

P. J. Safarik University

Faculty of Science

WEB BASED DATA-MINING ASSISTANT

THESIS

Field of Study:

Computer Science

Institute:

Institute of Computer Science

Tutor:

RNDr. Tomáš Horváth, PhD.

Košice 2015

Bc. Štefan Bocko

Thanks

Thanks to my supervisor RNDr. Tomáš Horváth, PhD. for his valuable efforts in supervising this project.

Abstrakt

Cieľom práce je implementovať tzv. proof-of-concept webovskú aplikáciu na manažovanie a podporu data mining projektov. Hlavná požiadavka na systém je jeho jednoduchosť ako aj identifikovanie hlavných prekážok a slabých stránok takéhoto systému.

Abstract

The goal of the thesis is to implement a proof-of-concept pilot of a web-based system for managing and supporting data mining projects. The main requirement is that the system should be very simple and user-friendly. Also, main challenges and obstacles of such a system should be detected in the thesis.

Contents

Introduction	6
Requirements	6
Existing solutions	7
Thesis structure	8
 1 Data preprocessing	 9
1.1 Data cleaning	9
1.2 Data transformation	10
 2 Data-mining algorithms	 13
2.1 k-nearest neighbor	13
2.2 Logistic regression	15
2.3 Decision tree	17
2.3.1 TDIDT Algorithm	19
2.4 Support vector machines	20
2.4.1 Linearly separable data	21
2.4.2 Linearly inseparable data	24
 3 Model recommendation	 27
3.1 Definitions	27
3.2 Gridsearch	28
3.3 Meta-learning	30
3.3.1 Statistical characterization approach	32
3.3.2 Landmarking approach	33
3.4 Hybrid approach	34
3.5 Experiments	35
3.5.1 Initial setup	36
3.5.2 Performance	39

4 Design and implementation	48
4.1 The CRISP-DM methodology	48
4.2 Project design	50
4.3 Use case	53
4.4 Implementation	62
4.4.1 Web application	63
4.4.2 Database	66
4.4.3 Computational server	67
Conclusion	70
Resumé	72
Bibliography	74
Attachments	77

Introduction

Nowadays there are many situations that require knowledge of data analysis. Examples include students who have to analyze data measured during their experiments. Mostly they are not data mining experts nor have experience in data analysis. They do not want to study numerous algorithms and techniques for acquiring knowledge from datasets.

Various software exist for solving data mining tasks, such as Weka, RapidMiner... The problem is, that all of them are designed for technical users, who have at least minimum knowledge of data mining algorithms. This assumption is not true in our case. We will assume, that our users do not have any experience with data analysis.

In this thesis we design and implement a proof-of-concept pilot of a web-based system for managing and supporting data mining projects.

Requirements

Our application should guide a user through data-mining process step by step in a way that is clear even for non technical user. We can not assume that the user has knowledge about data preprocessing or modelling. We will assume, that our user has a single file with his measurements and he wants to perform some analyzes on it.

The basic use case can look like this:

- User visits our web application from his browser
- User uploads file with data for analysis
- Our software will analyze given data to decide, which algorithm fits the best
- Our software will make an analysis of a given dataset
- The user-friendly report is generated as an output.

There are several problems, which we have to solve. First of all, there is a problem of finding suitable algorithm for analysis. This is because data mining algorithm, which performs great for a given dataset can perform poorly for another one. This is basically known as “no free lunch” theorem. Then if we choose an algorithm, there is another problem of finding its exposed hyperparameters for given dataset. We proposed a solution of this problem in Chapter 3.

Other problem is the process of gaining knowledge from a user about his data, taking into account, that the user does not understand data mining terminology. We have to take this into account, when displaying the results too.

Existing solutions

There are many data-mining tools available today. They can be divided into two major groups

- Desktop applications – Weka, RapidMiner ...
- Web applications – OpenML, BigML ...

