasoned. Bold font indicates the best result for a dataset.

A one-tailed Wilcoxon signed-rank test indicated that AUC-ROC of propositionalization on detrended and deseasoned data with `slope` as an additional feature generative function was statistically significantly higher than AUC-ROC of baseline propositionalization with $p < 0.00048$.

9.7 Discussion

The empirical results suggest that all the tested enhancements: detrending, deseasoning, and `slope` feature generative function are overall improving the accuracy of the classification models. This improvement is statistically significant, although when averaged over all the tested datasets, the improvement is modest (1 percent point). In the following paragraphs, we explain the observation and list the limitations of the performed study and the implemented method.

When the method works A good candidate dataset for temporal normalization has the following properties:

1. The dataset is affected by a linear trend and the dataset contains data for long enough to observe the effect of the trend.
2. The training set captures at least 2 full seasonal cycles, from which the seasonal pattern could be estimated.
3. Each `target id` contains at least 10 observations, from which the `slope` could be estimated.

When the method fails While sport is affected by the seasons (e.g.: Winter sports are generally more popular during the cold seasons than during the hot seasons), the tasks associated with the sports datasets in the relational repository (BasketballMen, BasketballWomen, Lahman, NBA, NCAA, PremierLeague) is the prediction of which of the teams wins the match. And we did not observe strong seasonality associated with this task.

Limitations of the study We have only evaluated the impact of detrending and deseasoning on numerical attributes while ignoring categorical attributes. But in principle, a categorical attribute can always be converted to a set of numerical attributes with the existential quantifier or count aggregates[201].

Another limitation is that we do not model a mixture of seasonal patterns. E.g., we can identify two groups of clients in Financial dataset based on their income: clients that are getting the 13. and the 14. paycheck (they have almost twice as big incomes in June and December than in any other month) and clients that are not getting the 13. and the 14. paycheck (they have the same paycheck each month). We are currently treating these two groups as a single group. Hence, when we deseason the data, the transactions

of clients with the 13. and the 14. paycheck get currently undercorrected, while the transactions in the second group of the clients get overcorrected. In principle, the two groups can be identified with clustering based on dynamic time warping distance. And each cluster can then have its seasonal pattern. But fixing these deficiencies is future work.

9.8 Conclusions

We removed trend and seasonality from relational data with Hierarchical Generalized Additive Models and found that it improves the predictive accuracy of classifiers built on top of the data.

Chapter 10

Generalized Aggregates

In this chapter, we define an approximate generalization of aggregate functions for relational data with temporal attributes. This generalization is parametrized, to allow simulation of a range of common aggregate functions and optionally take into account time. The parameters are not optimized but we rather rely on repeated stochastic sampling of the parameters. We then apply a common regularized linear model to train a model on this high-dimensional space. Experimental results on 11 datasets suggest that there are datasets, where incorporating time dimension into the model leads to an improvement in the predictive accuracy of the trained models.

10.1 Introduction

In this chapter, we approximate aggregate functions commonly used in propositionalization like `min`, `max`, `avg`, `count`, `sum`, and `var` with a weighted sum, where each sample instance in a set is assigned a weight.

What is propositionalization Propositionalization is a process of converting relational data, which consist of multiple tables, into a propositional (a single table) form, which can be analyzed with conventional machine learning algorithms. The key problem that propositionalization solves is how to deal with data that are in $n : 1$ relationship (e.g., for each customer we have a log of n their own past transactions). Propositionalization algorithms solve this $n : 1$ problem with aggregate functions, which take a vector and return a scalar (e.g., `min` function returns the smallest value in the vector).

Why the generalization Because it allows us to put more weight on recent samples (e.g., weight with exponential weighting) or model temporal dependencies (e.g., model an event that happens with a two-week delay).

Why it might work Instance weighting is the low-level mechanism of a whole class of supervised models: boosting algorithms (e.g., AdaBoost [79] or Extreme Gradient Boosting [36]) and bagging algorithms (e.g., sampling with replacement, as used in Random Forest [23], can be seen as a binarized form of instance weighting). If instance weighing works so well for propositional data, why it should not work well also for relational data?

What is the complication A slight inconvenience to the idea of aggregation thru instance weighting is that we have to support aggregation of vectors of variable length. E.g., when we find a weight $\mathbf{w}$ to work well on daily data from January, we would like to see weight $\mathbf{w}$ work well even on daily data from February, even though February has fewer days than January. We solve the issue by using a parametric definition of the weights (i.e., the count of the parameters is independent of the vector's length). In this chapter, we parametrize the weights with logistic sigmoids.

Why the sigmoid It was shown that a neural network with a sigmoidal activation function is a universal approximator [43, 112]. Hence, if some aggregate function can be expressed with a vector of weights $\mathbf{w}$, we can approximate the weight vector $\mathbf{w}$ with a set of sigmoids to approximate the aggregate function.

How do we optimize it The weights can be either optimized as in boosting algorithms or assigned randomly, as in bagging algorithms. In this chapter, we opt for random assignment of the weights.

The remainder of this chapter is organized as follows: In Section 10.2, we review related literature about relational learning, and particularly relational neural learning. In Section 10.3, we derive the generalized aggregate function. In Section 10.4, we describe the setup of the experiments we conduct to evaluate the generalized aggregate function. Section 10.5 presents the empirical results. In Section 10.6 and Section 10.7, we discuss extensions of our work and summarize the results.

10.2 Related Work

Representation One of the key issues in relational learning is feature extraction from sets. In traditional Inductive Logic Programming (ILP), like in PROGOL [185], we use existential quantifier, while in traditional propositionalization algorithms, like in RELAGGS [144], we use aggregates like `min`, `max`, `avg`, `count`, `sum`, and `var`. Many other "set descriptors" were described in the literature (see e.g., [149, 257, 252, 201]). Of particular interest for us is [194], where Neville et al. applied a model directly on the instances in the sets, giving each instance in the set unit weight. Schulte and Routley [235] then extended this approach by giving each instance in the set weight $1/n$, where n is the count of instances in the set. The idea was that samples at the target level should

by default have unit weight regardless of the set size. We take the idea of weighting the instances in a set a step further and allow each instance in the set to have its own weight.

Optimization Another key issue in relational learning is how to optimize the parameters. In the traditional ILP, we would use A*-like search, while in the traditional propositionalization, we would use brutal-force enumeration. From related alternatives, we have to mention work from Gjorgjioski and Džeroski [89], where they used stochastic sampling to reduce the search space. From propositional algorithms, we have to mention Extreme Learning Machine (ELM) [112], which randomly and non-linearly transforms the features into a high-dimensional space to train a regularized linear model on top of the high-dimensional space, exploiting the "blessing of the dimensionality" [91]. Since ELM was reported to have good accuracy to runtime tradeoff [68], we take a similar approach by randomly generating non-linear combinations of instances in a set followed by a regularized linear model.

Neural networks Since we use a sigmoid function, which is a traditional activation function in neural networks, we would expect sigmoidal aggregate functions, like our proposal, to be used in relational neural networks. However, the two dominant approaches how to adapt a neural network to relational data is to either first extract features in a propositional form, be it with a relational rule learner [242] or a relation random walk [127], or to use recurrent neural networks [19, 253, 231].

10.3 Method

We denote vectors with bold lower-case (e.g., $\mathbf{x}$), matrices with bold upper-case (e.g., $\mathbf{X}$), and scalars with italic lower-case (e.g., k) letters. We write dot product as $\mathbf{w} \cdot \mathbf{x}$, and length of vector $\mathbf{x}$ as $|\mathbf{x}|$. In the following text, $\mathbf{X}$ is the table with attributes to aggregate and $\mathbf{x}$ is a numerical attribute in $\mathbf{X}$.

We can approximate aggregate functions of attribute $\mathbf{x}$ with a weighted sum:

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w} \cdot \mathbf{x}, \tag{10.1}$$

where $\mathbf{w}$ is a vector of the weights, $w \in [0, 1], \forall x \in \mathbf{w}$. With this equation, we can express aggregate functions like `min`, `max`, or `sum`. However, to be able to approximate aggregate functions, like `count`, `avg`, or `var`, we introduce optional normalization with variable $n, n \in \{1, |\mathbf{x}|\}$, power $p, p \in \{0, 1, 2\}$ and power $q, q \in \{1, 2\}$:

$$f(\mathbf{x}, \mathbf{w}, n, p, q) = \left(\frac{1}{n}\mathbf{w} \cdot \mathbf{x}^p\right)^q. \tag{10.2}$$

To be able to use the same weight $\mathbf{w}$ regardless of $\mathbf{x}$ length, we parametrize the weight $\mathbf{w}$ with logistic sigmoid:

$$\text{sigmoid}(\mathbf{r}, k, r_0) = \frac{1}{1 + e^{-k(\mathbf{r} - r_0)}}, \tag{10.3}$$

where $\mathbf{r}$ is a rank (`rankdata` with `ordinal` argument in SciPy) of the ordering attribute $\mathbf{o}$ linearly normalized into the range $[0, 1]$, k defines the steepness of the curve and r_0 is the $\mathbf{r}$ value of the sigmoid's midpoint. E.g.:

$$\mathbf{o} = (5, 12, 10, 10) \to \mathbf{r} = (0, 1, 1/3, 2/3). \tag{10.4}$$

When we work with static datasets, the ordering attribute $\mathbf{o}$ is traditionally equivalent to the attribute $\mathbf{x}$, while in temporal datasets, the ordering attribute $\mathbf{o}$ is traditionally the time of the sample measurement $\mathbf{t}$. But in theory, $\mathbf{o}$ can be any attribute in data $\mathbf{X}$.

The final approximate aggregate function is a combination of the weighted sum Equation (10.2) the sigmoid Equation (10.3):

$$f(\mathbf{x}, \mathbf{o}, k, r_0, n, p, q) = \left(\frac{1}{n}\,\text{sigmoid}(\text{rank}(\mathbf{o}), k, r_0) \cdot \mathbf{x}^{pT}\right)^q. \tag{10.5}$$

An example list of parameters to use to approximate common aggregate functions is in Table 10.1.

Function	k	r_0	n	p	q
`sum`	1	-100	1	1	1
`count`	1	-100	1	0	1
`avg`	1	-100	|x|	1	1
`max`	10000	0.9999	1	1	1
`min complement`	10000	0.0001	1	1	1
`avg of squares`	1	-100	|x|	2	1
`square of avg`	1	-100	|x|	1	2

Table 10.1: List of parameters to approximate common aggregate functions.

Vector $\mathbf{w}$ for `sum` should ideally be all-ones vector but we approximate it with a sigmoid, where the sigmoid's midpoint r_o is a negative number (e.g., -100). Since we use the sigmoid only in the interval [0,1], we use only the upper part of the sigmoid, which is approximately flat (see Figure 10.1). We approximate `min` aggregate with `sum` − `min complement` aggregates. And we approximate `var` with `avg of squares` − `square of avg`, following the textbook $\text{var}(x) = \text{E}[x^2] - \text{E}[x]^2$.

10.4 Experiments

We performed experiments to answer the following questions:

1. Can we approximate common aggregate functions (`min`, `max`, `avg`, `count`, `sum`, and `var`) without loss of predictive accuracy?

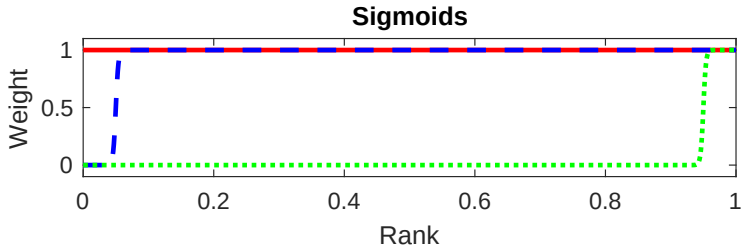

Figure 10.1: Example sigmoids, which can be used to approximate aggregates. Red: `sum`. Green: `max`. Blue: `min complement`. The midpoints and slopes of `max` and `min complement` sigmoids are less extreme than in Table 10.1 to improve legibility of the graph. We assume $\mathbf{o} = \mathbf{x}$.

2. Does the random sampling of the parameters in the approximate aggregates deliver better features than the set of predefined parameters, which simulate the common aggregate functions?
3. Does the attribute ordering by time add something new that a downstream model can utilize to improve predictive accuracy?

To answer these questions, we empirically compared accuracies of models build on 11 real (non-artificial) temporal datasets from relational repository [A.12]. From each dataset, we selected two tables. One table with the target to predict. And another table, which is in n:1 relationship to the target table and which contains numerical attributes to aggregate to the level of targets in the target table (see Table 10.2 for the setup).

Dataset	Target table	Target column	Target id	Target timestamp	Aggregate table	Classes
Accidents	nesreca	klas_nesreca	id_nesreca	cas_nesreca	oseba	B, not B
BasketballMen	teams	rank	tmID, year	year	players_teams	≤ 4, >4
BasketballWomen	teams	playoff	tmID, year	year	players_teams	N, Y
Financial	loan	status	account_id	date	trans	A, B
FNHK	pripady	Delka_hospitalizace	Identifikace_pripadu	Datum_prijeti	vykony	≤ 7, >7
Geneea	hl_hlasovani	vysledek	id_hlasovani	datum	hl_poslanec	A, R
Lahman	salaries	salary	teamID, playerID, lgID	yearID	fielding	≤ 500000, >500000
NBA	Game	ResultOfTeam1	GameId	Date	Actions	-1, 1
NCAA	target	team_id1_wins	id	season	results	0, 1
PremierLeague	Matches	ResultOfTeamHome	MatchID	MatchDate	Actions	-1, 1
Stats	users	Reputation	Id	LastAccessDate	posts	≤ 10, >10

Table 10.2: List of the used relational datasets from relational repository [A.12]. Regression and polynomial classification problems were converted to binary problems with the logic described in Classes column.

As a baseline, we aggregate the numerical attributes with the common set of aggregates. The first alternative uses approximate aggregates with a fixed set of parameters listed in Table 10.1. The purpose of this experiment is to validate the hypothesis that we can approximate the common aggregates good enough that it does not hurt the predictive accuracy of the downstream models.

The second alternative uses approximate aggregates with random parameters. We use uniform sampling for $k, k \in [0, 10000]$, $r_0, r_0 \in [-100, 1]$, $n, n \in \{1, |\mathbf{x}|\}$, power $p, p \in \{0, 1, 2\}$, and power $q, q \in \{1, 2\}$. Since we do not optimize the parameters, we substitute the quality with quantity and generate 100 aggregates per attribute rather than 6 like in the first alternative. The purpose of this experiment is to validate the hypothesis that stochastic sampling of the parameters is good enough that it does not hurt the predictive accuracy of the downstream models.

The third alternative is the same as the second alternative, but orders the attributes based on sampling time $\mathbf{t}$ (i.e., $\mathbf{o} = \mathbf{t}$), rather than the attribute $\mathbf{x}$ itself. The purpose of this experiment is to validate the hypothesis that ordering by time $\mathbf{t}$ generates useful features.

The final alternative combines 50 approximate aggregates ordered by $\mathbf{x}$ and 50 approximate aggregates ordered by $\mathbf{t}$. The purpose of this experiment is to validate the hypothesis features generated with $\mathbf{o} = \mathbf{x}$ are different from features generated with $\mathbf{o} = \mathbf{t}$.

We evaluate the quality of the generated features with 10-fold cross-validated regularized logistic regression [278] and AUC-ROC. We have chosen logistic regression because it is an additive model, which allows us to simulate `min` aggregate with `sum` and `min complement` aggregates without the need to explicitly calculate the `min`. We chose AUC-ROC because it is generally more sensitive to the changes in the prediction than thresholding measures, like classification accuracy, because AUC-ROC takes into account all possible thresholds while thresholding measures take into account only one threshold.

10.5 Results

Empirical results in Table 10.3 suggest that approximate functions can approximate the common set of aggregate functions. However, random parametrization of the weights is not a sufficient replacement for the baseline set of the aggregate functions. But once we combine $\mathbf{o} = \mathbf{x}$ features with $\mathbf{o} = \mathbf{t}$ features, we get better or equal results than the baseline on 7 datasets from 11 datasets. The remaining 4 datasets, where we did not observe an improvement (BasketballWomen, Lahman, NBA, PremiereLeague), share one common characteristic: they are all sport-related. And while we may expect temporal dependencies in sport (e.g., Autumn-born children are better at sport [230]), in all these datasets we are asked to estimate the winning team and not the performance of the players.

Statistical test A one-tailed Wilcoxon signed-rank test indicated that the models built on generalized aggregates ordered by $\mathbf{t}$ and $\mathbf{x}$ had statistically significantly higher AUC-ROC than the models built on the common aggregates with $P = 0.015625$.

Aggregates	Common	Proposed	Proposed	Proposed	Proposed
Parameters	–	Fixed	Random	Random	Random
Ordering	–	**x**	**x**	**t**	**t**, **x**
Accidents	**0.55**	**0.55**	0.52	0.52	**0.55**
BasketballMen	0.63	0.63	0.61	0.52	**0.64**
BasketballWomen	**0.61**	**0.61**	0.58	0.52	**0.61**
Financial	0.75	0.74	0.77	0.58	**0.78**
FNHK	0.65	0.65	0.64	0.66	**0.67**
Geneea	**0.71**	**0.71**	0.68	0.56	**0.71**
Lahman	**0.53**	**0.53**	0.52	0.50	**0.53**
NBA	0.55	0.55	**0.56**	0.51	**0.56**
NCAA	**0.63**	**0.63**	0.62	0.52	**0.63**
PremierLeague	**0.61**	**0.61**	0.58	0.52	**0.61**
Stats	0.58	0.58	0.60	0.62	**0.67**
Average ↑	0.618	0.617	0.607	0.548	**0.633**
#Wins and ties ↑	6	6	1	0	**11**

Table 10.3: AUC-ROC for different aggregation methods. Bold font indicates the best result for a dataset.

10.6 Discussion

In this section, we discuss how to deal with the deficiencies of our approach, namely handling of categorical attributes and missing values.

Categorical attributes can be aggregated with a sigmoid, as described by Kazemi et al. in [128].

Missing or invalid values can be replaced with safe valid values, and further indicated by additional dummy variables [280] before the aggregation.

10.7 Conclusions

In this chapter, we generalized aggregate functions commonly used in propositionalization (`min`, `max`, `avg`, `count`, `sum`, and `var`). While the generalization by itself did not turn out to improve AUC-ROC of the downstream models, it allowed us to incorporate temporal attributes in the generated features. Based on the performed experiments on 11 temporal relational datasets, the incorporation of these time-dependent features improved the average AUC-ROC. Since many relational databases include temporal attributes and many relational learners utilize the mentioned aggregate functions, the described generalization of the aggregate functions might provide the means to upgrade a time-unaware

relational learner to a time-aware relational learner. In the future, we plan to take the idea of the generalized aggregate functions and directly optimize the parameters of the generalized aggregate functions.

Chapter 11

Meta-learning

Many feature selection methods were developed in the past, but in the core, they all work the same way – you pass a set of features to the algorithm and get a reduced set of the features. But can we perform a non-trivial feature selection without first observing the features? This is an important question because if we were actually able to predict feature importance before observing the features, we would reduce computation requirements of all stages of machine learning process beginning with feature engineering. In this chapter, we argue that it is possible to predict feature importance before feature vector observation. The trick is that we use meta-features about the features to perform the feature selection. We evaluate the concept on 15 relational databases. On average, it was enough to generate the top decile of all features to get the same model accuracy as if we generated all features and passed them to the model.

11.1 Introduction

Data in relational databases are in the form of many tables, but common classification algorithms require input data in the form of a single table. Propositionalization solves this discrepancy by converting data from the form of many tables into a single table.

But there are two significant problems with the propositionalization [210]. It produces a lot of features. And many of them are redundant. These two issues result in high computational requirements during both, propositionalization and classification.

Contrary to the common approach (e.g., [145], [120], [129]), we deal with these two issues by performing feature selection *before* the propositionalization and not *after* the propositionalization. The key idea is that we collect meta-data about the attributes in the database (e.g., attribute data type), meta-data about the feature generative functions (e.g., id of the feature function), calculate landmarking features on a small subset of all features and pass their performance to a meta-learner, which predicts the optimal order, in which the remaining features should be calculated.

11.2 Related Work

The presented work is at the border between feature engineering and feature selection. Hence, we review related work from both these disciplines.

11.2.0.1 Meta-learning for Feature Engineering

Meta-learning was originally concerned with algorithm selection[219]. Nevertheless, Nargesian [188] trained a neural network to predict, which feature transformations are going to improve the accuracy of a classifier based on the feature histograms.

We extend the idea of using the data-based meta-features (in Nargesian's case a histogram) for feature engineering with landmarking.

11.2.0.2 Meta-learning for Feature Selection

Reif [216] applies meta-learning to accelerate forward selection. The key concept is that the performance of all candidate feature subsets in each forward step is first estimated with a meta-learner. And only the top x percent of the candidates get evaluated on the data to get the true subset performance. Based on the reported results, it is sufficient to evaluate only the top 10% of all candidate subsets on the data to get results comparable to classical forward selection.

The difference between our approach and Reif's approach is that Reif calculates meta-features from the features, while we calculate meta-features directly from the attributes that are used to calculate the features (in Figure 11.1 we use only the left table, while Reif uses the right table). Consequently, in Reif's case, we have to calculate the features first, to perform feature selection. While in our case, we can perform the feature selection before feature calculation.

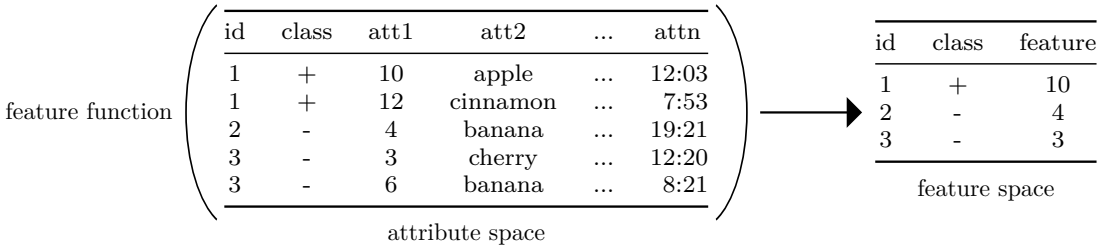

Figure 11.1: An example of a feature generative function *min* applied on attribute *att1*, which converts the multi-instance problem into a single-instance problem solvable with a common attribute value classifier. In this trivial example, the feature space contains only a single feature vector but it may generally contain thousands of feature vectors.

11.3 Method

A high-level schema of our approach is in Figure 11.2. The whole process is divided into two phases. During the offline phase, meta-features and feature performance are collected on many databases and passed to a meta-learner as training data. During the online phase, the trained meta-learner is used to rank candidate features in the descending order of their estimated utility. Following paragraphs define the *feature utility*.

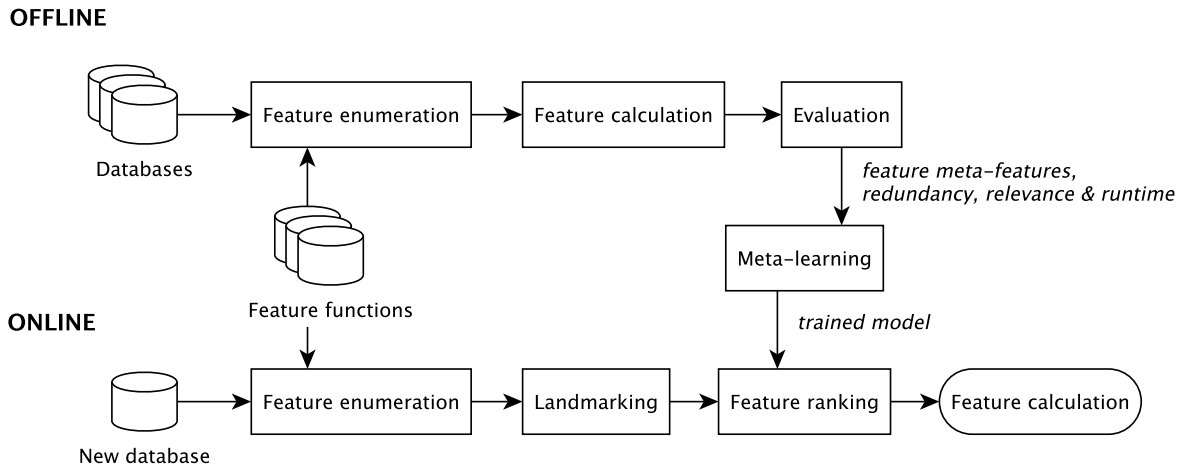

Figure 11.2: Flowchart of meta-learning on features.

There are many properties that a feature should posses [175], but we focus on predicting properties measurable directly from the data: relevance to the task, redundancy to other features and runtime of the feature calculation.

Relevance Without loss of generality, we assume that we want to utilize the calculated features for classification. We use $Chi^2 statistics$ [58, Section A.6.1] between the feature and the label as the measure of the relevance (if the feature is continuous, we first discretize the feature with equal-width binning). But in theory, any other measure can be used.

Runtime The runtime is defined as the time needed to calculate a particular feature vector. If two feature vectors are otherwise identical, we prefer the one that has a smaller runtime.

Redundancy In the analyzed databases (discussed further in chapter 11.4.0.1), 38% of all calculated features are redundant. We define that nominal feature f_1 is *redundant* to nominal feature f_2 iff a bijection exists between values in f_1 and f_2. A numerical feature

f_1 is redundant to numerical feature f_2 if a linear transformation from f_1 to f_2 and back exists.

We use this (weaker) definition of redundancy instead of the *identity* of the features because it corresponds better with the notion of redundancy in many models (e.g., in logistic regression with one shot encoding of categorical features). To speed up the identification of redundant features, we use Chi^2 as a hash function to identify potential redundant features [204, Section 2.1].

11.3.0.1 Feature Utility

We calculate features[1] in descending order of the estimated *relevance*/*runtime* ratio [1] since we prefer to calculate highly relevant and fast features first. Furthermore, we penalize the feature i proportionally to the estimated probability that the feature is redundant $\hat{p}_i$. Because each dataset has a different proportion of redundant features (see Table 11.1) and the tested meta-learning models had difficulties to model these differences, we employ median thresholding instead of a fixed threshold:

$$utility_i = (\hat{p}_i > median(\hat{p}) \; ? \; 1 - \hat{p}_i \; : \; 1)\frac{relevance_i}{runtime_i}, \tag{11.1}$$

where *redundancy* is a vector of estimated redundancy probabilities for a database.

