Meta-learning Templates: Beyond Algorithm Selection in Data Mining

by

Pavel Kordík

A habilitation thesis submitted to the Faculty of Information Technology, Czech Technical University in Prague

Specialization: Informatics

Prague, June 2016
Abstract and contributions

The problem of algorithm selection can be found in several domains. Choosing right algorithm for given problem is hard, given number of possible algorithms, their combinations and parameters. This habilitation thesis presents an unifying methodology proposing how to target this problem in a systematic way suggesting a multi stage process to reach the holy grail: an automated algorithm selection solution. We show, how this process can be deployed in several domains such as predictive modeling, data clustering, recommender systems, and others. We analyze and suggest how to overcome problems preventing full automation of the algorithm selection. We target mainly the problem of ill-posed performance evaluation that is present in all domains.

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

1. A unifying methodology for algorithm selection automation in multiple domains.

2. Successful deployment of this methodology in the domain of supervised modeling. Demonstration of automated algorithm selection including parameters and ensembles of predictive modeling algorithms. Analysis of results over large set of benchmarking datasets and for concept drift data.


5. Algorithm selection automation in data preprocessing, relational data mining, combinatorial optimization and data mining in general.

Keywords:
algorithm selection, data mining, meta-learning, predictive modeling, clustering, recommender systems, optimization, relational data mining, knowledge flows, machine learning, automation.
Acknowledgements

Big thank goes to my collaborators, mostly doctoral and master students. Without their continuous effort, we would not be able to elaborate the topic of automated algorithm selection so broadly, deploy it in so many domains, get interesting results and real-world applications. Jan Černý co-authored the research in the predictive modeling domain, Tomáš Bartoň in the data clustering domain, Tomáš Řehořek and Radek Baltyzar in the domain of recommender systems and knowledge flow optimization, Miroslav Čepek and Jan Motl in the data preprocessing domain and Oleg Kovářík in the combinatorial optimization domain.

Thanks to Pavel Tvrdík, dean and founder of our faculty for the environment he created and maintains. Thanks to Mirek Šnorek, my ex-supervisor for being still around.

Thanks to my friend Ivan Šimeček and Peter Fišer and who kindly shared all materials requested for the habilitation process making the administrative procedure far less painful for me.
Dedication

To my family, for their infinite patience, love and care.
# Contents

List of Figures viii

List of Tables xi

1 Introduction 1
   1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
   1.2 Structure of the Thesis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

2 Background and State-of-the-Art 5

3 Overview of our Approach 7
   3.1 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   3.2 Introducing general TASA process . . . . . . . . . . . . . . . . . . . . . . . 8
      3.2.1 Algorithm Quality and Evaluation Criteria . . . . . . . . . . . . . . . . 9
      3.2.2 Algorithm Ensembling Strategies . . . . . . . . . . . . . . . . . . . . . . 9
      3.2.3 Meta-Learning and Meta-Optimization Templates . . . . . . . . . . . . . . 10
      3.2.4 Template Optimization Strategies . . . . . . . . . . . . . . . . . . . . . . 11
      3.2.5 Meta-Features, Meta-Database and Meta-Data Mining . . . . . . . . . . . 12
      3.2.6 Algorithm Selection Automation and Applications . . . . . . . . . . . . . . 13

4 Algorithm Selection in Predictive Modeling 15
   4.1 Algorithm Quality and Evaluation Criteria . . . . . . . . . . . . . . . . . . . . 15
   4.2 Algorithm Ensembling Strategies . . . . . . . . . . . . . . . . . . . . . . . . 18
      4.2.1 Base algorithms and ensemble methods . . . . . . . . . . . . . . . . . . . . 18
         4.2.1.1 Base algorithms - models and classifiers . . . . . . . . . . . . . . . . . . . 18
         4.2.1.2 Ensembling algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
   4.3 Meta-Learning Templates . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
   4.4 Template Optimization Strategies . . . . . . . . . . . . . . . . . . . . . . . . 21
      4.4.1 Evolving templates by genetic programming . . . . . . . . . . . . . . . . . . . 22
         4.4.1.1 Encoding the template into genotype . . . . . . . . . . . . . . . . . . . . 22
         4.4.1.2 Fitness function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
         4.4.1.3 Initializing a population . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
         4.4.1.4 An evolutionary algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . 23
### 4. Feature Selection and Model Exploration

- 4.4.1.5 Feature selection ................................................. 25
- 4.4.1.6 Global heuristics ............................................. 26
- 4.4.2 Exploring models produced by templates ................ 27
- 4.4.3 Evolutionary Programming in anytime learning ........ 28

### 4.5 Meta-Features, Meta-Database and Meta-Data Mining

- 4.5.1 Data sets description ........................................... 29
- 4.5.2 Dividing the data ............................................... 30
- 4.5.3 Hierarchical ensemble ......................................... 31
- 4.5.4 Template overfitting ......................................... 32
- 4.5.5 Templates evolved for various data sets ............... 35
- 4.5.6 Similarity and substitutability of templates .......... 35
- 4.5.7 Meta-database for templates ............................... 37

### 4.6 Algorithm Selection Automation and Applications

- 4.6.1 Templates and transfer learning ............................. 40

### 5. Algorithm Selection in Clustering

- 5.1 Algorithm Quality and Evaluation Criteria ............... 43
  - 5.1.1 External validation ........................................ 46
  - 5.1.2 Usefulness of single criteria ............................ 46
- 5.2 Ensembling Strategies in Clustering .......................... 52
  - 5.2.1 Multi-objective ensembles ................................ 52
  - 5.2.2 Ensemble Generation Strategies ....................... 53
  - 5.2.3 Consensus Functions ....................................... 53
- 5.3 Meta-Learning Templates in Clustering ..................... 54
- 5.4 Template Optimization Strategies ............................ 55
  - 5.4.1 Evolutionary optimization of clustering ............... 55
- 5.5 Meta-Features, Meta-Database and Meta-Data Mining .... 58

### 6. Algorithm Selection in Recommender Systems

- 6.1 Algorithm Quality and Evaluation Criteria ................ 61
  - 6.1.1 Performance Measures in Recommender Systems ...... 62
    - 6.1.1.1 Accuracy ............................................. 62
    - 6.1.1.2 Catalog Coverage .................................. 64
  - 6.1.2 Popularity Bias ........................................... 64
  - 6.1.3 k-Nearest Neighbors Algorithm .......................... 65
  - 6.1.4 Association Rules ......................................... 66
    - 6.1.4.1 Rule-Quality Measures ............................ 67
    - 6.1.4.2 Rule-based Recommendation ....................... 68
  - 6.1.5 Pareto optimality ......................................... 68
  - 6.1.6 Biasing k-NN and Association Rules ................. 69
  - 6.1.7 Testing Datasets ......................................... 70
  - 6.1.8 Results ................................................... 71
  - 6.1.9 Online Evaluation ....................................... 72
CONTENTS

6.2 Algorithm Ensembling Strategies .............................................. 77
6.3 Meta-Learning Templates ....................................................... 77
6.4 Template Optimization Strategies ............................................ 79

7 Algorithm Selection in Other Domains ........................................ 87
  7.1 Algorithm selection in data preprocessing ................................ 87
  7.2 Automated workflow optimization ......................................... 89
    7.2.1 Structural aspects of KD processes .................................. 89
      7.2.1.1 Preprocessing Grammar ........................................ 90
      7.2.1.2 Learner Grammar ............................................... 90
    7.3 Other application domains ............................................... 94

8 Conclusion ................................................................................... 95

Bibliography .................................................................................. 97
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Algorithm optimization and selection framework as proposed in [115]</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Automated algorithm selection framework proposed in [136]</td>
<td>6</td>
</tr>
<tr>
<td>3.1</td>
<td>Problem space, algorithm space and performance space.</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>Proposed process Towards Algorithm Selection Automation (TASA)</td>
<td>8</td>
</tr>
<tr>
<td>3.3</td>
<td>Parallel and sequential algorithm ensembles</td>
<td>9</td>
</tr>
<tr>
<td>3.4</td>
<td>Templates of different complexity</td>
<td>10</td>
</tr>
<tr>
<td>3.5</td>
<td>General templates and specific templates</td>
<td>11</td>
</tr>
<tr>
<td>3.6</td>
<td>Meta-database for algorithm recommendation and seeding templates</td>
<td>12</td>
</tr>
<tr>
<td>3.7</td>
<td>Automated algorithm selection</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>Validation error and training time are two quality criteria</td>
<td>16</td>
</tr>
<tr>
<td>4.2</td>
<td>Size of training set to maximize performance in anytime learning</td>
<td>17</td>
</tr>
<tr>
<td>4.3</td>
<td>Algorithm selection and model selection</td>
<td>17</td>
</tr>
<tr>
<td>4.4</td>
<td>Meta-learning templates in predictive modeling</td>
<td>21</td>
</tr>
<tr>
<td>4.5</td>
<td>Executing the meta-learning template</td>
<td>22</td>
</tr>
<tr>
<td>4.6</td>
<td>Leaf node mutations in genetic programming of templates</td>
<td>24</td>
</tr>
<tr>
<td>4.7</td>
<td>Inner node mutations in genetic programming of templates</td>
<td>24</td>
</tr>
<tr>
<td>4.8</td>
<td>Evolutionary programming for template optimization</td>
<td>25</td>
</tr>
<tr>
<td>4.9</td>
<td>Meta-learning template evolved on the Spiral problem</td>
<td>28</td>
</tr>
<tr>
<td>4.10</td>
<td>Anytime performance of template optimization using different approaches</td>
<td>29</td>
</tr>
<tr>
<td>4.11</td>
<td>The workflow evaluating sensitivity to overfitting and stability of solution</td>
<td>32</td>
</tr>
<tr>
<td>4.12</td>
<td>Template overfitting and distribution of base algorithms</td>
<td>33</td>
</tr>
<tr>
<td>4.13</td>
<td>Distribution of base algorithms in templates</td>
<td>34</td>
</tr>
<tr>
<td>4.14</td>
<td>Templates evolved on individual data sets serialized into text description</td>
<td>36</td>
</tr>
<tr>
<td>4.15</td>
<td>Transfer learning with meta-learning templates</td>
<td>38</td>
</tr>
<tr>
<td>4.16</td>
<td>Average performance of individual templates over all data sets</td>
<td>39</td>
</tr>
<tr>
<td>4.17</td>
<td>The positive effect of meta-database to optimization speed</td>
<td>39</td>
</tr>
<tr>
<td>4.18</td>
<td>Strong overfitting of templates when data used repetitively</td>
<td>40</td>
</tr>
<tr>
<td>4.19</td>
<td>Updating templates and models</td>
<td>41</td>
</tr>
<tr>
<td>5.1</td>
<td>Sorted clustering results of Zoo data set with Point-BiSerial criterion</td>
<td>47</td>
</tr>
<tr>
<td>5.2</td>
<td>Visual comparison of two criteria on the Iris data</td>
<td>49</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.3</td>
<td>Clustering of unsupervised criteria and clustering problems.</td>
<td>51</td>
</tr>
<tr>
<td>5.4</td>
<td>Cluster ensemble</td>
<td>54</td>
</tr>
<tr>
<td>5.5</td>
<td>Example of meta-learning template for clustering.</td>
<td>55</td>
</tr>
<tr>
<td>5.6</td>
<td>Multi-objective clustering templates return more solutions.</td>
<td>57</td>
</tr>
<tr>
<td>5.7</td>
<td>Ranking clustering algorithms (templates) on benchmarking datasets.</td>
<td>59</td>
</tr>
<tr>
<td>6.1</td>
<td>Results for the User-kNN Algorithm on the MovieLens 10M Dataset</td>
<td>71</td>
</tr>
<tr>
<td>6.2</td>
<td>The effect of K-NN regularization on benchmarking data sets.</td>
<td>73</td>
</tr>
<tr>
<td>6.3</td>
<td>Results for the Association Rules Algorithm on the 1M Movielens Dataset</td>
<td>74</td>
</tr>
<tr>
<td>6.4</td>
<td>Recall-Coverage Tradeoff for Offline Evaluation on 1sthereanydeal.com</td>
<td>74</td>
</tr>
<tr>
<td>6.5</td>
<td>Recall-Coverage Tradeoff for Offline Evaluation (confidential dataset)</td>
<td>75</td>
</tr>
<tr>
<td>6.6</td>
<td>The value of Recall-Coverage harmonic mean for KNN (left) and AR (right) algorithms on the LastFM 2k dataset. The maximum can be observed for 50-NN with ( \beta = 0.3 ) and for ( s_{\text{min}} = 0.0001 ) with ( \beta = 0.5 ) in case of AR.</td>
<td>76</td>
</tr>
<tr>
<td>6.7</td>
<td>Comparison of Association Rules, kNN and their ensembles on LastFM 2k and Movielens 1M datasets.</td>
<td>78</td>
</tr>
<tr>
<td>6.8</td>
<td>Example of recommendation meta-learning template.</td>
<td>79</td>
</tr>
<tr>
<td>6.9</td>
<td>Online evaluation of recommender algorithms with fitness surrogate.</td>
<td>80</td>
</tr>
<tr>
<td>6.10</td>
<td>Penalization prevents multiple bellow average individuals in creating an above-average region.</td>
<td>83</td>
</tr>
<tr>
<td>7.1</td>
<td>Example dataset with subsequences of preprocessing methods.</td>
<td>88</td>
</tr>
<tr>
<td>7.2</td>
<td>Parameter search space for the \textit{N-Power Calculator} method and the Non Linear dataset [156].</td>
<td>88</td>
</tr>
<tr>
<td>7.3</td>
<td>Meta-learning templates for workflows are Directed Acyclic Graphs.</td>
<td>89</td>
</tr>
<tr>
<td>7.4</td>
<td>Preprocessing Grammar and its Semantics</td>
<td>91</td>
</tr>
<tr>
<td>7.5</td>
<td>Learners Grammar and Its Semantics.</td>
<td>92</td>
</tr>
<tr>
<td>7.6</td>
<td>Bagging Meta-Learner Grammar and Its Semantics.</td>
<td>93</td>
</tr>
<tr>
<td>7.7</td>
<td>A sample tree evolved on Ecoli dataset.</td>
<td>93</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Data sets are obtained mostly from the UCI repository and are small to medium-sized. ............................... 30
4.2 The table shows the difference between evolutions using learning/validation/testing sets or learning/validation sets respectively. .................................................. 31
4.3 Evolved templates under various depth constrains. ................................................................. 32
4.4 Frequency of occurrence in winning templates for all methods (Meta-models, Meta-classifiers, models and classifiers). .................................................. 36
5.1 Relation of unsupervised and supervised criteria on various datasets. .................. 50
5.2 Multi-objective evolution of clustering templates. ................................................................. 56
6.1 \(k\) vs. Number of Clicks in isthereanydeal.com On-line Experiment ............................................. 75
Chapter 1

Introduction

1.1 Motivation

Quality improvement of data mining models is not only an important research issue, but it is also daily live of most of data scientists and engineers in companies around the world. The quality of models have several components. For most researchers, generalization ability is the most important one. The other components are speed, interpretability, simplicity, robustness in terms of handling noisy or concept drift data, etc. In this thesis, we will focus mostly to improvement of generalization ability of data mining models, but the other components will also be taken into account. For example when we aim to improve generalization ability of predictive models (so called predictive power), we cannot completely ignore speed of training or recall. The aim is to optimize multiple quality criteria and provide data mining models that fit user needs. It is obvious that some criteria are contradicting - e.g. you often need to decrease the predictive power to get less complex and more interpretable models. The task of data mining algorithm and model selection often resembles constrained optimization problems; get algorithm/model with the best generalization accuracy under maximal training time and model complexity constrains.

In data mining, thousands of machine learning algorithms are available. Each algorithm was designed for a specific data mining task (e.g. classification, regression, recommendation or clustering). Algorithms have parameters that influence their behavior and need to be optimized for given problem. Often these parameters have so significant impact that it is hard to distinguish whether it is the same algorithm with different parameter value or a different algorithm (e.g. type of kernel in the Support Vector Machine algorithm [154]).

The problem of deciding which algorithm should be used for given problem is targeted in many studies [16, 127, 136], however solutions to algorithm selection are often very simple and suboptimal. As you can see in recent data mining competitions [3], most of approaches used were similar to trial and error of inefficient exhaustive search. We have to take into account that one algorithm can consist of several base algorithms (ensembles) and this what typically happens when you examine winning solutions over past years.

The algorithm selection problem is the most critical in domains with variety of algorithms available. In the field of supervised modeling or optimization, the problem is quite
well defined and measurable, in other domains (e.g. recommender systems, data clustering or visualization) the criterion of success is often complex metric which complicate algorithm selection problem even further.

This thesis addresses the problem of algorithm selection, optimization and automation in several domains. It attempts to answer several related questions such as:

- How to build the most accurate model that outperforms existing models?
- What is the best modeling algorithm to use for given problem?
- How to combine algorithms in a way that the quality of models is maximized?
- How to automate the process of algorithm/model selection?

We attempt to answer these questions in detail not only in the area of predictive modeling, but also for recommender systems and clustering.

Furthermore, we define an unified process that help us to answer algorithm selection questions over several domains. The process start with definition of model quality and typically analyses multiple model quality criteria. Then it considers model ensembling and introduces meta-learning templates into the domain. After that, state space of templates, constrains and search strategies are examined. Template optimization strategies and building blocks are addressed. Meta-features are constructed, benchmarks performed, a meta-database created. Meta-features are evaluated and redefined. In final stage, algorithm selection is automated using the outcomes obtained in preceding stages. Trough the thesis, we demonstrate how this process can be deployed in several domains.

1.2 Structure of the Thesis

The thesis is organized into eight chapters as follows:

1. Introduction: Describes the motivation behind our efforts together with our goals. There is also a list of contributions of this thesis.

2. Background and State-of-the-Art: Introduces the reader to the necessary theoretical background and surveys the current state-of-the-art.


5. Algorithm Selection in Clustering: Discusses our results in clustering.

6. Algorithm Selection in Recommender Systems: Discusses our results in recommender systems.
7. *Algorithm Selection in Other Domains*: Discusses our results in the domain of data preprocessing, relational data mining, data mining workflows and combinatorial optimization.

8. *Conclusions*: Summarizes outcomes of our research, suggests possible topics for further research, and concludes the thesis.
Chapter 2

Background and State-of-the-Art

The problem of algorithm selection is not new. No free lunch theorems [159] explain why the average performance of any pair of algorithms across all possible problems is identical. When several algorithm quality criteria are taken into account, no single algorithm is capable of pareto-dominating (see 6.1.5) other algorithms over large set of problems. Existence of many alternative algorithms with is unavoidable reality and many studies [123, 127, 68] targeted the algorithm selection problem in several domains. One of the first domain was the field of general black-box optimization [127], but recent overview [115] shows that it is still an unsolved problem and active research field.

Figure 2.1: Algorithm optimization and selection framework as proposed in [115].

It is an interdisciplinary problem as mentioned in [46] and there is not much overlap in the algorithm selection research over different domains, largely because different terminology. In this thesis, we are targeting this issue by proposing general unified algorithm selection framework. We applied this framework in multiple domains and corresponding background and state-of-the-art approaches are discussed separately in later sections.
Recent popularity of ensemble methods or combination of algorithms makes the problem of algorithm selection much harder. Luckily, a lot has been done in the field of meta-learning where the algorithm recommendation is well studied problem \cite{28, 49}. In the framework we propose, we extended meta-learning approaches to selection, recommendation and optimization of algorithm ensembles.

Optimization of algorithms and their ensembles is core of the machine learning research in all domains. Often, hyperparameter tunning is performed as a grid search or random search. Low efficiency of such approaches is obvious, moreover structural optimization algorithms has to be solved independently and that is not good, because the two optimization problems are tightly connected. In \cite{21}, Gaussian Processes and Tree-structured Parsen Estimators modeling distributions are shown to be significantly better. Memetic algorithms \cite{116} are another heuristics that can navigate the algorithm space and search for best performing algorithms and parameters. In our study \cite{94} we showed how to optimize structure and parameters of predictive models. Topology and Weight Evolving Artificial Neural Networks \cite{125} is another similar example.

The process of algorithm selection was already generalized to more domains. In \cite{136}, a general meta-learning framework streamlining automated algorithm selection and algorithm refinement is proposed (Figure 2.2). The goal of this thesis is the same, but we put the emphasis to different stages of the process. In our view, algorithm selection based on meta-features should not be the core component of the framework, but an optional component, because it is not always beneficial, as we show bellow.

Then we apply proposed process to predictive modeling, clustering, recommender systems and other domains. Domain specific background and state of the art are discussed in the individual chapters.
Chapter 3

Overview of our Approach

In this chapter, we introduce a process intended to unify algorithm selection methodology across multiple domains of data mining and optimization. The goal is to automate the process to the highest possible extent.