Desktop applications are intended to be used by professionals. They are very complex and require deep data-mining knowledge. They are quite difficult to install and run for non-technical users.

Therefore, we will focus in this section on web applications like OpenML [20] and BigML [21]. These applications have similar concept like our prototype. They are intended to be easy to use directly from web browser without any additional installations.

BigML is online tool for managing data mining projects as well as our prototype or OpenML. However, it is intended for professionals and technical users who can program their analyzes on their own. It is suitable for those who need to perform some analysis, but do not want to install complex tools. This tool simplifies the analysis process, but also offers many additional business and enterprise tools.

OpenML builds on its community. Datasets are shared between the community and anyone can access them and perform various analyzes. Other users can view these results. Therefore, there is a big chance that the dataset will be examined by an expert user. This is helpful especially for beginners who do not know which method to use. On the other side, users dataset can stay unnoticed for weeks, because it does

not look interesting for other users. Our system lacks the expert interaction, but it tries to automate the whole process and replace the expert support. Advantages and disadvantages of our system are summarized in the SWOT analysis below.

S automation of the process accessible from web browser performance simple UI	W missing expert support
O include more models community support	T small user interest existing systems

Figure 1: SWOT analysis of our prototype design.

Thesis structure

In Chapters 1 and 2 we describe mathematical and algorithmic background of pre-processing methods and data-mining models used in our solution. Chapter 3 formally defines a problem of model and hyperparameter recommendation. We analyze current approaches to this problem and we also propose our custom solution, that best fits our requirements.

In the second part of this work, we have implemented the proposed prototype application. The implementation overview is given in Chapter 4. There we describe the proposed system in more details from user as well as architectural point of view.

Chapter 1

Data preprocessing

One of the most difficult and time consuming data-mining step is preprocessing. At the same time it is a step that is crucial to the success of the whole process. The main goal of this step is to create such a data representation that is suitable for chosen processing method. In our case this is even more difficult, because we do not have specific data mining method at this point. We will use several data-mining techniques instead of just one to increase the accuracy of result.

In this chapter, we describe several known techniques of data preprocessing which we used in our system. These techniques include data cleaning, data transformation and discretization.

1.1 Data cleaning

Real-world datasets are often incomplete. Some values are not set, they are called missing values. Often, there is a special symbol defined which represents them (e.g. "?" or "missing"). These values are not eligible for data-mining modelling, because these models either can not be applied on such dataset or they give poor results. Therefore several approaches were developed to handle this problem.

In most cases, missing values are replaced by adequate values. For example one can create a list of all values in an attribute and replace its missing data with most common value. This can be further improved by creating a separate list for every class and then replacing with the most common value of class in which the instance belongs to. This method is most suitable for attributes with small number of distinct values such a boolean attributes with only two possible values.

Specifically for numeric attributes a mean value can be computed for an attribute

and missing data are replaced by this value. Also a simple constant value can be used.

There is no generally suitable methods for handling the missing values. Effectiveness of these techniques are highly dependant on attribute characteristics and type of data.

1.2 Data transformation

Data in datasets are mostly represented in a form that is more suitable for reading by humans. However, for machine-learning algorithms it is more convenient to represent data in different form (e.g. numeric attributes in 0–1 range). Every data-mining algorithm has specific needs about data representation. Therefore, there is no universal algorithm for data transformation. The known methods for transforming data include binarization, normalization, discretization . . .

Normalization and discretization are techniques which apply to numeric attributes. Numeric attribute is an attribute whose value consists of numbers (e.g. real, natural or integer). Discretization is the process of transferring continuous attributes (numeric) into their discrete counterparts (nominal attributes). We considered two discretization algorithms which differ in the way they divide the attribute data into discrete groups.

1. **Equidistant** – this method divides the range into n intervals of equal size. If l and h are the lowest and highest attribute values, the width of intervals will be $w = \frac{(h-l)}{n}$. It is the most straightforward method, but outliers may distort results. Also skewed data is not handled well.
2. **Equipotent** – also known as equal frequency partitioning, divides the range into n intervals, each containing approximately the same number of samples. The advantage of this technique is that it scales data well.