11.4 Experiment

11.4.0.1 Data

We used 15 databases listed in Table 11.1 from relational repository [A.12].

11.4.0.2 Features

For propositionalization, we used Relaggs [144], which was modified to work with 31 different feature (generative) functions, listed in Figure 11.2. The detail description of the employed feature functions is at `http://predictorfactory.com`.

11.4.0.3 Meta-features

We employ three sources of meta-features: landmarking features, database meta-data and feature function meta-data.

[1] In the production, we would calculate only the top n features that we would use to build a production classifier. But to demonstrate the meaningfulness of such approach, we calculate all features.

Table 11.1: Used databases. The range of relevant features is estimated with forward & backward selection with a decision tree (the percentage of features when meta-learning feature selection reaches accuracy corresponding to accuracy obtained on all the features).

Database	Domain	Attributes	Features	Redundant [%]	Relevant [%]
Accidents	Government	43	305	39	2–12 (7)
AustralianFootball	Sport	77	794	45	1–6 (1)
BasketballMen	Sport	195	865	41	1–49 (1)
Biodegradability	Medicine	17	71	25	6–66 (4)
Chess	Sport	45	127	16	65–72 (91)
Financial	Financial	55	493	32	1–59 (7)
Hepatitis	Medicine	26	152	42	4–42 (5)
Mondial	Geography	167	1524	45	1–9 (1)
Mutagenesis	Medicine	14	65	40	6–46 (6)
Nations	Geography	118	191	76	2–21 (3)
PremierLeague	Sport	217	667	23	2–27 (7)
PTE	Medicine	76	691	58	1–33 (1)
StudentLoan	Education	15	41	7	15–66 (21)
VisualGenome	Education	20	42	64	10–10 (14)
Walmart	Retail	27	545	19	1–22 (4)
average		74	438	38	8–36 (11)

Landmarking features Just like the accuracy of a few classifiers can be used as meta-features for the recommendation of the best classifier on the data (e.g., [202]), we define a subset of feature functions as landmarking feature functions for the recommendation of the best features.

Without loss of generality, we used following set of landmarking features: Direct field (a simple copy of the value), Aggregate (e.g., *min*, *max*,...), WOE (Weight of Evidence), Count (of tuples), Aggregate WOE, Time aggregate since. These feature functions were selected for their low runtime (see Table 11.3) and good coverage of different data types (numerical/character/temporal) and relationships between the label and the data (1:1/1:n). Note that we do not use multivariate feature functions for landmarking due to the potential combinatorial explosion.

Database meta-data Basic descriptive and statistical meta-features are frequently employed in meta-learning (e.g., [160]) and we do not differ in this respect. A noteworthy difference is that we do not calculate statistics of the attributes but rather reuse statistics maintained by the relational database for query plan optimization [A.8]. This slight deviation allows us to collect estimates of the statistics in time independent on the count of tuples (records) in the database.

	Univariate		Multivariate
1:1	Direct field (any) Text length (c) Time day part (t) Time is weekend (t) Time part (t) Time since (t) WOE (c)		Time diff (t+t)
1:n	Aggregate (n) Aggregate distinct (n) Aggregate range (n) Aggregate text length (c) Aggregate WOE (c) Coefficient of variation (n) Count (any) Distinct count (any) Duplicate ratio (any)	Existential count (any) Log product (n) Null ratio (any) Time aggregate (t) Time aggregate since (t) Time aggregate since event (t) Time frequency (t) Time range (t) Time WOE (t)	Aggregate frame (n+t) Correlation (n+t) Intercept (n+t) Slope (n+t) Time aggregate diff (t+t)

Table 11.2: Taxonomy of feature functions (data type they work on: c-character, n-numeric, t-temporal). The horizontal axis differentiates between feature functions working on a single attribute and multiple attributes. The vertical axis differentiate between feature functions working on a single tuple and multiple tuples.

Feature function meta-data Feature function meta-data consists of feature function name (e.g., Aggregate) and feature function parameters (e.g., *min*).

11.4.0.4 Measures

Anytime algorithm We formulate feature engineering as anytime algorithm [277], which aims to deliver the best subset of calculated features in any time. The quality of anytime algorithm can be expressed with a performance profile, where we measure quality of the solution at the given time (see example in Figure 11.3). To assign a single number to the performance profile, we calculate the area between the *archived* curve $a(t)$ and the expected *random* curve $r(t)$ (which we obtain from averaging the curve from many random permutations), divided by the area between the *perfect* curve $p(t)$ and the expected random curve $r(t)$:

$$POP = \frac{\int a(t)dt - \int r(t)dt}{\int p(t)dt - \int r(t)dt}, \tag{11.2}$$

where t is time. The obtained ratio then represents the "percentage of perfect" solution [22]. In our case, $a(t)$, $r(t)$ and $p(t)$ are the Chi^2 of the feature calculated at time t. The only difference between these functions is then the order, in which the features are calculated. The perfect feature ordering is based on a complete knowledge of relevance, redundancy and runtime of all the features. While archived ordering is based only on

Feature function	Relevance	Runtime	Redundancy	Utility
Aggregate frame	−2.11	−0.17	0.49	−2.19
Time aggregate diff	−1.53	0.05	0.54	−1.95
Time diff	−0.92	−0.01	0.48	−1.34
Time day part	−1.33	0.02	0.02	−1.31
Time since	−0.89	−0.04	0.45	−1.22
Null ratio	−0.99	−0.02	0.18	−1.06
Time frequency	−0.17	0.35	−0.10	−0.71
Existential count	−0.67	−0.06	0.06	−0.47
Slope	−0.49	0.03	0.03	−0.47
Time WOE	0.51	0.38	0.07	−0.42
Time is weekend	−1.03	−0.12	−0.21	−0.40
Time part	−0.45	0.00	0.05	−0.40
Text length	−0.76	−0.07	−0.09	−0.37
Intercept	1.42	0.36	0.37	−0.21
Correlation	1.32	0.26	0.37	−0.05
Time aggregate	0.57	0.05	0.21	0.19
Aggregate text length	−0.24	−0.05	−0.15	0.29
Aggregate range	0.34	−0.02	0.15	0.34
Duplicate ratio	0.32	0.17	−0.26	0.39
Aggregate distinct	0.56	0.03	0.11	0.44
Time range	0.06	0.06	−0.26	0.45
Coefficient of variation	0.36	0.01	−0.07	0.62
Time aggregate since event	0.19	0.01	−0.24	0.75
Direct field	0.26	−0.05	−0.09	0.79
Distinct count	0.21	−0.04	−0.16	0.82
Aggregate	0.49	0.04	−0.16	0.82
Log product	0.62	−0.02	0.02	0.87
Time aggregate since	1.25	0.27	−0.24	0.95
Count	0.30	−0.08	−0.18	1.13
WOE	1.27	−0.03	−0.01	1.69
Aggregate WOE	1.04	0.01	−0.39	2.05

Table 11.3: Expected standardized relevance (bigger is better), runtime (smaller is better) and redundancy (smaller is better) of feature functions (sorted by the feature utility).

the estimates of these feature properties (the only exception are landmarking features, which are calculated in a pseudorandom order).

Individual models To assess the ability of relevance and runtime prediction models to rank, we use Spearman correlation (ρ). The quality of redundancy estimation (a

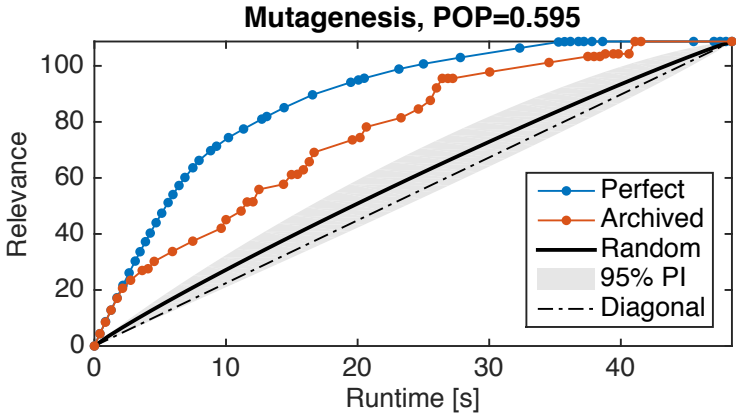

Figure 11.3: Performance profile. The shaded area represents the 95% prediction interval for a random curve.

classification task) is evaluated with area under receiver operating characteristic curve (AUROC).

11.4.0.5 Methodology

Algorithms were evaluated with leave-one-out validation, where all but the tested database was used for the training of the models. Obtained accuracies are reported for three different algorithms: *generalized linear model* (GLM), *gradient boosting machine* (GBM) and *deep learning* (DL), all from H2O.

Permutation testing To assess, whether the obtained performance profiles are significantly better than random, we generate 1 000 random orderings of the features to estimate 95% prediction intervals.

11.5 Results

First, we report the accuracy of the individual models. Second, we comment on the meta-feature importance as reported by L1 & L2 regularized GLM. Third, we report the obtained POPs.

11.5.0.1 Accuracy

The obtained accuracies are depicted in Table 11.4. Since the difference between the models is not significant, we use GLM for all following experiments.

Algorithm	Relevance [ρ]	Runtime [ρ]	Redundancy [AUROC]
DL	**0.558** $\pm$ 0.255	0.302 $\pm$ 0.186	0.798 $\pm$ 0.097
GBT	0.556 $\pm$ 0.252	0.206 $\pm$ 0.259	0.787 $\pm$ 0.106
GLM	0.551 $\pm$ 0.276	**0.369** $\pm$ 0.267	**0.810** $\pm$ 0.102

Table 11.4: Leave-one-out accuracy of individual models.

11.5.0.2 Feature Importance

Relevance The most important meta-features for feature relevance prediction is the average relevance of the landmarking features on individual attributes and the type of the employed feature function (see Table 11.5).

Meta-feature	Comment	Weight
landmark_relevance	average on the attribute	9.81
feature_function	e.g., null_ratio is inferior to direct field	5.58
feature_parameters	e.g., aggregate=min is inferior to aggregate=avg	1.90
data_type	e.g., enums are superior to datetimes	0.34
avg_length	extremely long attributes like text are subpar	0.25
is_primary_key	surrogate primary keys make inferior features	0.10

Table 11.5: Meta-features for relevance prediction.

Redundancy There are two main sources of redundant features [145]: redundancy in the input data and redundancy introduced by the feature functions. The redundancy in the input data is covered by landmarking *landmark_is_redundant* and *data_type*. While the introduced redundancy is explained with *feature_function* and *feature_parameters* (see Table 11.6).

Meta-feature	Comment	Weight
landmark_redundancy	average on the attribute	16.70
feature_parameters	e.g., aggregate=min is inferior to aggregate=avg	13.65
feature_function	e.g., null_ratio is inferior to count	2.61
data_type	e.g., integers are inferior to doubles	0.02

Table 11.6: Meta-features for redundancy prediction.

Runtime The runtime of a feature function calculation is a function of two factors: the type of the feature function and data property. Nevertheless, these two factors are dominated by the landmarking *landmark_runtime* (see Table 11.7).

Meta-feature	Comment	Weight
landmark_runtime	average on the attribute	5.00
feature_function	complicated features take more time	3.34
table_rows	more data means higher runtime	0.10

Table 11.7: Meta-features for runtime prediction.

11.5.0.3 Percentage of Perfect

The quality of anytime learning for all 15 datasets is reported in Table 11.8 in the penultimate column.

redundancy	0	1	0	0	1	1	0	1	
relevance	0	0	1	0	1	0	1	1	
runtime	0	0	0	1	0	1	1	1	PI
Accidents	0.11	0.45	0.61	−0.18	**0.68**	0.44	0.60	0.67	0.33
AustralianFootball	0.03	0.33	0.35	−0.09	0.40	0.33	0.35	**0.40**	0.36
BasketballMen	0.08	0.53	−0.52	0.28	0.62	0.56	−0.37	**0.62**	0.11
Biodegradability	−0.18	0.31	−0.44	0.24	**0.34**	−0.21	−0.32	0.32	0.31
Chess	0.05	0.46	0.72	0.31	0.71	0.80	**0.90**	0.87	0.29
Financial	−0.01	0.24	−0.02	−0.01	0.30	0.29	0.02	**0.35**	0.32
Hepatitis	0.07	0.37	−0.01	0.03	0.34	**0.48**	0.04	0.37	0.28
Mondial	−0.00	0.19	0.30	−0.09	0.32	0.27	0.26	**0.32**	0.11
Mutagenesis	0.05	0.14	0.09	0.20	0.24	**0.63**	0.62	0.59	0.22
Nations	0.16	0.59	0.87	0.18	0.75	0.79	**0.88**	0.76	0.34
PremierLeague	−0.07	0.12	0.27	0.08	0.17	0.35	**0.35**	0.34	0.17
PTE	0.01	0.39	0.32	−0.43	0.54	0.31	0.24	**0.53**	0.20
StudentLoan	0.25	0.17	0.62	0.14	0.61	0.53	**0.65**	0.61	0.39
VisualGenome	−0.11	0.44	0.95	−0.03	0.95	0.74	**0.96**	0.94	0.82
Walmart	−0.18	0.20	0.77	−0.06	0.49	0.17	**0.81**	0.52	0.42
average	0.02	0.33	0.32	0.04	0.50	0.43	0.40	**0.55**	0.31
win count	0	0	0	0	2	2	**6**	5	0

Table 11.8: POPs for all databases based on the used individual models. PI column contains the upper 95% prediction interval of POPs for random ordering of the features. The best values are in bold.

11.6 Discussion

11.6.0.1 What is the contribution of the individual models to POP?

To evaluate the contribution of the individual models to POP, we performed an experiment with a 2-level full factorial design for presence/absence of runtime, relevance and redundancy models (8 combinations in total) on all databases. To deal with the variability across databases (some are easier than others), we treat the database name as a random factor.

Conclusion: The result of the factor analysis is in Table 11.9. As expected, the intercept is not significantly different from zero, since POP measure should on average be 0 when we randomly rank the features. The biggest contributions to the accuracy are from redundancy and relevance prediction. The interaction between redundancy and relevance has a negative estimate because we do not reward calculation of redundant features even if they are highly relevant. Hence, prediction of the relevance helps only on the subset of unique features from the set of all candidate features.

	Estimate	Std. Error	Pr(> |t|)	
(Intercept)	0.020	0.058 6	0.687 7	
relevance	0.292	0.014 9	$4.052\,7 \times 10^{-5}$	***
redundance	0.318	0.014 9	$2.909\,5 \times 10^{-5}$	***
runtime	0.053	0.012 2	0.012 4	*
rel:red	−0.172 3	0.024 4	0.002 1	**

Table 11.9: Contribution of models to POP. Adjusted R^2: 0.564.

11.6.0.2 What is the effect of meta-learning on model accuracy?

To evaluate the effectivity of the meta-learning, we iteratively train a classification model on increasing percentage of the top features, as estimated with meta-learning. As the classification model, we use a decision tree because it can model interactions between the features, it is undemanding on data preprocessing and it is reasonably fast. As the evaluation measure, we use misclassification error as all databases have reasonably balanced classes in the label.

An example of the obtained curve is depicted in Figure 11.4, where we can observe that the decision tree slightly overfits when we use all the features. Nevertheless, forward selection still outperforms meta-learning feature selection, as it can observe all the features (our approach does not) and it is a wrapper (our approach is a filter [101]).

Conclusion: The result of the factor analysis is in Table 11.10. Prediction of relevance significantly reduces misclassification error. Prediction of runtime insignificantly increases the misclassification error, because this evaluation does not reward fast features. Redundancy prediction does not significantly decrease the classification error. Based on

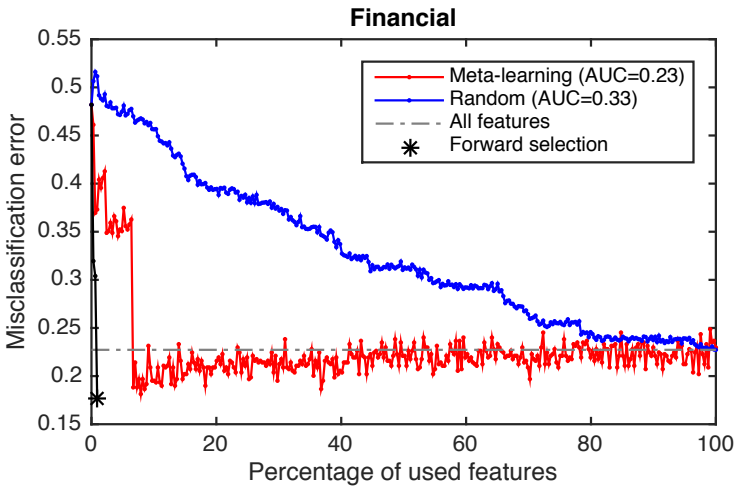

Figure 11.4: Area under misclassification error. Smaller is better.

our inspection of the results, this is because this evaluation rewards early discovery of a few highly relevant features much higher (since the best possible decision tree may use just a few features) than it penalizes redundancy (a redundant feature only pushes all subsequent features one step later).

	Estimate	Std. Error	Pr(> |t|)	
(Intercept)	0.294	0.020	$1.335\,7 \times 10^{-5}$	***
relevance	−0.053	0.016	0.001 6	**
redundance	−0.018	0.014	0.240 2	
runtime	0.010	0.014	0.516 7	

Table 11.10: Contribution of models to reduction of the area under misclassification error curve. Adjusted R^2: 0.486.

11.6.0.3 Which meta-features are important?

To analyze the importance of the three categories of the meta-features (*landmarking*, *database*, *feature-function*), we design an experiment, in which we vary the set of the used meta-features.

Conclusion: Based on the results reported in Table 11.11, only landmarking meta-features help to significantly[2] reduce the count of features that have to be engineered to reach model accuracy obtained on all features. Table 11.11 also tells us that if all meta-

[2]The reported p-values do not incorporate correction for repeated evaluation of serially correlated observations

	Estimate	Std. Error	Pr(> |t|)	
(Intercept)	43.328	11.569	0.001 3	**
database	−0.741	11.007	0.947 2	
featureFunction	−3.755	11.007	0.737 7	
landmarking	−30.586	11.007	0.014 0	*

Table 11.11: Contribution of meta-feature categories to the reduction of the count of engineered features needed to reach or surpass model accuracy obtained on a complete set of features. Adjusted R^2: 0.308.

features are used, it is in average sufficient to engineer only the top 8.25% of the features to match or surpass the classification accuracy of the model trained on all features.

11.6.0.4 Do we need so many feature functions?

We may wonder whether it is not enough to just engineer the 6 *landmarking features* and do not continue with the engineering of the remaining 25 (e.g., multivariate) features. We compared accuracies of the models trained only on the landmarking features with accuracies obtained on all features. Based on Wilcoxon signed-rank test, we have to reject the null hypothesis that the additional features do not improve accuracy (p-value = 0.00048). The median improvement is 1.2 percent point in classification accuracy (average improvement is 2.7 percent point).

Conclusion: The additional features improve the accuracy of the model over the accuracy of the model build only on the landmarking features by a small but significant amount.

11.6.0.5 Feature Selection vs. Feature Meta-learning

The described feature meta-learning bears similarity with filter-type feature selection methods like *Correlated Feature Selection* (CFS)[102] and *Minimum Redundancy Maximum Relevance* (mRMR)[199]. Both these methods attempt to quickly select relevant non-redundant features. And so does our method. But in comparison to these methods, we perform feature selection before the feature engineering.

Difficulty It can be argued that feature meta-learning is at least as difficult problem as feature selection since we can always convert feature selection problem to feature meta-learning by throwing away the computed features (and recalculating them on request).

11.6.0.6 Limitations

We performed experiments only on relational data and features from propositionalization. Propositionalization is known to produce a lot of duplicate features (38% on average on

the tested databases) and many of the features are irrelevant to the task (64% on average on the tested databases based on backward selection). These properties make it possible to obtain substantial gains from feature selection. However, the performed experiments do not tell us how the described approach is going to generalize on non-relational data.

Another limitation of the reported work is that it ignores interactions between the feature vectors in the downstream model. This can reduce the accuracy of the downstream model because a univariate oraculum meta-learner would not recommend calculation of features that are useful only in the combination with other features (a trivial example where this may happen is XOR problem [179]).

11.6.0.7 Applications

Feature meta-learning is desirable in domains, where a single universal approach to feature extraction does not exist or is not known ahead. An exemplary domain are relational data, which may contain highly diverse content ranging from structured to unstructured data.

Additionally, feature selection before feature engineering is applicable to complex or large data, where it is not feasible or convenient to calculate and evaluate all possible features due to limited resources.

11.7 Conclusions

In this chapter, we evaluated an idea of performing feature selection *before* feature engineering. To guide the search, we exploited *meta-learning*. Nargesian [188] used meta-features calculated from the original data. But we found out that landmarking meta-features work better. When we evaluated the implementation on 15 databases, we concluded that it is on average enough to engineer only *the top decile* of features to get accuracy comparable to accuracy obtained on all features. This finding is similar to Reif's [216] finding, who applied meta-learning to feature selection. However, Reif performs feature selection *after* feature engineering while we perform feature selection *before* feature engineering. The used code is published at `https://github.com/janmotl/metalearning`.

Chapter 12

Learning on Stream of Features

12.1 Online Random Forest

We study an interesting and challenging problem, supervised learning on a stream of features, in which the size of the feature set is unknown, and not all features are available for learning while leaving the number of observations constant. In this problem, the features arrive one at a time, and the learner's task is to train a model equivalent to a model trained from "scratch". When a new feature is inserted into the training set, a new set of trees is trained and added into the current forest. However, it is desirable to correct the selection bias: older features has more opportunities to get selected into trees than the new features. We combat the selection bias by adjusting the feature selection distribution. However, while this correction improves accuracy of the random forest, it may require training of many new trees. In order to keep the count of the new trees small, we furthermore put more weight on more recent trees than on the old trees.

Problem formulation One of the common issues in machine learning is changing data and the need to keep the machine learning models up to date with the changing data. One of the successful simplifications is to assume that over time we are getting new samples. However, we concerned with the orthogonal problem – fast updates of models when new features arrive (see Figure 12.1).

Motivation Our original need for learning on a stream of features was due to our interest into propositionalization [142]. Propositionalization is a data preprocessing step, which converts relational data into a single data. And one of the persistent problems of propositionalization is that it generates a wast quantity of redundant and/or unpredictive features (e.g.: [142, 124]). Would not it be interesting to intelligently guide the propositionalization in order to avoid wasteful generation of these irrelevant features? Our previous research [A.7] answered this question positively – based on *univariate feature selection* on a stream of features, we obtained 10-fold acceleration of the propositionalization (while maintaining the accuracy of the downstream model comparable to accuracy

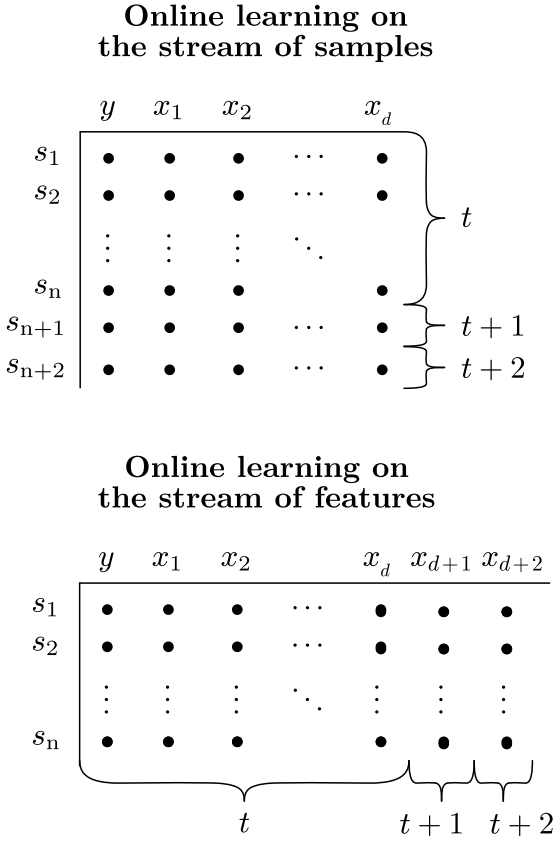

Figure 12.1: The difference between learning from a stream of samples (top) and a stream of features (bottom). In both cases, we have n samples and d features at time t. But at time $t + 1$, we either have one more sample (top) or one more feature (bottom).

obtained on exhaustive propositionalization). However, our former research had evident weakness: it neglected to take into account possible interactions between features. This chapter attempts to address this weakness.

Why not a feature selection filter? Features that are currently unpredictive may become predictive, as new features appear. For example, consider XOR problem, in which the binary label is determined by two binary features f_1 and f_2: $y = xor(f_1, f_2)$. Features f_1 and f_2 are individually unpredictive. But together, they define the label. Univariate feature selection filters (e.g.: based on information gain ratio) cannot correctly identify the change or the first feature relevance as the second feature is added in XOR problem. But models capable of modeling feature interactions (like random forests) can eventually identify these features as important.

Application Beside propositionalization, learning on a stream of features has another interesting use-case: Kaggle competitions. In these challenges, competitors are given a

dataset and the team with the best model wins[1]. Based on the analysis of solutions of the past winners[2], one of the common differentiating factors is extensive feature engineering. However, competitive feature engineering is generally not a one-time task but rather an iterative process:

1. formulate a hypothesis (e.g.: log transformation of features will improve the accuracy of the downstream model),
2. test the hypothesis (e.g.: evaluate the change of accuracy of the downstream model),

where the choice of the next round of hypotheses is influenced based on the success of the previously evaluated hypotheses. Traditionally, the evaluation of the hypothesis required retraining of the model from scratch. Our solution is to update the current model. The benefit is evident: the update of the current model takes less time than retraining the model from scratch. And consequently, that gives us the freedom to test more hypotheses.

Random forest We take random forest [23] as a starting model to expand into an online implementation because it can deal with dirty data (e.g.: missing values, outliers, mix of numerical and nominal attributes,...) and given an implementation of a decision tree, it is easy to implement and reason about.