3.1 Problem Statement

Similarly to [127, 115], we define algorithm selection problem as follows: \( F \) is the problem space such as predictive modeling or discrete optimization. \( f \in F \) are individual instances of problems for which we select an algorithm \( \alpha_0 \in A \), where \( A \) is the algorithm space. In the original framework, algorithms \( \alpha_0, \alpha_1, ..., \alpha_n \) are carefully selected diverse and complementary algorithms. In our case, algorithms can be even complex hierarchical ensembles of algorithms. The performance space \( P \) contains several performance criteria \( \rho_0, \rho_1, ... \rho_n \) to evaluate performance components of given algorithm on particular problem \( \rho_i(f_j, \alpha_k) \).

The goal of the algorithm selection is to find an algorithm \( \alpha_0 \) or set of algorithms maximizing performance criteria for given problem. Such definition is very vague given performance criteria are not specified. We can assume, that most important performance criteria are the accuracy of solution and the speed of the algorithm. When speed is not an issue, task is to find an algorithm returning best performing solution in term of accuracy. However even this is not as simple, as it seems to be. The quality of the solution has to

![Diagram](image)

Figure 3.1: For given problem from the problem space, one can select an algorithm or algorithms from the algorithm space that optimize given performance criteria from the performance state.
be estimated, and in most domains, it is very difficult, as we show through the thesis. For multiple performance criteria, the goal of the algorithm selection can be returning set of algorithms $\alpha_0, \alpha_1, ..., \alpha_l$ pareto-dominating (see 6.1.5) performance criteria $\rho_0, \rho_1, ..., \rho_n$.

Parameters of algorithms and ensembling techniques make algorithm space huge. It is also very difficult to measure similarity of algorithms as small change in parameters or in the ensemble structure can lead to a significant difference in the algorithm performance.

Above mentioned difficulties make problem of algorithm selection very hard and complex. With exploding popularity of big data, data analytics and machine learning, number of data science projects multiplies in the industry. In most of these projects, the algorithm selection problem is present and solved with various degree of success. Many data engineers and even scientists strive to perform algorithm selection efficiently. Often, their methodology has several flaws that harm the overall performance of system. Moreover, there is not enough of them, because they are needed not only to implement new projects but also to maintain existing ones. Problem often changes over the time, and algorithm selection has to be revisited to prevent loss in efficiency or complete failure of the system to deliver useful solution.

It is apparent, that the algorithm selection needs to be automated to the large extent to reduce implementation and maintenance costs of data science projects.

Therefore, we decided to introduce a general framework or process as a guideline for algorithm selection and pathway to automated algorithm selection. In later chapters we show, how this process can be deployed in several domains.

### 3.2 Introducing general TASA process

The process has several stages, see Figure 3.2. The later stages often require previous stages to be solved. It is hard to define what exactly solving the stage involves on a general level, and there is a lot of freedom to decide which stages requires more attention and which will be solved by first hand solution.
3.2. INTRODUCING GENERAL TASA PROCESS

3.2.1 Algorithm Quality and Evaluation Criteria

In this stage, one has to define and analyze exact performance criteria in the performance space $P$ (Figure 3.1).

The output of this stage is a method, allowing us to measure, evaluate and estimate algorithm quality using performance criteria. This is needed for later stages where algorithms are combined optimized or recommended.

Whereas this stage is straightforward in some domains (e.g. predictive modeling, optimization), it is extremely complicated in other domains (e.g. data clustering). One can finish the process with the first stage, generate set of candidate algorithms, evaluate them and select the best performer. To improve the performance of algorithms, one should consider ensemble strategies and continue with the next stage. It is also possible to make a shortcut in the path towards algorithm selection automation and proceed with optimization strategies to automate parameter tuning of individual algorithms or build a meta-database using base algorithms in their default settings (see Figure 3.2). Such shortcuts can significantly reduce performance of the final solution.

3.2.2 Algorithm Ensembling Strategies

In many domains, algorithm ensembles already proved that the concept of ensembling can improve the quality of result. There are various ways, how algorithms can be combined into an ensemble. Algorithms can be executed in parallel and their solutions can be combined or later algorithms can improve results of preceding algorithms as demonstrated in Figure 3.3. Only some combinations are meaningful and efficient. We have to follow ensembling best practices in order to achieve performance improvement with reasonable complexity (both structural and computational). It is good to be aware of the fact, why ensembling actually works and is capable of boosting the performance above performances of individual algorithms.
CHAPTER 3. OVERVIEW OF OUR APPROACH

Figure 3.4: Examples of meta-learning templates of different complexity ranging from simple algorithm to a complex hierarchical ensemble.

algorithms. Good reference is the doctoral thesis [31], presenting results of bias-variance error decomposition in the field of predictive modeling. In general, ensembling weak learners [43] or weak optimizers [167] can reduce bias of base algorithms, improve performance and make the final algorithm more robust.

In each domain, many ensembling strategies can be considered as we show in later chapters. The output of this stage is a set of algorithm ensembling strategies that help us to produce meta-algorithms. These algorithms can be again evaluated by performance criteria obtained in Stage 1 of the TASA process and best performing algorithm selected.

3.2.3 Meta-Learning and Meta-Optimization Templates

Complex hierarchical ensembles are increasingly popular in Kaggle competitions [5]. Our article [97] shows, that hierarchical ensembles are often beneficial and can bring increased performance, at least in predictive modeling.

The goal of meta-learning templates is to represent group of similar algorithms from the algorithm space. In [109] the concept of schemas is developed for genetic algorithms. The fitness of a schema is computed as average fitness of individuals in the population belonging to the schema.

In a similar way, we can define a meta-learning template as a representative of set of algorithms from the population of algorithms in the algorithm space $A$. The fitness or performance of the template is a single value or a performance vector that represent maximal (average) performance of algorithms generated from the template. The maximal performance is not robust to noise in the performance landscape and it is often good idea to replace it by average or median performance.

Figure 3.4 show templates of different complexity and specificity. The template a) is fully specified hierarchical template with predefined parameters. When this template is applied to some problem, algorithm is always the same, but the solution produces might differ because of stochastic nature of many base algorithms and ensembles. The trivial template b) with the wild-card represents all single algorithm templates. When executed,
3.2. INTRODUCING GENERAL TASA PROCESS

Figure 3.5: Where specific templates have all parameters assigned and algorithms decided, general templates represent class of algorithms.

Any base algorithm with any parameter settings can be applied to a problem. Template c) represents ensembles of algorithms. When you compare fitness of b) and c), you compare fitness of all single base algorithms in the population with all simple ensembles. Complex hierarchical template d) must have at least one base algorithm and one ensemble, but the top level ensemble can contain any combination of base algorithms and ensembles.

It is straightforward to generate algorithms from specific templates. When templates get more general, they represent more and more possible templates (a subspace of the algorithm space \( A \)). To generate algorithms from general templates, some heuristics is needed. A random search is first in hand. The advantage of replacing wild-cards by random algorithms with random parameters is that diverse algorithms are generated. Disadvantage is that these generated solutions can be very suboptimal and it is necessary to fine-tune them e.g. by a simple greedy approach. Other option can be generating algorithms using candidates from the meta-database described below.

It is apparent, that we can translate from general templates to specific templates by reducing number of wildcards and replacing general labels with specific algorithms. Such translation is demonstrated in Figure 3.5. Starting form a general template a), we can specify number of base algorithms in second layer ensembles and specify particular ensembling strategy with parameters b). Fully predefined template c) shows that the final ensemble consists of 100 base algorithms A. On the right, we give an example of this particular template in the predictive modeling domain.

3.2.4 Template Optimization Strategies

Deciding which template performs best for given problem involves exploring infinite state space of all possible combinations in the algorithms space \( A \), where the evaluation of a single state is often expensive as demonstrated in the Figure 4.3 or ill defined as explained in the data clustering domain.

Similarly to the approach suggested in [159], our goal is to benchmark the behavior of a range of algorithms on specific problems rather than specific algorithms over a range
CHAPTER 3. OVERVIEW OF OUR APPROACH

Figure 3.6: For a new problem, meta-features are computed, then similar problems are found in the meta-database. Best performing templates on similar problems are recommended or seeded into the template optimization process. Optimized template is then stored into meta-database.

of problems. A generalized meta-learning template represents class of algorithms and we estimate its quality in the optimization process and benchmarks on individual problems.

Our strategy when we optimize templates for a new problem is to start with general templates and find out e.g. whether simple algorithms are worse than ensembles. Then we translate into more specific templates that seem to be efficient in solving this particular problem. The advantage of this strategy is that it reduces the risk of premature convergence and the algorithm space $A$ can be sufficiently explored before particular regions of this space are explored more extensively.

To optimize the structure of a template and its parameters, several methods can be used. In [21] Gaussian processes and Tree-structured Parsen estimator is used for hyperparameter optimization. We use genetic programing, genetic strategies and other approaches in following chapters.

3.2.5 Meta-Features, Meta-Database and Meta-Data Mining

The search for good performing templates can be streamlined by meta-learning. Typically, best performing templates for different problems are stored into a meta-database. It is important to design and extract meta-features allowing us to measure similarity of problems. For a new problem, meta-features are computed and templates performing best on similar datasets are recommended for new problem. These templates can also seed the optimization process and their parameters might be better adjusted for given problem (see Figure 3.6).

As we show in domain applications, the optimization process can be significantly accelerated. However there ale also perils connected with the meta-database accelerated optimization. One is the template overfitting as reported in the next chapter. The other is the premature convergence (when the optimization is seeded with high quality solutions, exploring distant parts of the algorithm space where the global optima might be hiding is difficult for most heuristics.
3.2. INTRODUCING GENERAL TASA PROCESS

Figure 3.7: Two stage approach to algorithm selection and automation. First, a template is prepared, then it is used to generate an algorithm. Second stage is straightforward given that the template is specific. With problem update (new data) Stage 2 can be executed directly on updated problem possibly improving performance of algorithm.

3.2.6 Algorithm Selection Automation and Applications

The last step is putting everything together to automate algorithm selection problem. Automation of algorithm selection can be employed not only for new problems, but also for the algorithm maintenance on problem updates (new data, change in constrains, etc.).

Figure 3.7 shows our approach to algorithm selection automation. By continuous performance monitoring of selected algorithm and template, we can switch to updated algorithms/templates asynchronously. Updating the algorithm can be performed more often (e.g. every problem update), because this step is far less computationally expensive than updating the template.

This is the last stage of the TASA process. With algorithm selection automated, we are able to construct and maintain large number of templates/algorithms tailored to individual problems.

In following chapters, we demonstrate how to deploy the TASA process in several domains. We start with the predictive modeling domain.
Chapter 4

Algorithm Selection in Predictive Modeling

The area of predictive modeling and classification is ahead of other domains. Problem of algorithm selection has been studied extensively and meta-learning has been elaborated to detail [19, 122, 24, 28, 65, 157, 13]. The most important is to define algorithm quality and evaluation criteria in order to maximize it.

4.1 Algorithm Quality and Evaluation Criteria

In predictive modeling, the generalization performance of algorithms is measured on validation and testing data. Algorithm scalability estimation involves measuring speed of training, speed of recall, memory requirements on data set of increasing size. In anytime learning, one should consider both generalization performance and speed of algorithms [139]. We performed a simple experiment with the Million song data set [22]. The number of instances was scaled from 464 to 463715 in the training set, the size of the validation set was constant - 51630 instances. Deep learning neural net algorithm is Pareto-dominating other algorithms in both generalization performance and training speed on this particular dataset (Figure 4.1). Note that implementations of random forest in H2o and Spark MLlib frameworks have significantly different results.

It is not always true that the more data is used, the better models can be produced. As you see in Figure 4.1, the generalization performance does not increase for training sets larger than 200 thousand instances. Especially when we consider the training time as the algorithm quality criterion, it is necessary to balance the time spent training candidate algorithms and the time spent choosing the algorithm with the best generalization. Most medium-sized to big data sets contain redundant instances. The numerosity of such data sets should be reduced before the data set is used for evolution of templates.

The ratio of the reduction is, of course, a data dependent problem. For the collection of small data sets introduced bellow, we have experimentally distinguished the upper bound to the size of the data set as 500 instances; see Figure 4.2, where the performance of the
Algorithm is Pareto dominant in the generalization-training time performance space.

If the chosen value is too low, the data meaning may change radically and the result might be configuration optimized for completely different data (for an example of this behavior see Figure 4.12). This is also why the value is not too small to begin with. During the template optimization each time convergence is detected, the data sample is doubled in order to use assigned time effectively.

Another algorithm quality criterion is the interpretability of models generated by an algorithm. This is however hard to measure and we do not use it in this thesis.

Standard approach to algorithm selection is to focus on algorithm generalization performance only. It involves the estimation of generalization performance for candidate algorithms. It is relatively easy for large dataset, where enough instances can be reserved for model evaluation. For smaller datasets, cross validation and bootstrap sampling are often necessary. Such validation is quite computationally intensive, because it involves training and evaluating several models on bootstrap data samples (as shown on Figure 4.3). Moreover, it is often not sufficient and the performance estimates are too noisy and unreliable. In such cases multiple cross-validation should be used.
4.1. ALGORITHM QUALITY AND EVALUATION CRITERIA

Figure 4.2: The figure displays dependence of classification accuracy on sample size of the spread data. The time spent on training was fixed for all models. Redundant instances have negative impact on accuracies measured on bigger data sets. Therefore in anytime learning size of training sets need to be carefully controlled avoiding negative effect of redundant data on learning performance.

Figure 4.3: To estimate generalization power of an algorithm, performance on validation data samples need to be evaluated. For stable estimate, validation is repeated on several samples.
4.2 Algorithm Ensembling Strategies

When ensembling algorithms were introduced, the selection of the optimal classification or regression algorithm became even more complex. Base algorithms can be combined in several possible ways \cite{28, 271, 101, 160, 161, 78}. Decision Forests, Boosting of DTs and similar algorithms produce models with better generalization performance on many data sets \cite{128}. In the search for the best algorithm, ensembling algorithms (and their parameters) have to be considered.

Recently, aggregation or hierarchical combination of ensembles has been studied \cite{12, 37, 144} intensively. In particular, gradient boosting \cite{54} and multi-level stacking of neural networks \cite{16} were parts of the winning solution in the Netflix competition \cite{150, 18}. In \cite{5}, ad hoc hierarchical ensembles are called ”Frankenstein ensembles” due to their complexity.

These hierarchical ensembles are single purpose architectures that provide the best results on one particular problem (data set), but very likely fail with different data. That is why many authors develop tools for algorithm recommendation \cite{59} or evolve the best architectures (topologies) for a given data set.

In \cite{76}, genetic programming is used to evolve trees of ensemble models, but only to a limited degree with only one type of ensemble, and the article deals with cancer data only.

The Neural Network ensembling method (GEMS), proposed in \cite{152} trains models independently and then combines them using genetic programming into a simple hierarchical ensemble using weighted average. Weights are evolved by means of genetic programming rather than derived from model performance, as in Boosting for instance.

In modern proprietary modeling solutions such as IBM Modeler, one can manually specify which algorithms and parameters are to be evaluated and the top performing algorithms are then ensembled by standard Bagging.

4.2.1 Base algorithms and ensemble methods

In our experiments we use several base learners, both regression models and classifiers.

4.2.1.1 Base algorithms - models and classifiers

Base algorithms cannot be further decomposed (they are not ensemble algorithms).

- Regression base algorithms
  - Sigmoid, Sine, Polynomial, Gaussian, Exponential, Linear

- Classification base algorithms
  - Classification Neural Network (NN), Support Vector Machine (SVM), Bayes classifier, Decision Tree (DT), KNN
4.2. ALGORITHM ENSEMBLING STRATEGIES

Regression algorithms are adopted from the FAKE GAME environment [4] and optimized by methods from the JCOOL library [93, 23]. Classification algorithms are adopted from the Rapidminer environment [2].

The classification task itself can be decomposed into regression subproblems by separation of single classes from the others. These binary problems can be solved by regression models - by estimating continuous class probabilities. In our experiments, we therefore use regression models and meta-models (produced by ensemble algorithms adjusted for regression purposes). The classifier consisting of regression models is referred to as ClassifierModel. In similar way, regression problems can be transformed to classification problems and final ensembles can therefore contain base algorithms from both classification and regression field [5].

4.2.1.2 Ensembling algorithms

The performance of models can often be increased by ensembling [27, 101, 160, 131, 161, 78] base algorithms, particularly in cases where base algorithms produce models with insufficient plasticity or models are overfitted to training data [30].

A detailed description of the large variety of ensemble algorithms can be found in [26]. We briefly describe the ensembling algorithms that are used in our experiments. Bagging [29] is the simplest one; it selects instances for base models randomly with repetition and combines models with simple average. Boosting [131] specializes models on instances incorrectly handled by previous models and combines them with weighted average. Stacking [160] uses a meta model, which is learned from the outputs of all base models, to combine them. Another ensemble utilizing meta models is the Cascade Generalization [56], where every model except the first one uses a data set extended by the output of all preceding models. Delegating [50] and Cascading [11, 88] both use a similar principle: they operate with certainty of model output. The latter model is specialized not only in instances that are classified incorrectly by previous models, but also in instances that are classified correctly, but previous models are not certain in terms of their output. Cascading only modifies the probability of selecting given instances for the learning set of the next model. Arbitrating [120] uses a meta-model called referee for each model. The purpose of this meta-model is to predict the probability of correct output. In this study, we use selected ensemble methods implemented within the FAKE GAME project [4]:

- Regression ensembles
  - Bagging, Boosting, Cascade Generalization, Stacking, Area Specialization, Divide ensemble

- Classification ensembles
  - Bagging, Boosting, Cascade Generalization, Stacking, Cascading, Delegating, Arbitrating
4.3 Meta-Learning Templates

Several major research projects in recent years have targeted meta-learning in predictive modeling [96]. ESPRIT Statlog [145] compared the performance of numerous classification algorithms on several real-world data sets. In this project, metadata (statistical features describing the data sets) were used for algorithm recommendation. The MetaL project [20], built upon Statlogs outcomes, utilized landmarking metadata (results of fast algorithms, executed on a data set in order to determine its complexity). Another project was METALA [24], an agent-based distributed data mining system, supported by meta-learning. Again, the goal was to select from among available data mining algorithms the one producing models with the best generalization performance for the given data.

Although the results of these projects are interesting, they are not widely adopted for algorithm selection. Recently, a meta-learning infrastructure [155] has been developed to collect the results of different algorithms on different problems. Specific queries can be used to recommend the best algorithm, but the system is not designed to evaluate complex hierarchical ensembles with anytime properties.

In the predictive modeling domain, we represent algorithms by a hierarchical template as already demonstrated in the previous chapter in Figure 3.5. Some of our templates can also incorporate data preparation methods with direct impact on quality of models, namely feature selection of inputs or data normalizers/class balancers.

Predictive modeling templates use two stages to generate predictive models - algorithm selection and model building.

In the first stage, an appropriate algorithm (or combination of algorithms) together with their parameters is to be found. The result of this stage is called meta-learning template. In the second stage, this template is executed on training data and the model is built instantly. The second stage (model building) can be repeated multiple times adapting the prediction system to incoming data. When the distribution of incoming data is changed significantly, the first stage has to be repeated and the template should be replaced.

The meta-learning template [96] is a prescription how to build supervised models. In the most complex case, it can be a collection of ensembling algorithms, modeling and classification algorithms combined in a hierarchical manner. Models or classifiers deeper in the hierarchy can be more specialized in a particular subset of data samples or attributes. This scheme decomposes the problems into subproblems and combines (generalizes) the final solution (model) from subsolutions. The problem decomposition procedure depends on ensembling methods. Typically, it distributes data to member models and when all outputs are available, they are combined to the ensemble output. Ensemble algorithms act as inner nodes, whereas base algorithms act as leaves in the tree representing the hierarchy. Note that meta-learning templates are not data mining models, but algorithms. Models are produced when templates are executed.

Figure 4.4 shows an example of a meta-learning template. When executed, the full training data set is passed to a top level bagging that generates 4 bootstrap training data sets for members of the ensemble. The second bootstrap training data set is used to train a KNN classifier by boosting and samples where this classifier demonstrates high error are
4.4 Template Optimization Strategies

The meta-learning template can be designed manually using expert knowledge (for example, bagging boosted decision trees showed good results on several problems) so it is likely to perform well on a new data set. This is, however, not guaranteed even when the meta-data distance of data sets is small.

Bellow, we show how to construct templates automatically using the genetic programming [100].
Figure 4.5: An ensemble classifier can be produced by the hierarchical combination of algorithms depicted in Figure 4.4. Executing the template will distribute data to leaf base models according to procedures specified by ensembling algorithms. Base models and ensembles are constructed until the root ensemble (base model) is finished. Using the model involves propagating and presenting an input vector to leaf models and combining their outputs by ensembling procedures.