Normalization is a technique of adjusting the values with various ranges to more standard range like $[0, 1]$ or $[-1, 1]$. This data representation is more suitable for data-mining models, because attributes have equal weight when they are normalized. For example, one attribute measured in millimeters have much bigger values than similar attribute measured in meters and this can cause inequality during analysis. Therefore, normalization is very important preprocessing method [1]. We considered two different methods.

1. **zero-one** – this type normalizes the values to $[0, 1]$ range using the equation below

$$v_i = \frac{x_i - \min(x)}{\max(x) - \min(x)},$$

where v_i is the i^{th} normalized value, x_i is the original value at position i , $\min(x)$ is the minimum value of attribute x and $\max(x)$ is the maximum. For example if attribute can obtain values from 10 to 20 and we are normalizing value 15, the new value will be $v = \frac{15-10}{20-10} = \frac{1}{2}$.

2. **z-score** – attribute values are normalized based on the attribute mean and standard deviation. Value x_i of an attribute x is normalized to v_i by calculating

$$v_i = \frac{x_i - \bar{x}}{\sigma_x},$$

where \bar{x} is mean and σ_x is standard deviation of x . Standard deviation can be computed using the equation

$$\sigma_x = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}.$$

Lets suppose that we want to normalize attribute with mean value of 1200 and standard deviation equal to 350. A value $x = 1100$ is normalized to $v = \frac{1100-1200}{350} = -\frac{2}{7}$.

Another technique for data-preprocessing is called binarization. This method can be applied to any type of attribute (numeric, nominal, ...). It is intended to transform arbitrary attribute into its boolean counterpart based on a specific attribute value.

Lets assume, that we have dataset with attribute risk which can contain three different values (*low*, *medium* and *high*) as shown in Figure 1.2. The original data are shown in table on the left. Suppose we only want to avoid high risk cases. Hence, the values *low* and *medium* are equally good for us so we do not need to distinguish between them. Thus, we can binarize this attribute with respect to *high* value. A completely new attribute will be created with value 1 if the previous value was *high* or 0 otherwise. The original attribute is then replaced with this new one. On the right side of Figure 1.2 is binarized attribute which corresponds to the original attribute with respect to *high* value.

...	risk	risk-high	...
	medium				0	
	high				1	
	high				1	
	low				0	
...	medium	...	→	...	0	...
	low				0	
	low				0	
	high				1	
	medium				0	
	low				0	

Figure 1.2: Binarization of attribute. Original values are shown on the left. New attribute binarized with target value *high* is shown on the right.

More advanced type of binarization creates a new attribute for each value. The example is shown on Figure 1.3. Attribute risk is replaced by three new attributes (risk-low, risk-medium and risk-high). For each row, exactly one of these attributes has value 1 in accordance with original value and the rest of them have 0.

...	risk	risk-low	risk-medium	risk-high	...
	medium		\rightarrow		0	1	0	
	high				0	0	1	
	high				0	0	1	
	low				1	0	0	
...	medium	0	1	0	...
	low				1	0	0	
	low				1	0	0	
	high				0	0	1	
	medium				0	1	0	
	low				1	0	0	

Figure 1.3: Advanced binarization method. New attribute is created for every value of original attribute.

Chapter 2

Data-mining algorithms

Data mining comprises the algorithms that enable us to gain fundamental insights and knowledge from massive data. It is an interdisciplinary field merging concepts from areas like databases, statistics, pattern recognition, . . . In fact, data mining is a part of larger knowledge discovery process, which includes pre-processing tasks like data extraction, data cleaning, data reduction and feature construction, as well as post-processing steps like model interpretation, hypothesis confirmation and so on. This process tends to be highly iterative and interactive. The algebraic, geometric and probabilistic viewpoints of data play a key role in data mining methods. There are a lot of types of algorithms which solve data-mining tasks. They include exploratory data analysis, frequent pattern discovery, data clustering and classification models [3].