The key idea behind random forest classifier is that we make an ensemble of decision trees. In order to create diversity between the trees, it employs two strategies: bagging and random feature selection. Bagging is based on a random sampling of training instances with repetition. While random feature selection is without repetition. The count of features to select is one of the most tunable parameters of random forests [206] and multiple heuristics for the optimal value were provided in the literature. For simplicity of the following analysis, we assume that the count of the selected features is a fixed ratio of the count of all the features. We call the ratio *mtry*.

12.1.1 Implementation

Bias Whenever a new feature x_{new} arrives, we may train n new trees. And add the newly trained trees into the current random forest. Unfortunately, with this approach, the new features would be underrepresented in the forest in comparison to old features simply because the *old features had multiple opportunities* to get used in a tree while the *new feature had only one opportunity* to get used in a tree.

Consequently, earlier features would have a bigger impact (weight) in the forest than the newer features. This presents a bias, which is generally undesirable.

[1] See a list of all possible challenges at https://www.kaggle.com/competitions.
[2] See: http://blog.kaggle.com/category/winners-interviews/.

Variable count of trees The first intuitive improvement is to make sure that the new feature is actually always passed to the new trees (instead of leaving it on the chance). And instead of generating an arbitrary count of the trees, we can calculate the optimal count n that minimizes the random feature selection bias.

First, we introduce the notation. Let c be the count of how many times a feature x was passed to decision trees. And let *old* subscript describe some old feature and *new* subscript to describe the new feature. If we want to avoid the random feature selection bias, following should hold:

$$c_{new} = c_{old}. \tag{12.1}$$

Since

$$c_{new} = n \tag{12.2}$$

because the new feature is always selected and

$$c_{old} = mtry \cdot d_{old} + mtry \cdot n, \tag{12.3}$$

where d_{old} is the count of the old features, we get:

$$n = mtry \cdot d_{old} + mtry \cdot n. \tag{12.4}$$

Hence, we get the optimal n with:

$$n = \frac{mtry \cdot d_{old}}{1 - mtry}. \tag{12.5}$$

The issue with this approach is that if we keep adding d features one-by-one, the total count of the trees in the ensemble grows quadratically.

Tree weighting If we want to avoid the quadratic growth of the random forest, we may weight the late trees more than the former trees. While we could have calculated the tree weight analytically, we provide an algorithmic solution in Algorithm 1. In praxis, the advantage of the algorithmic solution is that it is self-correcting – if some of the assumptions are not fully fulfilled (e.g.: When we have 11 features and the feature selection ratio is 0.5, we can either select 5 or 6 features but not 5.5.), the error is not ignored (as it would be in a closed-form analytical solution) but is encoded in `weightedFeatureUseCount`. And each call of Algorithm 1 directly minimizes the error.

When scoring new samples, we evaluate trees in the ensemble and calculate the weighted average of the predictions (each generation of trees share the same treeWeight).

Algorithm 1: Random forest update, when a new feature arrives. Function `featureCnt()` returns count of features to sample.

Input: X: training data, y: training label, *col*: index of the new feature, *treeCnt*: cnt of trees to train, $weightedFeatureUseCnt$: bookkeeping vector initialized to zeros, *ensemble*: collection of trees.

Output: *ensemble*, *treeWeight*, $weightedFeatureUseCnt$.

```
featureUseCnt = zeros (col);
for i=1:treeCnt do
    oldFeatures = choice (1:col-1, featureCnt (col-1), replacement=False);
    features = [oldFeatures, col];
    samples = choice (nrow (x), nrow (x), replacement=True);
    tree = fitTree (X[samples, features], y[samples]);
    ensemble = [ensemble, tree];
    featureUseCnt[features]++ ;
end
treeWeight = avg (weightedFeatureUseCnt[1:col-1]) / (featureUseCnt[col] -
  avg (featureUseCnt[1:col-1]));
weightedFeatureUseCnt = weightedFeatureUseCnt +
  treeWeight*featureUseCnt;
```

12.1.2 Experiments

We compare two online random forest implementations: **baseline** and **challenger**. In baseline, features are selected with uniform probability (like in ordinary random forest). In the challenger model, the new feature is always selected while the old features are selected with uniform probability[3]. Furthermore, we train an **offline** random forest with the same meta-parameters as the online random forest in order to depict the value of the online learning.

Protocol For each data set, we performed the following procedure 10 times: We randomly split the data set into training/testing subsets with stratified sampling with 2:1 ratio. Then we randomly permutate the feature order in the data set (because our proposal should work regardless of the feature ordering). Finally, on online random forests we perform incremental learning feature-by-feature (i.e.: first we train the random forest on the first feature, then we add the second feature into the forest ... and continue until the last feature is added into the forest). After adding the last feature, the final model is evaluated on the testing set with AUC (Area Under the Receiver Operating Characteristics). In the case of the offline random forest, we train the random forest just once on all the features.

[3]This probability is smaller in the challenger model than in the baseline model in order to keep the final count of features in challengers' trees identical to the count of features in baselines' trees.

Meta-parameters At each generation (addition of a new feature), we train 30 new trees. This value is recommended by Breiman [23] and we decided to go with it. For feature selection ratio, we used $2/3$.

Data sets We used all 232 data sets (see Table 12.1) from OpenML [254] that have a binary label (because we evaluate the models with AUC), less than 200000 samples (because of runtime) and less than 15 features (again, because of the runtime).

2dplanes	biomed	echoMonths	house_8L	rabe_166
abalone	blogger	ecoli	houses	rabe_176
acute-inflammations	blood-transfusion	electricity	humandevel	rabe_265
aids	BNG(breast-w)	elusage	hungarian	rabe_266
Amazon_employee_access	BNG(tic-tac-toe)	fertility	hutsof99_logis	rabe_97
analcatdata_apnea1	bolts	fishcatch	ilpd	rmftsa_ctoarrivals
analcatdata_apnea2	boston	fri_c0_100_10	iris	rmftsa_ladata
analcatdata_apnea3	braziltourism	fri_c0_100_5	irish	rmftsa_sleepdata
analcatdata_asbestos	breast-cancer	fri_c0_1000_10	jEdit_4.0_4.2	Run_or_walk_information
analcatdata_bankruptcy	breast-cancer-dropped	fri_c0_1000_5	jEdit_4.2_4.3	sa-heart
analcatdata_birthday	breast-w	fri_c0_250_10	kdd_el_nino-small	schlvote
analcatdata_bondrate	breastTumor	fri_c0_250_5	kidney	sensory
analcatdata_boxing1	bridges	fri_c0_500_10	kin8nm	servo
analcatdata_boxing2	car	fri_c0_500_5	lowbwt	sleep
analcatdata_broadway	cars	fri_c1_100_10	lupus	sleuth_case1102
analcatdata_broadwaymult	chatfield_4	fri_c1_100_5	machine_cpu	sleuth_case1201
analcatdata_challenger	cholesterol	fri_c1_1000_10	MagicTelescope	sleuth_case1202
analcatdata_chlamydia	chscase_adopt	fri_c1_1000_5	mammography	sleuth_case2002
analcatdata_creditscore	chscase_census2	fri_c1_250_10	mbagrade	sleuth_ex1221
analcatdata_cyyoung8092	chscase_census3	fri_c1_250_5	mfeat-morphological	sleuth_ex1605
analcatdata_cyyoung9302	chscase_census4	fri_c1_500_10	mofn-3-7-10	sleuth_ex1714
analcatdata_dmft	chscase_census5	fri_c1_500_5	monks-problems-1	sleuth_ex2015
analcatdata_draft	chscase_census6	fri_c2_100_10	monks-problems-2	sleuth_ex2016
analcatdata_fraud	chscase_funds	fri_c2_100_5	monks-problems-3	socmob
analcatdata_germangss	chscase_geyser1	fri_c2_1000_10	mozilla4	solar-flare
analcatdata_gsssexsurvey	chscase_health	fri_c2_1000_5	mu284	space_ga
analcatdata_gviolence	chscase_vine1	fri_c2_250_10	mux6	stock
analcatdata_japansolvent	chscase_vine2	fri_c2_250_5	mv	strikes
analcatdata_lawsuit	chscase_whale	fri_c2_500_10	newton_hema	tae
analcatdata_michiganacc	cleve	fri_c2_500_5	no2	threeOf9
analcatdata_neavote	cleveland	fri_c3_100_10	nursery	tic-tac-toe
analcatdata_negotiation	Click_prediction_small	fri_c3_100_5	page-blocks	Titanic
analcatdata_olympic2000	cloud	fri_c3_1000_10	parity5	transplant
analcatdata_reviewer	cm1_req	fri_c3_1000_5	parity5_plus_5	vertebra-column
analcatdata_runshoes	cmc	fri_c3_250_10	pc1_req	veteran
arsenic-female-bladder	datatrieve	fri_c4_250_10	pollen	visualizing_hamster
arsenic-female-lung	delta_ailerons	fri_c4_500_10	postoperative-patient-data	visualizing_livestock
arsenic-male-bladder	delta_elevators	fried	prnn_crabs	visualizing_slope
arsenic-male-lung	diabetes	fruitfly	prnn_fglass	visualizing_soil
autoMpg	diabetes_numeric	glass	prnn_synth	vowel
badges2	diggle_table_a1	grub-damage	profb	wholesale-customers
balance-scale	diggle_table_a2	haberman	puma8NH	wilt
balloon	disclosure_x_bias	hayes-roth	pwLinear	wine
banana	disclosure_x_noise	heart-c	quake	witmer_census_1980
bank8FM	disclosure_x_tampered	heart-h	qualitative-bankruptcy	
banknote-authentication	disclosure_z	heart-statlog	rabe_131	
baskball	dresses-sales	hip	rabe_148	

Table 12.1: List of used data sets.

Results In 87% (201/232), the challenger model had higher average testing AUC than the baseline. Sign test on this statistic gives one-tail P-value $< 10^{-29}$. The average difference of the testing AUC across all the data sets was 2.10 percent point. Furthermore, in 71% (164/232), the challenger model had higher average testing AUC than the offline model (P-value $< 10^{-8}$). The table with the results and the code that generated the table is available from https://github.com/janmotl/rf.

12.1.3 Discussion

Overhead Challenger model, in comparison to baseline model, uses 3 more variables: `featureUseCount`, `weightedFeatureUseCount` and `treeWeight`. Each of these variables is (or fill in) a vector of length d, the count of features. Ignoring the differences in the data types, the total memory overhead is equivalent to 3 more training data samples. The computational complexity of updating these 3 variables, when a new feature is added, is $\mathcal{O}(d)$ since `treeCount` is a constant.

Limitation Our experiment suffers from one limitation: while we make sure that the feature selection rate is uniform, we ignore interactions between the features. This could be a topic of further research.

Extension One of possible extensions of our work, which we did not pursue further, is pruning of the oldest trees from the ensemble. The idea is simple: the older generations of the trees have so small weight, that they hardly influence the final prediction.

12.1.4 Conclusions

We have extended random forest to work on a stream of features. The idea was simple: when a new feature arrives, extend the forest with a new set of trees. However, with this strategy, older features end up used more frequently than the new features. When we fix this feature selection bias, it improves the testing AUC on average by 2 percent points. The proposed algorithm for feature selection bias correction is fast, easy to implement and robust. The code was open-sourced at https://github.com/janmotl/rf.

12.2 Online Discriminant Analysis

Online learning is a well-established problem in machine learning. But while online learning is commonly concerned with learning on a stream of samples, this chapter is concerned with learning on a stream on features. We propose to modify quadratic discriminant analysis (QDA) to work on a stream of features because it is fast, capable of modeling feature interactions, and we can obtain an analytical solution. The proposed updatable QDA was compared to scikit-learn QDA on standard benchmarking datasets, and it showed a 1000-fold increase in speed than the scikit-learn QDA. When a new

feature is inserted into a training set, our implementation of QDA has a lower computational complexity by a factor of d, the number of features, compared with retraining QDA from scratch. Experimental results also show that an update with the online version is three times faster than a complete recalculation of QDA from scratch. Fast learning on a stream of features provides a data scientist with timely feedback about the importance of new features during the feature engineering phase. In the production phase, it reduces the cost of updating a model when a new source of potentially useful features appears.

12.2.1 Introduction

A common issue in machine learning is the need to update machine learning models based on changing data. This issue can be simplified by assuming that new samples will be available over time. However, we are concerned with the orthogonal problem – the fast updating of models with the arrival of new features (see Figure 12.2).

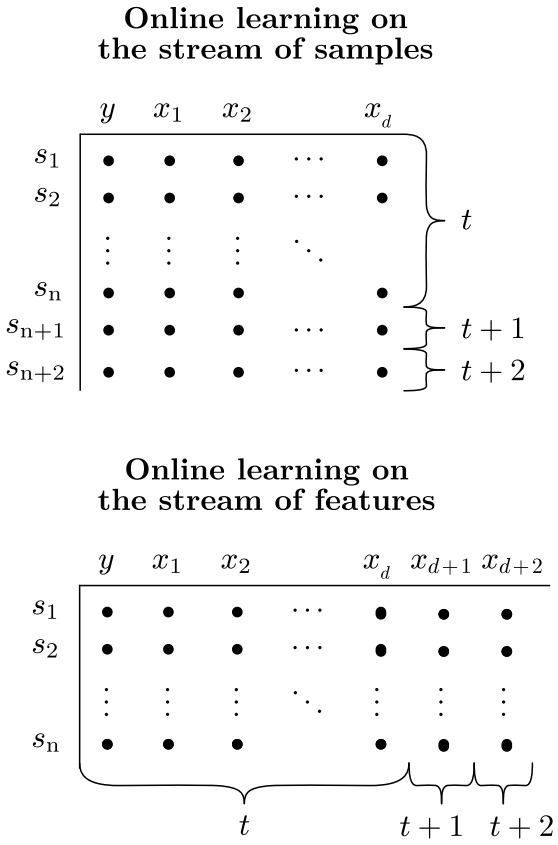

Figure 12.2: Difference between learning from a stream of samples (left) and a stream of features (right). In both cases, we have n samples and d features at time t. However, at time $t + 1$, we either have one more sample (left) or one more feature (right).

Interestingly, we did not find any publicly available implementation of a classifier,

which would satisfy following requirements:

1. it is faster to update the model with a new feature than to train the model from scratch,
2. is more accurate than naive Bayes, which can be straightforwardly updated to learning on a stream of features,
3. and is faster than logistic regression, random forest, and gradient boosted trees.

The proposed update of the quadratic discriminant analysis (QDA) to learning on a stream of features satisfies all these requirements.

12.2.1.1 Application

When we have an important model in production, we obtain new features that could improve the accuracy of the model. While we can always train a new model from scratch, this can take a considerable amount of time. If we could retrain the current model in a fraction of the time that it takes to build a model from scratch, it would allow us to:

1. test more hypotheses within the same time budget,
2. deploy the improved model quicker.

12.2.1.2 Quadratic Discriminant Analysis

We extend QDA because it is a simple (but nontrivial) model that can be solved analytically. Furthermore, QDA is capable of modeling interactions between features without explicitly defining them in the data preprocessing step. This property differentiates QDA from linear discriminant analysis (LDA). Nevertheless, it is easy to reduce QDA into LDA by using the same covariance matrix estimate for each class. LDA is known to have a good tradeoff between accuracy and runtime [249], is more efficient than logistic regression [62], and can be used for semi-supervised learning (e.g., by shrinking the class conditional covariance matrix estimates toward the shared covariance matrix).

This chapter is structured as follows. First, we discuss related literature in Section 12.2.2. Then we describe the proposed online QDA in Section 12.2.3. Empirical comparisons of the obtained accuracy and speed up are in Section 12.2.4. The paper closes with a discussion of the alternatives to the taken implementation of QDA in Section 12.2.5 and the conclusions in Section 12.2.6.

12.2.2 Related work

Feature stream processing was introduced by [274], where it was used for feature selection. The current state-of-the-art algorithm in this field is online streaming feature selection (OSFS) [262]. OSFS works as a feature-selection filter, which evaluates incoming features. Only when a new feature is evaluated by OSFS to be relevant and non-redundant, the feature is passed to a conventional downstream model, which is retrained from scratch.

While OSFS dramatically decreases the number of features, the downstream model retrains them; the time to retrain the downstream model remains an unsolved bottleneck [268].

Examples of other feature selection methods that work on a stream of features are a scalable and accurate online approach for feature selection (SAOLA) [269], online stream feature selection method based on mutual information (OSFSMI) [213], online stream feature selection method with self-adaptive sliding window (OSFSASW) [267], geometric online adaption (GOA) [237], and streaming feature selection considering feature interaction (SFS-FI) [275].

Our approach is a departure from the feature selection mindset; it focuses on *updating an offline classifier into an online classifier*.

This update approach was already successfully applied on a one-layer artificial neural network by [20], where they solve the problem analytically, and by [163], where they use dynamic programming on discretized features.

12.2.2.1 Unrelated work

Our implementation of QDA can be described as an *incremental algorithm*. However, this term can have several meanings, and we feel the need to explicitly state what our implementation is not, to avoid confusion.

A considerable amount of literature is associated with updating LDA when new samples arrive (see references in [51, Table 1]). However, we deal with an orthogonal problem when new features arrive. In addition, we do not calculate an approximate solution of QDA (e.g., by iteratively approximating the first few eigenvectors [44]) but calculate an *exact solution*.

12.2.3 QDA Algorithm

We denote by capital letters matrices, e.g., $Z \in \mathbb{R}^{n \times d}$, vectors by boldface characters, e.g., $\boldsymbol{z} \in \mathbb{R}^d$, and scalars by lowercase characters, e.g., $z \in \mathbb{R}$. n denotes the count of rows and d denotes the count of columns. Unless stated otherwise, we assume vectors to be column-vectors, i.e., we assume that $\boldsymbol{z}^\intercal \boldsymbol{z}$ is a scalar. We index vectors and matrices with square brackets, e.g., $Z[i, j]$ for i^{th} row and j^{th} column, to be able to use the same notion in mathematical equations and pseudocode. And we permit array broadcasting along the vertical, e.g., $Z - \boldsymbol{z} = Z - 1_n \boldsymbol{z}^\intercal$, where 1_n is a vector of ones.

A single data instance $\boldsymbol{s}$ can be scored with QDA using the following equation:

$$Z_k(\boldsymbol{s}) = -\frac{1}{2}(\boldsymbol{s} - \mu_k)^\intercal \Sigma_k^{-1}(\boldsymbol{s} - \mu_k) - \frac{1}{2}\ln|\Sigma_k| + \ln p_k. \tag{12.6}$$

where μ_k is the estimated mean value of instances in class k, Σ_k^{-1} is the inverse of the covariance matrix for the class k, $|\Sigma_k|$ is the determinant of the covariance matrix Σ_k, and p_k is the prior probability of the class k. Instance $\boldsymbol{s}$ is then classified into the class with the highest Z value.

12.2.3.1 Covariance

The class-conditional sample covariance matrix $\Sigma_{t,k}$ at time t is estimated from the class-conditional data matrix $X_{t,k}$ with

$$\Sigma_{t,k} = \frac{1}{n_k - 1}(X_{t,k} - \mu_{t,k})(X_{t,k} - \mu_{t,k})^\intercal, \tag{12.7}$$

where $\mu_{t,k}$ is a class-conditional vector of the feature means, and n_k is the class-conditional sample count. When a new feature $\boldsymbol{x}_{t+1}$ is appended into the data matrix, X_t becomes X_{t+1}. We then update the readily available $\Sigma_{t,k}$ to $\Sigma_{t+1,k}$ with:

$$\begin{aligned} \boldsymbol{x}^*_{t+1,k} &= \boldsymbol{x}_{t+1,k} - \bar{\boldsymbol{x}}_{t+1,k}, \\ X^*_{t+1,k} &= [X^*_{t,k}, \boldsymbol{x}^*_k], \\ \Sigma_{t+1,k} &= \left[\Sigma_{t,k}, \quad \tfrac{1}{n_k - 1}(X^*_{t+1,k} \cdot \boldsymbol{x}^*_{t+1,k})\right], \end{aligned} \tag{12.8}$$

where we store the covariance matrices in a packed (triangular) format because covariance matrices are always symmetric. For convenience, we use the * superscript to mark the centered matrices through the text. $\bar{\boldsymbol{x}}_{t+1,k}$ is the estimated mean value (a scalar) of vector $\boldsymbol{x}_{t+1,k}$.

12.2.3.2 Inverse

Whenever we see an inverse of a matrix (as in Equation (12.6)), we should generally never explicitly calculate the inverse because faster and more numerically stable methods exist [46]. Suppose that we want to solve $\boldsymbol{x} = A^{-1}\boldsymbol{y}$, where $\boldsymbol{x}$ is the unknown vector. Whenever matrix A is symmetric and positive definite, we can efficiently obtain $\boldsymbol{x}$ via the Cholesky decomposition `chol` [90, Section 4.2.3] followed by backward substitution `backward` [90, Section 3.1.6]:

$$\begin{aligned} R &= \texttt{chol}(A), \\ \boldsymbol{x} &= \texttt{backward}(R, \boldsymbol{y}). \end{aligned} \tag{12.9}$$

The Cholesky factorization decomposes matrix A into an upper triangular matrix R such that the product of R and its transpose $R^\intercal$ yields A:

$$A = RR^\intercal. \tag{12.10}$$

Backward substitution then solves the problem from bottom to top (hence, the name):

$$\begin{aligned} \boldsymbol{x}[d] &= \boldsymbol{y}[d]/R[d,d], \\ \boldsymbol{x}[i] &= \left(\boldsymbol{y}[i] - \sum_{j=i+1}^{d} R[i,j]\boldsymbol{x}[j]\right)/R[i,i], \quad i \in [1, d-1]. \end{aligned} \tag{12.11}$$

Herein, an important factor is that both the Cholesky decomposition and backward substitution can be efficiently updated when a new column is added to matrix A. The Cholesky decomposition can be updated with `cholinsert`, which is a built-in function in Octave or Julia (a reference implementation is given in Algorithm 4 in the appendix). The update of backward substitution is visually represented in Figure 12.3. Since the sample covariance matrices are always symmetric and positive definite when the features are linearly independent, we can use the Cholesky decomposition in QDA. Further discussions regarding the steps to be taken if this assumption is violated are detailed in Section 12.2.3.6.

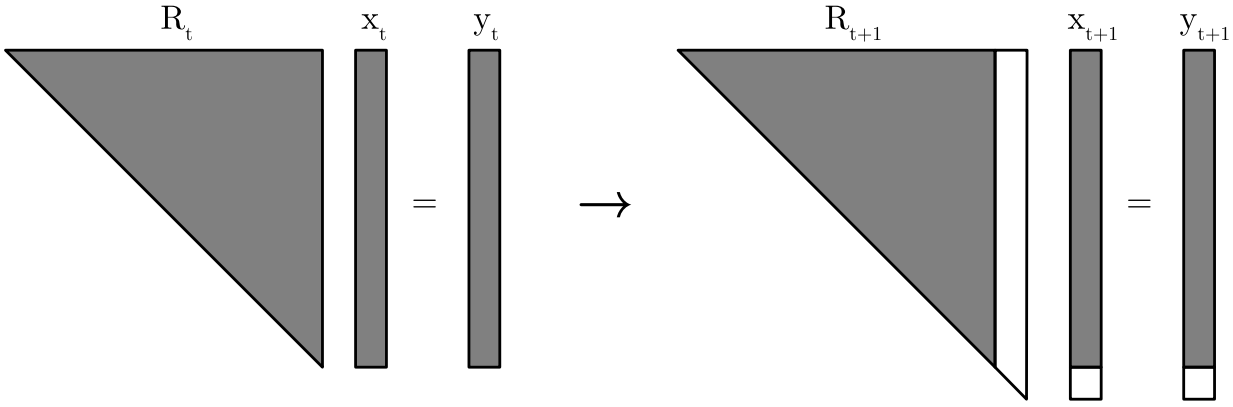

Figure 12.3: Updating the system of equations by appending a single column into triangular matrix R.

12.2.3.3 Determinant

We can obtain the determinant of matrix A as the product of the squared diagonal elements of $\texttt{chol}(A)$:

$$|A| = \prod_{i=1}^{d} R[i,i]^2, \tag{12.12}$$

where d is the size of R.

Whenever we need the logarithm of the determinant, as in Equation (12.6), we can avoid unnecessary overflows by summing the logarithms of the diagonal elements of $\texttt{chol}(A)$:

$$\ln|A| = \ln \prod_{i=1}^{d} R[i,i]^2 = 2 \cdot \sum_{i=1}^{d} \ln(R[i,i]). \tag{12.13}$$

Because `cholinsert` does not change the values of the old Cholesky decomposition R_t (it simply appends a new column), we can update the logarithm of the determinant in real time:

$$\ln|A_{t+1}| = \ln|A_t| + 2 \cdot \ln(R_{t+1}[t+1, t+1]), \tag{12.14}$$

where $R_{t+1}[t+1, t+1]$ is the bottom right number in triangular matrix R_{t+1}.

12.2.3.4 Vectorization

When scoring new samples, we may wish to score all the samples at once. However, Equation (12.6) is given only for a single sample. To remedy this, first note that:

$$\begin{aligned} \boldsymbol{b}^\intercal A^{-1}\boldsymbol{b} &= \boldsymbol{b}^\intercal (RR^\intercal)^{-1}\boldsymbol{b} \\ &= \boldsymbol{b}^\intercal (R^{-1})^\intercal R^{-1}\boldsymbol{b} \\ &= (R^{-1}\boldsymbol{b})^\intercal (R^{-1}\boldsymbol{b}), \end{aligned}$$

where R is the Cholesky decomposition of A. From this, we obtain a scoring function for matrix X:

$$Z_k(x) = -\frac{1}{2}\sum_d (R_k^{-1}(X - \mu_k)) \circ (R_k^{-1}(X - \mu_k)) - \frac{1}{2}\ln|\Sigma_k| + \ln p_k. \tag{12.15}$$

where $\circ$ is element-wise multiplication, μ_k is the class-conditional mean of the training data, R_k is the Cholesky decomposition of the Σ_k covariance matrix, and $R_k^{-1}X$ is solved with backward substitution.

12.2.3.5 Online version

The QDA update process is described in Algorithm 2, where `cholinsert` is a built-in function as implemented in Octave or Julia. In addition, `solveinsert` updates the solution of the triangular system of equations with multiple right-hand sides using the algorithm described in Algorithm 3 (backward substitution, but scaled to work with a matrix X instead of vector $\boldsymbol{x}$).