4.4.1 Evolving templates by genetic programming

Applying genetic programming involves resolving a) representation of individual, b) fitness function formulation, c) initial population, and d) design of genetic operators and evolution.

4.4.1.1 Encoding the template into genotype

Encoding is straightforward, because genetic programming works with trees. First, two classes of meta-learning templates need to be defined. Predefined templates represent an intuitive one-to-one relation between a template and the resulting model. As shown in Figure 4.8, the structure of the template is the same as the structure of the model produced. This type of template is also more difficult to optimize, because it allows to specify hierarchical ensembles in full details. Our experiments showed that it is beneficial to evolve a general template structure before predefined templates are optimized.

In a Generalized template, the number of member models in the ensemble is determined by a single integer variable. When the template is executed, member models are randomly selected and created from a template pool. To describe an example from Figure 4.8 (template execution in the Evaluator): Template ensemble E has 2 member templates (S and K) and variable numberOfModels which equals 5. The resulting ensemble model E has 5 member models generated by randomly selected templates from S and K pools.
4.4. TEMPLATE OPTIMIZATION STRATEGIES

4.4.1.2 Fitness function

Fitness is computed by estimating the generalization performance of models produced by the template. Because the Evaluator needs to train the models, it must be task specific, and we have so far Evaluators for classification, regression and order prediction problems. Also, the evaluation setup is determined by a context such as data division, data handling (e.g. cross-validation or intelligent data sampling) and result evaluation (e.g. simple accuracy or weighted loss matrix). Here, we evaluate classification templates only, and the standard 0-1 loss matrix is used. The implicit data handling method is multiple cross-validation: the fitness of a template is proportional to the average performance of models generated on training folds and evaluated on testing folds, while the data are divided into folds multiple times. Later, we address the issue of data handling, because the evaluation of the fitness function consumes most of the time during the evolution. We have to keep it simple but maintain good estimate of template quality at the same time. With more time available, our fitness estimates are refined with additional fitness evaluations starting with most promising estimates with high variation of recent evaluations. This is tightly connected with metrics defined in Stage 1 of the TASA process.

4.4.1.3 Initializing a population

An initial population is generated from a minimal form. All primitive templates, consisting of one base algorithm, are evaluated on the data set and stored in the meta-database. Then the population is initialized by templates from the meta-database. Meta-features based on instant base models landmarking and data statistics are computed. The meta-features vector is then compared to other vectors stored in the meta-database and the most similar records are returned. The records contain a list of best templates which are inserted into the initial population. During the computation, the fitness of each template is cached in the Evaluator, and when computation terminates, the best templates are saved as a new record into the meta-database or corresponding records are updated with the new templates. The Section 4.5.7 provides experimental results showing that using the templates from a meta-database is beneficial for most of the data sets. For very small data sets, it is better to initialize the population by a random substrate.

4.4.1.4 An evolutionary algorithm

The core algorithm is the evolution of templates which is shown in the simplified form in Figure 4.8. The fitness computation is handled by Evaluator and the selection pressure is implemented by tournament. Examples of possible mutations are shown in Figures 4.6 and 4.7.

- **Node change mutation** Structural mutation where node is replaced by another one. If a leaf if replaced by an inner node, the leaf is connected to the new inner node, which is how the tree grows.

- **Node add mutation** Also structural mutation which adds a leaf node to an inner node.
CHAPTER 4. ALGORITHM SELECTION IN PREDICTIVE MODELING

Figure 4.6: Example of all possible mutations of leaf node (DT in the figure). Behavior of the node change mutation depends on type of new node. If the new node is a leaf, the old node is simply replaced by the new one. If the new node is an inner node (i.e., ensemble), the old node is added as a successor to the new node. Explanatory Notes: SVM = Support Vector Machine, DT = Decision Tree; depth and models are parameters of corresponding algorithms.

Figure 4.7: Example of all possible mutations of inner node (Boosting ensemble in the Figure). Behavior of the node change mutation depends on new node type. If the new node is inner node, old node is simply replaced by the new one. If the new node is a leaf, the whole tree under the old node is replaced by the new single leaf node.
Figure 4.8: The figure represents a top level scheme of the evolutionary algorithm and how its individual parts interact. In this view the algorithm is divided into 3 main layers: The control layer is the top layer and contains the logic how to combine different parts and heuristics. The middle layer consists of the problem independent algorithms like evolutionary algorithm for evolving tree structures, template selector and the meta layer. The lowest layer consists of the problem specific tasks which are represented here by the Evaluator object and Meta-database.

- **Variable mutation** Variable of a node is mutated applying Gaussian noise to the current value.

Exploration versus exploitation capabilities of the evolution are automatically controlled by mutation limits (number of changes that can be made by mutation to a template is limited based on the relative fitness of templates).

### 4.4.1.5 Feature selection

For optimal input selection an optimal template needs to be known, but for the optimal template, optimal inputs need to be known. There is no easy way out of this loop, so it is advantageous to evolve inputs along with the template structure. Our experiments show that evolution itself cannot generate sufficient selective pressure specialized solely in inputs. The percentage of templates with wrong inputs is quite high and the computation of these
templates is essentially a waste of time. Even if the modeling algorithm represented by the
tree is the optimal one, the evolution algorithm cannot recognize that, because with wrong
inputs its fitness is the same as all other modeling algorithms. And the same problem arises
from the other side as well - in case of optimal inputs combined with a wrong modeling
algorithm.

We utilized the principle of coevolution \[110\] to deal with this problem. Hence, the
optimization of inputs has a selective pressure of its own.

We have designed and implemented the evolution so it can utilize any number of co-
algorithms (not necessarily evolutionary algorithms) which are responsible for suggesting
new values for mutation. These co-algorithms are variable- or object-bound and, when a
mutation of a given variable or object occurs, they are queried for a new value instead of
a random mutation used in a standard evolution. Fitness of the provided value is then
passed back to the co-algorithm. Once every \(N\) individual evaluations in the template
evolution, the next iteration of the co-algorithm is run.

Input coevolution is a standard type of evolution, evolving a binary gene (bits represent
whether corresponding inputs are used by evolved templates or not). As a stand-alone
algorithm it can exert more selective pressure on promising inputs and recommend them
more often than in the case of evolving inputs and tree structure together.

To illustrate the power of coevolution, we created an artificial data set by enriching
Spiral data with three irrelevant random input attributes. An optimal solution consists of
selecting the right two inputs plus optimizing a tree composed of a specific ensemble (we
disabled all instance based algorithms to make the problem more difficult).

We performed experiments to compare evolution with a) randomly generated inputs,
b) mutation of input genes, and c) coevolution of inputs. For each setup, the evolution ran
for 5 minutes and the resulting percentage of templates with optimal inputs was averaged
over 200 runs.

On average, evolution with randomly generated inputs contained only 3.23% optimal
input configurations, the evolution with mutation of input genes contained 6.74% and
evolution with coevolution of inputs contained 17.65% of optimal input configurations.
It demonstrates that coevolution can induce the level of selection pressure required to
multiply good inputs in the population of templates.

We also allowed input coevolution to be turned on only if convergence occurred. This
often happens quite early in case of problems where input optimization is necessary. On the
other hand, if input optimization is not needed, convergence conditions ensure that a good
template was already found and, in combination with the selection pressure of coevolution,
this minimizes the number of unpromising templates to be created. This approach provides
a means to optimize both inputs and model structure at the same time, while minimizing
the negative impact of input optimization for problems where it is not needed.

4.4.1.6 Global heuristics

The global optimization workflow has been experimentally derived. It navigates the search
in the space of all possible template topologies together with a detailed selection of the
4.4. TEMPLATE OPTIMIZATION STRATEGIES

As shown in Figure 4.8, there are three instances of evolutionary algorithm, each with different state space search capabilities.

First, genetic programming evolves generalized templates and finds promising parts of the state space (model and ensemble type combinations). Then, if a convergence occurs and there is more time available, evolution of the predefined templates is activated. It starts from the promising templates evolved earlier and attempts to define a specific structure of the template. The last type of evolution focuses on evolving parameters only, leaving the structure of the template intact. Note that during the evolution of templates, parameters are also modified; therefore the last evolution run is just to fine-tune them.

As a future work, we plan to experiment with continuous shift from general to specific templates within one evolution.

When the template evolution terminates, candidate templates are selected. Additional testing is needed to select the best template, because the fitness computed by the Evaluator is only a rough estimate of the true quality of the templates. These candidate solutions are then processed by the Final Selector, which performs extensive testing by multiple runs of the Evaluator to measure the performance of each template more accurately.

The other heuristics control adaptive settings of parameters like maximum template depth, maximum allowed computational complexity of template, input optimization trigger, intervals of base algorithm parameters, size of data samples, etc. The size of the state space is adaptively increased to maximize the probability of finding best possible solution in the given time.

4.4.2 Exploring models produced by templates

The final template is comprehensively tested and the generalization performance of models generated by this template should be the highest among candidate templates. The quality of the selected template can also be seen in the shape and consistency of decision boundaries of its models. As an example, we ran the evolution on the Two Intertwined Spirals data set [86] (10 minutes on a standard PC). The template that was finally selected can be written as: 

\[
\text{ClassifierCascadeGenProb}\{4 \times \text{KNN}(k=2, \text{vote=true, measure=ManhattanDistance})\}
\]

We used our RapidMiner plugin [1] to visualize the structure and behavior of the classifier produced when this template was executed. The template contains the ClassifierCascadeGenProb ensemble of three 2NN classifiers. In the Cascade Generalization [56] ensemble, every model except the first one uses a data set extended by the output of all previous models. In this particular case, the first 2NN classifier is produced on the Spiral data set, the input of the second 2NN classifier is enriched by two outputs of the first classifier (probabilities of membership in one of the two intertwined spirals). The third classifier receives two original ‘spiral’ inputs plus four output probabilities from the already generated classifiers, etc.

This behavior can be observed in Figure 4.9 a). As can be seen in the thumbnail images, where the background color should match the color of data points for the perfect classifier, the first KNN algorithm is capable of making a nearly perfect model, except for small regions with absent learning data. The other classifiers specialize in these re-
CHAPTER 4. ALGORITHM SELECTION IN PREDICTIVE MODELING

Figure 4.9: The Cascade generalization ensemble of three 2NN classifiers was produced by a meta-learning template evolved on the Spiral problem. The thumbnail images show the response of classifiers to the change of their two most relevant inputs.

regions, so the final cascade ensemble classifies the Spiral data even better. Figure 4.9 b) shows the decision boundaries of a recently evolved template that outperformed the Cascade generalization of KNN classifiers: ClassifierModeloutputs x LocalPolynomialModel. The LocalPolynomialModel was added to our base algorithm recently and it apparently performs better than KNN on this problem.

We can guess that both the Cascade-generalization of 2NN and the ClassifierModel of LocalPolynomialModels templates evolved on the Spiral data would also produce good models for similar problems, e.g. for any other complex separation problem in two dimensions. Experiments described in the next section are to reveal the universality of templates, their similarity and show how a meta-database can be used to streamline the optimization of templates.

We can conclude that the evolution introduced above is capable of template optimization when generalization performance is a single quality criterion.

4.4.3 Evolutionary Programming in anytime learning

One can argue that evolutionary programming is not suited for anytime learning and a greedy search from a minimal form in the space of hierarchical templates should be more efficient. Therefore we compared three approaches to anytime template optimization. First approach is the random search. We generate random templates, with random parameter values and random (small) depth and evaluate it. The second approach is more advanced greedy search. Greedy search starts from base algorithms only and then constructs new individuals by probabilistic selection based on node’s fitness. Result of this experiment can be seen in the Figure 4.10 and it shows that genetic programming can outperform other approaches at any given time.
4.5 Meta-Features, Meta-Database and Meta-Data Mining

Popular meta-learning approach is to use a knowledge base containing information that might help to improve the performance of learning. One possible approach [137] is to describe the data set by a vector of meta-features and based on this vector, select the best performing algorithm for the nearest data set from the knowledge base.

There are several ways to obtain meta-features. The oldest and most widely used tools are summary statistics and related meta-features. The landmarking meta-features [122] are obtained when simple, diverse and fast learning methods are executed on the data set and their results are collected.

To evaluate the properties of meta-learning templates and data treatment techniques, it is necessary to experiment with a wide range of data sets. First of all, data sets used in our experiments are described.

### 4.5.1 Data sets description

Table 4.1 lists the data sets used as well as their size, dimensionality and number of classification classes (outputs). Most of the data sets are taken from the UCI repository [52]. Other data sets (mostly artificial) are tailored to evaluate data separation capabilities of algorithms for low dimensional problems. The Spirals data set was used in the previous section and was designed as a benchmark for global approximation methods.
Table 4.1: Data sets are obtained mostly from the UCI repository and are small to medium-sized.

<table>
<thead>
<tr>
<th>data set</th>
<th>Origin</th>
<th>Inputs</th>
<th>Classes</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glass</td>
<td>UCI</td>
<td>9</td>
<td>7</td>
<td>213</td>
</tr>
<tr>
<td>Balance</td>
<td>UCI</td>
<td>5</td>
<td>3</td>
<td>624</td>
</tr>
<tr>
<td>Breast</td>
<td>UCI</td>
<td>9</td>
<td>2</td>
<td>698</td>
</tr>
<tr>
<td>Diabetes</td>
<td>UCI</td>
<td>8</td>
<td>2</td>
<td>767</td>
</tr>
<tr>
<td>Ecoli</td>
<td>UCI</td>
<td>8</td>
<td>7</td>
<td>335</td>
</tr>
<tr>
<td>Heart</td>
<td>UCI</td>
<td>13</td>
<td>5</td>
<td>269</td>
</tr>
<tr>
<td>Texture1</td>
<td>TSB</td>
<td>4</td>
<td>3</td>
<td>750</td>
</tr>
<tr>
<td>Texture2</td>
<td>TSB</td>
<td>4</td>
<td>10</td>
<td>2500</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>UCI</td>
<td>34</td>
<td>2</td>
<td>350</td>
</tr>
<tr>
<td>Spirals</td>
<td>TIS</td>
<td>2</td>
<td>2</td>
<td>192</td>
</tr>
<tr>
<td>Vehicle</td>
<td>UCI</td>
<td>18</td>
<td>8</td>
<td>845</td>
</tr>
<tr>
<td>Wine</td>
<td>UCI</td>
<td>13</td>
<td>3</td>
<td>177</td>
</tr>
<tr>
<td>Segment</td>
<td>UCI</td>
<td>19</td>
<td>7</td>
<td>2309</td>
</tr>
<tr>
<td>Fourier</td>
<td>UCI</td>
<td>76</td>
<td>10</td>
<td>1999</td>
</tr>
<tr>
<td>Spread</td>
<td>EVO</td>
<td>2</td>
<td>19</td>
<td>2500</td>
</tr>
</tbody>
</table>

*Spread* is a two-dimensional artificial data set, which was created with an evolutionary algorithm to be unsolvable by the basic classification algorithms currently available in RapidMiner. The fitness function was inversely proportional to the performance of the best classifier and the chromosomes contained parameters of a data set generator.

Data sets (*Texture1* and *Texture2*) come from a generator of images for pattern recognition [3]. Four features were extracted from these images, one using the local binary pattern and the other three with a 5x5 convolution matrix for each color component (rgb). We generated balanced data sets with 250 instances for each class (segment). *Texture1* was formed by three segments (750 instances) and *Texture2* by ten segments (2500 instances).

### 4.5.2 Dividing the data

Use of learning and testing sets to avoid overfitting is a well known principle. In the process of evolution of templates it is necessary to estimate the quality of templates and to balance well the data used for learning and for evaluation.

1. A *training set* is used for model learning.

2. A *validation set* is then used for testing generalization performance of templates during the evolution. Performance on the validation set is then translated into template fitness.

3. A *testing set* is never used during the evolution of templates (except plotting the
4.5. META-FEATURES, META-DATABASE AND META-DATA MINING

Table 4.2: The table shows the difference between evolutions using learning/validation/testing sets or learning/validation sets respectively.

<table>
<thead>
<tr>
<th>Algorithm settings</th>
<th>data sets - classification accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>spirals</td>
</tr>
<tr>
<td>test set used</td>
<td>0.8768</td>
</tr>
<tr>
<td>no test set</td>
<td>0.8818</td>
</tr>
</tbody>
</table>

Performance of evolution for different sets is shown in Table 4.2. Without the testing set the performance is much lower. Using only two sets may be beneficial for small specific data, where removing instances from the learning set cause a significant drop in the quality of templates and training data overfitting is not a big issue.

We conducted several experiments in order to answer questions formulated in the introduction. The following experiments examine properties of meta-learning templates and of the evolutionary process producing them.

Methodology of experiments is designed as follows. The evolutionary processes with benchmarked configurations (properties) were executed for given CPU time on the same machine. The running time was chosen for each experiment individually, considering the difficulty and size of the data set. This setup was repeated up to 200 times and outputs were averaged. Therefore a generalization performance of the configuration on certain data set averages 200 runs and each resulting configuration was tested 40 times (8000 model evaluation in total averaged into one number). Results with this precision are needed to recognize significant difference among configurations in a statistical sense.

### 4.5.3 Hierarchical ensemble

In this subsection we would like to answer the question if hierarchical ensembles are capable of improving the generalization performance of data mining models. We compare three configurations of the evolutionary algorithm. In the first configuration, we restrict the search to trivial templates with base algorithms only (depth 0). The second configuration allows the evolution to consider also ensembles of base algorithms (depth 1). The third configuration extends the search to the second level hierarchies - ensembles of ensembles (depth 2). All three variants run for the same time.

Table 4.3 shows that the classification accuracy of templates on test data increases with depth. We can conclude that for this particular problem (Glass data) the hierarchical ensemble represents significant improvement over base models or regular ensembles.

Of course, ensembles and hierarchical ensembles are not always beneficiary (for example when problems are linearly separable, there is no need for ensembles, because most of base
Table 4.3: The trivial template (maximal template depth limited to 0) was evolved on the Glass data set for given time (this is equivalent to the selection among base algorithms with optimized parameters), then we run same experiment with maximal depth 1 (simple ensembles of base algorithms allowed) for the same amount of time and so on. Results are averaged from 50 runs for each depth limit. Table shows the best template found for given depth limit and their maximal and average classification accuracies on test data sets.

<table>
<thead>
<tr>
<th>Depth limit</th>
<th>Test accuracy:</th>
<th>The best meta-learning template found in the search space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>average</td>
</tr>
<tr>
<td>0</td>
<td>0.671875</td>
<td>0.643958</td>
</tr>
<tr>
<td></td>
<td>DecisionTree(depth=9,conf=0.04,alt=7)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.766667</td>
<td>0.719318</td>
</tr>
<tr>
<td></td>
<td>ClassifierBagging{40x DecisionTree(depth=46,conf=0.494,alt=3)}</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.78972</td>
<td>0.748752</td>
</tr>
<tr>
<td></td>
<td>ClassifierBoosting{9x ClassifierBoosting{8x DecisionTree(depth=12,conf=0.5,alt=2)}}</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.11: The workflow evaluating sensitivity to overfitting and stability of solution.

Our algorithm explores trivial templates first and then gradually extends the search space to more complex hierarchies.

4.5.4 Template overfitting

The next experiment is to examine the sensitivity of meta-learning templates to data overfitting. A the same time, we will explore the robustness of our approach in terms of generating stable solutions for very similar problems.

The experimental setup is rather complicated so we use Figure 4.11 to illustrate it. The Ecoli data set was divided into two folds of equal size (Training and Testing). The Training fold was subsequently divided into Learn and Validation folds multiple times, with division ratio iterating from 0.1 (10% Learning, 90% Validation) to 1 (100% Learning, 0% Validation). Learning sets of increasing size were used to evolve meta-learning templates and to produce models by executing templates on the same data. These models
Figure 4.12: The difference between Test and Validation errors is not significant. Glyphs indicate percentages of base algorithms and meta-algorithms in winning templates. The same glyphs are depicted with percentages in the logarithmic scale in the next Figure.

were evaluated on the Training set and on the Testing set producing the Validation and Test errors. Whereas the Test errors are unbiased estimates of model performances, the Validation errors gradually translate to the Training errors (possibly biased) as the size of the Learn set increases. Note that for 100% Learn data fraction, full Training set is used to evolve templates and build models so the Validation error becomes the Training error.

We averaged results from 20 repetitions of this setup and plotted the development of errors (see Figure 4.12). The level of data overfitting is reasonably low. Even when the same (Training) data set is used to evolve the template, build the model and estimate its error, the error is not significantly different from the unbiased estimate computed on independent Testing set. This is mainly due to the fitness function used in the evolution of templates, which favors templates generating models performing well on unseen data. Glyphs summarize numbers of base algorithms and meta-algorithms appearing in evolved templates for each division ratio. As explained bellow, for tiny learning data sets (ratio bellow 0.3) diverse templates were evolved in each repetition, whereas for ratios above 0.8, evolved templates were almost identical.