Our prototype of web based data mining assistant now supports four classification algorithms. We could integrate many more algorithms into our system, but this is out of the scope of this prototype design. These four algorithms are namely k -nearest neighbor, Logistic regression, Decision tree and Support vector machines. Next we describe these algorithms in detail, because they play an important role in our solution.

2.1 k -nearest neighbor

The k -nearest neighbor algorithm belongs among the simplest data-mining models. Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The training tuples are described by n attributes. Each tuple represents a point in an n -dimensional space. In this way, all the training tuples are stored in an n -dimensional pattern space. When given an unknown tuple, a k -nearest neighbor classifier searches the pattern space for

the k training tuples that are closest to the unknown tuple. These k training tuples are the k "nearest neighbors" of the unknown tuple.

"Closeness" is defined in terms of a distance metric, such as Euclidean distance. The Euclidean distance between two points or tuples, e.g. $X_1 = (x_{1,1}, x_{1,2}, \dots, x_{1,n})$ and $X_2 = (x_{2,1}, x_{2,2}, \dots, x_{2,n})$, is computed as follows

$$dist(X_1, X_2) = \sqrt{\sum_{i=1}^n (x_{1,i} - x_{2,i})^2}. \quad (2.1)$$

In other words, for each numeric attribute, we take the difference between the corresponding values of that attribute in tuple X_1 and in tuple X_2 , square this difference, and accumulate it. The square root is taken of the total accumulated distance count. Basically, we normalize the values of each attribute before using Eq. 2.1. This helps to prevent attributes with initially large ranges (e.g. price) from outweighing attributes with initially smaller ranges (e.g. binary attributes). Min-max normalization can be used to transform a value v of a numeric attribute A to v' in the range $[0, 1]$ by computing

$$v' = \frac{v - min_A}{max_A - min_A} \quad (2.2)$$

where min_A and max_A are the minimum and maximum values of attribute A .

The previous metrics assumes that the attributes used to describe the tuples are all numeric. For nominal attributes (e.g. color), a simple method is to compare the corresponding value of the attribute in tuple X_1 with that in tuple X_2 . If the two are identical, then the difference between the two is taken as 0. If the two are different, then the difference is considered to be 1. Other methods may incorporate more sophisticated schemes for differential grading (e.g., where a larger difference score is assigned for example for blue and white color than for blue and black color).

For k -nearest neighbor classification, the unknown tuple is assigned the most common class among its k -nearest neighbors. When $k = 1$, the unknown tuple is assigned the class of the training tuple that is closest to it in pattern space. Nearest neighbor classifiers can also be used for numeric prediction, that is, to return a real-valued prediction for a given unknown tuple. In this case, the classifier returns the average value of the real-valued labels associated with the k -nearest neighbors of the unknown tuple [2].

The exposed parameter k can be used to obtain more precise results on the same dataset with this method just by taking into account more or less number of nearest

neighbors. In general, the larger the number of training tuples, the larger the value of k will be, so it is highly dependent on concrete dataset.

2.2 Logistic regression

To describe the Logistic regression model we have to understand the Linear regression first, because Logistic regression is based on this method. Linear regression belongs to regression methods which are designed to predict numerical attributes. The main difference between classification and regression is that in classification there are countable many types of classes to choose from. On the other side, in regression there are infinite number of them (e.g. real numbers).