12.2.3.6 Regularization

One of the key problems of QDA is how to reliably estimate the covariance matrices. In Equation (12.7), we presented an empirical estimate of the sample covariance matrices. However, these sample covariance matrices suffer from a high variance of the parameter estimates as the count of the parameters to estimate grows quadratically with the number of features d and linearly with the number of the classes k.

When the count of the samples is small relative to d and k, it is frequently beneficial to assume that the covariance matrices are identical and use a single shared covariance matrix Σ_t everywhere we would use $\Sigma_{t,k}$:

$$\Sigma_t = \frac{(n_1 - 1)\Sigma_{t,1} + (n_2 - 1)\Sigma_{t,2} + \ldots + (n_k - 1)\Sigma_{t,k}}{(n_1 - 1) + (n_2 - 1) + \ldots + (n_k - 1)}, \tag{12.16}$$

where n with the index is the number of samples in the class. This variant of QDA is known as linear discriminant analysis (LDA).

Algorithm 2: Updating QDA when a new feature is inserted.

Input: new feature, $\boldsymbol{x}_{t+1}$; label conditional sample count, m; label conditional logarithm of prior probability, lp; label conditional feature means, μ_t; label conditional centered features, X_t^*; label conditional Cholesky decomposition, R_t; and label conditional logarithm of determinant, ld_t; t is the count of features, k is the class

Output: $\mu_{t+1}, X_{t+1}^*, R_{t+1}, ld_{t+1}, Z_{t+1}$

foreach $k \in K$ **do**

- $\mu_{t+1,k} = [\mu_{t+1,k}, \bar{\boldsymbol{x}}_{t+1,k}]$ `/* Mean vector */`
- $X_{t+1,k}^* = [X_{t,k}^*, \boldsymbol{x}_{t+1,k}^* - \bar{\boldsymbol{x}}_{t+1}]$ `/* Centered matrix */`
- $\boldsymbol{\sigma}_k = \frac{1}{m_k - 1} \cdot \boldsymbol{x}_{t+1,k}^* \cdot X_{t+1,k}^*$ `/* Covariance vector */`
- $R_{t+1,k} = \texttt{cholinsert}(R_{t,k}, \boldsymbol{\sigma}_k)$ `/* Cholesky decomposition */`
- $ld_{t+1,k} = ld_{t,k} + 2 \cdot \ln(R_{t+1,k}[t+1, t+1])$ `/* Log determinant */`
- $A_{t+1,k} = \texttt{solveinsert}(A_{t,k}, R_{t+1,k}, X_{t+1}^*)$ `/* Solve equations */`
- $Z_{t+1,k} = lp_k - \frac{1}{2} \sum (A_{t+1,k} \circ A_{t+1,k}) + ld_{t+1,k}$ `/* Discrimination score */`

end

Algorithm 3: Function `solveinsert` updates the solution of a triangular system of equations with multiple right-hand side for $A \cdot R = X$.

Input: A_t, R_{t+1}, X_{t+1}, where t is the count of features

Output: A_{t+1}

$A_{t+1} = [A_t, (X_{t+1}[:, t+1] - A_t[:, 1:t] \cdot R_{t+1}[1:t, t+1]) / R_{t+1}[t+1, t+1]]$

Another common issue with QDA is the invertibility of the sample covariance matrices [117]. This can be remedied with covariance shrinking toward the identity matrix [158]:

$$\Sigma_{t,k} = (1 - \lambda)\Sigma_{t,k} + \lambda I, \tag{12.17}$$

where I is the identity matrix of the size $\Sigma_{t,k}$ and λ is a shrinkage coefficient, $\lambda \in (0, 1)$.

In regularized discriminant analysis (RDA) by [80], the covariance matrix is a linear combination of the empirical shared covariance matrix, the empirical class conditional covariance matrix, and the identity matrix:

$$\Sigma_{t,k} = (1 - \alpha - \lambda)\Sigma_t + \alpha\Sigma_{t,k} + \lambda I, \tag{12.18}$$

where α is a tunable weight, $\alpha \in (0, 1 - \alpha)$.

12.2.4 Experiments

12.2.4.1 Datasets

We evaluated the proposed implementation of QDA on the OpenML-CC18 benchmarking suite by [15]. This suite includes 72 datasets, from which we excluded datasets

that include nominal features or missing values, as the implementation does not directly support them.

We compared two algorithms: QDA learning from scratch with scikit-learn 0.24.0 QDA as the reference versus the proposed QDA update by feature insert. For each of these two scenarios, we recorded the training and scoring time in seconds on each of the datasets. We also compared Brier scores (a calibration measure for classification) between the implementations for each of the datasets to validate that the quality of the predictions is equivalent. Since the Brier scores were indeed identical between the implementations, we do not report them.

Because some datasets have collinear features, as indicated by the high correlation coefficients in the last column in Table 12.2, we used regularized covariance matrices: $(1 - \lambda) \sum_k + \lambda I$, where $\lambda = 0.02$. A constant regularization was used, as the aim of the experiment was not to find the best possible regularization coefficients but to compare runtimes.

Median runtimes from 15 iterations are provided in Table 12.2.

dataset	scikit-learn	proposed	speed up	samples	features	classes	corr
mfeat-morphological	0.015	0.014	1.10	2 000	6	10	0.97
balance-scale	0.004	0.004	1.15	625	4	3	0
wilt	0.006	0.004	1.38	4 839	5	2	0.96
diabetes	0.005	0.003	1.71	768	8	2	0.54
vehicle	0.010	0.005	1.77	846	18	4	1.00
blood-transfusion	0.005	0.003	1.79	748	4	2	1
segment	0.022	0.010	2.21	2 310	19	7	1.00
banknote-authentication	0.007	0.003	2.44	1 372	4	2	0.79
climate-model	0.006	0.003	2.45	540	20	2	0.11
phoneme	0.010	0.004	2.45	5 404	5	2	0.32
kc2	0.008	0.003	2.94	522	21	2	1.00
wdbc	0.009	0.003	3.50	569	30	2	1.00
steel-plates-fault	0.042	0.012	3.61	1 941	27	7	1
letter	0.334	0.090	3.71	20 000	16	26	0.85
jungle_chess_2pcs	0.081	0.021	3.92	44 819	6	3	0.02
pendigits	0.097	0.023	4.18	10 992	16	10	0.86
pc1	0.013	0.003	5.00	1 109	21	2	1
kc1	0.019	0.003	6.30	2 109	21	2	1
qsar-biodeg	0.019	0.003	7.06	1 055	41	2	0.92
mfeat-zernike	0.090	0.013	7.08	2 000	47	10	1.00
wall-robot-navigation	0.049	0.007	7.31	5 456	24	4	0.63
pc3	0.025	0.003	8.02	1 563	37	2	1
texture	0.153	0.019	8.07	5 500	40	11	1
satimage	0.105	0.011	9.40	6 430	36	6	0.96
pc4	0.026	0.003	9.64	1 458	37	2	1
mfeat-karhunen	0.127	0.013	9.82	2 000	64	10	0.57
analcatdata_authorship	0.052	0.005	10.38	841	70	4	0.71
GesturePhaseSegmentation	0.136	0.012	11.17	9 873	32	5	0.94
mfeat-fourier	0.172	0.013	13.17	2 000	76	10	0.68
optdigits	0.243	0.017	14.42	5 620	64	10	0.93
first-order-theorem-proving	0.184	0.011	17.11	6 118	51	6	1
numerai28.6	0.765	0.033	23.32	96 320	21	2	0.86
spambase	0.091	0.004	23.62	4 601	57	2	1.00
ozone-level-8hr	0.085	0.003	29.33	2 534	72	2	1.00
semeion	0.624	0.012	50.07	1 593	256	10	0.81
mfeat-factors	0.686	0.014	50.67	2 000	216	10	1
mfeat-pixel	0.796	0.010	83.44	2 000	240	10	0.94
cnae-9	0.986	0.011	88.31	1 080	856	9	1
isolet	11.465	0.051	224.41	7 797	617	26	1.00
madelon	1.893	0.004	515.26	2 600	500	2	0.99
har	9.689	0.014	670.19	10 299	561	6	1
Fashion-MNIST	101.438	0.086	1 177.68	70 000	784	10	0.96
Devnagari-Script	731.853	0.484	1 511.97	92 000	1 024	46	0.97
mnist_784	145.914	0.091	1 600.96	70 000	784	10	1
Bioresponse	40.582	0.004	9 068.45	3 751	1 776	2	0.98
CIFAR_10	2 130.479	0.084	25 385.96	60 000	3 072	10	0.98

Table 12.2: Training and scoring times in seconds for scikit-learn QDA and the proposed updatable QDA, when we insert the last feature. Dataset metadata: count of samples, features, classes, and the maximal absolute Pearson's correlation coefficient between two different features in the dataset. Predictions from scikit-learn and the proposed QDA are identical. The datasets are ordered by the obtained speed up.

12.2.4.2 Accuracy

Because we are generally not only interested in the runtime but also in the accuracy of the classifiers, we compare the accuracy of QDA to a few selected classifiers. Due to the lack of classifier implementations that work on a stream of features, we use offline classifiers and evaluate them in offline use case.

Classifier selection Because of the focus on learning on a stream of features, we have to include naive Bayes (NB) and k-nearest neighbors (kNN) as potential competitors. As an approximation[4] of the online one-layer neural network as described in [20] we use logistic regression with L1 and L2 regularization (LR) [278]. Furthermore, we include random forest (RF) by [23], and gradient boosted trees (GBT) by [81] as examples of common offline classifiers. For all these algorithms, we use implementations from scikit-learn 0.24.0. As variants of QDA, we also include LDA, which uses a shared covariance matrix, and regularized discriminant analysis (RDA) by [80], which uses a linear combination of the shared covariance matrix and the class conditional covariance matrix. For an exhaustive comparison of QDA to other classifiers, we refer the keen reader to Delgado's empirical comparison of QDA to other 178 classifiers on 121 datasets by [68].

Methodology We use the same dataset as in Section 12.2.4.1 and estimate the testing ROC-AUC with 10-fold cross-validation. Because kNN (with Euclidean distance) requires normalized features, we normalize the features with z-score normalization for all the classifiers. The classifier hyperparameters were initialized to the recommended settings by [195] and greedily optimized based on the average testing ROC-AUC over all datasets. The used hyperparameters are listed in Table 12.3.

Table 12.3: The used classifier hyperparameters. The classifiers are described in Section 12.2.4.2.

Classifier	Parameters
GBT	n_estimators=500, learning_rate=0.1, max_features='log2'
kNN	n_neighbors=29
LDA	shrinkage=0.0001
LR	C=1.5, penalty='elasticnet', l1_ratio=0.5
NB	var_smoothing=10e-12
QDA	shrinkage=0.02
RDA	shrinkage=0.0001, alpha=0.2
RF	n_estimators=300, max_features=0.25, criterion='entropy'

[4] We contacted the authors but we were told that the implementation is not public.

Results The results are in Table 12.4. However, the Devnagari-Script, Fashion-MNIST, mnist_784, and numerai28.6 are absent in the comparison because too many algorithms didn't finish on these datasets in 3 hours. QDA underperforms on har, madelon, pc4, phoneme, and wall-robot-navigation datasets as QDA is incapable of modeling XOR-like problems because each class is represented by only a single prototype. To illustrate the character of these datasets, we cite a part of wall-robot-navigation dataset description: "The wall-following task and data gathering were designed to test the hypothesis that this apparently simple navigation task is indeed a non-linearly separable classification task"[5]. We just note that the problem is neither separable with a quadratic decision border. On the other end, decision tree based algorithms (RF, GBT) are capable of representing XOR-like problems and perform well on these datasets. A possible remedy of this QDA deficiency is to use multiple prototypes per class as in mixture discriminant analysis by [105] or map the data into a high-dimensional space as in generalized discriminant analysis by [8]. However, evaluation of these alternatives is out of the scope of this chapter.

On the other end, QDA performs well on mfeat-zernike and vehicle datasets, which are both separable with quadratic decision boundaries.

Statistical significance Statistically significant differences between the classifiers are depicted in Figure 12.4 following the procedure by [48]. NB is significantly worse than any other evaluated classifier. RF is the best. QDA, kNN, LDA, RDA, LR, and GBT are comparable.

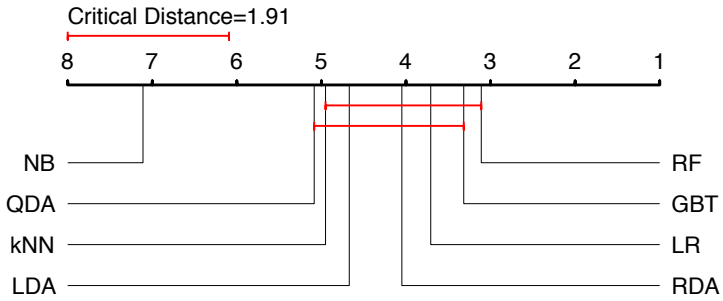

Figure 12.4: Comparison of the classifiers based on ROC-AUC. Groups of classifiers that are not significantly different (at p=0.01) are connected.

Pareto optimality The comparison of the average classifier ROC-AUC over datasets in Table 12.4 versus the average classifier total runtime on the cross-validation (training and scoring time) is depicted in Figure 12.5. LDA and RDA are not only Pareto optimal, but are also on the knee-point of the Pareto frontier.

[5] https://www.openml.org/d/1497

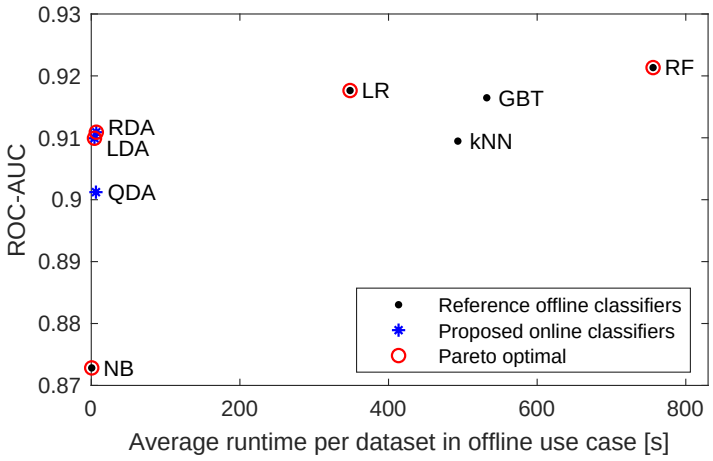

Figure 12.5: Classifier ROC-AUC and cross-validation runtime tradeoff. Regularized discriminant analysis (RDA) and linear discriminant analysis (LDA) are on the knee-point. The runtime is for the *offline* use case (all features are available at once). If we evaluated the classifiers on a stream of features, the *average* runtime of some of the offline classifiers would be in days.

Dataset	NB	kNN	LR	LDA	RDA	QDA	RF	GBT
Bioresponse	0.575	0.805	0.845	0.749	0.749	0.639	0.880	0.867
GesturePhaseSegmentation	0.670	0.706	0.743	0.735	0.684	0.674	0.763	0.747
analcatdata_authorship	0.994	1	1	1	1	0.997	0.999	1
balance-scale	0.889	0.916	0.964	0.951	0.955	0.971	0.714	0.872
banknote-authentication	0.938	1	1	1	1	1	1	1
blood-transfusion	0.801	0.733	0.947	0.932	0.927	0.877	0.548	0.610
climate-model	0.952	0.919	0.956	0.948	0.944	0.858	0.949	0.926
cnae-9	0.926	0.985	0.996	0.968	0.968	0.968	0.989	0.997
diabetes	0.816	0.813	0.830	0.830	0.831	0.812	0.833	0.809
first-order-theorem-proving	0.608	0.753	0.680	0.678	0.698	0.679	0.810	0.805
har	0.959	0.993	0.997	0.998	0.998	0.993	0.999	0.999
isolet	0.988	0.997	0.999	0.999	0.999	0.997	0.999	0.999
jungle_chess_2pcs	0.800	0.860	0.790	0.791	0.799	0.815	0.785	0.813
kc1	0.788	0.769	0.794	0.790	0.797	0.794	0.757	0.745
kc2	0.827	0.812	0.825	0.818	0.825	0.816	0.780	0.707
letter	0.957	0.998	0.980	0.967	0.982	0.996	1	1
madelon	0.644	0.622	0.585	0.587	0.587	0.527	0.913	0.703
mfeat-factors	0.993	0.998	0.998	0.999	0.999	0.997	0.999	0.999
mfeat-fourier	0.966	0.970	0.978	0.976	0.980	0.979	0.984	0.980
mfeat-karhunen	0.996	0.996	0.996	0.997	0.999	0.998	0.998	0.998
mfeat-morphological	0.946	0.959	0.966	0.965	0.964	0.963	0.956	0.958
mfeat-pixel	0.985	0.998	0.997	0.996	0.996	0.998	0.999	0.999
mfeat-zernike	0.960	0.978	0.982	0.979	0.981	0.980	0.969	0.968
optdigits	0.972	0.999	0.999	0.998	0.996	0.994	1	1
ozone-level-8hr	0.817	0.864	0.896	0.869	0.877	0.846	0.872	0.868
pc1	0.720	0.804	0.853	0.832	0.819	0.793	0.819	0.793
pc3	0.772	0.804	0.801	0.830	0.824	0.795	0.856	0.819
pc4	0.843	0.872	0.907	0.894	0.890	0.894	0.946	0.945
pendigits	0.979	0.999	0.997	0.990	0.997	0.999	1	1
phoneme	0.818	0.921	0.812	0.814	0.825	0.842	0.965	0.940
qsar-biodeg	0.822	0.905	0.919	0.904	0.912	0.908	0.917	0.920
satimage	0.956	0.986	0.977	0.972	0.976	0.975	0.991	0.991
segment	0.957	0.984	0.978	0.973	0.976	0.978	0.996	0.995
semeion	0.956	0.993	0.995	0.991	0.991	0.992	0.996	0.998
spambase	0.922	0.949	0.961	0.943	0.936	0.938	0.978	0.983
steel-plates-fault	0.884	0.884	0.893	0.895	0.899	0.894	0.907	0.902
texture	0.968	0.999	1	1	1	1	1	1
vehicle	0.776	0.903	0.945	0.943	0.944	0.956	0.934	0.938
wall-robot-navigation	0.820	0.918	0.890	0.847	0.868	0.870	0.999	0.999
wdbc	0.985	0.991	0.996	0.993	0.995	0.990	0.991	0.995
wilt	0.841	0.933	0.956	0.966	0.960	0.958	0.985	0.988
Avg (larger is better)	0.873	0.909	0.918	0.910	0.911	0.901	0.921	0.916
Rank (smaller is better)	7.110	4.951	3.707	4.671	4.049	5.085	3.110	3.317

Table 12.4: Testing ROC-AUC from cross-validation.

12.2.5 Discussion

12.2.5.1 Computational complexity

Inserting a new feature leads to computational complexity $\mathcal{O}(nd + d^2)$, while the calculation of QDA from scratch leads to computational complexity $\mathcal{O}(nd^2 + d^3)$. A detailed analysis of the computational complexity is given in Appendix A.3. The empirical results for randomly generated dense matrices are shown in Figure 12.6. The measurements were taken in Octave 4.4. The empirical results for 46 standard benchmarking datasets, where the proposed updatable QDA is compared to scikit-learn QDA[6], are provided in Section 12.2.4.1. In this comparison, the updatable QDA was always faster than scikit-learn QDA. Notably, there was a 1000-fold increase in speed in five cases. When we examined the scikit-learn implementation, we identified SVD decomposition as the bottleneck. For the description of how to use SVD for discriminant analysis, see for example [117]. The code for the resulting replication is available at github.com/janmotl/qda.

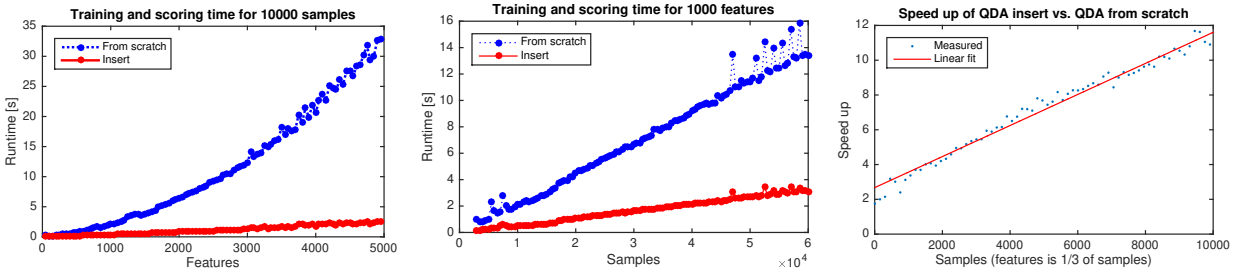

Figure 12.6: Inserting a new feature into QDA is faster and scales better than the calculation of QDA from scratch.

12.2.5.2 Feature selection

The ability to quickly insert new features can be used, for example, to speed up forward variable selection with LDA, as implemented in `greedy.wilks()` [223].

12.2.5.3 Model size

The size of the trained incremental QDA does not differ from the textbook QDA Equation (12.6).

However, whenever we want to insert a new feature into the model, we need more variables. All necessary variables can be calculated from the features directly. However, the calculation of $R_k^{-1}X_k^*$ from scratch takes the $\mathcal{O}(nd^2)$ with the `TRSM` subroutine from LAPAC. Hence, if we did not store $R_k^{-1}X_k^*$, it would increase the overall computational complexity of the online QDA update from $\mathcal{O}(nd + d^2)$ to $\mathcal{O}(nd^2 + d^2)$. Nevertheless, based on the empirical experiments on 1000 samples and 3000 features, even if we do not

[6]Scikit-learn is a free software learning library for Python.

store any variable beyond the first three in Table 12.5, an update with the online version is still $\sim 3\times$ faster than a complete recalculation of QDA from scratch.

Table 12.5: Model size per class for scoring (the top 3 lines) and model update (the bottom 3 lines). The offline implementation is in Equation (12.6), and the online version is in Equation (12.15).

Offline		Online	
Variable	Size	Variable	Size
Σ_k^{-1}	$d \times d$	R_k	$d \times d$
$\ln p_k - \frac{1}{2} \ln \lvert\Sigma_k\rvert$	1	$\ln p_k - \frac{1}{2} \ln \lvert\Sigma_k\rvert$	1
μ_k	d	μ_k	d
-	-	$R_k^{-1} X_k^*$	$n \times d$
-	-	X_k^*	$n \times d$
-	-	m_k	1

12.2.5.4 Limitations

The online version of QDA inherits all the disadvantages of offline QDA, namely high sensitivity to outliers and difficulties with the accurate estimation of the covariance matrix/matrices, which can be addressed by the use of robust statistical estimates [116] and covariance shrinking [158], respectively. Missing values in the estimate of the covariance matrix can be handled as described by [168]. And a mixture of numerical and nominal features can be handled with one of the methods described by [113].

12.2.5.5 Feature tracking

To update the model with new features, we must remember all the past features to be able to update the covariance matrix. This issue is shared with the offline version QDA and even with *streaming feature selection* methods [274], such as OSFS, because an irrelevant feature may become relevant once a new feature is inserted. A canonical example of this is an XOR problem: $\boldsymbol{y} = \text{XOR}(\boldsymbol{x}_1, \boldsymbol{x}_2)$, where x_1 and x_2 are features.

If it is inconvenient to remember all observed features (e.g., because of their sheer number and size), we can either recommend "upgrading" models that assume (conditional) independence of the features, such as naive Bayes, or using distance-based models. The advantage of naive Bayes is that for each feature, we have to only remember sufficient statistics [72], for example, the mean and variance of each continuous feature. The disadvantage of naive Bayes, in comparison to QDA, is that it does not model interactions between the features.

A significant advantage of a distance-based approach is that the size of the Gram/similarity/distance matrix does not grow with the number of features. This property is exploited, for example, in a support vector machine with an RBF kernel. The disadvantage

of the distance-based method is that the memory requirements grow quadratically with the number of instances. There are notable exceptions, such as low-rank kernel approximations [71] that were designed specifically to tackle this issue. QDA training, however, can be implemented to have memory requirements oblivious to the number of samples.

12.2.6 Conclusions

The key to learning on a stream of features with QDA is to use the Cholesky decomposition and `cholinsert` function in place of the matrix inverse. This method, together with an efficient update of the covariance matrices, reduced the computational complexity of the QDA update when a new feature was added, from $\mathcal{O}(nd^2 + d^3)$ to $\mathcal{O}(nd + d^2)$. This is equivalent to the reduction by d (the number of features). The proposed QDA returns identical predictions to scikit-learn QDA while being 1000 times faster. In the accuracy comparison, we identified the need to regularize the estimated covariance matrices. The proposed regularized discriminant analysis on a stream of features is statistically significantly more accurate than naive Bayes (at p=0.01) and comparable to regularized logistic regression, k-nearest neighbors, and gradient boosted trees while in runtime it is much closer to naive Bayes.

Chapter 13

Discussion

In this chapter, we describe the experience from deployment of Predictor Factory at multiple clients.

One of the questions that a consultant working with data must decide is whether the analysis will be performed on the client's infrastructure or on own hardware. With Predictor Factory, both approaches are feasible. Following paragraphs describe advantages and disadvantages of each approach.

Possibly the biggest advantage of working on a client's infrastructure is that the client does not give the data away. Hence, the client does not have to worry, whether the data contains sensitive informative that must not leave a company. Nor the client has to worry about data leaks.

Consequently, it is commonly easier to persuade a client to give you access to their data, if the analysis is performed at the client's infrastructure. Also, it commonly takes less time to get access to the data, if the analysis is performed on the client's infrastructure than if the analysis is performed out of the company, since "only" a terminal and appropriate privileges must be set up and no lawyer has to check that the data may actually leave the company. Consequently, if you work on the client's infrastructure, you may get your hands on a much wider range of data than you would get if you asked the company for data export. Finally, a non-negligible advantage is that you do not have to own a secure supercomputer for the analysis – all heavy lifting is done by the company's server.

However, the work at the client's infrastructure has its own disadvantages. The first complication is, that each client has a different infrastructure and it takes time to adapt to the new environment. The second disadvantage is that the client may not be overly happy if you consume too much computation resources.