It is apparent from decrease of errors (Figure 4.12), that when the division ratio exceeds 0.5 (84 instances in the Learn set), the quality of templates becomes reasonable and above 0.8 (134 instances in the Learn set) almost constant. It is also confirmed by the analysis of produced templates. For each division ratio, we counted numbers of base algorithms and meta-algorithms contained in winning templates. We plot their percentages in Figure 4.13. Note that these shapes correspond to glyphs from the previous Figure, except that the logarithmic scale is used in Figure 4.13. When the Learn set contains just 17 instances (10% of the Train set), it is apparently not enough to express decision boundaries of dominant
Figure 4.13: Development of percentages (in log scale) of base algorithms and meta-algorithms contained in winning templates. For tiny Learn sets (below 20%), almost all models perform well, for full Learn set Random Forests dominate.
classes in the Ecoli data set. The Local polynomial regression and the separation using Polynomial models work slightly better than other algorithms, but the difference is not high and all algorithms can be found in winning templates. Due to the random sampling, the stability of templates is low for such tiny varying data sets. For bigger division ratios (0.3 - 0.7) the KNN algorithm is often part of the winning solution and the importance of the Random forest grows. For Learning sets of more than hundred instances Random forests dominate evolved templates accompanied by advanced ensembling methods: Boosting, Stacking and Cascade Generalization. Note that this behavior was examined on the Ecoli data set, but can be observed also on other data sets.

4.5.5 Templates evolved for various data sets

Templates evolved on data sets (see Figure 4.14) were serialized to a text description representing their internal structure. As you can see in the Figure 4.14, for some data sets trivial templates were evolved (for example the KNN algorithm for Heart and Pendigits), for other datasets a regular ensemble performed best (for example the Boosting of Decision Trees for Segment) and hierarchical templates were the best solution for Vehicle or Texture2, Wine or Breast data sets and others. Note that depicted templates are representatives of final templates selected in multiple runs on benchmarking datasets. In each independent (no meta-database) run of the evolution on a single dataset, final template can differ. Diversity of final templates may be minimal for some dataset and significant for other, however they are very similar in terms of functionality and complexity.

The occurrence of individual algorithms can be counted for evolved templates. Almost 40% of solutions were hierarchical templates, the same percentage contained the ClassifierModel decomposing the classification problem into $N$ regression problems of class probability estimation. It is surprising that regression models are present so often in final classification templates. One possible explanation is that our optimization algorithms for predictive modeling are very efficient and fast. Therefore the evolution can explore many more variants in given time than in case of KNN, Neural nets or other classification algorithms that tend to be slower.

It is apparent that ensembles and particularly their hierarchical variants significantly outperform optimized base algorithms for several data sets. The hierarchical template $StackingProbab\{5X StackingProb\{5X NeuralNetClassifier\}\}$ evolved in our previous experiments on Texture2 data set (Ttext2) achieved also the best overall performance being the most universal template for our set of problems.

4.5.6 Similarity and substitutability of templates

In our contribution [97] we analyzed the similarity of templates in terms of performance on individual datasets. We executed each template on all datasets and measured the performance of generated classifiers. We split each dataset randomly into two folds, one is used for learning and the second for evaluating the classifier, then the folds are exchanged.
CHAPTER 4. ALGORITHM SELECTION IN PREDICTIVE MODELING

Meta-learning templates evolved on benchmarking datasets

<table>
<thead>
<tr>
<th>Class</th>
<th>Meta-learning template</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glass</td>
<td>StackingProbabilities {4x KNN(k=2, vote=true, measure=ManhattanDistance)}</td>
</tr>
<tr>
<td>Balance</td>
<td>Boosting {57x ClassifierModel {&lt;outputs&gt;x PolynomialModel(degree=4)}}</td>
</tr>
<tr>
<td>Breast</td>
<td>ClassifierModel {&lt;outputs&gt;x CascadeGenModel {7x CascadeGenModel{5x GaussianModel}}}</td>
</tr>
<tr>
<td>Diabetes</td>
<td>SVM(kernel=dot)</td>
</tr>
<tr>
<td>Ecoli</td>
<td>ClassifierArbitrating {2x ClassifierBagging {3x SVM(kernel=anova)}}</td>
</tr>
<tr>
<td>Heart</td>
<td>KNN(k=15, vote=false, measure=CosineSimilarity)</td>
</tr>
<tr>
<td>Texture1</td>
<td>ClassifierArbitrating {4x ClassifierModel {&lt;outputs&gt;x PolynomialModel(degree=2)}}</td>
</tr>
<tr>
<td>Texture2</td>
<td>CascadeGenProb {8x Boosting {2x ClassifierModel {&lt;outputs&gt;x ExpModel}}}</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>DecisionTree(maxdepth=20, conf=0.25, alt=10)</td>
</tr>
<tr>
<td>Spirals</td>
<td>CascadeGenProb {8x ClassifierArbitrating {4x KNN(k=3, vote=false, measure=MixedEuclideanDistance)}}</td>
</tr>
<tr>
<td>Pendants</td>
<td>KNN(k=3, vote=false, measure=CosineSimilarity)</td>
</tr>
<tr>
<td>Vehicle</td>
<td>ClassifierArbitrating {6x ClassifierModel {&lt;outputs&gt;x DivideModel(mult=6.68){7x PolynomialModel(degree=3)}}}</td>
</tr>
<tr>
<td>Wine</td>
<td>CascadeGenProb {9x ClassifierModel {&lt;outputs&gt;x BoostingRTModel(tr=0.1){8x GaussianModel}}}</td>
</tr>
<tr>
<td>Spambase</td>
<td>ClassifierModel {&lt;outputs&gt;x CascadeGenModel {9x SigmoidModel}}</td>
</tr>
<tr>
<td>Segment</td>
<td>Boosting {17x DecisionTree(maxdepth=24, conf=0.082, alt=0)}</td>
</tr>
<tr>
<td>Fourier</td>
<td>NeuralNetClassifier(net=-1x0, epsilon=0.00001, learn=0.3, momentum=0.2)</td>
</tr>
<tr>
<td>Spirals+3</td>
<td>KNN(k=3, vote=true, measure=EuclideanDistance)</td>
</tr>
<tr>
<td>Spread</td>
<td>CascadeGenProb {5x CascadeGenProb {3x KNN(k=9, vote=true, measure=CosineSimilarity)}}</td>
</tr>
</tbody>
</table>

Figure 4.14: Templates evolved on individual data sets serialized into text description.

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNN Classifier</td>
<td>Classifier</td>
<td>6</td>
</tr>
<tr>
<td>Cascade Generalization</td>
<td>Meta-classifier</td>
<td>5</td>
</tr>
<tr>
<td>Cascade Generalization</td>
<td>Meta-model</td>
<td>3</td>
</tr>
<tr>
<td>Boosting Classifiers</td>
<td>Meta-classifier</td>
<td>3</td>
</tr>
<tr>
<td>Arbitrating Classifiers</td>
<td>Meta-classifier</td>
<td>3</td>
</tr>
<tr>
<td>Polynomial Model</td>
<td>Model</td>
<td>3</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>Classifier</td>
<td>2</td>
</tr>
<tr>
<td>Gaussian Model</td>
<td>Model</td>
<td>2</td>
</tr>
<tr>
<td>SVM Classifier</td>
<td>Classifier</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.4: Frequency of occurrence in winning templates for all methods (Meta-models, Meta-classifiers, models and classifiers).
4.5. META-FEATURES, META-DATABASE AND META-DATA MINING

Due to the noise in results, we repeated this procedure 25 times so each template was evaluated using 25x two fold cross-validation on all datasets.

The results summarized in Figure 4.15 show that three data sets (Breast, Wine and Texture1) are very easy to classify, no matter which template (algorithm) is used. The set of evolved templates was slightly different than that listed here (Figure 4.14). Although we have added some base algorithms and improved global heuristics of the evolution since last experiments published in [97], the winning templates are quite consistent.

There is a group of four data sets (Ecoli, Heart, Ionosphere, Segment), that can be solved by most of the templates except those based on Polynomial models. These models are trained by the Least squares algorithm [95]. For certain data (noisy with binary inputs and overlapping instances) the algorithm fails to deliver a solution due to a non-invertible matrix, the parameters of polynomials are set randomly and the result is poor. There are also two complex data sets (Spirals and Spread) that can be solved almost exclusively by their templates and one complex noisy data set (Texture2) where only ensemble (or hierarchical ensemble) of algorithms can deliver satisfactory results.

Based on these experiments we can conclude that hierarchical templates evolved on particular complex problems (data sets) have often capacity to solve other complex problems very well. This is often the case for complex general-purpose templates containing universal algorithms such as neural nets. On the other hand, some problems (Spirals, Spread) require specific algorithms (KNN, Local polynomial regression). Note that in our previous work [97] the template \texttt{CascadeGenProb\{9x ClassifierModel\{(outputs) x ExpModel\}}} was evolved for the Spread problem and it failed to produce good classifiers on the Spiral data set.

When the performances of templates on individual data sets are averaged, we get the "universality" of templates. Figure 4.16 shows that templates based on polynomial models are least universal (with 60% average performance due to the limited learning capabilities of our implementation of the Least Squares). On the other hand, the most universal is the Texture2 template (double stacking of neural nets). With an average performance over 80% on all data sets, the top three templates (Texture2, Tspread, Tspirals) contain hierarchical ensembles. It is apparent that without using the evolution, a hierarchical meta-learning template is algorithm of the first choice. This is in accordance with our previous research regarding the GAME algorithm [92].

4.5.7 Meta-database for templates

Meta-data system in Figure 4.8 utilizes experience from previous runs. After each run of the evolutionary algorithm, best $N$ templates are stored to the database together with meta-data consisting from statistical descriptors of the data set and landmarking performances of base algorithms. These stored templates are further used to enrich the population of templates being evolved for similar data sets.

When the evolution is executed for a new data set, the initial population is seeded with templates stored for similar data sets based on meta data vectors distance. When the evolution is finished, the data set and evolved templates are added to the meta database.
Figure 4.15: Performances of meta-learning templates on individual data sets visualized as a starplot matrix. Labels of templates in the legend are derived from data sets used to evolve them. For Balance data, the template evolved on Segment data (Tseg) performs worst whereas the template evolved on Balance data (Tbal) performs best - expected behavior.
4.5. META-FEATURES, META-DATABASE AND META-DATA MINING

Figure 4.16: Average performance of individual templates over all data sets.

Figure 4.17: The improvement in the convergence of the evolution can be observed for all tested data sets when the seeding from the meta database is used.

The probability that a template is selected for seeding is inversely proportional to the squared distance of meta data vectors and proportional to a robust performance of the template. The robust performance is defined as average rank of template performance on data sets within the group.

The positive influence of seeding is demonstrated in Figure 4.17. The best solution (template) found in the initial population is far better when the meta database is used for all tested data sets. The improvement is bigger for complex tasks, such as the Spiral problem, where hierarchical templates have to be discovered and the evolution of such templates from randomly initialized population takes many generations.

There is a danger of data overfitting when metadatabase is used. By storing the best template, the dataset used is no longer validation or testing. When the same data set is presented over and over again and each time the best template from the metadatabase is inserted into the initial population of templates, overfitting occurs as shown in Figure 4.18.
Figure 4.18: Templates can overfit the data when the metadatabase is used and the same data set is presented to the system over and over again. The dataset here is Borelia collected in [111].

despite a template does have just a fraction of parameters of a model generated from it.

4.6 Algorithm Selection Automation and Applications

The last stage of the TASA process wraps all efforts done in previous stages and examines automated framework presented in Figure 3.7.

4.6.1 Templates and transfer learning

The concept of meta-learning templates allows to decompose the modeling process to the algorithm selection and the model building phases. Although the algorithm selection is computationally intensive task (evolution of a meta-learning template), it can be streamlined by selection of pre-evolved template from a meta-database as demonstrated above. The model building phase (template execution) can be performed several times to adapt the prediction system to incoming data. This is very useful especially for transfer learning [6], where predictive models need to reflect changes in data distribution. The question is, how often the model and the template should be replaced.

Both model and template will perform well on new data coming from the same distribution as historical data used for template evolution and model creation. Unfortunately stationary distribution is often not the case in real-world applications. We designed an experiment to verify, whether the concept of meta-learning templates can be applied to transfer learning problems.
Figure 4.19: Updating templates and models with new data (a batch of last 250 instances). Model update happens all the time, template switch is useful in the 3900-4250, where the simpler template produces better models - weak learners not overfitting the data.
The Gaussian data set obtained from [114] was used in the experiment. It has six thousand instances in total generated from non-stationary Gaussian distributions. We use a time window of 250 instances. At first, we prepared two data sets (starting from the instance 4000 and 5500 respectively) and evolved two meta-learning templates ($T_{4000-4250}$ and $T_{5500-5750}$).

Then our plugin to RapidMiner was used to execute these templates on Gaussian data. The sliding window of 250 instances was used as the training set for both templates and the following 250 templates were used to evaluate the performance of generated models.

Figure 4.19 shows the generalization performances of classifiers repetitively built using two templates $T_{4000-4250}$ (NaiveBayes using one input only) and $T_{5500-5750}$ (Arbitrating of NeuralNets using both input attributes) on recent 250 instances. Two templates are continuously executed for most recent 250 instances and the performance of models is measured on following 250 instances. The template evolved for instances 4000-4250 generates models that are superior only in around that time (when cluster means of Gaussian distributions used to generated the data are in line). Otherwise the second template generates more precise models. It is apparent that small shift in data distribution can lead to considerably different templates. The interesting interval is 3950-4150, where models generated by the first template outperform models generated by the second template.$^1$

Note that although we used the same data to evolve the template, no overfitting can be observed.

We analyzed the behavior of models and plotted the training data together with the output of classifier encoded to color of the background (two thumbnail images on the top of the Figure4.19). In the first image a classifier built using the first template (Naive Bayes) on instances (4000-4250) needs only one input to classify data into three classes. The behavior of classifiers generated by the second template (neural nets ensembles) matches perfectly to images obtained from the Gaussian data video [114] illustrating changing data distribution.

In order to maximize the performance of classifiers on data with changing distribution, we have to evaluate the quality of candidate templates continuously. The actual models should be generated using the best template. This approach is more flexible than other approaches used for the incremental learning (e.g. updating weights of models in the ensemble). It allows us to switch to trivial algorithms when the complexity of the modeling task become low as illustrated on this simple example.

In the two stage modeling, we can continuously evaluate the quality of models and templates, replacing them when significantly better candidates are found. When smooth changes are needed, $N$ recent models can be averaged.

Our future work is to extend the automated algorithm selection framework for large datasets.

---

$^1$It means for example that classifiers constructed using instances (3750-4000) and validated on instances (4000-4250) should use the Naive Bayes algorithm instead of neural network ensembling.
Chapter 5
Algorithm Selection in Clustering

In clustering, several algorithms exist and cluster ensembles are of growing popularity recently. Datasets with diverse data density and distributions, dimensinality, noise and other problems need a special treatment and diverse clustering approaches are applicable. Moreover, the performance of clustering algorithms is heavily dependent on their parameters such as the similarity measure. In order to obtain robust clusters, ensembles of clustering algorithms are employed. Problem of clustering algorithm selection therefore exhibit similar attributes as algorithm selection in predictive modeling. However there are also significant differences and the most important difference is that measuring the performance of clustering is hard, as discussed in the next section.

5.1 Algorithm Quality and Evaluation Criteria

One of the key issues in clustering is how we measure the quality of clusterings, since naturally we would like to know which algorithms produce better clusterings. However, creating a universal measure to compare the performance any two clusterings seems to be very hard task.

Many validation indexes have been proposed in the literature. An extensive survey of validation metrics is available in [70]. In [17] we evaluated several unsupervised criteria and measured their effects and similarities.

The idea behind clustering validation is simple, and most metrics express a ratio between cluster compactness and intra-cluster distances (separatedness).

An example of one such validation criterion is the C-index, which was introduced by Hubert and Levin [81] in 1976. It is computed as

\[
p_{\text{C-index}}(C) = \frac{d_{w} - \min(d_{w})}{\max(d_{w}) - \min(d_{w})}
\]

where \(d_{w}\) is the sum of the intra-cluster distances. This index was found to exhibit excellent recovery characteristics by Milligan [104], who used the minimum value across the hierarchy levels to indicate the optimal number of clusters [105].
CHAPTER 5. ALGORITHM SELECTION IN CLUSTERING

Deviation \[72\] minimizes the distances to a centroid within a cluster. It is defined as:

\[
Dev(C) = \sum_{C_k \in C} \sum_{i \in C_k} \delta(i, \mu_k) \tag{5.1}
\]

where \( C \) is the set of all clusters, \( \mu_k \) is the centroid of the cluster \( C_k \) and \( \delta(., .) \) is a chosen distance function (in our case Euclidean distance).

Connectivity \[72\] reflects the connectedness of items in a cluster. Clusterings with a low value of connectivity might have arbitrary shapes (non-spherical shapes), unlike solutions typically produced by algorithms like \( k \)-means. Connectivity evaluates the degree to which neighbouring data-points have been placed in the same cluster. It is computed as:

\[
Conn(C) = \sum_{i=1}^{N} \left( \sum_{j=1}^{L} x_{i,nn_{ij}} \right), \tag{5.2}
\]

where
\[
x_{r,s} = \begin{cases} 1, & \text{if } \not\exists C_k : r \in C_k \land s \in C_k \\ 0, & \text{otherwise}, \end{cases}
\]

\( nn_{ij} \) is the \( j \)th nearest neighbour of item \( i \), \( N \) is the size of the data set and \( L \) is a parameter determining the number of neighbours that contribute to the connectivity measure.

The Davies-Bouldin index \[39\] combines two measures, one related to dispersion and the other to separation between different clusters. Mathematically,

\[
p_{DB} = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} \left( \frac{\bar{d}_i + \bar{d}_j}{d(C_i, C_j)} \right) \tag{5.3}
\]

where \( d(C_i, C_j) \) corresponds to the distance between the centre of clusters \( C_i \) and \( C_j \), and \( \bar{d}_i \) is the average within-group distance for cluster \( C_i \).

\[
\bar{d}_i = \frac{1}{|C_i|} \sum_{l=1}^{|C_i|} d(x_i(l), \bar{x}_i) \tag{5.4}
\]

It is desirable for the clusters to have maximum separation from each other, therefore we seek clustering that minimizes the Davies-Bouldin index.

The Akaike information criterion \[9\] is typically used in supervised learning when trying to estimate model error. Essentially it attempts to estimate the optimism of the model and add it to the model error \[74\].

\[
p_{AIC} = -2 \cdot \log \left( \frac{L_n(k)}{n} \right) + 2 \cdot \frac{k}{n} \tag{5.5}
\]

where \( L_n(k) \) is the maximum likelihood of a model with \( k \) parameters based on a sample of size \( n \).
5.1. Algorithm Quality and Evaluation Criteria

BIC (the Bayesian information criterion) works in a similar way.

The SD index was introduced in 2000 by Halkidi et al. [69] and it is based on concepts of average cluster scattering and total separation of clusters which were previously used by Rezaee et al. [126] for evaluation of fuzzy clusterings.

The average scattering is defined as:

$$Scatt(k) = \frac{1}{k} \sum_{i=1}^{k} \frac{\|\sigma(\bar{c}_i)\|}{\|\sigma(X)\|}$$  \hspace{1cm} (5.6)

where $\|x\|$ is the norm of a vector, $\bar{c}_i$ is the centroid of the $i$-th cluster, $\sigma(X)$ is the variance of the input dataset.