When all the attributes are numeric, we can use linear regression technique, which is a staple method in statistics too. The idea is to express the class as a linear combination of the attributes, with predetermined weights

$$x = w_0 + w_1a_1 + w_2a_2 + \cdots + w_k a_k, \quad (2.3)$$

where x is the class, a_1, a_2, \dots, a_k are the attribute values and w_0, w_1, \dots, w_k are weights. The weights are calculated from the training data. The predicted value for the first instance's class can be written as follows

$$w_0a_0^{(1)} + w_1a_1^{(1)} + w_2a_2^{(1)} + \cdots + w_k a_k^{(1)} = \sum_{j=0}^k w_j a_j^{(1)}, \quad (2.4)$$

where the superscript denotes that it is the first instance. Moreover, it is notationally convenient to assume an extra attribute a_0 , with a value that is always 1. The value from (2.4) is predicted, not the actual value for the class. The difference between the predicted and actual values is important and should be as small as possible. The goal of the linear regression method is to choose the coefficients w_j to minimize the sum of the squares of these differences over all the training instances. If there are n training instances then the sum of the squares of differences is

$$\sum_{i=0}^n \left(x^{(i)} - \sum_{j=0}^k w_j a_j^{(i)} \right)^2 \quad (2.5)$$

where the expression inside the parentheses is the difference between the actual class of the i th training instance and its predicted class.

Linear regression can easily be used for classification in domains with numeric attributes. The trick is to perform a regression for each class, setting the output equal to 1 for training instances that belong to the class and 0 for those that do not. The result is a linear expression for the class. Then, given a test example of unknown class, calculate the value of each linear expression and choose the one that is the largest.

This approach often yields good results in practice. However, it has two drawbacks. First, the membership values it produces are not proper probabilities because they can fall outside the range $[0, 1]$. Second, least-squares regression assumes that the errors are statistically independent and also normally distributed with the same standard deviation, an assumption that is violated when the method is applied to classification problems because the training examples only take on the values 0 or 1.

A Logistic regression technique does not suffer from these problems. Instead of approximating the 0 and 1 values directly, logistic regression builds a linear model based on a transformed target variable. Lets suppose first that we have only two classes. Logistic regression replaces the original target variable

$$P[1|a_1, a_2, \dots, a_k] \quad (2.6)$$

by variable

$$\log \left(\frac{P[1|a_1, a_2, \dots, a_k]}{1 - P[1|a_1, a_2, \dots, a_k]} \right) \quad (2.7)$$

which is no longer constrained to the interval $[0, 1]$, but can lie anywhere between negative and positive infinity. Now we can approximate the transformed variable using a linear function just like the ones generated by Linear regression. The resulting model is

$$P[1|a_1, a_2, \dots, a_k] = \frac{1}{1 + \exp(-w_0 - w_1 a_1 - \dots - w_k a_k)}, \quad (2.8)$$

where w are weights.

In Logistic regression model weights are found using the log-likelihood of the model instead of using the squared error like in Linear regression. This is given by formula

$$\sum_{i=1}^n (1 - x^{(i)}) \log \left(1 - P[1|a_1^{(1)}, a_2^{(2)}, \dots, a_k^{(k)}] \right), \quad (2.9)$$

where $x^{(i)}$ are either 0 or 1.

The weights w_i have to be chosen to maximize the log-likelihood. This can be done for example by iteratively solving a sequence of weighted least-squares regression problems until the log-likelihood converges to a maximum.

One can generalize Logistic regression to several classes by approach described above for Multiresponse Linear regression by performing Logistic regression independently for each class [4].

The Logistic regression classifier implemented in Weka package exposes two hyperparameters to fine tune its results. First hyperparameter corresponds to the maximum number of iterations given to algorithm for finding the maximum of log-likelihood of the model. If the algorithm does not find the maximum of log-likelihood after specific number of iterations, then the best weights found so far are used. The Logistic regression classifier implemented in Weka uses penalized maximum likelihood estimation with a quadratic penalty function. This likelihood estimation exposes the second hyperparameter of this model. It is the penalty hyperparameter, which stands for the weight of the penalty. The higher the value of the penalty parameter (aka ridge parameter) the closer to zero are the penalized maximum likelihood estimates.

2.3 Decision tree

In this section we look at a widely-used method of constructing a model from a dataset in the form of a set of decision rules. This representation of the data has the advantage compared with other approaches of being meaningful and easy to interpret. Therefore we explain this model using the example below.