Conclusion Overall, the biggest advantage of Predictor Factory is that it fits well into the infrastructure and processes of companies that store their data in relational databases and analyze their data with analytical tools like SAS. Predictor Factory changes a bare minimum of processes in a data mining process – just feature extraction. And Predictor

factory utilizes present infrastructure for the feature creation. No application have to be installed[1]. Hence, while Predictor Factory can be utilized on both, client's and consultant's side, Predictor Factory shines particularly on the client's side. The user manual for Predictor Factory is in Appendix A.4. The website is http://predictorfactory.com and the source code is at https://github.com/janmotl/PredictorFactory.

[1]Predictor Factory, nor Java Runtime Machine, have to be installed – they can be just copied.

Chapter 14

Conclusions

In Chapter 1, relational learning was introduced. And while relational learning is known from 1976, it was pointed out that the time dimension is not commonly correctly treated in relational learning.

Background and State-of-the-Art Propositionalization algorithms were decomposed into 7 building blocks and empirically compared in Chapter 2. The identified state-of-the-art algorithms in relational supervised learning were: Wordification [201], Link-Based propositionalization [55], Simple decision Forest [14], MVC with weighted voting [180], Path Independent Classifier (PIC) [233], and Ensemble of Probabilistic Relational Neighbours (EPRN) [205].

Predictor Factory Description of Predictor Factory, an implementation of a propositionalization algorithm, was given in Chapter 3. Predictor Factory was designed to do just one thing (propositionalization) and do it right. Hence, many technical details were discussed and addressed, like the naming convention or which data to process.

Implementation The architecture of Predictor Factory and the choice of the used technologies were described and justified in Chapter 4. During the user testing, 7 subjects out of 9 delivered a predictive model with Predictor Factory in less than an hour.

Relational repository With the help of Oliver Schulte, Václav Ostrožlík, and other contributors, a public repository of relational datasets, described in Chapter 5, was created to promote relational learning. The repository is used by scientist (there is over 70 citations at Google Scholar[1]), students (there is over 1 000 code results at GitHub[2]) as well as corporates including IBM[3] and Oracle.

[1] See: https://scholar.google.cz/scholar?q=relational.fit.cvut.cz.

[2] See: https://github.com/search?q=relational.fit.cvut.cz.

[3] See: https://github.com/chiragsahni/lale-gpl/blob/master/lalegpl/datasets/multitable/fetch_datasets.py.

Back in time, repositories like UCI Machine Learning Repository did not include relational datasets and the commonly used relational datasets in ILP community, like Train or Mutagenesis datasets, were not suitable for temporal relational learning as they did not contain any attribute that would be changing in time. The existence of the repository attracted contributors and there is now 22 non-synthetic (real) temporal relational datasets, which makes it possible to produce statistically significant results.

Empirical Evaluation The empirical comparison of Predictor Factory to other propositionalization tools in Chapter 6 chapter shows that Predictor Factory has classification accuracy comparable to other implementations. Nevertheless, there is one aspect in which Predictor Factory appears to be superior to the examined implementations – the ability to process diverse datasets.

Foreign key constraint discovery In relational learning, it is common to predict a target stored in one table based on the content of other tables. However, to use the content of these tables, you have to know their relationship to the target. In nicely designed relational datasets, it suffices to follow the foreign key constraints defined in the database. In reality, foreign key constraints are frequently absent, be it because of the limitations of the storage medium (e.g., CSV files) or simply because the constraints were never defined.

Algorithms for the estimation of the foreign key constraints already existed. However, they had $\mathcal{O}(n^2)$ computational complexity with regard to the number of attributes in a database. And that means that they did not scale well. What helps is to collect attribute statistics in one pass over the database and then run the $\mathcal{O}(n^2)$ algorithm just on these statistics, instead of the actual and potentially large data. Can we do better? Another step could be to collect sampled statistics instead of the exact statistics, reducing the one pass over the database to a fraction of a single pass. However, the empirical results in Chapter 7 suggest that we do not have to see even a single datum from the data to make sound estimates. How is that possible? Modern databases already collect attribute statistics for execution plan optimization. Hence, all that was necessary to do was to check whether these statistics could be reused for foreign key constraint discovery. The empirical results suggest that these statistics work equally well as statistics explicitly developed for foreign key constraint discovery. Hence, we now have a fast open source implementation for the detection of the relationships in the database.

Multi target stratified sampling Some of the features, which Predictor Factory generates, are supervised – we need the target to calculate them. However, while these supervised features can be highly predictive of the target, they are unlikely to be predictive of a different target. To make Predictor Factory practical in scenarios when we need to predict multiple targets (e.g., propensities to buy different products), Predictor Factory supports multiple targets in a single run. Unfortunately, the standard stratified sampling, which is used to evaluate features' relevancy, does not work with more than

one target. An exact solution to stratified sampling based on multiple targets is provided in Chapter 8.

Detrending and deseasoning Detrending and deseasoning are common techniques in time series analysis. However, the common techniques from time series analysis are generally not directly applicable to relational data. A method how to detrend and deseason relational data is described in Chapter 9.

Generalized aggregate While one of the biggest advantages of propositionalization is that it is trivial to add new feature functions, it leads to the explosion of the number of feature functions. A family of aggregates, which generalizes common aggregates like *sum*, *min* or *max* was defined in Chapter 10.

Meta learning Achilles heel of propositionalization is that it produces a vast quantity of irrelevant and redundant features. However, since the features are generated sequentially and not all at once, we can build a meta model to predict the features' univariate relevancy, redundancy and runtime. Based on the empirical results Chapter 11, meta learning reduces propositionalization runtime to one-tenth of the original runtime without loss of accuracy of the downstream model.

Learning on stream of features The disadvantage of the implemented meta learning is that it is unaware of the used downstream model. To remedy that, we modified two classifiers, random forest Section 12.1 and discriminant analysis Section 12.2, to work efficiently on a stream of features, making it practical to train and use a meta model, which predicts improvement of the accuracy of the chosen downstream model, when we calculate another feature.

Discussion The author's experience with the application of Predictor Factory on clients' data is provided in crefchap:discussion. One of the biggest practical advantages of Predictor Factory is that all the heavy computation is happening directly in the client's (testing) database. And once Predictor Factory finishes, a SQL script which replicates the run is produced. Hence, no new hardware or software has to be obtained for production.

14.1 Contributions

The main contributions of the dissertation thesis are as follows:

1. Relational dataset repository for benchmarking relational algorithms (83 datasets).
2. Algorithm for discovering relationships between tables in a database (scales independently of the row count in the tables).

3. Optimal algorithm for stratified partitioning based on multiple columns for cross-validation (with exact solution).
4. Predictor Factory for automatic feature extraction from relational data for classification and regression.
5. Approach for removing trend and seasonal variation from relational data (improves accuracy, P=.0005).
6. Generalization of common aggregate functions to temporal data (improves accuracy, P=.016).
7. Meta-learning for faster feature extraction (reduces runtime by 90%).
8. Classifiers for learning on a stream of features (up to 10 000 times faster than scikit-learn).

14.2 Future Work

The author of the dissertation thesis suggests designing a neural network, which exploits the idea of generalized aggregates to temporal data. Traditionally, attributes in n:1 relationship to the target were handled with recurrent neural networks (RNN). However, while RNNs are powerful, they also require a large training set. Generalized aggregates use fewer parameters than RNNs, but are still at least as powerful as commonly used aggregates in statistics and time-series analysis. Hence, they should outperform RNNs in data-scarce situations. Research questions to answer: Are generalized aggregates suitable for transfer learning like convolutional neural networks (CNN)? CNNs were found to produce patterns similar to Gabor filter. Can we find a similar relationship between the pre-trained generalized aggregates and the commonly used aggregates?

Bibliography

[1] S. M. Abdulrahman and P. Brazdil. Measures for combining accuracy and time for meta-learning. *CEUR Workshop Proc.*, 1201:49–50, 2014.

[2] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *VLDB J.*, 24(4):557–581, 2015.

[3] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proc Int. Conf. Very Large Databases*, volume 1215, pages 487–499, Santiage, Chile, 1994. Morgan Kaufmann.

[4] J. Anderson, M. Gaare, J. Holguín, N. Bailey, and T. Pratley. *Professional Clojure*. John Wiley and Sons, 2016.

[5] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn. Design and Implementation of the LogicBlox System. In *Proc. 2015 ACM SIGMOD Int. Conf. Manag. Data - SIGMOD '15*, pages 1371–1382, New York, New York, USA, 2015. ACM Press.

[6] A. Atramentov, H. A. H. Leiva, and V. Honavar. A Multi-relational Decision Tree Learning Algorithm – Implementation and Experiments. In T. Horváth and A. Yamamoto, editors, *Inductive Log. Program.*, volume 2835 of *LNAI*, pages 38–56. Springer Berlin Heidelberg, 2003.

[7] F. Bajersvej. *Relational Classification - Decision Tree Approach*. PhD thesis, Aalborg University, 2005.

[8] G. Baudat and F. Anouar. Generalized discriminant analysis using a kernel approach. *Neural Comput.*, 12(10):2385–2404, 2000.

[9] S. D. Bay, D. Kibler, M. J. Pazzani, and P. Smyth. The UCI KDD archive of large data sets for data mining research and experimentation. *ACM SIGKDD Explor. Newsl.*, 2(2):81–85, dec 2000.

[10] J. G. Bennett, P. A. Gee, and C. E. Gayraud. System and Methods Including Automatic Linking of Tables for Improved Relational Database Modeling with Interface, 1997.

[11] C. Bergmeir, R. J. Hyndman, and B. Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Comput. Stat. Data Anal.*, 120(April):70–83, apr 2018.

[12] P. Berka. Workshop notes on Discovery Challenge PKDD'99. Technical report, University of Economics, Prague, 1999.

[13] A. Bernard and P. van Elteren. A generalization of the method of m rankings. In *Proc. Int. Math. Congr.*, volume 2, pages 275–276, Amsterdam, sep 1954.

[14] B. Bina, O. Schulte, B. Crawford, Z. Qian, and Y. Xiong. Simple decision forests for multi-relational classification. *Decis. Support Syst.*, 54(3):1269–1279, 2013.

[15] B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. G. Mantovani, J. N. van Rijn, and J. Vanschoren. OpenML Benchmarking Suites. *arXiv*, pages 1–6, 2017.

[16] H. Blockeel. Learning, attribute-value. Technical report, Katholieke Universiteit Leuven, 2010.

[17] H. Blockeel. Learning, relational. Technical report, KU Leuven, Leuven, 2013.

[18] H. Blockeel, S. Džeroski, B. Kompare, S. Kramer, B. Pfahringer, and W. V. Laer. Experiments in Predicting Biodegradability. *Appl. Artif. Intell.*, 18(2):157–181, feb 2004.

[19] H. Blockeel and W. Uwents. Using neural networks for relational learning. *Work. Notes ICML-2004 Work. Stat. Relational Learn. Connect. to Other Fields*, pages 23–28, 2004.

[20] V. Bolon-Canedo, D. Fernández-Francos, D. Peteiro-Barral, A. Alonso-Betanzos, B. Guijarro-Berdiñas, and N. Sánchez-Maroño. A unified pipeline for online feature selection and classification. *Expert Syst. Appl.*, 55:532–545, 2016.

[21] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley, 2015.

[22] T. Brandenburger and A. Furth. Cumulative Gains Model Quality Metric. *J. Appl. Math. Decis. Sci.*, 2009:1–14, jun 2009.

[23] L. Breiman. Random forest. *Mach. Learn.*, 45(5):1–35, 1999.

[24] N. E. Breslow and D. G. Clayton. Approximate Inference in Generalized Linear Mixed Models. *J. Am. Stat. Assoc.*, 88(421):9, 1993.

[25] G. Brown, A. Pocock, M.-J. Zhao, and M. Lujan. Conditional Likelihood Maximisation: A Unifying Framework for Mutual Information Feature Selection. *J. Mach. Learn. Res.*, 13:27–66, 2012.

[26] A. M. Bujang, N. Sa'at, M. T. A. B. Sidik, and L. C. Joo. Sample Size Guidelines for Logistic Regression from Observational Studies with Large Population. *Malaysian J. Med. Sci.*, 25(4):122–130, 2018.

[27] P. Buryan. *Refinement Action-Based Framework For Utilization Of Softcomputing In Inductive Learning*. PhD thesis, Czech Technical University, 2013.

[28] D. Cai, X. He, and J. Han. Training linear discriminant analysis in linear time. *Proc. - Int. Conf. Data Eng.*, 00:209–217, 2008.

[29] M. Ceci and A. Appice. Spatial associative classification: Propositional vs structural approach. *J. Intell. Inf. Syst.*, 27(3):191–213, 2006.

[30] M. Ceci, A. Appice, H. L. Viktor, D. Malerba, E. Paquet, and H. Guo. Transductive relational classification in the co-training paradigm. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 7376 LNAI:11–25, 2012.

[31] P. Cerda, G. Varoquaux, and B. Kégl. Similarity encoding for learning with dirty categorical variables. *Mach. Learn.*, 107(8-10):1477–1494, 2018.

[32] S. H. Cha and S. N. Srihari. On measuring the distance between histograms. *Pattern Recognit.*, 35(6):1355–1370, 2002.

[33] D. D. Chamberlin. Early history of SQL. *IEEE Ann. Hist. Comput.*, 34(4):78–82, 2012.

[34] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Comput. Electr. Eng.*, 40(1):16–28, 2014.

[35] H. Chen, H. Liu, J. Han, and X. Yin. Exploring Optimization of Semantic Relationship Graph for Multi-relational Bayesian Classification. *Decis. Support Syst.*, 48(1):112–121, 2009.

[36] T. Chen, T. He, and M. Benesty. Extreme Gradient Boosting. *arXiv*, pages 1–4, 2016.

[37] Z. Chen, V. Narasayya, and S. Chaudhuri. Fast Foreign-Key Detection in Microsoft SQL Server PowerPivot for Excel. *VLDB Endow.*, 7(13):1417–1428, 2014.

[38] J. Cheng, C. Hatzis, H. Hayashi, M.-A. Krogel, S. Morishita, D. Page, and J. Sese. KDD Cup 2001 report. *ACM SIGKDD Explor. Newsl.*, 3(2):47, jan 2002.

[39] N. R. Chrisman. Rethinking Levels of Measurement for Cartography. *Cartogr. Geogr. Inf. Sci.*, 25(4):231–242, mar 1998.

[40] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. STL: A seasonal-trend decomposition. *J. Off. Stat.*, 6(1):3–73, 1990.

[41] W. G. Cochran. Sampling Techniques, 1977.

[42] I. Coursac and N. Duteil. PKDD 2001 Discovery Challenge - Medical Domain. *PKDD 2001 Discov. Chall. 2001*, 3(2), 2002.

[43] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control. Signals, Syst.*, 2(4):303–314, 1989.

[44] I. Dagher. Incremental PCA-LDA algorithm. *CIMSA*, (4):97–101, 2010.

[45] G. Das. Sampling Methods in Approximate Query Answering Systems. In *Encycl. Data Warehous. Mining, Second Ed.*, pages 1702–1707. IGI Global, 2009.

[46] T. A. Davis. Algorithm 9xx: FACTORIZE: An object-oriented linear system solver for MATLAB. *ACM Trans. Math. Softw.*, 39(4):1–18, 2013.

[47] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *J. Med. Chem.*, 34(2):786–797, feb 1991.

[48] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

[49] P. J. Denning. The locality principle. *Commun. ACM*, 48(7):19–24, jul 2005.

[50] M. M. Deza and E. Deza. Encyclopedia of Distances. *Encycl. Distances*, pages 291–305, 2013.

[51] T. I. Dhamecha, R. Singh, and M. Vatsa. On incremental semi-supervised discriminant analysis. *Pattern Recognit.*, 52:135–147, 2016.

[52] N. A. Diamantidis, D. Karlis, and E. A. Giakoumakis. Unsupervised stratification of cross-validation for accuracy estimation. *Artif. Intell.*, 116(1-2):1–16, 2000.

[53] T. G. Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees : Bagging , Boosting , and Randomization. *Mach. Learn.*, 40(2):139–157, 2000.

[54] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artif. Intell.*, 89(1-2):31–71, 1997.

[55] Q.-T. Dinh, M. Exbrayat, and C. Vrain. Link-Based Method for Propositionalization. *Inductive Log. Program.*, pages 2–7, 2012.

[56] B. Dolšak and S. Muggleton. The Application of Inductive Logic Programming to Finite Element Mesh Design. In S. Muggleton, editor, *Inductive Log. Program.*, pages 453–472, London, 1992. Academic Press.

[57] R. Dubčáková. Eureqa: Software review. *Genet. Program. Evolvable Mach.*, 12(2):173–178, 2011.

[58] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley Interscience, 2 edition, 2000.

[59] J. Durbin. Incomplete blocks in ranking experiments. *Br. J. Stat. Psychol.*, 4(2):85–90, jun 1951.

[60] Dušan Petković. Modern Temporal Data Models: Strengths and Weaknesses. *Commun. Comput. Inf. Sci.*, 521:136–146, 2015.

[61] S. Džeroski, S. Schulze-Kremer, K. R. Heidtke, K. Siems, and D. Wettschereck. Applying ILP to Diterpene Structure Elucidation from 13C NMR Spectra. *Appl. Artif. Intell.*, 12:41–54, aug 1996.

[62] B. Efron. The efficiency of logistic regression compared to normal discriminant analysis. *J. Am. Stat. Assoc.*, 70(352):892–898, 1975.

[63] W. Emde and D. Wettschereck. Relational Instance-Based Learning. *ICML*, 1996.

[64] G. Farnadi. Statisticcal Relational Learning with Soft Quantifier. Technical report, Universiteit Gent, 2015.

[65] S. Fatehi. SchemaCrawler, 2017.

[66] T. Fawcett. An introduction to ROC analysis. *Pattern Recognit. Lett.*, 27(8):861–874, jun 2006.

[67] X. Feng, A. Kumar, B. Recht, and C. Ré. Towards a unified architecture for in-RDBMS analytics. In *SIGMOD Conf.*, pages 325–336, 2012.

[68] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *J. Mach. Learn. Res.*, 15:3133–3181, 2014.

[69] FICO. Leveraging Transactional Data with Genetic Algorithms. Technical Report November, FICO, 2006.

[70] J. J. Filho and J. Wainer. HPB: A Model for Handling BN Nodes with High Cardinality Parents. *J. Mach. Learn. Res.*, 9:2141–2170, 2008.

[71] S. Fine and K. Scheinberg. Efficient SVM Training Using Low-Rank Kernel Representations. *JMLR*, 1:243–264, 2000.

[72] R. A. Fisher. On the Mathematical Foundations of Theoretical Statistics. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, 222(594-604):309–368, 1922.

[73] P. A. Flach and N. Lachiche. Naive Bayesian Classification of Structured Data. *Mach. Learn.*, 57(3):233–269, 2004.

[74] G. Forman, G. Forman, M. Scholz, and M. Scholz. Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement. *HP labs*, 12(1):49–57, 2009.

[75] J. Foulds and E. Frank. A review of multi-instance learning assumptions. *Knowl. Eng. Rev.*, 25(01):1, mar 2010.

[76] M. V. M. França, G. Zaverucha, and A. D'Avila Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Mach. Learn.*, 94(1):81–104, 2014.

[77] R. Frank, F. Moser, and M. Ester. A Method for Multi-relational Classification Using Single and Multi-feature Aggregation Functions. In *Knowl. Discov. Databases PKDD 2007*, volume 4702 of *LNAI*, pages 430–437. Springer Berlin Heidelberg, Berlin, 2007.

[78] P. Frasconi, F. Costa, L. De Raedt, and K. De Grave. kLog: A Language for Logical and Relational Learning with Kernels. *Artif. Intell.*, 7(2007):117–143, 2012.

[79] Y. Freund and R. E. Schapire. Experiments with a New Boosting Algorithm. *Int. Conf. Mach. Learn.*, pages 148–156, 1996.

[80] J. H. Friedman. Regularized discriminant analysis. *J. Am. Stat. Assoc.*, 84(405):165–175, 1989.

[81] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Stat.*, 29(5), oct 2001.

[82] M. Friedman. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *J. Am. Stat. Assoc.*, 32(200):675–701, dec 1937.

[83] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):1–37, mar 2014.

[84] R. García-Durán, F. Fernández, and D. Borrajo. Prototypes Based Relational Learning. In *Artif. Intell.*, chapter 3, pages 130–143. Springer, Berlin, 2008.

[85] A. H. Gazala and W. Ahmad. Multi-Relational Data Mining A Comprehensive Survey. In M. Usman, editor, *Improv. Knowl. Discov. through Integr. Data Min. Tech.* IGI Global, jan 2015.

[86] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[87] A. S. Ghanem, S. Venkatesh, and G. West. Learning in imbalanced relational data. *2008 19th Int. Conf. Pattern Recognit.*, pages 1–4, 2008.

[88] L. Ghionna and G. Greco. Boosting tuple propagation in multi-relational classification. In *Proc. 15th Symp. Int. Database Eng. & Appl. - IDEAS '11*, pages 106–114, New York, sep 2011. ACM Press.

[89] V. Gjorgjioski and S. Džeroski. Stochastic propositionalization of relational data using aggregates. Technical report, Jožef Stefan Institute, 2008.

[90] G. H. Golub and C. F. V. Loan. *Matrix Computations*. Johns Hopkins, 4th edition, 2013.

[91] A. N. Gorban and I. Y. Tyukin. Blessing of dimensionality: Mathematical foundations of the statistical physics of data. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, 376(2118), 2018.

[92] S. B. Green. How Many Subjects Does It Take To Do A Regression Analysis. *Multivariate Behav. Res.*, 26(3):499–510, 1991.

[93] T. Gunasegaran and Y. N. Cheah. Evolutionary cross validation. *ICIT 2017 - 8th Int. Conf. Inf. Technol. Proc.*, pages 89–95, 2017.

[94] H. Guo. *Learning from Multirelational Data through Multiple Views*. PhD thesis, University of Ottawa, 2010.

[95] H. Guo and H. L. Viktor. Mining relational databases with multi-view learning. *Proc. 4th Int. Work. Multi-relational Min.*, pages 15–24, 2005.

[96] H. Guo and H. L. Viktor. Mining imbalanced classes in multirelational classification. In *Proceeding 6th Int. Work. Multi-Relational Data Min.*, pages 46–57, 2007.

[97] H. Guo and H. L. Viktor. Learning from Skewed Class Multi-relational Databases. *Fundam. Informaticae*, 89(1):69–94, 2008.

[98] H. Guo and H. L. Viktor. Multirelational classification: a multiple view approach. *Knowl. Inf. Syst.*, 17(3):287–312, feb 2008.

[99] I. Guyon and A. Elisseefl. An introduction to feature extraction. *Stud. Fuzziness Soft Comput.*, 207:1–25, 2006.

[100] I. Guyon, M. Nikravesh, S. Gunn, and L. A. Zadeh, editors. *Feature Extraction*, volume 207 of *Studies in Fuzziness and Soft Computing*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[101] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. An Introduction to Variable and Feature Selection Isabelle. *Mach. Learn.*, 46(1/3):389–422, 2002.

[102] M. A. Hall. *Correlation-based Feature Selection for Machine Learning*. PhD thesis, The University of Waikato, 1999.

[103] T. Halpin and S. Rugaber. *LogiQL: A Query Language for Smart Databases*. CRC Press, 2014.

[104] T. Hastie and R. Tibshirani. Generalized Additive Models. *Stat. Sci.*, 1(3):297–310, 1986.

[105] T. Hastie and R. Tibshirani. Discriminant Analysis by Gaussian Mixtures. *J. R. Stat. Soc. Ser. B*, 58(1):155–176, 1996.

[106] C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The Predictive Toxicology Challenge 2000-2001. *Bioinformatics*, 17(1):107–108, jan 2001.

[107] J. D. Hirst, R. D. King, and M. J. E. Sternberg. Quantitative structure-activity relationships by neural networks and inductive logic programming. I. The inhibition of dihydrofolate reductase by pyrimidines. *J. Comput. Aided. Mol. Des.*, 8(4):405–420, 1994.

[108] D. S. Hochbaum. Integer Programming and Combinatorial Optimization. Technical report, University of California, 2010.

[109] C. C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *Int. J. Forecast.*, 20(1):5–10, 2004.

[110] A. G. Howard. *Missing Data in Non-Parametric Tests of Correlated Data*. PhD thesis, University of North Carolina at Chapel Hill, 2012.

[111] Huan Liu and R. Setiono. Chi2: feature selection and discretization of numeric attributes. In *Proc. 7th IEEE Int. Conf. Tools with Artif. Intell.*, pages 388–391. IEEE Comput. Soc. Press, 1995.

[112] G. B. Huang, Q. Y. Zhu, and C. K. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[113] C. J. Huberty, J. M. Wisenbaker, J. D. Smith, and J. C. Smith. Using Categorical Variables in Discriminant Analysis. *Multivariate Behav. Res.*, 21(4):479–496, oct 1986.

[114] T. N. Huynh and R. J. Mooney. Discriminative structure and parameter learning for Markov logic networks. In *Proc. 25th Int. Conf. Mach. Learn. - ICML '08*, volume 307, pages 416–423, New York, New York, USA, 2008. ACM Press.

[115] J. Jambeiro Filho and J. Wainer. Using a hierarchical Bayesian model to handle high cardinality attributes with relevant interactions in classification problem. In *IJCAI Int. Jt. Conf. Artif. Intell.*, pages 2504–2509, 2007.

[116] J.-h. Jiang, Z.-p. Chen, C.-j. Xu, and R.-q. Yu. Robust Linear Discriminant Analysis for Chemical Pattern Recognition. *J. Chemom.*, 13(January 1998):3–13, 1999.

[117] J. Kalina, Z. Valenta, and J. D. Tebbens. Computation of Regularized Linear Discriminant Analysis. *Comput. Stat. Int. Conf.*, pages 128–133, 2015.

[118] T. Kamishima, H. Kazawa, and S. Akaho. A Survey and Empirical Comparison of Object Ranking Methods. In J. Fürnkranz and E. Hüllermeier, editors, *Prefer. Learn.*, pages 181–201. Springer, Berlin, 2010.

[119] J. M. Kanter and O. Gillespie. Label, Segment, Featurize : a cross domain framework for prediction engineering. In *IEEE DSAA 2016*, page 10. IEEE, 2016.