$\sigma(X) \in \mathbb{R}^m$ with $m$ being the number of dataset dimensions. Variance for a dimension $d$ is defined as:

$$\sigma^d = \frac{1}{n} \sum_{i=1}^{n} (x_i^d - \bar{x}^d)^2$$

$$||\sigma(X)|| = \sqrt{\sum_{i=1}^{m} (\sigma^d)^2}$$

The total separation is given by:

$$Dis(k) = \frac{D_{max}}{D_{min}} \sum_{i=1}^{k} \left( \sum_{j=1}^{k} ||\bar{c}_i - \bar{c}_j|| \right)^{-1}$$ \hspace{1cm} (5.7)

where $D_{max}$ is the maximum distance and $D_{min}$ is the minimum distance between cluster centers ($\bar{c}_i$) and $k$ is the number of clusters.

$$D_{max} = \max_{i,j \in \{1, \ldots, k\}} (||\bar{c}_i - \bar{c}_j||)$$ \hspace{1cm} (5.8)

$$D_{min} = \min_{i,j \in \{1, \ldots, k\}} (||\bar{c}_i - \bar{c}_j||)$$ \hspace{1cm} (5.9)

Then we can define the SD validity index as follows:

$$f_{SD} = \alpha \cdot Scatt(k) + Dis(k)$$ \hspace{1cm} (5.10)

where $\alpha$ should be a weighting factor equal to $Dis(c_{max})$ with $c_{max}$ being the maximum number of clusters [69]. This makes perfect sense for fuzzy clustering (as was proposed in [126]), however it is rather unclear how to compute $c_{max}$ in the case of crisp clustering,
when $c_{\text{max}} \gg k$ without running another clustering with $c_{\text{max}}$ as the requested number of clusters. Nonetheless, [69] mentions that “SD proposes an optimal number of clusters almost irrespective of $c_{\text{max}}$, the maximum number of clusters”, thus we consider the special case where $c_{\text{max}} = k$:

$$f_{\text{SD}} = \text{Dis}(k) \cdot \text{Scat}(k) + \text{Dis}(k)$$

$$= \text{Dis}(k) \cdot (\text{Scat}(k) + 1)$$

Robustness [158] is important performance criterion and many cluster ensembles explained below are designed to improve this measure.

### 5.1.1 External validation

Another group of similarity measures consists of indices that can be used for comparing two clusterings, or a single clustering to external labels that are however often not available. Albatineh et al. [10] made a comprehensive list of 22 different indices of this type, and found that after a correction of chance some of those indices become equivalent. The most popular index seems to be the Adjusted Rand index [80], which was introduced by Hubert and Arabie in 1985. This index was also used for the evaluation of results in studies [72] and [47].

Yet another important group of measures is built upon fundamental concepts from information theory. The mutual information between two clusterings measures how much knowing one of these clusterings reduces our uncertainty about the other [119]. Kvalseth [102] introduced Normalized Mutual Information (NMI) which was later used by Strehl and Ghosh [142] and sometimes is referred as $\text{NMI}_{\text{sum}}$. This measure is more stable than the Adjusted Rand Index, especially when dealing with a higher number of clusters. This measure was also used in several follow-up works in the context of ensemble clustering [151], [75].

Note that the external validation is often not applicable as soon as target variables are not available. In many datasets, the target variable is intended for classification purposes and classes are not natural data clusters - more likely data density around class decision boundaries is high in order to help classifiers construct robust class separation manifolds. In such case, external validation using these target variables would be an ill-posed task.

It is therefore preferable to focus on unsupervised criteria.

### 5.1.2 Usefulness of single criteria

Similarly to performance evaluation of predictive models, we would like to see if any of single clustering criterion is good predictor of clustering quality. Using a brute-force approach we generated 250 unique clustering results using the Iris data set, which we sorted according to 21 unsupervised relative indexes. An ideal sorting criterion should be able to distinguish between good and bad results. As a reference for “correct” sorting we used external class labels that were not used during the clustering process.
5.1. ALGORITHM QUALITY AND EVALUATION CRITERIA

Our clustering results were produced using a hierarchical agglomerative clustering algorithm with various settings and data preprocessing. The quality of the results is varied and the number of clusters ranges from 2 to 147 (where almost each instance is in a separate cluster). Figure 5.2 presents a visualization of the sorting results. Each bar represents a clustering result; the height of the bar is proportional to a supervised metric (Adjusted Rand Index or NMI), and its position on axis $x$ corresponds to an unsupervised sorting. The best results should be towards the left, the worst results on the right.

A single objective function for sorting a set of clusterings is very hard to find for real-world data sets. Single criteria can only correlate with external criteria for simple data sets such as the Zoo data set, as can be observed for the Point-BiSerial criterion [106] (Figure 5.1).

Figure 5.2a shows that Silhouette is quite a good estimator of both the Adjusted Rand Index (ARI) and NMI (Figure ) external measures. However, some clusterings with higher Silhouette values do not have high supervised index values.

Sorting using the Davies-Bouldin index [39] (Figure 5.2c) places clusterings with low ARI values next to items with high ARI values. Red bars signify incorrect placement, since the clusterings should be on the opposite side of the scale. This makes the Davies-Bouldin index unusable (at least for the Iris data set) because we can not distinguish between the good and the bad solutions.

A similar problem is found with AIC sorting (Figure 5.2e); there are a few results among those on the left hand side which are false positive results, marked in red. Looking at Figure 5.2f we notice that according to NMI the clusterings are not as bad as it would seem with ARI. Moreover, it should be noted that AIC does not project all results to a single value: the clusterings are almost evenly divided across the range.

C-index [81] produces slightly better results, as the first half of the clusterings produced would probably include a few high-quality clusterings. However, the best C-index values do not correlate with the best ARI or NMI values (Figure 5.2g, 5.2h). A “golden standard” clustering (a hand made clustering based on class labels), marked yellow in the figure, is located in the middle of the C-index scale which might suggest that ideal clustering in the

Figure 5.1: Sorted clustering results of Zoo data set with Point-BiSerial criterion.
Iris data set is given by average C-index values.

Moreover, each criterion might be suitable for a different type of data. Our visualization illustrates this complexity, and demonstrates that no single objective function can be used as performance criterion alone. Except for simple data sets, like Zoo data, we were not able to find any combination of internal and external criteria which are correlated.

We do not need correlation (linear dependence) between the unsupervised criterion and NMI. We are looking for a criterion, that produces similar ranking. This can be expressed by the Spearman’s correlation rank (SCR) which summarize the strength and direction (negative or positive) of a relationship between two variables (even non-linear one).

Table 5.1 shows SCR between clusterings sorted by a supervised criteria (NMI$_{sqrt}$) and unsupervised metrics on several data sets.

Again, as you see, it is hard to find an unsupervised performance measure that would estimate NMI score with a reasonable confidence for all datasets. However for each dataset, at least one unsupervised criterion might be used to navigate in the space of possible clusterings in the right direction.

We extended our experiment to other criteria and in Figure 5.3 we show SCR ranking encoded into color. There are two major groups of unsupervised criteria based on similarity of their rankings. One group gives emphasis on cluster compactness, the other to separation of clusters. When you look at the Iris data, compactness of clusters measured by unsupervised criteria is in accordance to the supervised measure NMI. The is however not the case of unsupervised criteria measuring the cluster separation. For Iris data, supervised labels are misleading because there are in fact two physical clusters, not three, as labels indicate.

As you see, compactness and separation based criteria are contradicting and it is hard to maximize both at the same time. It is interesting, that criteria that combine both compactness and separation are not very successful in ranking clustering according to their NMI. However the goal of clustering is having both compact and well separated clusters so we are experimenting with multi-objective approaches bellow.

Our future work is to employ meta-learning methods and recommend proper set of unsupervised criteria for given data set based on meta-features only. In later section we assume, that we already found unsupervised criteria to evaluate the performance of clustering algorithms.

---

1. [https://github.com/deric/clustering-benchmark](https://github.com/deric/clustering-benchmark)
2. Data was generated based on visual similarity to an image in the referred paper, we were unable to obtain the dataset from the original author.
3. C-index
4. Calinski-Harabasz
5. Gamma
6. PointBiSerial
7. Silhouette
8. Ratkowsky-Lance
5.1. Algorithm Quality and Evaluation Criteria

(a) Silhouette – ARI

(b) Silhouette – NMI

(c) Davies-Bouldin – ARI

(d) Davies-Bouldin – NMI

(e) AIC – ARI

(f) AIC – NMI

(g) C-index – ARI

(h) C-index – NMI

Figure 5.2: Visualization of Iris data set clustering results sorted by internal validation index compared to external (supervised) validation index. The color of the bar signifies its distance from ideal placement (green – correct, black – misplaced by half the scale, red – incorrect placement). Yellow bars represent clustering according to external labels.
Table 5.1: Spearman’s correlation rank (SCR) between sorted clusterings by NMI and given unsupervised (internal) metric. 18 clusterings were generated by 10 different clustering algorithms used in our benchmark (non deterministic algorithms were repeated 5 times). The SCR value is an average from 10 independent runs. There is no universal metric that would be working on all problems. Data files used in experiments are available online.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>dim</th>
<th>size</th>
<th>classes</th>
<th>DB</th>
<th>Dunn</th>
<th>C-ind.</th>
<th>C-H</th>
<th>Silhou</th>
<th>Gamma</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-spiral</td>
<td>2</td>
<td>312</td>
<td>3</td>
<td>-0.655</td>
<td>-0.038</td>
<td>-0.007</td>
<td>0.638</td>
<td>-0.690</td>
<td>0.424</td>
</tr>
<tr>
<td>aggregation</td>
<td>2</td>
<td>788</td>
<td>7</td>
<td>0.310</td>
<td>0.429</td>
<td>-0.278</td>
<td>-0.307</td>
<td>0.310</td>
<td>-0.224</td>
</tr>
<tr>
<td>atom</td>
<td>3</td>
<td>800</td>
<td>2</td>
<td>0.602</td>
<td>0.243</td>
<td>-0.763</td>
<td>0.444</td>
<td>-0.299</td>
<td>0.483</td>
</tr>
<tr>
<td>balance-scale</td>
<td>4</td>
<td>625</td>
<td>3</td>
<td>-0.558</td>
<td>0.872</td>
<td>-0.269</td>
<td>0.739</td>
<td>0.412</td>
<td>-0.844</td>
</tr>
<tr>
<td>chainlink</td>
<td>3</td>
<td>1000</td>
<td>2</td>
<td>-0.396</td>
<td>-0.062</td>
<td>-0.677</td>
<td>0.595</td>
<td>-0.195</td>
<td>0.618</td>
</tr>
<tr>
<td>compound</td>
<td>2</td>
<td>399</td>
<td>6</td>
<td>-0.540</td>
<td>0.109</td>
<td>-0.400</td>
<td>0.111</td>
<td>-0.382</td>
<td>0.204</td>
</tr>
<tr>
<td>ds-850</td>
<td>2</td>
<td>850</td>
<td>5</td>
<td>0.408</td>
<td>0.562</td>
<td>-0.122</td>
<td>-0.608</td>
<td>0.736</td>
<td>-0.502</td>
</tr>
<tr>
<td>ecoli</td>
<td>7</td>
<td>336</td>
<td>7</td>
<td>0.108</td>
<td>0.301</td>
<td>0.752</td>
<td>-0.164</td>
<td>0.224</td>
<td>-0.848</td>
</tr>
<tr>
<td>flame</td>
<td>2</td>
<td>240</td>
<td>2</td>
<td>0.570</td>
<td>0.582</td>
<td>0.465</td>
<td>-0.562</td>
<td>0.549</td>
<td>-0.556</td>
</tr>
<tr>
<td>glass</td>
<td>9</td>
<td>217</td>
<td>7</td>
<td>0.761</td>
<td>0.527</td>
<td>0.160</td>
<td>-0.634</td>
<td>0.834</td>
<td>-0.801</td>
</tr>
<tr>
<td>heart-statlog</td>
<td>13</td>
<td>270</td>
<td>2</td>
<td>0.443</td>
<td>0.513</td>
<td>0.789</td>
<td>-0.009</td>
<td>0.372</td>
<td>-0.413</td>
</tr>
<tr>
<td>ionosphere</td>
<td>34</td>
<td>351</td>
<td>2</td>
<td>0.299</td>
<td>0.288</td>
<td>0.449</td>
<td>0.346</td>
<td>0.186</td>
<td>-0.829</td>
</tr>
<tr>
<td>iris</td>
<td>4</td>
<td>150</td>
<td>3</td>
<td>0.235</td>
<td>0.356</td>
<td>0.150</td>
<td>-0.645</td>
<td>0.273</td>
<td>-0.342</td>
</tr>
<tr>
<td>jain</td>
<td>2</td>
<td>373</td>
<td>2</td>
<td>-0.548</td>
<td>0.284</td>
<td>-0.581</td>
<td>0.295</td>
<td>-0.443</td>
<td>0.447</td>
</tr>
<tr>
<td>long1</td>
<td>2</td>
<td>1000</td>
<td>2</td>
<td>-0.206</td>
<td>0.399</td>
<td>-0.290</td>
<td>0.199</td>
<td>-0.201</td>
<td>0.170</td>
</tr>
<tr>
<td>longsquadre</td>
<td>2</td>
<td>900</td>
<td>6</td>
<td>-0.086</td>
<td>0.247</td>
<td>0.252</td>
<td>-0.287</td>
<td>0.146</td>
<td>-0.440</td>
</tr>
<tr>
<td>lsun</td>
<td>2</td>
<td>400</td>
<td>3</td>
<td>0.385</td>
<td>0.123</td>
<td>-0.824</td>
<td>0.476</td>
<td>0.347</td>
<td>0.550</td>
</tr>
<tr>
<td>new-thyroid</td>
<td>5</td>
<td>215</td>
<td>3</td>
<td>0.400</td>
<td>0.487</td>
<td>0.200</td>
<td>-0.331</td>
<td>0.427</td>
<td>-0.589</td>
</tr>
<tr>
<td>pathbased</td>
<td>2</td>
<td>300</td>
<td>3</td>
<td>-0.287</td>
<td>0.429</td>
<td>0.529</td>
<td>-0.235</td>
<td>-0.064</td>
<td>-0.108</td>
</tr>
<tr>
<td>sonar</td>
<td>60</td>
<td>296</td>
<td>2</td>
<td>0.430</td>
<td>0.104</td>
<td>0.750</td>
<td>0.148</td>
<td>0.243</td>
<td>-0.259</td>
</tr>
</tbody>
</table>
5.1. ALGORITHM QUALITY AND EVALUATION CRITERIA

Figure 5.3: The benchmark in previous Figure was extended and criteria were clustered based on similarity of their results on datasets. There are three main classes of datasets based on results of unsupervised criteria. First class are simple, mostly 2D datasets with well separated clusters, where traditional compactness criteria often fail e.g. because compactness cannot be measured by the Euclidean distance. Second class are datasets with compact clusters where cluster separation is difficult to measure and third major group in our benchmarking portfolio are ill-posed datasets similar to second class but here is no unsupervised criterion that correlates strongly with NMI and can be used to optimize data clustering.
5.2 Ensembling Strategies in Clustering

The goal of ensemble methods is to find a consensus partition that agrees as much as possible with all the base partitions. Strehl and Ghosh [142] proposed three heuristics for solving the ensemble problem. The CSPA (Cluster-based Similarity Partitioning Algorithm) combines multiple clusterings based on their pairwise similarity. The HGPA (Hyper-Graph Partitioning Algorithm) partitions a hypergraph where hyperedges represent clusters. Lastly, the MCLA (Meta-CLustering Algorithm) identifies groups of clusters (meta-clusters) and consolidates them.

It is worth noticing that all these advanced algorithms still rely on traditional clustering algorithms, which can be very effective in revealing the type of structure they were designed for. A disadvantage of these ensemble methods is, however, that the final result is usually a single clustering.

### 5.2.1 Multi-objective ensembles

Faceli et al. [47] combined a multi-objective approach to clustering with ensemble methods and the resulting algorithm is called MOCLE (Multi-Objective Clustering Ensemble Algorithm). The objectives used in the MOCLE algorithm are the same as those used in MOCK [72]: deviation (eq. 5.1) and connectivity (eq. 5.2). Unlike MOCK, in this case the evolutionary algorithm used is NSGA-II [40].

Ensemble Clustering based on Quadratic Mutual Information (QMI) [148] searches for the consensus clustering by maximizing the mutual information between the consensus partition and the ensemble partitions via EM algorithm [41].

MOCLE is capable of selecting clusterings with a high Adjusted Rand Index value, however it should be noted that the algorithm is initialized with high-quality partitions produced by carefully configured algorithms with a number of clusters close to the optimal value. In real world scenarios this kind of initialization is impossible especially when exploring an unknown data set.

The main objective of clustering ensembles is to combine multiple clusterings into one, preferably high-quality solution.

Let \( X = \{x_1, x_2, \ldots, x_n\} \) be a set of \( n \) data points, where each \( x_i \in X \) is represented by a vector of \( d \) attributes. A cluster ensemble is defined as \( \Pi = \{\pi_1, \pi_2, \ldots, \pi_m\} \) with \( m \) clusterings. Each base clustering \( \pi_i \) consists of a set of clusters \( \pi_i = \{C_1^i, C_2^i, \ldots, C_{k_i}^i\} \), such that \( \bigcup_{j=1}^{k_i} C_j^i = X \), where \( k_i \) is the number of clusters in a given ensemble member (it does not have to be the same for all members).

The problem is how to obtain a final clustering \( \pi^* = \{C_1^*, C_2^*, \ldots, C_K^*\} \), where \( K \) is the number of clusters in the result and \( \pi^* \) summarizes the information from ensemble \( \Pi \).

The process consists of two major steps. Firstly, we need to generate a set of clustering solutions and secondly we need to combine the information from these solutions. A typical result of the ensemble process is a single clustering. It has been shown in supervised ensembles that the best results are achieved when using a set of predictors whose errors are dissimilar [89]. Thus it is desirable to introduce diversity between ensemble members.
5.2. Ensemble Generation Strategies

Several approaches have been used to initialize clustering solutions in order to create an ensemble.

- **Homogeneous ensembles**: Base clusterings are created using repeated runs of a single clustering algorithm. This is quite a popular approach, repeated runs of k-means with random center initialization have been used in [53]. When using k-means the number of clusters is typically fixed to \(\lceil \sqrt{n} \rceil\), where \(n\) is the size of the dataset [82].

- **Varying \(k\)**: Repeated runs of k-means with random initialization and \(k\) [60], a golden standard is using \(k\) in the range from 2 to \(\sqrt{n}\).

- **Random subspacing** An ensemble is created from base clusterings that use different initial data. This could be achieved by projecting data onto different subspaces [48], [66] choosing different subsets of features [142], [163], or using data sampling techniques [44].

- **Heterogeneous ensembles**: Diversity of solutions is introduced by applying different algorithms on the same dataset.

5.2.3 Consensus Functions

Having multiple clusterings in an ensemble, many functions have been proposed to derive a final clustering. When only one solution is considered as the result, it is usually referred as a consensus function, unlike meta clustering where the output is a set of multiple clusterings [34].

There are several approaches as to how to represent information contained in base clusterings, some use matrices while others use graph representation.

- **Pairwise similarities**: A pairwise similarity matrix is created and afterwards a clustering algorithm (e.g. hierarchical agglomerative clustering) is applied to group together items that were most frequently together in the same cluster in all the base clusterings [53]. the Cluster-based Similarity Partitioning Algorithm (CSPA) from Strehl and Ghosh [142] uses METIS [87] for partitioning a similarity matrix into \(k\) components.

- **Feature-based approach**: The ensemble problem is formulated as categorical data clustering. For each data point an \(m\)-dimensional vector containing labels in base clusterings is created. The goal is to find a partition \(\pi^*\) which summarizes the information gathered from the partitions \(\Pi\) [117, 149, 25].

- **Graph based**: Many methods use graph representation for capturing relationships between base clusterings. Strehl and Ghosh [142] also proposed the HyperGraph-Partitioning Algorithm (HGPA), where vertices correspond to data points and a
CHAPTER 5. ALGORITHM SELECTION IN CLUSTERING

Figure 5.4: Cluster Ensemble: Firstly using the same input dataset we generate multiple clusterings, then select best clusterings for the ensemble and combine them into a single solution. Diversity of base clusterings can be promoted by using different algorithms or by using multi objective approach to cluster candidates generation.

hyperedge represents clusters. Another approach chooses COMUSA [108] which increases the weight of the edge for each occurrence of data pairs in the same cluster. Afterwards the nodes are sorted by the attachment score, which is defined as the ratio between the sum of the node’s weights and its number of incident edges. The nodes with the highest attachment score are then used as a foundation for new clusters. This approach is relatively fast to compute, however it might fail to capture complex relationships between very diverse clusterings.

There are many ways to produce a final consensus, nonetheless in this work we focus on the selection of high-quality and diverse clusterings. This is in accordance to some studies [168] in predictive modeling.

In order to optimize ensemble member selection and generate ensembles with diverse base clusterings, we apply a multi-objective optimization. This approach is somehow similar to Negative correlation learning in predicive modeling [147]. The whole ensembling process is shown in Fig. 5.4.

5.3 Meta-Learning Templates in Clustering

Resembling ensembles in predictive modeling, clustering ensembles can be also organized in hierarchies. Great overview of recent ensembling approaches in clustering is available in [162]. As you see, the list of possible ensembling methods is really broad. Ensembles work in two phases distributing problem and generating consensus. For each phase many methods are available and number of possible combinations is very high. Parameters of clustering methods are of the large importance and should not be ignored.

One example of a cluster meta-learning template is depicted on Figure 5.5. It illustrates consensus ensembles described in [162]. This template is specific, because all methods and parameters are specified. One can generalize this template and experiment whether similar templates are able to generate better clustering.
5.4 Template Optimization Strategies

The optimization of meta-learning templates is not easy, given that the performance of templates is difficult to measure/estimate. We have three options - the first and preferable option is to recommend the combination of performance measures by meta-learning. This option is however hard to implement and our result and results of other researchers [138] are still not good enough to use these criteria as objective function.

The second option is using manually selected criteria and run a multi objective evolutionary algorithm to find Pareto dominant (see 6.1.5) clusterings.

5.4.1 Evolutionary optimization of clustering

The first multi-objective evolutionary clustering algorithm was introduced in 2004 by Handl and Knowles [73] and is called VIENNA (the Voronoi Initialized Evolutionary Nearest-Neighbour Algorithm).

In MOCK, each individual is represented as a directed graph which is then translated into clustering. The genotype is encoded as an array of integers whose length is same as the number of instances in the data set. Each number is a pointer to another instance (an edge in the graph), since it is connected to the instance at a given index. This easily enables the application of mutation and crossover operations.

As a Multi-Objective Evolutionary Algorithm (MOEA) MOCK employs the Pareto Envelope-based Selection Algorithm version 2 (PESA-II) [36], which keeps two populations, an internal population of fixed size and a larger external population which is exploited to explore good solutions. Two complementary objectives, deviation and connectivity, are used as objectives in the evolutionary process.

A clear disadvantage of MOCK is its computation complexity, which is a typical characteristic of evolutionary algorithms. Nevertheless, the computation time spent on MOCK should be rewarded with high-quality solutions. Faceli et al. [47] reported that for some high-dimensional data it is not guaranteed that the algorithm will complete, unless the control front distribution has been adjusted for the given data set.
In order to do that, we employed a multi-objective evolution algorithm which uses NSGA-II [40].

Our evolutionary algorithm operates on a meta-algorithm (template) level. There are many parameters which can be adjusted in order to obtain reasonable clustering. It is difficult to decide which normalization will be better for a given data set before analysing the data. Therefore typical steps of data preprocessing are encoded in the individual. We use normalizations typically used in cluster analysis [107]: z-score formula, min-max normalization, division by the maximum value of that attribute (transforming the value into a value between $-1$ and 1) and standardized measurement (subtracting the mean value of the attribute and dividing by its mean absolute deviation) or no standardization. Another parameter determines whether to logarithm the whole data set or not.

Table 5.2: Comparison of the best clusterings found during multi-objective evolution with combination of two objectives. Best ARI is the value of the solution with the highest Adjusted Rand Index value found during evolution. BF ARI and BF NMI are Adjusted Rand Index values and Normalized Mutual Information values found by a brute-force algorithm which tried all possible settings of the given clustering algorithms.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Best objectives</th>
<th>Best ARI</th>
<th>BF ARI</th>
<th>BF NMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>AIC/BIC &amp; Calinski-Harabasz</td>
<td>0.88</td>
<td>0.88</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>C-index &amp; Deviation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C-index &amp; Davies-Bouldin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Glass</td>
<td>C-index &amp; Hybrid Centroid Similarity</td>
<td>0.70</td>
<td>0.70</td>
<td>0.31</td>
</tr>
<tr>
<td></td>
<td>AIC &amp; PointBiserial</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AIC/BIC &amp; Sum of AVG parwise sim.</td>
<td>0.67</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Deviation &amp; Sum of AVG parwise sim.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AIC/BIC &amp; Sum of squared errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wine</td>
<td>C-index &amp; min-max cut</td>
<td>0.90</td>
<td>0.95</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>min-max cut &amp; Sum of Centroid Sim.</td>
<td>0.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BIC &amp; Silhouette</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AIC &amp; min-max cut</td>
<td>0.81</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To simplify the problem, we use a limited class of clustering templates. As a base clustering algorithm we use hierarchical agglomerative clustering with different linkage
Figure 5.6: Final Pareto front produced by NSGA-II algorithm (10 generations, population 20, crossover 50%, mutation 20%). Each dot represents a clustering result and is colored according to the Adjusted Rand Index value. The number next to each result marks the number of clusters.

methods (single linkage, complete linkage, average linkage and Ward’s linkage). After producing a dendrogram we applied a heuristic to convert a hierarchical structure into a flat partitioning. One heuristic is based on a hill-climbing algorithm. It cuts the dendrogram at each level and computes a relative validation index for each cut. When going to the next level would not improve the index, the algorithm stops.

An individual is encoded as an array of integers and each index in the genome is mapped to a certain parameter, which can have any range. All clustering parameters are nominal values. We used a polynomial mutation as it is implemented in the original NSGA-II and Simulated Binary Crossover (SBX). The probability of crossover was 50% and mutation probability 20%. For each dataset we evaluated $\binom{21}{2} = 210$ combinations of two objective functions to find out the optimum of supervised criteria.

In Table 5.2 the best combinations of objectives for evolution of clusterings with two objectives are presented. Introducing AIC (or BIC) as one objective can significantly improve the results of multi-objective evolution. We managed to find a pair of objectives for each data set, whereby the final Pareto front with at most 10 individuals includes a solution matching the best solution found by the brute-force approach, or whose ARI value is very close to the optimum. Each evolution run was repeated 5 times; an example of a
resulting Pareto front is shown in Figure 5.6. Some objectives always converge to the same set of solutions, others produce very diverse results. We would like to examine these properties in our follow up work.

The third optional approach to evolution of templates is so called interactive evolution [15]. We performed several experiments and our preliminary results suggest that number of fitness evaluation human has to perform is to high for real-world data sets. Our future work is to experiment with surrogate modeling to decrease number of required interactions.

5.5 Meta-Features, Meta-Database and Meta-Data Mining

Meta-learning can be another solution to the problem of ill-defined performance measures in clustering and too big search space. We propose to recommend set of unsupervised criteria suitable for given clustering problem. It is also possible to recommend suitable algorithm [138], but the accuracy is still not very high and we have to deal with the optimization of parameters as well.

Preliminary example of performance (average NMI of generated clusterings) in the Figure 5.7. As a future work, we plan to run extensive experiments to evaluate true potential of individual algorithms (templates) on benchmarking problems. This results will be used in the meta-database to recommend templates for new problems.

So far, we were not able to automate algorithm selection in clustering domain. We overcome lot of difficulties with the performance criteria and our future work is to prepare evolutionary system with anytime properties similar to those presented for supervised models.
Figure 5.7: The results are preliminary, much more extensive benchmarks have to be executed. Parameters of algorithms have to be considered. For each algorithm one has search the space of possible parameter settings to evaluate true potential. In this setup we made just ten random evaluations.
Chapter 6

Algorithm Selection in Recommender Systems

Recommender systems are in fact special type of predictive models. They work with large sparse matrices of interactions and need to be able producing predictions for millions of users - that is why algorithms differ significantly. Sometimes traditional predictive models can be used in recommender system domain - for example on transformed and reduced data and recommender systems can be used in predictive modeling e.g. to recommend parameters of algorithms [135]. In this chapter, we apply the TASA process to the domain of recommender systems.

6.1 Algorithm Quality and Evaluation Criteria

When evaluating algorithm performance in Recommender Systems (RSs), one of the most prominent approaches is to use predictive accuracy measures, such as the root mean squared error (RMSE) and the mean absolute error (MAE) [77]. The task is to predict rating of an item a user have not rated yet. RMSE is computed as the difference between predicted and actual rating.

Recent research suggests that lower RMSE may not necessarily lead to better recommendations. In [32], it is not only shown that lower RMSE may not improve classification measures such as the recall or the precision, but that these measures may even be in direct conflict with the RMSE. Very similar conclusions are also derived in [38], stating that classification measures (precision, recall) are of much greater practical importance than the RMSE. The main objective of vast majority of RSs is to find good items to solve the Top-N recommendation task, where rating prediction is only an instrument, not the goal.

Accepting that the predictive accuracy measures are generally ill-posed, the subsequent natural candidates are the aforementioned recall and precision. In contrast to RMSE/MAE, these measures consider recommendation as a binary classification rather than numerical regression task. In case of Top-N recommendation, several methodologies [32, 140] have been proposed to only use recall as the only model-quality measure, elim-
inatating the necessity to also measure the \textit{precision} or to put both together into so-called F-measure.

Even though using recall as the model quality measure gives the evaluation a meaning and the values obtained are much easier to interpret in comparison to the RMSE, it turns out that another problems appear. One of the most important is the good model’s bias towards the most \textit{popular items}. In fact, to achieve high recall, it seems as a quite good strategy to only recommend the most popular items in the database, completely ignoring the so-called \textit{long-tail} items. This, however, contradicts other important goals of Recommender Systems, such as the \textit{serendipity} and the \textit{novelty}, i.e., the ability to offer surprisingly interesting items that the users would normally miss.

Recently, there were several approaches published to take the serendipity and the novelty into account. A common way to penalize the focus solely on the popular items is to bias the testing set for each user, preferring the long-tail items \cite{38, 121}. In \cite{141}, another approach, called the \textit{popularity-stratified recall}, is presented, assigning relevant items from the testing set a weight inverse proportional to how frequently the item is being rated in the training data. Furthermore, a modified training method for a low-rank matrix factorization model in presented in \cite{141} for the model to better fit the biased measure.

We proposed a novel way of model evaluation, taking the recommendation for the long tail into account. In contrast to \cite{141}, we do not bias the recall measure and still consider it good enough in its pure form. To our opinion, it is quite hazardous to optimize the measure rather than the model itself. Instead, we propose \textit{additional measures} to be taken into account and being evaluated \textit{simultaneously}, resulting in a \textbf{multi-objective optimization problem}. Specifically, we couple the recall together with the \textit{catalog coverage}, measuring how many different items the model recommends. We demonstrate our approach on two algorithms, the \textit{User-based k-Nearest Neighbors} (\textit{k-NN}) and the \textit{Association Rules}, which we intentionally selected as being conceptually very different. We show that despite their dissimilarity, they share very similar behavior in the Recall-Coverage plane. Also, we demonstrate that the regularization technique proposed in \cite{141} for the matrix decomposition algorithm may be used for both the \textit{k-NN} and the Association Rules as well.

### 6.1.1 Performance Measures in Recommender Systems

#### 6.1.1.1 Accuracy

When evaluating CF systems, most of the approaches only focus on the their \textit{predictive accuracy} \cite{57}. Typically, a set of test users is used to evaluate a model, possibly as a part of split or cross-validation process \cite{32}. For each test user, some of the ratings in his/her history are submitted to a model, and the rest are compared to the model’s response. Concerning accuracy, based on the nature of the recommendation model, there is a whole range of possible measures.

A lot of research has been done in algorithms focusing on \textit{rating prediction} (e.g. number of stars) that users would assign to individual items. Systems based on rating
predictions typically predict the ratings of all the items in the database and recommend the items with the highest predictions for a given user. For rating predictions, the accuracy is typically expressed in terms of numerical error measures such as the RMSE or Normalized Mean Average Error (NMAE) [67]. These have been thoroughly studied in the literature for years.

The task of recommending items might also be viewed as **binary classification** [77], aiming to predict items as being **relevant** or **irrelevant** to a given user [140]. For the algorithms focusing primarily on this task, it becomes possible to use the measures known from Information Retrieval, such as the **precision** and the **recall**. Supposing the system being free to choose the number of items it recommends, one may express

\[
\text{precision} = \frac{\#TP}{\#TP + \#FP}, \quad \text{recall} = \frac{\#TP}{\#TP + \#FN} \tag{6.1}
\]

where \#TP (true positives) is the number of items recommended and being truly relevant, \#FP (false positives) is the number of items irrelevant yet wrongly recommended, and \#FN (false negatives) is the number of relevant items missed by the recommender. These two may be put together, forming F1 measure, which combines the precision and recall into a single number [77, 32].

Unlike in Information Retrieval, in Recommender Systems, the number of items to be recommended, \(N\), is usually fixed, leading to task known as the **Top-N recommendation** [38]. Fixing \(N\), however, brings some issues into measuring the precision and the recall. In many situations, it is important to handle the so-called cold-start problem, which emerges for users with not enough ratings (implicit or explicit) in their history [132]. This is especially true for the environment of World Wide Web with continuous arrival on newcomers that websites are facing. Therefore, we consider particularly important for the Recommender System to be able to offer good recommendations to the new users as well as to the long-term ones. This makes using the precision and the recall a bit challenging. Consider a test user with \(k\) relevant items hidden to a model. If \(k < N\), then the precision will always be \(\leq k/N\), i.e., the value of 1.0 might be unreachable for some test users. Symmetrically, for a long term users, having \(k > N\), it is the recall which cannot reach the value of 1.0, as the theoretical maximum is \(N/k\).

Several approaches have been proposed to solve this precision-recall dilemma. In [140], using only recall is suggested as it is proportional to precision with a user-dependent proportionality ratio. Nevertheless, because the ratio is user dependent, we don’t consider using pure recall fair. Instead, in our experiments, we adopt the leave-one-out cross-validation methodology as proposed in [32], hiding one relevant item a time, and letting the model to predict Top-N items for each hidden item (with constant \(N = 5\) in [32]). For a given user, the recall is computed as the proportion of relevant items which have appeared in the corresponding Top-N list during the cross-validation. This way of recall computation allows us to average the recall over all the test users, being fair to all of them.

**Definition 6.1.1** Let \(O_I\) be a set of all possible observations on items \(I\) and let \(N \in \mathbb{N}\).
A Top-N recommendation model is a projection $O_T \to 2^I$ such that for any
$
\{(i_1^1, r_1), \ldots, (i_m^r, r_m)\} \mapsto \{i_1, \ldots, i_k\}$ it holds:

1. $\{i_1^1, \ldots, i_m^r\} \cap \{i_1, \ldots, i_k\} = \emptyset$.

2. $k \leq N$.

Based on observed rating of some items, the model tries to recommend a set of $N$ items
different from those being observed. The actual number of recommended items $k$ may be
smaller than $N$ if the model is unable to use all $N$ slots.

### 6.1.1.2 Catalog Coverage

Despite its dominant position in the literature, accuracy alone may not reflect the business
needs or the pleasurable of user experience. This comes from the fact that the accuracy
is typically measured on a set of test users, where the task is to predict known interactions
with items that are hidden to the model for each user. This often pushes an accurate model
to specialize mainly on mainstream items for which there is a public consensus among the
users. This, however, contradicts another important goal of recommender systems, which
is discovery of novel, interesting items, that the user would normally miss.

To overcome the shortages of accuracy, other, complementary measures have been in-
troduced. One of the most prominent is the **Catalog coverage**, which measures the
number of items that the system is able to recommend [77, 57]. It is expressed simply as
the number of different items that the model recommends on a whole set of test users:

$$
catalog - coverage(m) = \frac{|\{Top - N(m, u) | u \in U_{test}\}|}{|I|}.
$$

If the system tends to recommend only the mainstream, it may reach high accuracy, but
will exhibit low coverage. That is why in many systems coverage decreases as a function
of accuracy [62]. Bellow, we examine this interaction thoroughly.

### 6.1.2 Popularity Bias

As discussed in [38, 121], it is a common case that in terms of recall, non-personalized
models, recommending only the most popular items, exhibit performance similar to much
more sophisticated, state-of-the-art ones. As pointed out in [141], the recall tends to
decrease towards the long tail. Moreover, in [141], two concepts were introduced that
we consider particularly important: the modified measure referred to as the **popularity-
stratified recall**, and a **popularity-biasing parameter** $\beta$, used in model training. While
we present a method different from the popularity-stratified recall to evaluate a model, we
still recognize the huge importance of the $\beta$ parameter used to control a model. Incidentally,
we re-discovered the exact same parameter independently when examining recall-coverage
trade-offs, and hence we fully agree with [141] and also use it in our experiments, yet in
context of different models.
Supposing a model predicts, for a given user-item pair \((u, i)\), the rank of a the given item, \(\text{rank}(u, i)\), one may further divide this rank by the item’s overall popularity \(\text{popularity}(i) \in (0.0, 1.0]\), to get popularity-stratified rank,

\[
\text{rank}_{PS}(u, i) = \frac{\text{rank}(u, i)}{\text{popularity}(i)}. \tag{6.3}
\]

Because such a bonification of unpopular items may be too drastic, especially for the least popular items for which \(\text{popularity}(i) \to 0\), it is the \(\beta\) parameter which is used to soften it:

\[
\text{rank}_{PS}^{\beta}(u, i) = \frac{\text{rank}(u, i)}{\text{popularity}(i)^\beta}, \tag{6.4}
\]

where \(\beta \in [0.0, 1.0]\) may be fine-tuned to achieve a reasonable compromise.

In [141], \(\text{popularity}(i)\) is computed, with reference to [140], as the proportion of users in the training set who gave the item \(i\) the maximal possible rating. Here, we use slightly different notion of popularity, simply as the \(\text{sum of all the ratings}\) of the item \(i\) in the training set:

\[
\text{popularity}(i) = \sum_{u \in U} (\hat{r}_{u,i} - \bar{r}_u). \tag{6.5}
\]

### 6.1.3 k-Nearest Neighbors Algorithm

In our experiments, we are using user-based \(k\)-Nearest Neighbors algorithm with cosine similarity and voting, a typical representative of neighborhood based collaborative filtering methods. In [38], such an algorithm is referred to as the \(\text{Non-normalized Cosine Neighborhood}\). To rank items in a user’s neighborhood, the following formula is used:

\[
\text{rank}(u, i) = \sum_{\hat{u} \in N_k(u)} \text{sim}(u, \hat{u}) \cdot (\hat{r}_{\hat{u},i} - \bar{r}_u). \tag{6.6}
\]

While such rank is traditionally divided by \(\sum_{\hat{u} \in N_k(u)} \text{sim}(u, \hat{u})\) to normalize the result and keep it within the original rating scale, omitting normalization brings two advantages:

1. There is no information loss about certainty. Such a ranking can be viewed as voting rather than averaging, preferring items relevant to more users in the neighborhood.

2. The method is especially suitable for the Top-\(N\) recommendation task and can easily be applied to unary and binary ratings.

In general, the \(k\)-nn algorithm (both item and user based) works according to Alg. 1.
Algorithm 1: General outline of neighborhood algorithms

**input**: Number of items to be recommended \( N \in \mathbb{N} \),
Number of neighbors used for ranking \( k \in \mathbb{N} \),
User to recommend items to \( u \),
List of all items \( \text{Items} \),
User-Item matrix of ratings \( R \)

**output**: \( N \) items to be recommended

```plaintext
foreach item \( \in \text{Items} \) do
    if item \( \notin u.rated\_items\) then
        item.rank \( \leftarrow \) rank\_according\_to\_nearest\_neighbors\((k, u, item)\)
    descending\_rank\_sort(\text{Items})

return top(\(N, \text{Items}\))
```

### 6.1.4 Association Rules

Besides k-NN, we focus on Association Rules (AR) which allow us to construct pattern-driven recommendations. We chose AR models, because they are largely different from neighborhood-based approaches, and our aim is to explore behavior of multiple algorithms in Recall-Coverage plane.

The problem of association rules mining was stated in [7] and further extended in [8]. We are given \((\mathcal{I}, \mathcal{D}, s_{\min})\), where \(\mathcal{I}\) is the set of items, \(\mathcal{D} = T_1, \ldots, T_m\) is a set of transactions and \(s_{\min} \in [0.0, 1.0]\) it the minimal support. Each transaction \(T_j \in \mathcal{D}\) is a subset of \(\mathcal{I}\). In the context of Recommender Systems, we may identify the set of transactions with the set of users, such as each transaction contains exactly the items which are relevant to a given user, based on either explicit or implicit ratings. To ease the readability in the context of recommendation domain, we will denote the transactions \(\mathcal{U}\) (for users) instead of \(\mathcal{D}\).

The original definition of association rule mining problem also includes an extra parameter, \(\alpha_{\min}\) called the minimal confidence, but because the confidence is not the only quality measure for a rule to be used here, we omit \(\alpha_{\min}\) at this point.

Given \((\mathcal{I}, \mathcal{U}, s_{\min})\), we are to find a set of association rules holding the minimal support. An association rule is an implication \(X \Rightarrow Y\) such that \(X \cup Y \in \mathcal{U}\), \(X \neq \emptyset\), \(Y \neq \emptyset\), \(X \cap Y = \emptyset\). We say that association rule \(X \Rightarrow Y\) holds the minimal support if

\[
\frac{|\{T \in \mathcal{U} \mid X \cup Y \in T\}|}{|\mathcal{U}|} \geq s_{\min},
\]

i.e. we require at least \(s_{\min} \cdot 100\%\) of users to contain \(X \cup Y\) as relevant items in their rating history.