[120] J. M. Kanter and K. Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE Int. Conf. Data Sci. Adv. Anal.*, pages 1–10. IEEE, oct 2015.

[121] A. Karalič and I. Bratko. First Order Regression. *Mach. Learn.*, 26(2-3):147–176, 1997.

[122] T. Karunaratne and H. Boström. Graph propositionalization for random forests. *8th Int. Conf. Mach. Learn. Appl. ICMLA 2009*, pages 196–201, 2009.

[123] T. Karunaratne, H. Boström, and U. Norinder. Pre-processing structured data for standard machine learning algorithms by supervised graph propositionalization - A case study with medicinal chemistry datasets. *Proc. - 9th Int. Conf. Mach. Learn. Appl. ICMLA 2010*, pages 828–833, 2010.

[124] V. Kassarnig and F. Wotawa. Evolutionary propositionalization of multi-relational data. *Proc. 30th Int. Conf. Softw. Eng. Knowl. Eng.*, 2018:629–690, 2018.

[125] S. Kaufman and S. Rosset. Leakage in data mining: Formulation, detection, and avoidance. *KDD '11 Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. data Min.*, pages 556–563, 2012.

[126] M. Kaufmann, P. Fischer, N. May, and D. Kossmann. Benchmarking Bitemporal Database Systems: Ready for the Future or Stuck in the Past? *EDBT*, pages 738–749, 2014.

[127] N. Kaur, G. Kunapuli, S. Joshi, K. Kersting, and S. Natarajan. Neural Networks for Relational Data. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 11770 LNAI:62–71, 2020.

[128] S. M. Kazemi, D. Buchman, K. Kersting, S. Natarajan, and D. Poole. Relational logistic regression. *Proc. Int. Work. Tempor. Represent. Reason.*, pages 548–557, 2014.

[129] C. S. Kheau, R. Alfred, and L. H. Keng. Dimensionality Reduction in Data Summarization Approach to Learning Relational Data. *ACIIDS*, 7802:166–175, 2013.

[130] A. Kiel. Datomic. Technical report, University Leipzig, 2013.

[131] W. O. N. Kim, B.-j. J. Choi, S.-k. K. Kim, E. K. Hong, S.-k. K. Kim, and D. Lee. A Taxonomy of Dirty Data. *Data Min. Knowl. Discov.*, 7(1):81–99, 2003.

[132] R. Kimball and M. Ross. *The data warehouse toolkit: The definitive guide to dimensional modeling*. John Wiley & Sons, 2013.

[133] R. D. King, M. Sternberg, and A. Srinivasan. Relating chemical activity to structure: An examination of ILP successes. *New Gener. Comput.*, 13(3-4):411–433, 1995.

[134] I. Kitzmann, C. Konig, D. Lubke, and L. Singer. A Simple Algorithm for Automatic Layout of BPMN Processes. *2009 IEEE Conf. Commer. Enterp. Comput.*, pages 391–398, 2009.

[135] D. J. Klein. Centrality measure in graphs. *J. Math. Chem.*, 47(4):1209–1223, 2010.

[136] A. J. Knobbe. *Multi-relational data mining*. PhD thesis, Universiteit Utrecht, 2004.

[137] A. J. Knobbe, M. de Haas, and A. Siebes. Propositionalisation and Aggregates. *Lect. Notes Comput. Sci.*, 2168:277–288, 2001.

[138] G. T. Knofczynski and D. Mundfrom. Sample sizes when using multiple linear regression for prediction. *Educ. Psychol. Meas.*, 68(3):431–442, 2008.

[139] R. Kohavi. A study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *IJCAI*, 5(0):7, 1995.

[140] S. Kok, M. Summer, M. Richardson, P. Singla, H. Poon, D. Lowd, J. Wang, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, University of Washington, 2009.

[141] J. Kranjc, V. Podpečan, and N. Lavrač. ClowdFlows: A Cloud Based Scientific Workflow Platform. In *ECML PKDD*, pages 816–819, Bristol, 2012.

[142] M.-A. Krogel. *On Propositionalization for Knowledge Discovery in Relational Databases*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, 2005.

[143] M.-A. Krogel, S. Rawles, F. Železný, P. A. Flach, N. Lavrač, S. Wrobel, F. Železný, P. A. Flach, N. Lavrač, S. Wrobel, M.-A. Krogel, N. Lavrač, S. Rawles, P. A. Flach, and S. Wrobel. Comparative Evaluation of Approaches to Propositionalization. In *Proc. 13th Int. Conf. Inductive Log. Program.*, volume 2835, pages 194–217, 2003.

[144] M.-A. Krogel and S. Wrobel. Transformation-Based Learning Using Multirelational Aggregation. In *ILP*, pages 142–155. Springer, London, 2001.

[145] M.-A. Krogel and S. Wrobel. Propositionalization and Redundancy Treatment. In *Databases, Doc. Inf. Fusion*, pages 1–13, Hannover, 2002. CEUR.

[146] M.-A. Krogel and S. Wrobel. Facets of aggregation approaches to propositionalization. In *Inductive Log. Program.*, pages 30–39. Springer, Berlin, 2003.

[147] S. Kruse, T. Papenbrock, H. Harmouch, and F. Naumann. Data Anamnesis: Admitting Raw Data into an Organization. *IEEE Data Eng. Bull.*, pages 8–20, 2016.

[148] K. Kulkarni and J. E. Michels. Temporal features in SQL:2011. *SIGMOD Rec.*, 41(3):34–43, 2012.

[149] O. Kuželka. *Fast Construction of Relational Features for Machine Learning*. PhD thesis, Czech Technical University, 2013.

[150] O. Kuželka and F. Železný. Block-wise construction of tree-like relational features with monotone reducibility and redundancy. *Mach. Learn.*, 83(2):163–192, 2011.

[151] N. Lachiche. Good and Bad Practices in Propositionalisation. In *AI*IA 2005 Adv. Artif. Intell.*, volume 3673 LNAI, pages 50–61. Springer, Berlin, 2005.

[152] M. Lan, C. L. Tan, and H. B. Low. Proposing a new term weighting scheme for text categorization. *Proc. Natl. Conf. Artif. Intell.*, 1:763–768, 2006.

[153] N. Landwehr. nFOIL: Integrating Naive Bayes and FOIL. *J. Mach. Learn. Res.*, 8:481–507, 2007.

[154] N. Landwehr, K. Kersting, and L. D. Raedt. Integrating Naïve Bayes and FOIL. *JMLR*, pages 481–507, 2007.

[155] N. Landwehr, A. Passerini, L. De Raedt, and P. Frasconi. kFOIL: Learning simple relational kernels. *AAAI*, 6:389–394, 2006.

[156] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[157] N. Lavrač, S. Džeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. *Proc. 5th Eur. Work. Sess. Learn.*, 482(OCTOBER 2006):265–281, 1991.

[158] O. Ledoit and M. Wolf. Improved estimation of the covariance matrix of stock returns with an application to portfolio selection. *J. Empir. Financ.*, 10(5):603–621, dec 2003.

[159] H. A. Leiva, A. Atramentov, and V. Honavar. A multi-relational decision tree learning algorithm. *Proc. 13th Int. Conf. Inductive Log. Program.*, pages 38–56, 2002.

[160] C. Lemke, M. Budka, and B. Gabrys. Metalearning: a survey of trends and technologies. *Artif. Intell.*, 44(1):117–130, 2015.

[161] W. S. Li and C. Clifton. SEMINT: a tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data Knowl. Eng.*, 33(1):49–84, 2000.

[162] H. Liu, X. Yin, and J. Han. An efficient multi-relational Naïve Bayesian classifier based on semantic relationship graph. In *Proc. 4th Int. Work. Multi-relational Min.*, pages 39–48, New York, 2005. ACM Press.

[163] Y. W. Liyanage, D. S. Zois, and C. Chelmis. On–the–Fly Joint Feature Selection and Classification. *arXiv*, pages 1–12, 2020.

[164] H. Lodhi and S. Muggleton. Is Mutagenesis still challenging? *Proc. 15th Int. Conf. Inductive Log. Program.*, pages 35–40, 2005.

[165] N. Logan. Package stringdist. Technical report, CRAN, 2016.

[166] V. López, A. Fernández, and F. Herrera. On the importance of the validation technique for classification with imbalanced datasets: Addressing covariate shift when data is skewed. *Inf. Sci. (Ny).*, 257:1–13, 2014.

[167] F. M. Lord. On the Statistical Treatment of Football Numbers. *Am. Psychol.*, 8(12):750–751, 1953.

[168] K. Lounici. High-dimensional covariance matrix estimation with missing observations. *Bernoulli*, 20(3):1029–1058, 2014.

[169] R. Magoulas and J. King. *2013 Data Science Salary Survey*. O'Reilly, 2013.

[170] V. M. Markowitz. Safe referential integrity and null constraint structures in relational databases. *Inf. Syst.*, 19(4):359–378, jun 1994.

[171] V. M. Markowitz and J. A. Makowsky. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. *IEEE Trans. Softw. Eng.*, 16(8):777–790, 1990.

[172] P. Mateti and N. Deo. On Algorithms for Enumerating All Circuits of a Graph. *SIAM J. Comput.*, 5(1):90–99, mar 1976.

[173] N. D. Mauro and F. Esposito. Ensemble relational learning based on selective propositionalization. *CoRR*, pages 1–10, 2013.

[174] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of Internet Portals with machine learning. *Inf. Retr. Boston.*, 3:127–163, 2000.

[175] A. McNab and D. A. Ladd. Information quality: The importance of context and trade-offs. In *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, pages 3525–3532, 2014.

[176] L. Meurice, F. J. B. Ruiz, J. H. Weber, and A. Cleve. Establishing referential integrity in legacy information systems - Reality bites! *Proc. - 30th Int. Conf. Softw. Maint. Evol. ICSME 2014*, pages 461–465, 2014.

[177] D. Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explor. Newsl.*, 3(1):6, jul 2001.

[178] D. Michie, S. Muggleton, D. Page, and A. Srinivasan. To the international computing community: A new east-west challenge. Technical report, Oxford University Computing laboratory, Oxford, 1994.

[179] M. Minsky and S. A. Papert. *Perceptrons. An Introduction to Computational Geometry*. MIT, jan 1969.

[180] S. Modi. Relational Classification using Multiple View Approach with Voting. *Int. J. Comput. Appl.*, 70(16):31–36, 2013.

[181] J. Moeyersoms and D. Martens. Including high-cardinality attributes in predictive models: A case study in churn prediction in the energy sector. *Decis. Support Syst.*, 72:72–81, 2015.

[182] J. G. Moreno-Torres, J. A. Saez, and F. Herrera. Study on the impact of partition-induced dataset shift on k-fold cross-validation. *IEEE Trans. Neural Networks Learn. Syst.*, 23(8):1304–1312, 2012.

[183] C. N. Morris. Parametric empirical Bayes inference: Theory and applications. *J. Am. Stat. Assoc.*, 78(381):47–55, 1983.

[184] F. Mosteller. *Data Analysis and Regression: A Second Course in Statistics*. Addison-Wesley Publishing Company, 1977.

[185] S. Muggleton. Inverse entailment and progol. *New Gener. Comput.*, 13(3-4):245–286, dec 1995.

[186] S. Muggleton, M. Bain, J. Hayes-Michie, and D. Michie. An Experimental Comparison of Human and Machine Learning Formalisms. In *Proc. Sixth Int. Work. Mach. Learn.*, pages 113–118. Morgan Kaufmann, 1989.

[187] S. Muggleton and L. De Raedt. Inductive Logic Programming: Theory and methods. *J. Log. Program.*, 19-20:629–679, 1994.

[188] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. Turaga. Learning Feature Engineering for Classification. In *Proc. Twenty-Sixth Int. Jt. Conf. Artif. Intell.*, number August, pages 2529–2535, 2017.

[189] S. Natarajan, K. Kersting, S. Joshi, and S. Saldana. Early prediction of coronary artery calcification levels using statistical relational learning. In *IAAI*, pages 1557–1562, Bellevue, 2013. AAAI Press.

[190] R. F. O. Neto, P. J. L. Adeodato, A. C. Salgado, D. R. Filho, G. R. Machado, D. R. de Carvalho Filho, and G. R. Machado. CoMoVi: a Framework for Data Transformation in Credit Behavioral Scoring Applications Using Model Driven Architecture. *SEKE 2014*, pages 286–291, 2014.

[191] B. Neupane, D. Richer, A. J. Bonner, T. Kibret, and J. Beyene. Network meta-analysis using R: a review of currently available automated packages. *PLoS One*, 9(12):e115065, jan 2014.

[192] J. Neville and D. Jensen. Collective classification with relational dependency networks. *J. Mach. Learn. Res.*, 8:77–91, 2003.

[193] J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning relational probability trees. In *KDD*, pages 625–630. ACM Press, 2003.

[194] J. Neville, D. Jensen, and B. Gallagher. Simple estimators for relational bayesian classifiers. *Proc. - IEEE Int. Conf. Data Mining, ICDM*, pages 609–612, 2003.

[195] R. S. Olson, W. L. Cava, Z. Mustahsan, A. Varik, and J. H. Moore. Data-driven advice for applying machine learning to bioinformatics problems. In *Biocomput. 2018*, pages 192–203. World Scientific, jan 2018.

[196] Oracle. Oracle SQL Developer Data Modeler, 2017.

[197] E. J. Pedersen, D. L. Miller, G. L. Simpson, and N. Ross. Hierarchical generalized additive models in ecology: An introduction with mgcv. *PeerJ*, 2019(5), 2019.

[198] L. Pedro de Jesus and P. Sousa. Selection of Reverse Engineering Methods for Relational Databases. *Proc. Third Eur. Conf. Softw. Maint.*, pages 194–197, 1998.

[199] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE TPAMI*, 27(8):1226–1238, aug 2005.

[200] C. Perlich and F. Provost. Distribution-based aggregation for relational learning with identifier attributes. *Mach. Learn.*, 62(1-2 SPEC. ISS.):65–105, 2006.

[201] M. Perovšek, A. Vavpetič, J. Kranjc, B. Cestnik, and N. Lavrač. Wordification : Propositionalization by unfolding relational data into bags of words. *Expert Syst. Appl.*, 42(17-18):6442–6456, 2015.

[202] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-Learning by Landmarking Various Learning Algorithms. In *ICML*, volume 951, pages 743–750, 2000.

[203] U. Pferschy and R. Staněk. Generating subtour elimination constraints for the TSP from pure integer solutions. *Cent. Eur. J. Oper. Res.*, pages 1–30, 2016.

[204] A. Popescul and L. H. Ungar. Structural Logistic Regression for Link Analysis. In *MRDM*, number August, pages 92–106, 2003.

[205] C. Preisach and L. Schmidt-Thieme. Relational ensemble classification. *Proc. - IEEE Int. Conf. Data Mining, ICDM*, pages 499–509, 2006.

[206] P. Probst, M. N. Wright, and A. Boulesteix. Hyperparameters and tuning strategies for random forest. *WIREs Data Min. Knowl. Discov.*, 9(3):1–19, may 2019.

[207] F. Provost. Machine learning from imbalanced data sets 101. *Proc. AAAI'2000 Work.*, pages 1–3, 2000.

[208] F. Provost, C. Perlich, and S. a. Macskassy. Relational learning problems and simple models. In *Proc. Relational Learn. Work. IJCAI-2003*, pages 116–120, 2003.

[209] L. D. Raedt. Attribute-value Learning versus Inductive Logic Programming: the Missing Links. In *ILP*, volume 1446 of *Lecture Notes in Computer Science*, pages 1–8, Berlin, Heidelberg, 1998. Springer.

[210] L. D. Raedt. *Inductive Logic Programming*, volume 1446 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 1998.

[211] L. D. Raedt. *Logical and Relational Learning*. Cognitive Technologies. Springer, Berlin, 2008.

[212] L. D. Raedt and W. V. Laer. Inductive Constraint Logic. *Algorithmic Learn. Theory*, pages 80–94, 1995.

[213] M. Rahmaninia and P. Moradi. OSFSMI: Online stream feature selection method based on mutual information. *Appl. Soft Comput. J.*, 68:733–746, 2018.

[214] A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*, volume 67. Cambridge University Press, New York, NY, USA, 2011.

[215] J. Ramon. *Clustering and Instance Based Learning in First Order Logic*. PhD thesis, KU Leuven, 2002.

[216] M. Reif and F. Shafait. Efficient feature size reduction via predictive forward selection. *Pattern Recognit.*, 47(4):1664–1673, 2014.

[217] Z. Reitermanová. Data Splitting. In *Week Dr. Students*, pages 31–36. Charles University, 2010.

[218] P. Reutemann, B. Pfahringer, and E. Frank. A Toolbox for Learning from Relational Data with Propositional and Multi-Instance Learners. In G. Webb and X. Yu, editors, *AI 2004 Adv. Artif. Intell.*, volume 3339, pages 1017–1023. Springer, 2004.

[219] J. Rice. The Algorithm Selection Problem. *Adv. Comput.*, 15(C):65–118, 1976.

[220] A. M. Richard and C. R. Williams. Distributed structure-searchable toxicity (DSSTox) public database network: A proposal. *Mutat. Res. - Fundam. Mol. Mech. Mutagen.*, 499(1):27–52, 2002.

[221] M. Richardson and P. Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2 SPEC. ISS.):107–136, feb 2006.

[222] D. R. Roberts, V. Bahn, S. Ciuti, M. S. Boyce, J. Elith, G. Guillera-Arroita, S. Hauenstein, J. J. Lahoz-Monfort, B. Schröder, W. Thuiller, D. I. Warton, B. A. Wintle, F. Hartig, and C. F. Dormann. Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography (Cop.).*, 40(8):913–929, 2017.

[223] C. Roever, N. Raabe, K. Luebke, U. Ligges, G. Szepannek, M. Zentgraf, and M. U. Ligges. The klaR package, 2006.

[224] A. Rostin, O. Albrecht, J. Bauckmann, F. Naumann, and U. Leser. A machine learning approach to foreign key discovery. In *12th Int. Work. Web Databases*, pages 1–6, 2009.

[225] S. Roy, U. Roy, S. Sciences, and W. Bengal. Performance Evaluation of Various Distance-based Data-Mining Classifiers on Typing Patterns for User Authentication / Identification. *Int. J. Innov. Res. Dev.*, 5(2):148–156, 2016.

[226] U. Rückert and S. Kramer. Margin-based first-order rule learning. *Mach. Learn.*, 70(2-3):189–206, nov 2008.

[227] A. Said and A. Bellogín. RiVal – A Toolkit to Foster Reproducibility in Recommender System Evaluation. *RecSys 2014 Proc. 8th ACM Conf. Recomm. Syst.*, pages 371–372, 2014.

[228] C. Sammut and G. I. Webb, editors. *Encyclopedia of Machine Learning*. Springer, New York, 2011.

[229] M. Samorani. Automatically Generate a Flat Mining Table with Dataconda. In *2015 IEEE Int. Conf. Data Min. Work.*, pages 1644–1647. IEEE, nov 2015.

[230] G. R. Sandercock, A. A. Ogunleye, D. A. Parry, D. D. Cohen, M. J. Taylor, and C. Voss. Athletic performance and birth month: Is the relative age effect more than just selection bias? *Int. J. Sports Med.*, 35(12):1017–1023, 2014.

[231] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. *Adv. Neural Inf. Process. Syst.*, 2017-Decem(Nips):4968–4977, 2017.

[232] SAP. Event Log Aggregation Scenario. Technical report, SAP, 2015.

[233] O. Schulte, B. Bina, B. Crawford, D. Bingham, and Y. Xiong. A hierarchy of independence assumptions for multi-relational Bayes net classifiers. *Proc. 2013 IEEE Symp. Comput. Intell. Data Min.*, pages 150–159, 2013.

[234] O. Schulte, Z. Qian, A. E. Kirkpatrick, X. Yin, and Y. Sun. Fast learning of relational dependency networks. *Mach. Learn.*, 103(3):377–406, jun 2016.

[235] O. Schulte and K. Routley. Aggregating predictions vs. aggregating features for relational classification. In *2014 IEEE Symp. Comput. Intell. Data Min.*, pages 121–128. IEEE, dec 2014.

[236] K. Sechidis, G. Tsoumakas, and I. Vlahavas. On the Stratification of Multi-label Data. In *Lect. Notes Comput. Sci.*, volume 6913, pages 145–158. Springer, Berlin, Heidelberg, 2011.

[237] S. Y. Sekeh, M. R. Ganesh, S. Banerjee, J. J. Corso, and A. O. Hero. A Geometric Approach to Online Streaming Feature Selection. *arXiv*, 2019.

[238] R. She, K. Wang, Y. Xu, and P. S. Yu. Pushing feature selection ahead of join. In *Proc. 2005 SIAM Int. Conf. Data Min.*, pages 1–5, 2005.

[239] J. H. Skillings and G. A. Mack. On the Use of a Friedman-Type Statistic in Balanced and Unbalanced Block Designs. *Technometrics*, 23(2):171–177, may 1981.

[240] J. Sobehart, S. Keenan, and R. Stein. Validation methodologies for default risk models. *Credit*, 4(May):51–56, 2000.

[241] P. Sondhi. Feature Construction Methods : A Survey. Technical report, Univeristy of Illinois at Urbana Champaign, 2010.

[242] G. Šourek, V. Aschenbrenner, F. Železný, and O. Kuželka. Lifted relational neural networks. *CEUR Workshop Proc.*, 1583:1–9, 2015.

[243] A. Srinivasan. The Aleph Manual. Technical report, University of Oxford, 2001.

[244] A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis predictions using ILP. *Inductive Log. Program.*, 1297:273–287, 1997.

[245] S. S. Stevens, N. Series, and N. Jun. On the Theory of Scales of Measurement. *Science (80-.).*, 103(2684):677–680, jun 1946.

[246] J. Swamidass, J. Chen, J. Bruand, P. Phung, L. Ralaivola, P. Baldi, S. J. Swamidass, J. Chen, J. Bruand, P. Phung, L. Ralaivola, and P. Baldi. Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, 21(SUPPL. 1):359–368, jun 2005.

[247] P. Szymański and T. Kajdanowicz. A Network Perspective on Stratification of Multi-Label Data. In *Int. Work. Learn. with Imbalanced Domains Theory Appl.*, pages 22–35, 2017.

[248] P. Szymański, T. Kajdanowicz, and K. Kersting. How is a data-driven approach better than random choice in label space division for multi-label classification? *Entropy*, 18(8):1–23, 2016.

[249] L. Tjen-Sien, L. Wei-Yin, and Y.-S. Shih. A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms. *Mach. Learn.*, 229:203–229, 1992.

[250] B. Trawiński, M. Smetek, Z. Telec, T. Lasota, M. Smętek, Z. Telec, T. Lasota, M. Smetek, Z. Telec, T. Lasota, B. T. Nski, M. S. M. Etek, Z. Telec, T. Lasota, B. Trawiński, M. Smętek, Z. Telec, T. Lasota, B. Trawinski, M. Smetek, Z. Telec, and T. Lasota. Nonparametric statistical analysis for multiple comparison of machine learning regression algorithms. *Int. J. Appl. Math. Comput. Sci.*, 22(4):867–881, jan 2012.

[251] A. Trotman, A. Puurula, and B. Burgess. Improvements to BM25 and language models examined. *ACM Int. Conf. Proceeding Ser.*, 27-28-Nove:58–65, 2014.

[252] P. Urbanke, A. Uhlig, and J. Kranz. A Customized and Interpretable Deep Neural Network for High-Dimensional Business Data - Evidence from an E-Commerce Application. In *ICIS 2017 Transform. Soc. with Digit. Innov.*, number November, pages 1–18, 2018.

[253] W. Uwents and H. Blockeel. Classifying relational data with neural networks. *Lect. Notes Artif. Intell. (Subseries Lect. Notes Comput. Sci.*, 3625:384–396, 2005.

[254] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: networked science in machine learning. *ACM SIGKDD Explor. Newsl.*, 15(2):49–60, jun 2014.

[255] Y. L. Varol. An algorithm to generate all topological sorting arrangements. *Comput. J.*, 24(1):83–84, jan 1981.

[256] C. Vens, J. Ramon, and H. Blockeel. Refining Aggregate Conditions in Relational Learning. In *Eur. Conf. Princ. Data Min. Knowl. Discov.*, pages 383–394, 2006.

[257] C. Vens, A. Van Assche, H. Blockeel, and S. Džeroski. First order random forests with complex aggregates. *Lect. Notes Comput. Sci.*, 3194/2004:323–340, 2004.

[258] J. Wainer. Comparison of 14 different families of classification algorithms on 115 binary datasets. *arXiv*, pages 1–36, 2016.

[259] K. M. Wittkowski. Friedman-Type statistics and consistent multiple comparisons for unbalanced designs with missing data. *J. Am. Stat. Assoc.*, 83(404):1163–1170, dec 1988.

[260] K. M. Wittkowski, E. Lee, R. Nussbaum, F. N. Chamian, and J. G. Krueger. Combining several ordinal measures in clinical studies. *Stat. Med.*, 23(10):1579–1592, 2004.

[261] M. Wolters and M. Kirsten. Exploring the use of linguistic features in domain and genre classification. In *Proc. ninth Conf. Eur. chapter Assoc. Comput. Linguist.*, pages 142–149, Morristown, NJ, USA, jun 1999. Association for Computational Linguistics.

[262] X. Wu, K. Yu, H. Wang, and W. Ding. Online Streaming Feature Selection. *Int. Conf. Mach. Learn.*, pages 1159–1166, 2010.

[263] X. Z. Wu and Z. H. Zhou. A unified view of multi-label performance measures. *34th Int. Conf. Mach. Learn. ICML 2017*, 8:5778–5791, 2017.

[264] C. Yan. Ant-FOIL: Integrating Ant Colony System and FOIL. In *2015 7th Int. Conf. Intell. Human-Machine Syst. Cybern.*, pages 559–562. IEEE, aug 2015.

[265] Y. Yang and T. M. Hospedales. Deep multi-task representation learning: A tensor factorisation approach. In *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, pages 1–9, 2017.

[266] X. Yin, J. Han, J. Yang, and P. S. Yu. CrossMine: Efficient classification across multiple database relations. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 3848 LNAI(6):172–195, 2006.