One of the most popular algorithms to solve the AR-mining task is the APRIORI algorithm also proposed in [8]. APRIORI first searches for frequent itemsets in bottom-up
manner. From the frequent itemsets discovered, association rules are generated exhaustively. Since the original publication of APIRORI, many other algorithms have been proposed to mine association rules more efficiently, such as the ECLAT [165], the FP-growth [71], and many others. In our experiments, we use a modified, lazy version of the APRIORI algorithm, which finds the set of rules at the time of recommendation, pruning the search space of frequent itemsets with respect to a current test user.

### Algorithm 2: Weighted-Rules Recommendation

**input**: Set of train users $\mathcal{U}$, Test user $U \in \mathcal{U}$, Set of association rules $\mathcal{R}$, Number of items to recommend $N \in \mathbb{N}$

**output**: Top-$N$ recommendations $R(U) \subseteq \mathcal{I}$, $|R(U)| \leq N$

$\mathcal{R}^+ \leftarrow \{ (X \Rightarrow Y) \in \mathcal{R} \mid X \subseteq U \}$

$C \leftarrow \text{init\_table}()$

for $(X \Rightarrow Y) \in \mathcal{R}^+$ do

foreach $i \in (Y \setminus U)$ do

if $i \notin C$ then

$C[i] \leftarrow 0$

$C[i] \leftarrow C[i] + \text{measure}((X \Rightarrow Y), \mathcal{U})$

end if

end foreach

$S \leftarrow \text{descending\_sort\_by\_value}(C)$

$R(U) \leftarrow \emptyset$

for $i \leftarrow 1$ to $N$ do

$R(U) \leftarrow R(U) \cup \{S[i]\}$

end for

return $R(U)$

### 6.1.4.1 Rule-Quality Measures

There are multiple quality measures for association rules. Let us first denote the *support* of a set $A \subseteq \mathcal{I}$ as:

$$\text{supp}(A) = \frac{|\{U \in \mathcal{U} \mid A \subseteq U\}|}{|\mathcal{U}|}.$$  

The very first measure, proposed by the authors of the APRIORI algorithm in [8], is the *confidence*, defined as

$$\text{conf}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)},$$

i.e. as the proportion of users holding $X$ who also hold $Y$ in their relevant itemsets.

A comprehensive list of other measures is studied and evaluated in [14]. For the purposes of recommendation, we further investigate the *lift*, given as
The lift measure addresses the fact that some rules may have high confidence only by chance, even though $X$ and $Y$ are statically independent [14]. It is noteworthy that this corresponds to the problem of recommending the most popular items, ignoring the long tail. We further discuss this phenomenon in Sec. 6.1.5.

6.1.4.2 Rule-based Recommendation

Based on a set of association rules $\mathcal{R}$, it is possible to construct collaborative filtering algorithm, with a large variety of options.

In [130], the following simple algorithm has been proposed. When generating recommendations for user $U_j$, we collect all the rules $(X \Rightarrow Y) \in \mathcal{R}$ such that $X \subseteq U_j$. Then we sort the rules according to some measure and recommend the top-$N$ items from the right-hand sides of the rules from the top of the list. If an item is predicted by multiple rules, we only use the rule with the highest value of the measure function. The rule-quality measure proposed in [130] is the confidence. A similar, yet more detailed algorithm, which additionally uses support of the rule and the left-hand side cardinality to break ties, is proposed in [103].

The appropriateness of the confidence measure is disputed in [14], where many other measures, including the lift, are compared by performance in classification tasks. Furthermore, in [14], it is also mentioned that there are other possibilities than the best-rule method. Weighted voting method as proposed in [91] is discussed as one of the options. We further generalize such weighted voting for the purposes of recommendation as shown in Alg. 2.

We call the approach outlined in Alg. 2 the weighted-rules method.

6.1.5 Pareto optimality

Definition 6.1.2 Let $X$ be a set and $f_1, \ldots, f_n$ be real functions $f_i : X \to \mathbb{R}$, values of which we want to maximize (also called the objective functions). We say that $x \in X$ Pareto-dominates $\hat{x} \in X$ iff the following two conditions are satisfied:

1. $\forall i \in \{1, \ldots, n\} : f_i(x) \geq f_i(\hat{x}),$
2. $\exists i \in \{1, \ldots, n\} : f_i(x) > f_i(\hat{x}).$

We also say that $x^* \in X$ is Pareto-optimal state iff it is not Pareto-dominated by any $x \in X$ different from $x^*$. We call Pareto-optimal front the set $X^*$ of all Pareto-optimal states.

In our case of Recall-Coverage multi-objective evaluation, $X$ is the set of available models (by example, a set of $k$-NN models of different $k$s), and there are two objective
functions present: \( f_{\text{rec}} \) and \( f_{\text{cov}} \). These stay for the recall and the catalog coverage on a fixed dataset being examined. One model, \( m \), may be viewed as Pareto-dominating another model, \( \hat{m} \), if at least one of the two following conditions is met:

- \( f_{\text{rec}}(m) > f_{\text{rec}}(\hat{m}) \land f_{\text{cov}}(m) \geq f_{\text{cov}}(\hat{m}) \), i.e. \( m \) has higher recall than \( \hat{m} \), but still possesses at least as high catalog coverage as \( \hat{m} \).

- \( f_{\text{cov}}(m) > f_{\text{cov}}(\hat{m}) \land f_{\text{rec}}(m) \geq f_{\text{rec}}(\hat{m}) \), i.e. \( m \) has higher catalog coverage than \( \hat{m} \), but still possesses at least as high recall as \( \hat{m} \).

If we accept the set \( X \) of objective functions as the relevant measures of model quality, then \( m \) may be considered strictly better then \( \hat{m} \), and there is a good sense in using \( m \) instead of \( \hat{m} \).

It is also noteworthy that \( X = \{ f_{\text{rec}}, f_{\text{cov}} \} \) may be further extended by other measures, reflecting different qualities of a model. These could include, by example, the user coverage \([55]\), or even experimental measures such as the one proposed in \([58]\), aiming to estimate the serendipity.

Nevertheless, there is a dilemma of choosing the best model if there are multiple models in the Pareto-optimal front. The same dilemma as in the previous chapter discussing multi-objective clustering.

We propose the recommendation models to be evaluated in multi-objective plane, observing behavior of different measures and looking for Pareto-optimal models. In this article, the measures investigated are the recall and the catalog coverage.

Both the recall and the catalog coverage of recommendation models can be adjusted by various, model-dependent parameters, such the number of neighbors (k-NN), the number of latent features (SVD), the minimal support (AR), and by regularization terms. In the section below, as well as in our experiments, we show that it is also the biasing towards the long-tail items, which significantly affects these measures.

### 6.1.6 Biasing k-NN and Association Rules

We propose application of the model-biasing method presented in \([141]\) to be extended to other algorithms than those of matrix decomposition.

For the user-based \( k \)NN algorithm, putting together Eqs. (6.4), (6.5), and (6.6), we propose the **Popularity-Stratified, Non-normalized Cosine Neighborhood k-NN Algorithm**, which we will, for simplicity, keep referring to as the User-kNN throughout the rest of this chapter. For a given user-item pair \((u, i)\), the algorithm computes the rank as

\[
\text{rank}(u, i) = \frac{\sum_{\hat{u} \in N^k(u)} \text{sim}(u, \hat{u}) \cdot (r_{\hat{u}, i} - \bar{r}_u)}{\left( \sum_{u \in \mathcal{U}} (r_{\hat{u}, i} - \bar{r}_u) \right)^\beta}, \tag{6.7}
\]

where \( N^k(u) \) in the set of the \( k \) nearest neighbors, \( \mathcal{U} \) is the set of all the users in the database, and \( \beta \) is the long-tail biasing parameter as proposed in \([141]\).
In the context of association rules, it is particularly noteworthy that the concept of \( \beta \in [0.0, 1.0] \) parameter can be utilized to bias the confidence measure to get a very special meaning: in provides us a smooth transition from the confidence to the lift. Considering the rule-quality formula,

\[
\text{lift}(X \Rightarrow Y) = \frac{\text{conf}(X \Rightarrow Y)}{\text{supp}(Y)^\beta},
\]

we obtain the confidence for \( \beta = 0.0 \), and the lift for \( \beta = 1.0 \).

For the confidence measure, the association rules model tends to focus on the most popular items. This is somehow similar to the standard, unbiased \( k \)-NN algorithm. Consider, for a moment, a popular item \( i^* \) liked by most of the users in the database. Such an item would have high rating over many users in many neighborhoods, and hence would be very often recommended. Similarly, such a popular item would be implied by many association rules with high confidence, and hence would often be recommended by the AR algorithm as well. Moreover, consider a long-tail item \( i^- \). Such an item would be present only sparsely in most of the neighborhoods in the \( k \)-NN algorithm, and would be implied with a low confidence by association rules. Now, we could start moving \( \beta \) away from 0.0 towards 1.0. In the neighborhoods where present, the rank of \( i^- \) would start increasing. Similarly, the association rules having \( i^- \) on right-hand side would start receiving higher weights. At some point, for large enough \( \beta \), \( i^- \) would start receiving higher ranks than \( i^* \) across the \( k \)-NN neighborhoods, and the rules implying \( i^- \) would start overweighting those implying \( i^* \). This shows that despite being very conceptually different, both the models may be biased by \( \beta \) towards the long tail in the very same way.

### 6.1.7 Testing Datasets

To demonstrate our novel evaluation approach, we selected two publicly available datasets for the experiments:

- **MovieLens 1M (10M)** – 6040(71567) users, 3706(10681) items, 1M (10M) of explicit ratings given as the number of stars on the 1–5 scale [113].

- **HetRec 2011 Last.FM** – 1892 users, 17632 items, 92834 implicit ratings given as the number of plays [33].

In case of the MovieLens 1M dataset, we transformed all the ratings to lie within the \([-1.0, 1.0]\) interval, where \( r_{u,i} = -1.0 \) and \( r_{u,i} = 1.0 \) stay for the lowest and the highest possible rating, respectively, and \( r_{u,i} = 0 \) means neutral. This brings several advantages, namely a more faithful computation of the cosine similarity.

For the HetRec 2011 Last.FM dataset, we considered all the ratings implicitly relevant, as the dataset seemed to have been already preprocessed to only contain user-artist relations of high number of plays. For the purposes of the \( k \)-NN algorithm, we performed user-specific normalization into the \([0.0, 1.0]\) interval, inferring the rating proportionally
in number of plays, assigning the artist with maximal number of plays by a given user the value of 1.0. This has been done because of the cosine similarity being involved. For association rules, all the ratings were just assumed positive, without the need to transform into numerical scale.

To show that the results are robust we have employed number of other datasets (some of them are available online, some upon request).

- **LastFM 360k** – 359349 users, 160167 items, 17M of *implicit* ratings
- **Book crossing** – 105283 users, 340556 items, 716109 of *implicit* and 433671 of *explicit* ratings
- **Goout.cz** – 724768 users, 46901 items, 4M of *implicit* ratings
- **Nangu.TV** – 9004 users, 17558 items, 136494 of *implicit* ratings
- **Libimseti.cz** – 135359 users, 168791 items, 17M of *explicit* ratings
- **Icflix** – 13900 users, 11414 items, 232047 of *explicit* ratings

### 6.1.8 Results

As shown in Fig. 6.1, for the MovieLens 10M dataset, without the $\beta$ regularizer, the highest recall is obtained for the 37-NN model. Higher number of neighbors deteriorates both the recall and the coverage. Models based on thousands of neighbors are similar to recommendation of the most popular items (note that the recall of bestsellers is often quite high). In contrast, lowering the number of neighbors has negative impact on the recall due to overfitting, but the catalog coverage increases as the recommendations become more
specific for small, local groups of users. The effect of $\beta$ depends on the plasticity of a model. For models with higher plasticity (low number of neighbors), $\beta$ significantly reduces the recall and increases the coverage. The recall is maximized for the 66-NN model with $\beta = 0.1$.

The Figure 6.2 shows that similar behavior can be observed when k-NN is applied to other datasets.

Association rules (AR) based recommendation is conceptually different method. The sensitivity of the AR models to the minimal support ($s_{min}$) parameter resembles the sensitivity of the $k$-NN to number of neighbors. High value of a parameter implies biased models (weak learners), whereas lower values increase the plasticity of models, resulting in higher coverage. The term $\beta$ positively influences both the recall and the Coverage of models of weak learners when slightly above zero. For rules of high granularity (overfitted models) biasing has negative impact on Recall as depicted in Fig. 6.3.

### 6.1.9 Online Evaluation

To further test the Recall-Coverage evaluation method, we executed it offline and compared its results with online evaluation on a small online gaming discount portal. The portal aggregates the current prices of video games, covering about 20 online shops from all around the world. It has about 10K daily users, from who about 1K are logged in and uniquely identified. For each user, there are several interaction types available: the waitlist, providing a set of explicitly positive ratings, the collection of purchases, providing implicit positive ratings, and the ignore list, providing explicit negative ratings. We put all these together and normalized them into the $[-1, 1]$ rating scale, resulting in a dataset for offline evaluation. The dataset consisted of 26K users, 2K items (only the games with up-to-date discounts were allowed), and 3.2M ratings.

On the dataset, we performed simple offline experiment, using the $k$-NN algorithm and measuring the Recall and the Catalog Coverage for different values of $k$. The results of this experiment are shown in Figure 6.4. As seen in the figure, we obtained a standard curve presented earlier for the $k$-NN algorithm.

As seen on Fig. 6.4, there are multiple Pareto-optimal states, one of which needs to be chosen in order to meet the business goals. In many cases, there only a single business goal: to offer a model with the highest user response, i.e. the number of clicks. To explore the relation between the recall, the catalog coverage, and the number of clicks, we designed a the following simple A/B testing experiment. We divided the users into 5, equally sized groups, based on the results of the offline experiment:

- $k = 5$ – high catalog coverage, yet a quite low recall,
- $k = 25$ – balanced compromise between the catalog coverage and the recall,
- $k = 100$ – strict optimization for recall as the most widespread up-to-date approach,
Figure 6.2: The effect of K-NN regularization on benchmarking data sets.
Figure 6.3: Results for the Association Rules Algorithm on the 1M Movielens Dataset

Figure 6.4: Recall-Coverage Tradeoff for Offline Evaluation on the Isthereanydeal.com

- $k = 500$ – Pareto-dominated model biased highly towards the most popular items,
- $old$ – the old method used by the portal before the deployment of the recommendation; highly sophisticated, based on a series of prunes, removing DLC (downloadable content such as story expansions, new maps for multi-player games, etc.), taking prices into account and preferring the newest discounts.

We ran the experiment for a period of 7 days. The results of the experiment are shown in Table 6.1.

We ran the experiment for a period of 7 days. The results of the experiment were merged to Fig. 6.4. The size of circles is proportional to number of conversions.

Although the results might be considerably influenced by noise coming mainly from external sources, such as the newly announced discounts, it seems quite convincing that
Table 6.1: $k$ vs. Number of Clicks in isthereanydeal.com On-line Experiment

<table>
<thead>
<tr>
<th>Method</th>
<th># Clicks</th>
<th>Improvement over old method</th>
</tr>
</thead>
<tbody>
<tr>
<td>old</td>
<td>64</td>
<td>0%</td>
</tr>
<tr>
<td>5-NN</td>
<td>93</td>
<td>45.31%</td>
</tr>
<tr>
<td>25-NN</td>
<td>263</td>
<td>310.94%</td>
</tr>
<tr>
<td>100-NN</td>
<td>155</td>
<td>142.19%</td>
</tr>
<tr>
<td>500-NN</td>
<td>154</td>
<td>140.63%</td>
</tr>
</tbody>
</table>

the best performance was obtained for the 25NN model, possessing a balanced trade-off between the recall and the catalog coverage. Interestingly, it seems there is no difference between the 100NN and 500NN models, despite the latter is Pareto-dominated, oriented only towards the most popular items, and essentially non-personalized. The 5NN model seems to be too plastic and susceptible to noise.

Figure 6.5: Recall-Coverage Tradeoff for Offline Evaluation (confidential dataset)

Indeed, extensive conclusions cannot be drawn on a single experiment, but we observed the same behavior in all online A/B evaluations we have performed so far. We publish results of recommendation models in a very different domain (vendor wish to stay anonymous), with number of users 880407, number of items 113659 and 4M implicit ratings in the database. It is apparent (Fig. 6.5) that maximizing the Recall is not a good strategy leading to highest conversion rates. Although the database is very sparse and many users
Figure 6.6: The value of Recall-Coverage harmonic mean for KNN (left) and AR (right) algorithms on the LastFM 2k dataset. The maximum can be observed for 50-NN with $\beta = 0.3$ and for $s_{\text{min}} = 0.0001$ with $\beta = 0.5$ in case of AR.

have just a single interaction (low coverage of 1NN is due to inability to find useful item in the 1-neighborhood), results are consistent with the previous online experiment.

Results of online experiments suggest that the most useful recommendation models are not those with highest Recall, but those from the center of the Pareto front. To enable reproducibility of results and further exploration of this interesting field, we encourage researchers to run models in our platform. In a controlled environment models that pass offline tests can also be evaluated online.

To prevent annoying users during A/B testing with irrelevant recommendations, we suggest optimizing parameters of recommendation algorithms offline. The objective function we propose is to maximize the weighted harmonic mean of recall and catalog coverage measures. It is defined as follows:

$$RC_{\text{whm}} = \frac{\text{recall} \cdot \text{catalog} - \text{coverage}}{\alpha \cdot \text{recall} + (1 - \alpha) \cdot \text{catalog} - \text{coverage}}$$  \hspace{1cm} (6.9)

As seen in Figure 6.6 for $\alpha = 0.5$, this apparently convex optimization problem can be handled for example by the Powell’s method \[124\]. When ensembles of recommender
models and other parameters are taken into account, the optimization problem becomes more difficult.

We recommend optimizing models offline for several values of $\alpha$ in the range $(0.3, 0.9)$ and then use online A/B tests to confirm the best tradeoff for given database. The other option is to use the multi-objective optimization and select solutions from the center of the Pareto front, where recall and coverage are in balance.

### 6.2 Algorithm Ensembling Strategies

As in the other domains, recommendation algorithms can be combined in an ensemble. Winning solutions of the Netflix prize used ensembling intensively \[\text{[150]}\]. Again, there are several approaches how recommender systems can be blended. Straightforward ensembling strategy is to mix outputs of several diverse recommenders in one recommendation (usually, the task is to recommend at least 5 items). We use this simple strategy below. Mixing kNN with AR might be good idea, because these algorithms work very differently.

Interesting observation can be made when kNN and AR models are compared in the Figure 6.7. For the LastFM Dataset, rule-based models dominate neighborhood-based models, whereas for the Movielens 1M Dataset it is just the opposite. It can be explained by low number of ratings in the LastFM 2k Dataset that prevent neighborhood-based models to work efficiently. The Figure 6.7 also shows that ensembling diverse recommendation methods is very promising direction. The ensemble Pareto-dominated base models in the region where Recall and Coverage are in tradeoff.

Currently we experiment with including matrix factorization methods in the ensemble \[\text{[98]}\]. Also, we study methods that enforce diversity in the recommendation ensemble.

### 6.3 Meta-Learning Templates

Number of algorithms and options how to combine and parametrize them is again tremendous. In the Figure 6.8 one simple example of meta-learning template is presented.

This template is specific and contains all algorithms and parameters required to construct a recommendation algorithm. Parameters of the Mix ensemble are percentages (probabilities) in which particular ensemble members will be generated. The final recommendation here will be a mix of nearest neighbor, association rule, matrix factorization based recommendation together with recommendations based on the item attributes similarity and reminder of last interactions.

The template can be also generalized. One can then measure whether it is more beneficial to build simple models or hierarchical ensembles of recommendation algorithms.
Figure 6.7: Comparison of Association Rules, kNN and their ensembles on LastFM 2k and Movielens 1M datasets.
6.4 Template Optimization Strategies

Further research is needed to investigate how offline criteria can be good estimators of the online performance. Problem of offline evaluation is that one has to transfer all constrains and business rules that are influencing the final recommendation. Below, we focus to optimize the online performance of the recommender system.

We elaborated optimization of simple templates and their parameters in the domain of online recommendation.

A variant of A/B testing called Multivariable testing is employed to evaluate multiple configurations of the recommender system at the same time. Each of the configurations consists of multiple parameters allowing us to find the optimal combination of their values, which would be otherwise complicated because some of the parameters influence each other.

All users coming to the site are randomly distributed into groups of approximately even size and each of that groups is assigned to one of the tested recommender system configurations. For a relevant evaluation, a sample of certain minimal size needs to be gathered to reduce the noise. In our case, that means generating at least a couple thousands of recommendations by each of the tested variants before evaluating them by a chosen metric.

In most cases a conversion rate is the performance metric of the first choice. It represents a portion of recommendations that the user not only interacted with, but also consumed the recommended item, meaning that he purchased, watched or read it, with respect to what the current domain offers. This metric is a popular way of evaluating recommender systems because it is directly tied to increasing the number of satisfied users and sales.