[267] D. You, X. Wu, L. Shen, S. Deng, Z. Chen, C. Ma, and Q. Lian. Online Feature Selection for Streaming Features Using Self-Adaption Sliding-Window Sampling. *IEEE Access*, 7:16088–16100, 2019.

[268] K. Yu, W. Ding, D. A. Simovici, and X. Wu. Mining emerging patterns by streaming feature selection. *KDD*, pages 60–68, 2012.

[269] K. Yu, X. Wu, W. Ding, and J. Pei. Towards Scalable and Accurate Online Feature Selection for Big Data. *Proc. - IEEE Int. Conf. Data Mining, ICDM*, 2015-Janua(January):660–669, 2015.

[270] L. Yu and H. Liu. Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution. In *ICML*, pages 856–863, Washington, D.C., 2003.

[271] F. Železný and N. Lavrač. Propositionalization-based relational subgroup discovery with RSD. *Mach. Learn.*, 62(1-2 SPEC. ISS.):33–63, 2006.

[272] X. Zeng and T. R. Martinez. Distribution-balanced stratified cross-validation for accuracy estimation. *J. Exp. Theor. Artif. Intell.*, 12(1):1–12, 2000.

[273] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. On multi-column foreign key discovery. *Proc. VLDB Endow.*, 3(1-2):805–814, 2010.

[274] J. Zhou, D. Foster, R. Stine, and L. Ungar. Streaming feature selection using alpha-investing. *KDD*, pages 384–393, 2005.

[275] P. Zhou, P. Li, S. Zhao, and X. Wu. Feature Interaction for Streaming Feature Selection. *IEEE Trans. Neural Networks Learn. Syst.*, pages 1–12, 2020.

[276] X. Zhou. Shrinkage Estimation of Log-odds Ratios for Comparing Mobility Tables. *Sociol. Methodol.*, 45(1):320–356, 2015.

[277] S. Zilberstein. Using Anytime Algorithms in Intelligent Systems. *AI Mag.*, 17(3):73–83, 1996.

[278] H. Zou and T. Hastie. Addendum: Regularization and variable selection via the elastic net. *J. R. Stat. Soc. Ser. B (Statistical Methodol.*, 67(5):768–768, nov 2005.

[279] J.-D. Zucker and J.-G. Ganascia. Learning structurally indeterminate clauses. In *8th Int. Conf. Inductive Log. Program.*, pages 235–244, Madison, Wisconsin, 1998.

[280] N. Zumel and J. Mount. vtreat: a data.frame Processor for Predictive Modeling. *arXiv*, 2016.

Reviewed Publications of the Author Relevant to the Thesis

[A.1] J. Motl and P. Kordík. Aggregate Function Generalization to Temporal Data. In *ICTAI-2021*, pages 5. IEEE Computer Society, 2021.

[A.2] J. Motl and P. Kordík. Discriminant Analysis on a Stream of Features. In *ADMA2021*, pages 15. Springer International Publishing, 2021.

[A.3] J. Motl and P. Kordík. Trend and Seasonality Elimination from Relational Data. In *ICTAI-2021*, pages 4. IEEE Computer Society, 2021.

[A.4] J. Motl and P. Kordík. Stratified Cross-Validation on Multiple Columns. In *ICTAI-2021*, pages 6. IEEE Computer Society, 2021.

[A.5] J. Motl and P. Kordík. Learning on a Stream of Features with Random Forest. In *CEUR Workshop Proc.*, pages 79–83, 2019.

[A.6] J. Motl and P. Kordík. Violation of Independence of Irrelevant Alternatives in Friedman's Test. *CEUR Workshop Proc.*, 2203:59–63, 2018.

[A.7] J. Motl and P. Kordík. Do We Need to Observe Features to Perform Feature Selection? In *CEUR Workshop Proc.*, pages 44–51, 2018.

[A.8] J. Motl and P. Kordík. Foreign Key Constraint Identification in Relational Databases. In *CEUR Workshop Proc.*, pages 106–111. CEUR, 2017.

The paper has been cited in:

- M. Cruz-Luna, C. Luna-Trejo, and J. Urbina-Fernández. Aseguramiento de integridad de datos para el sistema de encuestas del ITSH. *Rev. Tecnol. Comput.*, 2(6):15–21, 2018.
- P. Rahman, A. Nandi, and C. Hebert. Amplifying Domain Expertise in Clinical Data Pipelines. *JMIR Med. Informatics*, 8(11):24, nov 2020.

- J. Yoo, J. Lee, P.-L. Rhee, D. K. Chang, M. Kang, J. S. Choi, D. W. Bates, and W. C. Cha. Alert Override Patterns With a Medication Clinical Decision Support System in an Academic Emergency Department: Retrospective Descriptive Study. *JMIR Med. Informatics*, 8(11):14, nov 2020.
- М. Л. Цымблер. *Методы и алгоритмы поддержки целостности реляционных баз данных в приложениях классов OLAP и OLTP*. PhD thesis, Южно-уральский Государственный Университет, 2019.

[A.9] J. Motl and P. Kordík. Benchmarking Classifier Performance with Sparse Measurements. In J. Yaghob, editor, *CEUR Workshop Proc.*, pages 172–178. CEUR, 2015.

The paper has been cited in:

- V. Kassarnig and F. Wotawa. An Approach to Automatically Extract Predictive Properties from Nominal Attributes in Relational Databases. In *2018 IEEE Int. Conf. Big Data (Big Data)*, pages 4932–4939. IEEE, dec 2018.

Remaining Publications of the Author Relevant to the Thesis

[A.10] J. Motl *Predictor Factory.* Ph.D. Minimum Thesis, Czech Technical University, 2016.

[A.11] J. Motl *Predictor Factory.* Pražská technika 2/2016, 2016.

[A.12] J. Motl and O. Schulte. The CTU Prague Relational Learning Repository. *arXiv*, pages 7, nov 2015.

The paper has been cited in:

- M. Bahri, G. Bahl, and S. Zafeiriou. Binary Graph Neural Networks. *CVPR*, pages 1–14, dec 2021.
- G. Benito and M. Luna. *Inferencia de dependencias funcionales mediante funciones de similitud en minería de datos.* PhD thesis, Cinvestav, 2019.
- A. O. Chudý. *Detecting similarities of data domains using machine learning methods.* Master's thesis, Czech Technical University, 2019.
- B. Du, C. Yuan, R. Barton, T. Neiman, and H. Tong. Hypergraph Pre-training with Graph Neural Networks. *arXiv*, pages 1–12, 2021.
- S. Gholami. *Upgrading Bayesian network scores for multi-relational data.* Master's thesis, Simon Fraser University, 2016.
- F. A. Hofmann. *Tracer: A Machine Learning Approach to Data Lineage.* Master's thesis, Massachusetts Institute of Technology, 2020.
- A. C. José. *Implementación de un repositorio de objetos de aprendizaje durante la enseñanza de la geometria analítica en la carrera de matemática del instituto superior de ciencias de la educación de sumbe.* PhD thesis, Universidad de Matanzas, 2016.

- A. Kakadiya, S. Natarajan, and B. Ravindran. Relational Boosted Bandits. In *Proc. AAAI Conf. Artif. Intell.*, volume 35, pages 12123–12130. aaai.org, 2020.
- A. Komanduri. *Improving Bayesian Graph Convolutional Networks using Markov*. Bachelor's thesis, University of Arkansas, 2021.
- N. Lavrač, B. Škrlj, and M. Robnik-Šikonja. Propositionalization and embeddings: two sides of the same coin. *Mach. Learn.*, 109(7):1465–1507, 2020.
- A. Nazi, B. Ding, V. Narasayya, and S. Chaudhuri. Efficient estimation of inclusion coefficient using hyperloglog sketches. In *Proc. VLDB Endow.*, volume 11, pages 1097–1109. dl.acm.org, 2018.
- A. d. C. Oliveira-Filho. *Benchmark para Métodos de Consultas por Palavras-Chave a Bancos de Dados Relacionais*. PhD thesis, Universidade Federal de Goiás, 2018.
- C. Y. Park and K. B. Laskey. MEBN-RM: A Mapping between Multi-Entity Bayesian Network and Relational Model. *Appl. Sci.*, (9):26, apr 2019.
- N. Patki, R. Wedge, and K. Veeramachaneni. The Synthetic Data Vault. In *Proc. - 3rd IEEE Int. Conf. Data Sci. Adv. Anal. DSAA 2016*, pages 399–410. ieeexplore.ieee.org, 2016.
- N. Patki. *The Synthetic Data Vault: generative modeling for relational databases*. Master's thesis, 2016.
- F. Riahi and O. Schulte. Propositionalization for unsupervised outlier detection in multi-relational data. In *Proc. 29th Int. Florida Artif. Intell. Res. Soc. Conf. FLAIRS 2016*, pages 448–453. aaai.org, 2016.
- P. Scherer, H. Andres-Terre, P. Lio, and M. Jamnik. Decoupling feature propagation from the design of graph auto-encoders. *arXiv*, pages 1–7, 2019.
- M. Schleich. Structure-Aware Machine Learning over Multi-Relational Databases. In *Proc. ACM SIGMOD Int. Conf. Manag. Data*, pages 736–748. dl.acm.org, 2021.
- J. Schouterden, J. Davis, and H. Blockeel. LazyBum: Decision Tree Learning Using Lazy Propositionalization. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 11770 LNAI:98–113, 2020.
- B. J. Schreck. *Towards An Automatic Predictive Question Formulation*. PhD thesis, Massachusetts Institute of Technology, 2016.
- O. Schulte and S. Gholami. Locally Consistent Bayesian Network Scores for Multi-Relational Data. In *Proc. Twenty-Sixth Int. Jt. Conf. Artif. Intell.*, pages 2693–2700, California, aug 2017. International Joint Conferences on Artificial Intelligence Organization.
- T. Sciammarella and G. Zaverucha. Weight Your Words: The Effect of Different Weighting Schemes on Wordification Performance. *Lect. Notes Comput.*

Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 11770 LNAI:114–128, 2020.

- H. Seki and M. Nagao. An efficient Java implementation of a GA-based miner for relational association rules with numerical attributes. In *2017 IEEE Int. Conf. Syst. Man, Cybern.*, pages 2028–2033. IEEE, oct 2017.
- H. Seki and M. Nagao. An FCA approach to mining quantitative association rules from multi-relational data. *Int. J. Comput. Intell. Stud.*, 6(4):366–383, 2017.
- H. Seki and M. Nagao. Mining correlated association rules from multi-relational data using interval patterns. In *Int. Conf. Concept Lattices Their Appl.*, volume CLA2018, pages 47–58. cla2018.inf.upol.cz, 2018.
- W. P. Sousa and J. C. Silva. *Uma Técnica para Ranqueamento de Interpretações SQL Oriundas de Consultas com Palavras-chave Uma Técnica para Ranqueamento de Interpretações SQL Oriundas de Consultas com Palavras-chave*. PhD thesis, Universidade Federal de Goiás, 2017.
- S. V. Srikanthan and S. Xu. *The K-Multiple Instance Representation*. PhD thesis, Case Western Reserve University, 2019.
- L. Stanković, B. Scalzo, D. Mandic, S. Li, M. Daković, A. G. Constantinides, and M. Brajović. Data analytics on graphs Part III: Machine learning on graphs, from graph topology to applications. *Found. Trends Mach. Learn.*, 13(4):332–530, 2020.
- R.-A. Stoica. *R2PG-DM: A direct mapping from relational databases to property graphs*. PhD thesis, Eindhoven University of Technology, 2019.
- K. K. Veeramachaneni, N. Patki, K. P. Durg, J. S. Wilkinson, and S. R. Nochilur. Methods and apparatus for transforming and statistically modeling relational databases to synthesize privacy-protected anonymized data, 2020.
- P. Wijegunawardana, R. Gera, and S. Soundarajan. Node Classification with Bounded Error Rates. In *Springer Proc. Complex.*, pages 26–38, 2020.

Relevant Theses Supervised by the Author

[A.13] E. Vondráčková. *Metaučení přínosu příznaků.* Master's thesis, Czech Technical University, 2020.

[A.14] R. Severa. *Lineární diskriminační analýza na proudu příznaků.* Bachelor's thesis, Czech Technical University, 2020.

[A.15] P. Mück. *Klasifikace na temporálních relačních datech.* Master's thesis, Czech Technical University, 2018.

[A.16] A. Blažková. *Meta-learning na relačních datech.* Master's thesis, Czech Technical University, 2018.

Remaining Publications of the Author

[A.17] J. Motl and P. Kordík. Fast and exact audit scheduling optimization. *Springer SN Applied Sciences*, pages 17, 2020.

[A.18] J. F. Fullard, M. E. Hauberg, J. Bendl, G. Egervari, M. D. Cirnaru, S. M. Reach, J. Motl, M. E. Ehrlich, Y. L. Hurd, and P. Roussos. An atlas of chromatin accessibility in the adult human brain. *Genome Res.*, 28(8):1243–1252, 2018.

The paper has been cited in:

- N. Almeida, M. W. H. Chung, E. M. Drudi, E. N. Engquist, E. Hamrud, A. Isaacson, V. S. K. Tsang, F. M. Watt, and F. M. Spagnoli. Employing core regulatory circuits to define cell identity. *EMBO J.*, 40(10), 2021.
- P. Baloni, C. C. Funk, B. Readhead, and N. D. Price. Systems modeling of metabolic dysregulation in neurodegenerative diseases. *Curr. Opin. Pharmacol.*, 60:59–65, oct 2021.
- J. Bendl, M. E. Hauberg, K. Girdhar, E. Im, J. M. Vicari, S. Rahman, P. Dong, R. Misir, S. M. Reach, P. Apontes, B. Zeng, W. Zhang, G. Voloudakis, R. A. Nixon, V. Haroutunian, G. E. Hoffman, J. F. Fullard, and P. Roussos. The three-dimensional landscape of chromatin accessibility in Alzheimer's disease. *bioRxiv*, 2021.
- V. Berge-Seidl, L. Pihlstrøm, and M. Toft. Integrative analysis identifies bHLH transcription factors as contributors to Parkinson's disease risk mechanisms. *Sci. Rep.*, 11(1), 2021.
- S. M. Bilinovich, K. Lewis, B. L. Thompson, J. W. Prokop, and D. B. Campbell. Environmental epigenetics of diesel particulate matter toxicogenomics. *Int. J. Environ. Res. Public Health*, 17(20):1–14, 2020.
- B. B. Boutwell and M. A. White. Gene regulation and the architecture of complex human traits in the genomics era. *Curr. Opin. Psychol.*, 27:93–97, 2019.

- K. Bowles, D. A. Pugh, K. Farrell, N. Han, J. TCW, and Y. Liu. 17q21. 31 Sub-Haplotypes Underlying H1-Associated Risk for Parkinson's Disease and Progressive Supranuclear Palsy Converge on Altered Glial Regulation. *Cell*, 2019.
- J. Bryois, N. Skene, T. F. Hansen, L. Kogelman, H. Watson, L. Brueggeman, G. Breen, C. Bulik, E. Arenas, J. Hjerling-Leffler, and P. Sullivan. Genetic Identification of Cell Types Underlying Brain Complex Traits Yields Novel Insights Into the Etiology of Parkinson's Disease. *bioRxiv*, 2019.
- D. Cameron. *Gene regulation in microglia and genetic risk for complex brain disorders*. PhD thesis, Cardiff University, 2019.
- N. V. Carullo and J. J. Day. Genomic Enhancers in Brain Health and Disease. *Genes (Basel).*, 10(1):43, jan 2019.
- N. V. Carullo, R. A. Phillips, R. C. Simon, S. A. Roman Soto, J. E. Hinds, A. J. Salisbury, J. S. Revanna, K. D. Bunner, L. Ianov, F. A. Sultan, K. E. Savell, C. A. Gersbach, and J. J. Day. Enhancer RNAs predict enhancer–gene regulatory links and are critical for enhancer function in neuronal systems. *Nucleic Acids Res.*, 48(17):9550–9570, 2020.
- J. Cedernaes and J. Bass. You are when you eat: On circadian timing and energy balance. *J. Clin. Invest.*, 131(1), 2021.
- A. Chawla, C. Nagy, and G. Turecki. Chromatin profiling techniques: Exploring the chromatin environment and its contributions to complex traits. *Int. J. Mol. Sci.*, 22(14), 2021.
- Y. Chen, X. Ding, S. Wang, P. Ding, Z. Xu, J. Li, M. Wang, R. Xiang, X. Wang, H. Wang, Q. Feng, J. Qiu, F. Wang, Z. Huang, X. Zhang, G. Tang, and S. Tang. A single-cell atlas of mouse olfactory bulb chromatin accessibility. *J. Genet. Genomics*, 48(2):147–162, 2021.
- M.-D. Cirnaru, S. Song, K.-T. Tshilenge, C. Corwin, J. Mleczko, C. G. Aguirre, H. Benlhabib, J. Bendl, P. Apontes, J. Fullard, J. C. Muncunill, A. Reyah, A. Nik, P. Carlsson, P. Roussos, S. Mooney, L. Ellerby, and M. Ehrlich. Transcriptional and epigenetic characterization of early striosomes identifies Foxf2 and Olig2 as factors required for development of striatal compartmentation and neuronal phenotypic differentiation. *bioRxiv*, 2020.
- M. R. Corces, A. Shcherbina, S. Kundu, M. J. Gloudemans, L. Frésard, J. M. Granja, B. H. Louie, T. Eulalio, S. Shams, S. T. Bagdatli, M. R. Mumbach, B. Liu, K. S. Montine, W. J. Greenleaf, A. Kundaje, S. B. Montgomery, H. Y. Chang, and T. J. Montine. Single-cell epigenomic analyses implicate candidate causal variants at inherited risk loci for Alzheimer's and Parkinson's diseases. *Nat. Genet.*, 52(11):1158–1168, 2020.
- P. Dong, G. E. Hoffman, P. Apontes, J. Bendl, S. Rahman, M. B. Fernando, B. Zeng, J. M. Vicari, W. Zhang, K. Girdhar, K. G. Townsley, R. Misir,

C. Consortium, K. J. Brennand, V. Haroutunian, G. Voloudakis, J. F. Fullard, and P. Roussos. Transcribed enhancers in the human brain identify novel disease risk mechanisms. *bioRxiv*, 2021.

- P. Dong, G. E. Hoffman, P. Apontes, J. Bendl, S. Rahman, M. B. Fernando, B. Zeng, J. M. Vicari, W. Zhang, K. Girdhar, K. G. Townsley, R. Misir, t. C. Consortium, K. J. Brennand, V. Haroutunian, G. Voloudakis, J. F. Fullard, and P. Roussos. Population-level variation of enhancer expression identifies novel disease mechanisms in the human brain. *bioRxiv*, 2021.
- G. Egervari, D. Akpoyibo, T. Rahman, J. F. Fullard, J. E. Callens, J. A. Landry, A. Ly, X. Zhou, N. Warren, M. E. Hauberg, G. Hoffman, R. Ellis, J. M. N. Ferland, M. L. Miller, E. Keller, B. Zhang, P. Roussos, and Y. L. Hurd. Chromatin accessibility mapping of the striatum identifies tyrosine kinase FYN as a therapeutic target for heroin use disorder. *Nat. Commun.*, 11(1), 2020.
- P. O. Estève, U. S. Vishnu, H. G. Chin, and S. Pradhan. Visualization and Sequencing of Accessible Chromatin Reveals Cell Cycle and Post-HDAC inhibitor Treatment Dynamics. *J. Mol. Biol.*, 432(19):5304–5321, 2020.
- J. F. Fullard, A. W. Charney, G. Voloudakis, A. V. Uzilov, V. Haroutunian, and P. Roussos. Assessment of somatic single-nucleotide variation in brain tissue of cases with schizophrenia. *Transl. Psychiatry*, 9(1), 2019.
- T. Gao and J. Qian. Eagle: An algorithm that utilizes a small number of genomic features to predict tissue/ cell type-specific enhancer-gene interactions. *PLoS Comput. Biol.*, 15(10), 2019.
- T. Gao and J. Qian. EnhancerAtlas 2.0: An updated resource with enhancer annotation in 586 tissue/cell types across nine species. *Nucleic Acids Res.*, 48(D1):58–64, 2020.
- M. García-Cabezas, H. Barbas, and B. Zikopoulos. Parallel Development of Chromatin Patterns, Neuron Morphology, and Connections: Potential for Disruption in Autism. *Front. Neuroanat.*, 12, 2018.
- K. Girdhar, G. E. Hoffman, J. Bendl, S. Rahman, P. Dong, W. Liao, L. Brown, O. Devillers, B. S. Kassim, J. R. Wiseman, R. Park, E. Zharovsky, R. Jacobov, E. Flatow, A. Kozlenkov, T. Gilgenast, J. S. Johnson, L. Couto, M. A. Peters, J. E. Phillips-Cremins, C.-G. Hahn, R. E. Gur, C. A. Tamminga, D. A. Lewis, V. Haroutunian, S. Dracheva, B. K. Lipska, S. Marenco, M. Kundakovic, J. F. Fullard, Y. Jiang, P. Roussos, and S. Akbarian. Acetylated Chromatin Domains Link Chromosomal Organization to Cell- and Circuit-level Dysfunction in Schizophrenia and Bipolar Disorder. *bioRxiv*, 2021.
- P. Giusti-Rodríguez, L. Lu, Y. Yang, C. A. Crowley, X. Liu, J. Bryois, I. Juric, J. S. Martin, S. C. Allred, N. Ancalade, J. J. Crowley, J. Guintivano, P. R. Jansen, G. J. Jurjus, G. Mahajan, G. Rajkowska, J. C. Overholser, S. Pochareddy, G. Santpere, J. E. Savage, Y. Shin, A. F. Pardiñas, M. C.

O'Donovan, M. J. Owen, D. Posthuma, N. Sestan, C. A. Stockmeier, J. T. R. Walters, G. E. Crawford, Y. Li, F. Jin, M. Hu, Y. Li, and P. F. Sullivan. Schizophrenia and a high-resolution map of the three-dimensional chromatin interactome of adult and fetal cortex. *bioRxiv*, 2018.

- P. Giusti-Rodríguez, L. Lu, Y. Yang, C. A. Crowley, X. Liu, I. Juric, J. S. Martin, A. Abnousi, S. C. Allred, N. E. Ancalade, N. J. Bray, G. Breen, J. Bryois, C. M. Bulik, J. J. Crowley, J. Guintivano, P. R. Jansen, G. J. Jurjus, Y. Li, G. Mahajan, S. Marzi, J. Mill, M. C. O'Donovan, J. C. Overholser, M. J. Owen, A. F. Pardiñas, S. Pochareddy, D. Posthuma, G. Rajkowska, G. Santpere, J. E. Savage, N. Sestan, Y. Shin, C. A. Stockmeier, J. T. Walters, S. Yao, G. E. Crawford, F. Jin, M. Hu, Y. Li, and P. F. Sullivan. Using three-dimensional regulatory chromatin interactions from adult and fetal cortex to interpret genetic results for psychiatric disorders and cognitive traits. *bioRxiv*, 2018.
- A. A. Golicz, P. E. Bayer, P. L. Bhalla, J. Batley, and D. Edwards. Pangenomics Comes of Age: From Bacteria to Plant and Animal Applications. *Trends Genet.*, 36(2):132–145, 2020.
- P. Gontarz, S. Fu, X. Xing, S. Liu, B. Miao, V. Bazylianska, A. Sharma, P. Madden, K. Cates, A. Yoo, A. Moszczynska, T. Wang, and B. Zhang. Comparison of differential accessibility analysis strategies for ATAC-seq data. *Sci. Rep.*, 10(1), 2020.
- G. B. Gutierrez. *Transcriptome dynamics of neurodegeneration using single-cell and long-read approaches*. PhD thesis, University of California, 2021.
- J. L. Haigh, A. Adhikari, N. A. Copping, T. Stradleigh, A. A. Wade, R. Catta-Preta, L. Su-Feher, I. Zdilar, S. Morse, T. A. Fenton, A. Nguyen, D. Quintero, S. Agezew, M. Sramek, E. J. Kreun, J. Carter, A. Gompers, J. T. Lambert, C. P. Canales, L. A. Pennacchio, A. Visel, D. E. Dickel, J. L. Silverman, and A. S. Nord. Deletion of a non-canonical regulatory sequence causes loss of Scn1a expression and epileptic phenotypes in mice. *Genome Med.*, 13(1), 2021.
- M. M. Halstead. *Dynamic Chromatin Accessibility in Livestock Genomes: Characterizing the Epigenetic Regulome from Fertilization to Differentiation*. PhD thesis, University of California, 2020.
- M. E. Hauberg, J. Creus-Muncunill, J. Bendl, A. Kozlenkov, B. Zeng, C. Corwin, S. Chowdhury, H. Kranz, Y. L. Hurd, M. Wegner, A. D. Børglum, S. Dracheva, M. E. Ehrlich, J. F. Fullard, and P. Roussos. Common schizophrenia risk variants are enriched in open chromatin regions of human glutamatergic neurons. *Nat. Commun.*, 11(1), 2020.
- P. W. Hook. *Leveraging mouse genomic data to prioritize genes and variants associated with common, complex neurological disease*. PhD thesis, Johns Hopkins University, 2020.