In case of web sites with low traffic (number of interactions) making reliable evaluations in a reasonable time is not feasible. Due to a small amount of visitors, an action rate is used as an evaluation metric instead. It captures the portion of recommendations that the user clicked on. Although interacting with the recommendation undoubtedly means that it caught the user’s attention, it does not have to mean that he found what he was looking for and it is therefore not clear whether will such recommendation lead to an increase of
revenues/customer satisfaction or not. On the other hand, we observed some correlation between actions and conversions during our experiments, meaning that while not perfect, the action rate appears to be a usable substitute for the conversion rate.

In order to adaptively optimize the recommender systems online through the use of multivariate A/B testing (see Figure 6.9), we propose a Surrogate Assisted Online Optimization Algorithm (SAOOA) that has been inspired by Evolutionary Strategies [42].

It is designed to explore the complex multidimensional space of algorithm space while overcoming the real world noise and frequent changes of the environment. To make that possible, a Gaussian mixture model, representing the probability that a certain configuration will be selected for evaluation, is being adapted each generation according to the current population of individuals. Individual, a real valued vector of parameter values, represents a single configuration of the recommender system.

An outline can be seen in Algorithm 3, where $N$ is the number of individuals added to the population in one generation. Because we are evaluating every individual through A/B testing, a larger $N$ requires more users for a proper fitness measurement, therefore it is usually small ($<10$).

$M$ is the number of last generations kept in population, more generations means more data, which translates into a more detailed mixture model, but it also means slower reaction to changes. Therefore, we set the $M$ according to the speed of change in the environment.
We have experimentally found that $M = 3$ works well under all sorts of different conditions.

A fitness of a template refers to the value received by evaluating it, and a gaussian refers to a single Gaussian function of the Gaussian mixture model.

The optimization algorithm starts by sampling an initial population from a Gauss distribution with a mean placed where we estimate the optimal configuration and a standard deviation chosen according to how sure are we about that estimate. If we cannot estimate the optimal configuration in any way, we simply place the function at the center of the configuration space and set the deviation large enough to enable generation of any configuration.

After initial population is generated, the actual optimization starts. Firstly, new individuals are evaluated, then the standard deviation of the Gaussian functions, making up the mixture model, is updated. After that, the whole mixture model is created as a sum of gaussians representing each of the individuals. Some of the gaussians are penalized, which is a process explained below, and all of them are multiplied by a weight exponentially correlating with the number of generations they persist in the population. Finally, new individuals are sampled from the mixture model and the whole cycle starts again. Because of the ever changing environment, it runs continuously as real-time algorithm.

The GMM, representing the probability of new individuals being generated at certain coordinates, is not technically a surrogate model, because we evaluate the selected individuals online each generation. On the other hand, its aim is to simplify the noisy real world environment and to provide a way to generate new individuals easily.

Each parameter is being optimized inside a defined interval of values. All used gaussians have diagonal covariance matrices with standard deviation in each dimension proportionate to the size of interval corresponding to the parameter representing that dimension.

When sampling new individuals, they cannot be generated closer than $\sigma/2$ from each other (in every dimension) because that would have two negative consequences. Firstly, it would reduce the exploration of the available space which is undesirable due to the expensive evaluation of each individual and secondly, the gaussians representing individuals very close to each other, or even on top of each other, would merge and create an impression of an individual with much higher fitness than it actually has. This is different from a scenario when gaussians with centers $\sigma/2$, or more, away from each other merge and create one very high "hill". That just means that we are very confident about this area having above average fitness, and therefore generate individuals in it with a high probability.

One of the most important aspects of building the mixture model is penalizing gaussians representing bellow-average individuals, which amplifies the differences between the individuals that would otherwise be hard to distinguish. It also prevents a case where multiple bellow-average individuals are generated close to each other, their gaussians merge and create an impression of an above-average area. Figure 6.10 shows how penalization solves the problem and transforms the model into a much more relevant estimate of promising regions.

Penalization is done by adding $c \ast (f_{K_i} - f_K)$ to aforementioned gaussians. $c$ is an experimentally found constant equal to $13$, if it was smaller, all the gaussians would merge
Algorithm 3: SAOOA

**input**: Number of individuals in one generation $N \in \mathbb{N}$,
Number of generations kept in population $M \in \mathbb{N}$,
Generation number $K \in \mathbb{N}$

**output**: $N$ new individuals

/* Initialization */
```
new_individuals ← sample N individuals from an initial gaussian
K ← 1
```

/* Endless loop of optimization */
```
while true do
    population ← population ∪ new_individuals
    old_individuals ← individuals older than $M$ generations
    population ← population \ old_individuals
    /* Evaluate the individuals */
    foreach individual ∈ new_individuals do
        individual.fitness ← evaluate(individual)
        individual.K ← $K$
        average_fitness[$K$] ← average_fitness(new_individuals)
    /* Update the standard deviation */
    γ ← recalculate_gamma(γ)
    σ ← $\sigma_d \ast γ$
    /* Create the Gaussian mixture model */
    for $i ← 1$ to length(population) do
        amplitude ← population[$i$].fitness
        mean ← population[$i$].genome
        gaussian ← Gauss(amplitude,mean,σ)
        /* Penalize the sub-average individuals */
        if individual[$i$].fitness < average_fitness(population[$i$].$K$) then
            gaussian ← gaussian + $c \ast (population[i].fitness - average_fitness[population[i].K])$
        /* Weight the gaussians */
        weight ← $(\frac{1}{2})^{K - population[i].K}$
        gaussians[$i$] ← gaussian * weight
        GMM ← $\sum_{i=1}^{\text{length(population)}}$ gaussians[$i$]
    /* Sample new individuals from the GMM */
    new_individuals ← sample(GMM)
    K ← $K + 1$
```
Figure 6.10: Robust surrogate fitness is important in noisy environment. The landscape can be modeled by penalization of bellow average individuals which create above average regions when in clusters are not penalized.
into a one large "hill" because the sub-average individuals would not be penalized enough.
On the other hand, if it was bigger, the penalization would be so severe that it would turn
the model into a one large "valley". Therefore the value 13 is a sought for balance between
these two extremes. So far, this number was acceptable for all databases we experimented
width.

\[ f_{K_i} \] is a fitness of an \( i \)-th individual from generation \( K \), and \( f_K \) is an average fitness of
individuals from generation \( K \).

Individuals are always compared only within one generation because each generation
can be evaluated for a quite long period of time which can result in different fitness values
than in previous generations thanks to the constant changes in the environment.

Another important feature of the proposed algorithm is updating the standard deviation
before creating the mixture model.

We generally do not want large changes of the individuals, because then we risk gener-
ating significantly suboptimal templates, which is undesirable due to the nature of online
A/B Testing. Therefore a default standard deviation is quite small, specifically a \( \frac{1}{100} \) of
parameter’s interval size.

In case we are not sure about the location of an initial area used to generate the
first individuals, we are forced to make it significantly larger and then risk that the first
population will be far away from optimum, making it very hard to reach with only small
changes made possible by the default standard deviation.

Solution to this problem is a standard deviation multiplier \( \gamma \) allowing us to recalculate
the standard deviation of all gaussians each generation as:

\[ \sigma = \sigma_d \times \gamma \]  \hspace{1cm} (6.10)

\( \sigma_d \) is a default standard deviation set at the start of the algorithm. \( \gamma \) is initially set to a
number large enough to allow the exploration of the whole space during the subsequent
process of slowly reducing the multiplier to its base level.

Therefore a method inspired by simulated annealing [61] is employed, reducing the
exploration rate of the algorithm with an increasing number of passed generations.

Because of the dynamic environment, we cannot converge to a specific point and remain
there, that is why we never decrease the multiplier past a certain base threshold, meaning
that exploration always continues.

If we detect a significant change in the environment exceeding typical noise, we increase
the multiplier to ensure that we do not get stuck in a suddenly sub-average area. In our
case, we consider a change significant if the difference is more than 25%. Changes are
detected by keeping the best individual from the previous generation and then comparing
the two measurements of its fitness value. This approach is called elitist selection and we
will call this particular individual an elitist.

Unfortunately, due to the large noise and complex environment, recognizing a true
global change has proven to be difficult. However the algorithm as described above is robust
enough to run in production. It proved to increase the performance of the recommender
system by 10% in average, when compared to static recommendation templates.
Above, we showed how to model and optimize continuous parameters of the recommender system. Discrete parameters and topology of templates can be optimized in a similar way by maintaining a surrogate model of their fitness. As future work, we plan to investigate how we can efficiently translate from generalized templates to specific templates and back in case of significant changes of the fitness landscape.

We also need to proceed with meta-learning experiments to be able to seed templates based on fitness landscape analysis.
CHAPTER 6. ALGORITHM SELECTION IN RECOMMENDER SYSTEMS
Chapter 7

Algorithm Selection in Other Domains

We automate the algorithms selection process in other domains as well. In this chapter we give you a brief overview of the most interesting results obtained.

7.1 Algorithm selection in data preprocessing

The dissertation thesis [156] deals with the automation of preprocessing methods selection. Goal is to find a sequences of preprocessing algorithms to be applied to each attribute. The problem encoding is in the Figure 7.1.

Performance criterion is the same as in case of supervised modeling - generalization performance of a simple model trained on preprocessed attributes. Ensembles in this case are subsequences of local and global methods. Example of local methods are missing value imputation, class balancing, local transformation and normalization. Global methods work on all attributes such as PCA projection etc. Such methods can be encoded into meta-learning template.

Several methods has been examined and their performance measured. The best performing heuristics was the genetic programming.

As an example of template optimization we present visualization of the search space when two parameters of the local transformation function (power by N). The Figure 7.2 shows the progress of the best-so-far solution in the Steepest descent search algorithm. The dots represent best-so-far solution and lines represent its development. In contrast to the other methods such as the Simulated annealing the Steepest descent shows lower number of iterations of the best-so-far solution, this is caused by fact, that the best-so-far solutions are moved only if the new states with higher fitness are found.

The conclusion of the thesis is that automated preprocessing (enabling better features) can increase the generalization accuracy of simple predictive models by five to ten percent.
CHAPTER 7. ALGORITHM SELECTION IN OTHER DOMAINS

Figure 7.1: Illustration of dataset with three attributes and corresponding local and global subsequences of preprocessing methods.

Figure 7.2: Visualisation of the parameter values search space for the N-Power Calculator preprocessing method and the Non Linear dataset. Blue marks shows positions of the best-so-far solution of the Steepest descent search algorithm.
7.2 Automated workflow optimization

In [64], templates are introduced to select/optimize full knowledge flow. Planning and optimization of workflows is also elaborated in [118], where meta model and AI planner are combined.

Another meta-learning approach [84] focuses on optimizing significant parts of data mining workflows, including the model selection that is close to our approach.

At first we have to modify the structure of meta-learning templates. Trees are not sufficient to express requirements of data mining workflows. Therefore we are using Directed Acyclic Graphs (DAGs) for this purpose.

7.2.1 Structural aspects of KD processes

The composition of actions can be expressed in a form of labeled oriented graphs called workflows. These workflows became a standard way of representing interconnected actions that need to be executed in order to obtain useful knowledge [166]. There are many different names for the same thing. In RapidMiner software, for example, these workflows are called Processes, which is the term that we will use further in the text. A sample RapidMiner process is shown in Figure 7.3, along with the oriented graph analogy.

When talking about processes, the term Process should be defined more properly. A general definition of a process is a question rather philosophical. In out context, a pro-
cess is understood as a DAG of labeled nodes and edges. The nodes may be thought as actions (i.e. transformations), whilst the edges serve as transition channels between the actions. An action represents a projection $A: \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ stand for input and output domain of the action, respectively. There are special nodes called inputs with $\mathcal{X} = \emptyset$, and outputs with $\mathcal{Y} = \emptyset$.

In order to optimize Processes, we will use a grammatical evolution [129]. Let us show selected rules bellow.

### 7.2.1.1 Preprocessing Grammar

The preprocessing phase consists of a chain of configured preprocessing actions. This needs to be reflected into the grammar. At the top level of process construction grammar, we have proposed the type `Preprocessing`.

We propose the type `Preprocessing` to be polymorphous. Firstly, the preprocessing sequence must be allowed to be empty. For that reason, we declare a terminal node called `PreprocessingTerminal`. This type of node can be placed at any location in the tree where an instance of `Preprocessing` is required. If the `PreprocessingTerminal` node is placed at the top-level of the hierarchy, i.e. as the first child of `ValidationProcess` node, there is no preprocessing at all.

Furthermore, two types of configurable preprocessing actions will be available: the PCA projection and the Select Attributes. To reflect this, we declare two more realizations of `Preprocessing` type: the PCA and the `SelectAttributes`. Wherever an instance of `Preprocessing` is required, these two node types may be inserted into the tree as well.

Indeed, both the PCA and `SelectAttributes` require specific attributes in order to be configured. As the PCA preprocessing action needs the number of principal components to be specified, the type `PCA` in the GP tree will have a descendant of type `NaturalNumber`. Similarly, the `SelectAttributes` type will have one descendant of type `SetOfPositiveIntegers` to specify the indices of attributes to be selected, and another descendant of type `BooleanConstant` to determine whether the selection should be inverted.

Finally, there must be chaining mechanism provided for the purpose of preprocessing methods composition. Hence we add another descendant of type `Preprocessing` to both the PCA and `SelectAttributes` types. At this point, the chaining mechanism is fully established as the child nodes may again of one the types `PCA`, `SelectAttributes`, or `ProcessingTerminal` that finishes the sequence. See the Fig. 7.4 that shows the preprocessing grammar.

### 7.2.1.2 Learner Grammar

To incorporate models, we propose the Learner node to be expandable into one of KNN, NaiveBayes, DecisionTree, MajorityVote, and Bagging node types. Similarly to preprocessing action-defining nodes, each of these node types comes with an ordered set of descendant types. These are shown Figure 7.5, and 7.6.
7.2. AUTOMATED WORKFLOW OPTIMIZATION

Preprocessing

\[ \text{Preprocessing} ::= \text{PCA}, \text{Preprocessing} \]

Select Attributes

\[ \text{Select Attributes} ::= \text{Select Attributes}(\text{SetOfPositiveIntegers}, \text{BooleanConstant}, \text{Preprocessing}); \]

PCA Projection

\[ \text{PCA} ::= \text{PCA}(\text{NaturalNumber}), \text{Preprocessing} \]

Preprocessing Terminal

Insert Principal Component Analysis (PCA) Dimensionality Reduction

Set number of components

Insert Feature Selection Dimensionality Reduction

Invert the selection?

Set indices of attributes to be selected

Do not insert any node

Preprocessing sequence so far (potentially empty)

Preprocessing sequence so far (potentially empty)

Insert next preprocessing here

Select Attributes

Set of Positive Integers

Boolean constant

Preprocessing

Preprocessing sequence so far (potentially empty)

Insert next preprocessing here

Preprocessing

Natural Number

Preprocessing

Figure 7.4: Preprocessing Grammar and its Semantics
CHAPTER 7. ALGORITHM SELECTION IN OTHER DOMAINS

\[ \text{Learner} \ ::= \text{<KNN>} \]
| \text{<NaiveBayes>} |
| \text{<DecisionTree>} |
| \text{<MajorityVote>} |
| \text{<Bagging>} |

K-NN Naive Bayes Decision Tree Majority Vote

\[ \text{Learning subprocess} \]
\[ \text{Training data} \]
\[ \text{K-NN Input Model Output} \]

\[ \text{Natural Number Boolean constant Distance Measure} \]

\[ \text{Use k-nearest neighbor learner} \]
\[ \text{Distance measure used?} \]
\[ \text{Weighted vote?} \]
\[ \text{Number of neighbors?} \]

\[ \text{Learning subprocess} \]
\[ \text{Use Naive Bayes Learner} \]
\[ \text{Laplace Correlation?} \]

\[ \text{Learning subprocess} \]
\[ \text{Use Decision Tree Learner} \]
\[ \text{Apply parameters} \]

\[ \text{Semantics} \]
\[ \text{Figure 7.5: Learners Grammar and Its Semantics.} \]
7.2. AUTOMATED WORKFLOW OPTIMIZATION

**Syntax**

![Syntax Diagram]

\[
<\text{Bagging}> ::= \text{Bagging}(\langle\text{Learner}\rangle)
\]

**Semantics**

![Semantics Diagram]

**Figure 7.6**: Bagging Meta-Learner Grammar and Its Semantics.

![Sample Tree Diagram]

**Figure 7.7**: A sample tree evolved on Ecoli dataset.
The **KNN** node type has three descendants. The first descendant is of type **NaturalNumber** and defines the number $k$ of neighbors to be examined. The second descendant is of **BooleanConstant** type and defines whether the vote of resulting class on the top of the $k$ neighbors collected should be weighted. Finally, the third descendant is of type **DistanceMeasure**, which is a constant specifying the measure to be used for distance evaluation.

The **NaiveBayes** node type has a single **Boolean** descendant determining whether Laplace correlation shall be used in the Naive Bayes learner.

The **DecisionTree** node type comes with several descendents (parameters). These types are shown in Figure 7.5 and are analogical to the parameter nodes of previously mentioned learning and preprocessing actions. Indeed, there must be special grammar defined for each of these node types.

**Meta-Learners Grammar** To demonstrate ensembling capabilities we define two simple meta-learners. These are represented by **MajorityVote** and **Bagging** node types. We propose these node types to allow arbitrary nesting. Nesting of meta-learning methods can significantly improve the classification accuracy when compared with base learners provided alone as we discuss in the predictive modeling chapter.

**Bagging Meta-Learner** It was mentioned that Bagging meta-learner requires an inner learner to be specified. Hence the **Bagging** node type comes with a single descendant of type **Learner**. This is shown of Fig. 7.6.

When we run the evolution for 30 generation, we obtain convergence for most of the data sets. Many interesting solutions have been found. As an example, we present a template which has been evolved for the Ecoli dataset in Figure 7.7.

Even though the set of preprocessing and learning actions was very limited, the algorithm managed to find a satisfactory solution for most of the datasets. In most cases, a good solution was in the first generation within the randomly generated individuals. Indeed, this is because the limited set of learners.

### 7.3 Other application domains

For our results on TASA deployment in the field of combinatorial optimization, please refer to [99]. In the continuous optimization domain some result can be found in [23]. In the field of relational data mining see the Predictor Factory and Prague Relational Learning Repository [112]. Recently, we started to elaborate with the Neural Turing Machine [146] composed of several building blocks including recurrent neural networks. Optimizing templates in the reinforcement learning domain is a big challenge because the evaluation of template performance is computationally very demanding.
Chapter 8

Conclusion

We elaborated new approach to algorithm selection using meta-learning templates and proposed general process towards algorithm selection automation (TASA). This process starts with elementary performance definition stage and continues with steps of increasing complexity. The advantage is that the result can be obtained from any stage (e.g. by manual selection) and some stages can be skipped (e.g. meta-database). This approach is both universal enough to enable deployment in different domains and specific enough to supply guidelines similar to standardized processes such as CRISP-DM.

We have deployed the TASA process in the field of supervised modeling and demonstrated capabilities of automated template selection and execution in anytime learning and under a concept drift. Proposed evolutionary programming search significantly outperforms greedy heuristics in optimization of proposed meta-learning templates. Templates can accommodate hierarchical ensembles and they are translated from general to specific in the optimization process. This is unique concept that prevents premature convergence similarly to simulated annealing.

In data clustering domain, we performed extensive experiments examining how to measure performance of clusterings/templates. We show similarity of unsupervised clustering criteria and their correlation with supervised measures. In evolutionary optimization of clustering templates, we employed multi-objective search with good results on benchmarking data. Employing a meta-database to recommend set of unsupervised criteria and initial templates and full automation of algorithm selection is our future work.

For recommender systems, we show that traditional performance measures are not very good in estimation of the online performance. We introduced multi-objective offline evaluation process based on recall and coverage criteria that correlates with the online performance of the system. Ensemble methods are examined and we show that ensemble can Pareto dominate solutions obtained by single algorithms. Recommendation templates were introduced and we proposed robust surrogate assisted online optimization algorithm that optimize their parameters. The algorithm is deployed in production and it brings substantial improvement in performance.

We tackle the problem of algorithm selection automation in other domains as well. As demonstrated in the last chapter, algorithm selection in data preparation and relational
data mining can improve performance of predictive models. Even more important is the ability to automate the algorithm maintenance. New attributes in a database can be automatically discovered by a relational algorithm and the performance of predictive modeling templates can be improved as well as coincidence.

We adjusted templates to represent data mining workflows and optimize it by a grammatical evolution. These preliminary results suggest that it would be better to build workflows from building blocks that are optimized independently. This is however hard task and our future work, together with optimization of reinforcement learning models.
Bibliography


[71] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. SIGMOD Rec., 29(2):1–12, May 2000.