- P. W. Hook and A. S. McCallion. Leveraging mouse chromatin data for heritability enrichment informs common disease architecture and reveals cortical layer contributions to schizophrenia. *Genome Res.*, 30(4):528–539, 2020.
- S. Jäkel and A. Williams. What Have Advances in Transcriptomic Technologies Taught us About Human White Matter Pathologies? *Front. Cell. Neurosci.*, 14, 2020.
- H. Jeong, I. Mendizabal, S. Berto, P. Chatterjee, T. Layman, N. Usui, K. Toriumi, C. Douglas, D. Singh, I. Huh, T. M. Preuss, G. Konopka, and S. V. Yi. Cell-type and cytosine context-specific evolution of DNA methylation in the human brain. *bioRxiv*, 2020.
- H. Jeong, I. Mendizabal, S. Berto, P. Chatterjee, T. Layman, N. Usui, K. Toriumi, C. Douglas, D. Singh, I. Huh, T. M. Preuss, G. Konopka, and S. V. Yi. Evolution of DNA methylation in the human brain. *Nat. Commun.*, 12(1), 2021.
- I. M. Kaplow, M. E. Wirthlin, A. J. Lawler, A. R. Brown, M. Kleyman, and A. R. Pfenning. Predicting lineage-specific differences in open chromatin across dozens of mammalian genomes. *bioRxiv*, 2020.
- R. K. Kawaguchi, Z. Tang, S. Fischer, R. Tripathy, P. K. Koo, and J. Gillis. Exploiting marker genes for robust classification and characterization of single-cell chromatin accessibility. *bioRxiv*, pages 1–18, 2021.
- K. Koshi-Mano, T. Mano, M. Morishima, S. Murayama, A. Tamaoka, S. Tsuji, T. Toda, and A. Iwata. Neuron-specific analysis of histone modifications with post-mortem brains. *Sci. Rep.*, 10(1), 2020.
- B. Lai, S. Qian, H. Zhang, S. Zhang, A. Kozlova, J. Duan, X. He, and J. Xu. Predicting Epigenomic Functions of Genetic Variants in 1 the Context of Neurodevelopment via Deep Transfer 2 Learning 3. *bioRxiv*, 2021.
- J. T. Lambert, L. Su-Feher, K. Cichewicz, T. L. Warren, I. Zdilar, Y. Wang, K. J. Lim, J. Haigh, S. J. Morse, T. W. Stradleigh, E. Castillo, V. Haghani, S. Moss, H. Parolini, D. Quintero, D. Shrestha, D. Vogt, L. C. Byrne, and A. S. Nord. Parallel functional testing identifies enhancers active in early postnatal mouse brain. *bioRxiv*, 2021.
- M. Langmyhr, S. P. Henriksen, C. Cappelletti, W. D. van de Berg, L. Pihlstrøm, and M. Toft. Allele-specific expression of Parkinson's disease susceptibility genes in human brain. *Sci. Rep.*, 11(1), 2021.
- C.-X. Lin, H.-D. Li, C. Deng, W. Liu, S. Erhardt, F.-X. Wu, X.-M. Zhao, J. Wang, D. Wang, and B. Hu. Genome-wide prediction and integrative 1 functional characterization of Alzheimer's 2 disease-associated genes. *bioRxiv*, 2021.
- Y. Liu. Clinical implications of chromatin accessibility in human cancers. *Oncotarget*, 11(18):1666–1678, 2020.

- D. Mancarella and C. Plass. Epigenetic signatures in cancer: proper controls, current challenges and the potential for clinical translation. *Genome Med.*, 13(1), 2021.
- L. Mangnier, C. Joly-Beauparlant, A. Droit, S. Bilodeau, and A. Bureau. Cis-Regulatory Hubs: a Relevant 3D Model to Study the Genetics of Complex Diseases with an Application to Schizophrenia. *bioRxiv*, 2021.
- J. K. Mich, L. T. Graybuck, E. E. Hess, J. T. Mahoney, Y. Kojima, Y. Ding, S. Somasundaram, J. A. Miller, B. E. Kalmbach, C. Radaelli, B. B. Gore, N. Weed, V. Omstead, Y. Bishaw, N. V. Shapovalova, R. A. Martinez, O. Fong, S. Yao, M. Mortrud, P. Chong, L. Loftus, D. Bertagnolli, J. Goldy, T. Casper, N. Dee, X. Opitz-Araya, A. Cetin, K. A. Smith, R. P. Gwinn, C. Cobbs, A. L. Ko, J. G. Ojemann, C. D. Keene, D. L. Silbergeld, S. M. Sunkin, V. Gradinaru, G. D. Horwitz, H. Zeng, B. Tasic, E. S. Lein, J. T. Ting, and B. P. Levi. Functional enhancer elements drive subclass-selective expression from mouse to primate neocortex. *Cell Rep.*, 34(13), 2021.
- L. Minnoye, G. K. Marinov, T. Krausgruber, L. Pan, A. P. Marand, S. Secchia, W. J. Greenleaf, E. E. M. Furlong, K. Zhao, R. J. Schmitz, C. Bock, and S. Aerts. Chromatin accessibility profiling methods. *Nat. Rev. Methods Prim.*, 1(1), 2021.
- J. Mukai, E. Cannavò, G. W. Crabtree, Z. Sun, A. Diamantopoulou, P. Thakur, C. Y. Chang, Y. Cai, S. Lomvardas, A. Takata, B. Xu, and J. A. Gogos. Recapitulation and Reversal of Schizophrenia-Related Phenotypes in Setd1a-Deficient Mice. *Neuron*, 104(3):471–487, 2019.
- D. Pal, S. Dutta, D. P. Iyer, U. Bhaduri, and S. M. R. Rao. Identification of PAX6 and NFAT4 as the transcriptional regulators of lncRNA Mrhl in neuronal progenitors. *bioRxiv*, 2021.
- C. J. Playfoot, J. Duc, S. Sheppard, S. Dind, A. Coudray, E. Planet, and D. Trono. Transposable elements and their KZFP controllers are drivers of transcriptional innovation in the developing human brain. *bioRxiv*, 2020.
- S. K. Powell, C. P. O'Shea, S. R. Shannon, S. Akbarian, and K. J. Brennand. Investigation of Schizophrenia with Human Induced Pluripotent Stem Cells. *Adv. Neurobiol.*, 25:155–206, 2020.
- P. Rajarajan and S. Akbarian. Use of the epigenetic toolbox to contextualize common variants associated with schizophrenia risk. *Dialogues Clin. Neurosci.*, 21(4):407–416, 2019.
- E. Ramamurthy, G. Welch, J. Cheng, Y. Yuan, L. Gunsalus, D. A. Bennett, L. H. Tsai, and A. Pfenning. Cell type-specific histone acetylation profiling of Alzheimer's Disease subjects and integration with genetics. *bioRxiv*, 2020.
- R. H. Reynolds, J. Hardy, M. Ryten, and S. A. Gagliano Taliun. Informing disease modelling with brain-relevant functional genomic annotations. *Brain*, 142(12):3694–3712, 2019.

- L. F. Rizzardi, P. F. Hickey, A. Idrizi, R. Tryggvadóttir, C. M. Callahan, K. E. Stephens, S. D. Taverna, H. Zhang, S. Ramazanoglu, K. D. Hansen, and A. P. Feinberg. Human brain region-specific variably methylated regions are enriched for heritability of distinct neuropsychiatric traits. *Genome Biol.*, 22(1), 2021.
- D. Rocks, I. Jaric, L. Tesfa, J. M. Greally, M. Suzuki, and M. Kundakovic. Cell type-specific chromatin accessibility analysis in the mouse and human brain. *Epigenetics*, 2021.
- P. Shooshtari, S. Feng, V. Nelakuditi, J. Foong, M. Brudno, and C. Cotsapas. OCHROdb: A comprehensive, quality checked database of open chromatin regions from sequencing data. *bioRxiv*, 2018.
- K. Spiess and H. Won. Regulatory landscape in brain development and disease. *Curr. Opin. Genet. Dev.*, 65:53–60, 2020.
- C. Srinivasan, B. N. Phan, A. J. Lawler, E. Ramamurthy, A. R. Brown, I. M. Kaplow, M. E. Wirthlin, and A. R. Pfenning. Addiction-associated genetic variants implicate brain cell type-and region-specific cis-1 regulatory elements in addiction neurobiology 2. *bioRxiv*, 2020.
- P. F. Sullivan and D. H. Geschwind. Defining the Genetic, Genomic, Cellular, and Diagnostic Architectures of Psychiatric Disorders. *Cell*, 177(1):162–183, 2019.
- I. Tripodi, M. Chowdhury, and R. Dowell. ATAC-seq signal processing and recurrent neural networks can identify RNA polymerase activity. *bioRxiv*, 2019.
- I. J. Tripodi. *Inferring mechanisms of toxicity from differential genomics and semantic knowledge representations*. PhD thesis, University of Colorado at Boulder, 2020.
- I. J. Tripodi, M. Chowdhury, M. Gruca, and R. D. Dowell. Combining signal and sequence to detect RNA polymerase initiation in ATAC-seq data. *PLoS One*, 15(4), 2020.
- K. Van den Berge, H.-J. Chou, H. Roux de Bézieux, K. Street, D. Risso, J. Ngai, and S. Dudoit. Normalization benchmark of ATAC-seq datasets shows the importance of accounting for GC-content effects. *bioRxiv*, 2021.
- Y. Wang, X. Zhang, Q. Song, Y. Hou, J. Liu, Y. Sun, and P. Wang. Characterization of the chromatin accessibility in an Alzheimer's disease (AD) mouse model. *Alzheimer's Res. Ther.*, 12(1), 2020.
- M. E. Wirthlin, I. M. Kaplow, A. J. Lawler, J. He, B. D. N. Phan, A. Brown, W. Stauffer, and A. Pfenning. The Regulatory Evolution of the Primate Fine-Motor System. *bioRxiv*, 2020.
- Y. Xiang, B. Cakir, and I. H. Park. Deconstructing and reconstructing the human brain with regionally specified brain organoids. *Semin. Cell Dev. Biol.*, 111:40–51, 2021.

- M. Xu, Q. Liu, R. Bi, Y. Li, C. Zeng, Z. Yan, Q. Zheng, X. Li, C. Sun, M. Ye, X.-J. Luo, M. Li, D.-F. Zhang, and Y.-G. Yao. A multiple-causal-gene-cluster model underlying GWAS signals of Alzheimer's disease. *bioRxiv*, 2021.
- F. Yan, D. R. Powell, D. J. Curtis, and N. C. Wong. From reads to insight: A hitchhiker's guide to ATAC-seq data analysis. *Genome Biol.*, 21(1), 2020.
- S. Yin, K. Lu, T. Tan, J. Tang, J. Wei, X. Liu, X. Hu, H. Wan, W. Huang, Y. Fan, D. Xie, and Y. Yu. Transcriptomic and open chromatin atlas of high-resolution anatomical regions in the rhesus macaque brain. *Nat. Commun.*, 11(1), 2020.
- D. T. Youmans. *Recruitment of Polycomb Repressive Complex 2 to Chromatin by Accessory Proteins*. PhD thesis, University of Colorado at Boulder, 2021.
- S. Zhang, H. Zhang, Y. Zhou, M. Qiao, S. Zhao, A. Kozlova, J. Shi, A. R. Sanders, G. Wang, K. Luo, S. Sengupta, S. West, S. Qian, M. Streit, D. Avramopoulos, C. A. Cowan, M. Chen, Z. P. Pang, P. V. Gejman, X. He, and J. Duan. Allele-specific open chromatin in human iPSC neurons elucidates functional disease variants. *Science (561-565).*, 369(6503):561–565, 2020.
- W. Zhang, G. Voloudakis, V. M. Rajagopal, B. Readhead, J. T. Dudley, E. E. Schadt, J. L. Björkegren, Y. Kim, J. F. Fullard, G. E. Hoffman, and P. Roussos. Integrative transcriptome imputation reveals tissue-specific and shared biological mechanisms mediating susceptibility to complex traits. *Nat. Commun.*, 10(1), 2019.
- B. Zhao, T. Li, S. M. Smith, D. Xiong, X. Wang, Y. Yang, T. Luo, Z. Zhu, Y. Shan, N. Matoba, Q. Sun, Y. Yang, M. E. Hauberg, J. Bendl, J. F. Fullard, P. Roussos, W. Lin, Y. Li, J. L. Stein, and H. Zhu. Common variants contribute to intrinsic human brain functional networks. *bioRxiv*, 22(919), 2020.
- B. Zhao, T. Li, Y. Yang, X. Wang, T. Luo, Y. Shan, Z. Zhu, D. Xiong and Y. Li. The comprehensive genetic architecture of brain white matter. *bioRxiv*, 2020.
- B. Zhao, T. Li, Y. Yang, X. Wang, T. Luo, Y. Shan, Z. Zhu, D. Xiong, M. E. Hauberg, J. Bendl, J. F. Fullard, P. Roussos, Y. Li, J. L. Stein, and H. Zhu. Common genetic variation influencing human white matter microstructure. *Science.*, 372(6548), 2021.
- Y. Zhou, Y. Sun, D. Huang, and M. J. Li. epiCOLOC: Integrating Large-Scale and Context-Dependent Epigenomics Features for Comprehensive Colocalization Analysis. *Front. Genet.*, 11, 2020.

[A.19] J. Motl and W. Nie. What Makes a Fairytale Five Factors of Fairytales. *Int. J. Comput. Linguist. Appl.*, 8(1):9, 2017.

The paper has been cited in:

- F. Muhabat, R. A. Mangrio, B. Kazemian, S. Sadia, and M. Noor. Arabic Fairy-Tales: An Analysis of Hatim Tai's Story within Propp's Model. *SSRN Electron. J.*, 1(2):21–25, 2015.

[A.20] J. Motl. *Supporting the Diagnosis of Borreliosis by Machine Learning Methods.* Master thesis, Czech Technical University, 2013.

The paper has been cited in:

- P. Kordík. *Meta-learning Templates: Beyond Algorithm Selection in Data Mining.* habilitation thesis, Czech Technical University, 2016.
- P. Kordík, J. Černý, and T. Frýda. Discovering predictive ensembles for transfer learning and meta-learning. *Mach. Learn.*, 107(1):177–207, 2018.

Appendix A

Appendix

A.1 Feature Functions

Feature function	Description
accented	Does the string contain an accented character?
affinity	Similar to Weight of Evidence (WoE), but without normalization and transformation with logarithm. For decision trees it should be as informative as WoE, but the calculation should be faster. For interpretability or logistic regression, WoE is still preferred.
aggregate	Apply aggregate functions on a numerical attribute. Following aggregate functions were selected for being supported in many databases: avg, min, max and $stddev\_samp$.
aggregate distinct	Apply aggregate functions on distinct values.
aggregate frame	Apply aggregate functions on a numerical attribute. In comparison to aggregate pattern, this pattern limits the history to the specific amount of days.
aggregate range	As many databases do not support range aggregate, calculate it with min and max. The value is given with: $f(x) = max(x) - min(x)$, where x is an attribute.
aggregate robust	Returns the n-th lowest/biggest value instead of the lowest-/biggest value. This pattern is also known as "ranked aggregate".
aggregate subgroup	Apply aggregate functions on a subgroup of numerical attributes. The subgroup is defined by a value of a nominal attribute.
aggregate text length	An aggregate of string lengths.
aggregate woe	An aggregate of weight of evidence.
change count	Count number of changes in time.

change ratio	Ratio of count of increases to count of decreases in time.
coefficient of variation	This is a special scenario of ratio of aggregates. It is defined as the ratio of the standard deviation to the mean.
concentration	Any statistic (e.g. percentage of the most frequent value) calculated from the distribution of the nominal child field with child filter F.
correlation	Calculate Pearson correlation of an attribute with time.
count	Number of records (including nulls).
decimals	Number of decimal digits.
direct field	Returns the attribute unchanged.
direct field as nominal	Converts a numeric attribute to nominal attribute.
distinct count	Number of distinct values (all nulls are counted as one value).
duplication ratio	Proportion of records with a duplicate value.
email domain	Extracts suffix after @ from a string.
entropy	Shannon entropy for discrete distributions.
entropy continuous	Entropy estimate for numerical attributes.
existential count	Count of occurrences of a specific value.
frequency	Relative frequency of the given nominal value in the population.
geometric mean	One of the three classical Pythagorean means.
harmonic mean	One of the three classical Pythagorean means.
integral	The integral (area under curve) computed with the trapezoidal rule.
intercept	Time, when the attribute crosses zero, as estimated with a regression line.
ip local	Is it a local IP address?
is null	Is the value null?
leave one out	A regularized version of WoE.
log product	Log product of numbers.
mean absolute difference	Measures dispersion (MAD).
mode	The most frequent value.
moments	Skewness and kurtosis.
nominal change	Count of changes of a nominal attribute in time.
null ratio	Real-world data commonly contain contain missing values. And sometimes they are not missing at random. Thus we mine them.
odds ratio	The pattern is from: Feature selection for unbalanced class distribution and Naive Bayes, Mladenic & Grobelnik.
percentile	Percentiles.
prefix	Extracts prefix of a constant length from a string.
product	Calculate product of numbers.
ratio	Ratio of the two attributes X and Y.

segment comparison	Value of a numerical attribute divided by an aggregate (avg, min,...) of the nominal field.
slope	Slope of an attribute in time as estimated with a linear regression.
suffix	Extracts suffix of a constant length from a string.
text woe	Converts a text attribute to bags-of-words, which get further converted to WoE.
text dirty woe	Converts a dirty text attribute to bags-of-character-trigrams, which get further converted to WoE.
text length	Returns number of characters.
time aggregate	Just like aggregate pattern, but on time attributes. In comparison to "Time since pattern", this pattern is not relative to the target date.
time aggregate since	Time since some date in the past till the time when the prediction is required.
time aggregate since event	Time since event E till the time when the prediction is required.
time date diff	Difference between two dates in days.
time date part	The day/day-of-week/month of the event.
time date since	Days since some date in the past till the moment when the prediction is required.
time day part	Identify, whether it's working hour, morning, evening or night.
time diff	Difference between two times.
time frequency	Count of records divided by duration.
time from peak	Time from the peak.
time is weekend	Divide the date into workday/weekend.
time month part	Identify, whether it's the beginning, middle or end of the month.
time of peak	Time when the signal peaks.
time part	The hour/minute of the event.
time range	Max time - min time.
time since	Time since some date in the past till the time when the prediction is required. The selection of the date is driven with aggregate functions like min, max and avg.
time woe	Weight of evidence for time.
title case	Is the string in "Title Case" format? In other words, are all initial letters in upper case and everything else in lower case?
woe	Weight of evidence.
woe limit	Weight of evidence. Considers only cases with at least 10 samples.
zero crossing	Count of how many times zero was crossed.
zeros	Returns the count of trailing zeros. If the value is zero, we return zero.

Table A.1: List of implemented feature functions.

A.2 Leaking Features

Leakage is the introduction of information about the label to the model, which should not be legitimately available [125]. Some of the most common sources of leaks in relational databases are discussed in the following paragraphs.

Target column Features that use *target column* naturally leak information about the target into the feature. If these features are not evaluated on records with the blinded *target column* but on the training data, they have overly optimistic predictive power.

Absent temporal constraint If the *target table* contains a *target timestamp*, but non-target tables are not constrained with temporal constraints, data from the future (relative to *target timestamp*) may leak into features.

Attributes from the target table In praxis, it is not uncommon that a *target table* contains "forgotten" attributes that were used as intermediate steps during the construction of the *target column*. These auxiliary attributes should not be used in features as they are not going to be available in the deployment.

A.3 Computational Complexity

The following text lists the computational complexity of a feature insert per class when the current QDA model contains n instances and d features. We assume that the scoring (testing) data have the same size as the training data and that they do not change from one iteration to another. The computation complexity of Algorithm 2 line-by-line:

2. $\mathcal{O}(n+d)$ to estimate the mean of the new feature of length n and extend the vector of the means from d to $d+1$ by the estimated mean. If we have to allocate a new vector and copy all the data into the new vector, the computational complexity is $\mathcal{O}(n+d)$. However, if the vector is pre-allocated to a size of at least $d+1$, we have to only write down a scalar, bringing the computational complexity to $\mathcal{O}(n)$.
3. $\mathcal{O}(nd)$ to center the new feature and extend the matrix of the centered features from $n \times d$ to $n \times (d+1)$. If we have to allocate a new matrix and copy all the data into the new matrix, the computational complexity is $\mathcal{O}(nd)$. However, if the matrix is pre-allocated to a size of at least $n \times (d+1)$, we only need to write down the vector of length n, which is $\mathcal{O}(n)$.
4. $\mathcal{O}(nd)$ to perform vector-matrix multiplication of vector length n and matrix $n \times (d+1)$. The complexity can be derived from the basic matrix multiplication of one

$m \times n$ matrix and one $n \times d$ matrix with a computational complexity of $\mathcal{O}(mnd)$. The matrix is then multiplied by a scalar with $\mathcal{O}(nd)$.

5. $\mathcal{O}(d^2)$ for Cholesky insert. The Cholesky insert of column x into the triangular matrix R_t can be computed with block Cholesky factorization, as described in [90, Algorithm 4.2.3]:

Algorithm 4: `cholinsert` of feature $\boldsymbol{x}$ into triangular R_t. The used functions are standard Octave functions.

Input: R_t, $\boldsymbol{x}$, where t is the count of features
Output: R_{t+1}

1 $R21 = \texttt{mrdivide}(\boldsymbol{x}[1:t]^\intercal,\ R_t)$
2 $R22 = \texttt{sqrt}(\boldsymbol{x}[\text{t+1}]$ - $R21 \cdot R21^\intercal)$
3 $R_{t+1} = [R_t,\ R21^\intercal; \texttt{zeros}(1, \texttt{size}(R_t, 2)), R22]$

The first line is a forward substitution of the $d \times d$ triangular matrix and a vector d. This is $\mathcal{O}(d^2)$ (see [90, Section 3.1]). The second line consists of a dot product of a vector of length d, one scalar subtraction, and one scalar square root. In total, this is $\mathcal{O}(d)$. The assembly of all the pieces together is $\mathcal{O}(d^2)$. The total computational complexity of `cholinsert` is $\mathcal{O}(d^2)$.

6. $\mathcal{O}(1)$ to calculate one scalar logarithm, one scalar multiplication, and one scalar addition.

7. $\mathcal{O}(nd)$ to perform scalar subtraction from a matrix $n \times (d+1)$, which is $\mathcal{O}(nd)$, and backward substitution update, which consists of a vector-matrix multiplication of a vector length d and a matrix $n \times d$, division of a vector of length d with a scalar, and subtraction of a vector length n with another vector of length n, which in total is also $\mathcal{O}(nd)$.

8. $\mathcal{O}(nd)$ to update the scores. The update consists of element-wise squaring of a matrix of size $n \times (d+1)$, column-wise summation of the $n \times (d+1)$ matrix, and two additions between a scalar and a vector of length n. In total, this is $\mathcal{O}(nd)$. However, only the right-most column of matrix A changes. This can be exploited to reduce the computational complexity of this code line to $\mathcal{O}(n)$.

The biggest terms in this breakdown are $\mathcal{O}(nd)$ and $\mathcal{O}(d^2)$. Because we do not enforce that $n \geq d$, it is appropriate to write the overall computational complexity as $\mathcal{O}(nd+d^2)$.

The computational complexity of QDA from scratch, as considered in the text, consists of the same steps, with the computational complexity multiplied by d, giving $\mathcal{O}(nd^2 + d^3)$. This result is comparable to $\mathcal{O}(ndt + t^3)$, where $t = \min(n, d)$, as reported by [28], where SVD decomposition is used instead of Cholesky factorization.

A.4 User Manual

> In theory, there is no difference between theory and practice. But, in practice, there is.
>
> Jan van de Snepscheut

The following paragraphs describe a common process of application Predictor Factory from the point of view of a consultant that is asked to extract a table of features. The process begins with data stored in a relational database and has the following steps (with time estimate in parenthesis):

- Get access to the data (1 day – 3 months)
- Data quality review and remedy (1 day – 2 weeks)
- Identify/build a target table (2 days)
- Run Predictor Factory (1 day)
- Check results (1 day)
- Model (1 day)
- Interpret the model (1 day)
- Test the model in the real world (2 months and more)

A.4.0.1 Data access

Obtaining access to the data is generally a difficult task because a consultant must pass multiple password prompts and configuration quests. Even if you are assured that everything is set up and tested, you had better reserve a day just to access the data.

A.4.0.2 Data quality

Once you pass the initial level and get access to the data, you have to identify a schema that contains your data. If the tables are spread over multiple schemas, bear yourself for a fight. Predictor Factory requires all the input tables to be in a single schema[1]. Furthermore, forget about a quick win if the tables are not linked together with foreign key constraints. If the foreign key constraints are not set, you may either define them in the database or in a supplementary XML file called `foreignConstraint.xml` in the config directory. If you can write into the database or make a copy of the database and change the database schema, it is highly advisable to set the constraints in the database rather than in the XML. The reason is that a database may warn you about a mismatch of data types in the foreign constraint and violating records.

[1] This requirement originated in the fact that foreign key constraints can be by default set only across tables in a single schema.

Experienced users may argue that 2 days is a gross underestimation of the time necessary to fix the data. However, these 2 days are to write off the unsalvageable data, not to fix them.

A.4.0.3 Target Table

Once we have all the data in a single schema and with foreign constraints set up, we have to identify the target table. In most cases, however, the target table is not readily available and must be manually created. The target table should contain target id – an identifier of the subject, for which we wish to make a prediction (typically *customer_id*), the target timestamp – a timestamp of the time when we wish to have the prediction (for example, for each first day in a month), and the target (the historically known outcomes that we wish to predict). While it may look overly generous to reserve 2 days for the definition of the target table, if the target table is wrong, everything, including features, model, and predictions, will be wrong. Hence, 1 day is reserved for the creation of the target table and another day for meditation, whether the target is well defined. As a rudimentary sanity check, check that a combination of the target id and the target are unique in the target table (for example, with a single unique constraint on both attributes together). Also check, that neither the target timestamp or the target id contain null values (for example, with not null constraints).

If you want to score new data, append (row-wise) the scoring data to the target table with null values in place of thetarget.

A.4.0.4 Run Predictor Factory

Start Predictor Factory by opening predictorFactory.jar. If java is not installed, call the following command from a command line to use the attached Java Runtime Environment (JRE)[2]:

```
jre\bin\java.exe -Xmx30G -d64 -jar PredictorFactory.jar
```

The flag `Xmx` specifies the maximum memory that Predictor Factory can allocate. The flag `d64` says to run it in a 64-bit environment to be able to utilize more than 4GB of memory.

After starting Predictor Factory, log in to the database. If you have problems with the connection, check the following:

- Validate that the same connection parameters work in another tool.
- It is not a firewall problem.
- The version of the JDBC driver in the *lib* directory of Predictor Factory is up to date.

[2]The attached JRE is for 64 bit Windows. If you use a different operating system, you have to install/copy JRE for your system.

After connecting to the database, set configuration in the tabs one after another from left to right. On the last tab, start the execution by clicking on "Run" button.

A.4.0.5 Check Results

It is advisable to follow the following checklist:

- Was *mainSample*, the table with calculated features, produced in the output schema?
- Does it have the same count or rows as the target table?
- Does it have some features?
- Were temporal constraints correctly identified? To check it, open *journal_table* in the output schema and look for *temporal_constraint* attribute.
- Were all the tables, which were selected for processing, correctly propagated? To check it, see *is_ok* attribute in *journal_table*.