The Table 2.1 contains data about a golfer who decides whether or not to play each day on the basis of the weather. First four attributes are related to weather and the last attribute is a golfer's decision to play or not to play. There are two numerical attributes, namely *Temp* and *Humidity*, and three categorical attributes. Attribute *Outlook* can contain these three possible values: sunny, overcast or rain. Attribute *Windy* is a boolean attribute, which means that it can contain only true or false values. *Class* attribute is a classification attribute and can contain only two possible values: play or don't play.

If we assume that the golfer is acting consistently, how can be the rules that determine the decision whether or not to play each day defined? One way of answering this is to construct a decision tree such as the one shown in Figure 2.4.

Outlook	Temp ($^{\circ}F$)	Humidity (%)	Windy	Class
sunny	75	70	true	play
sunny	80	90	true	don't play
sunny	85	85	false	don't play
sunny	72	95	false	don't play
sunny	69	70	false	play
overcast	72	90	true	play
overcast	83	78	false	play
overcast	64	65	true	play
overcast	81	75	false	play
rain	71	80	true	don't play
rain	65	70	true	don't play
rain	75	80	false	play
rain	68	80	false	play
rain	70	96	false	play

Table 2.1: Data for the Golf Example.

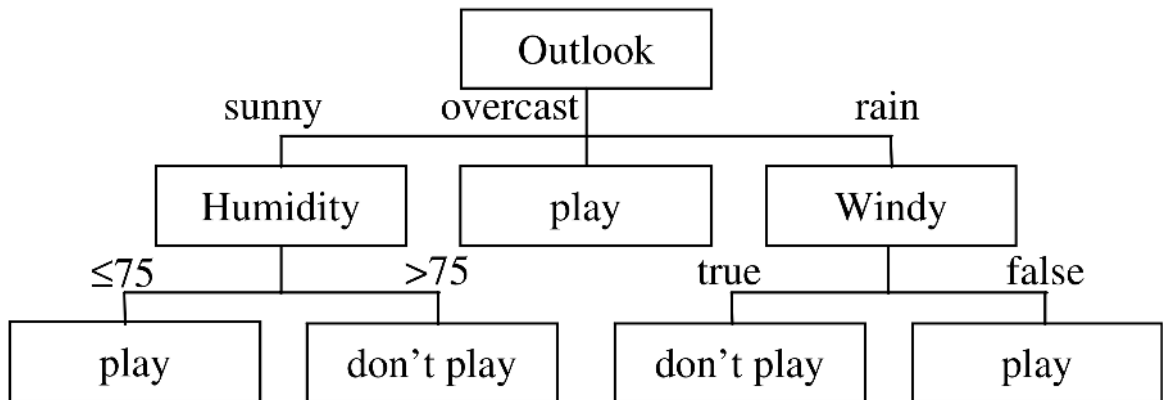


Figure 2.4: Decision tree for the golfer example [5].

A decision tree is created by a splitting on the value of attributes, i.e. testing the value of an attribute and then creating a branch for each of its possible values. In the case of continuous attributes the value is tested against *less than or equal to* or *greater than* comparison with a given split value. The splitting process continues until each branch can be labelled with just one classification.

A decision tree can be viewed as not just equivalent of the original training set but as a generalization of it which can be used to predict the classification of other

instances. For example if we want to determine the classification for the new set of weather conditions from this tree, the first look is at the value of *Outlook* attribute.

1. If the value of *Outlook* is sunny, the next attribute we consider is *Humidity*. If the value is less than or equal to 75, the decision is play. Otherwise, the decision is don't play.
2. If the value of *Outlook* is overcast, then the decision is play.
3. If the value of *Outlook* is rain, the next attribute we consider is *Windy*. If the value is true the decision is don't play, otherwise the decision is play.

2.3.1 TDIDT Algorithm

A Weka tool uses the TDIDT algorithm for constructing a decision tree from a training set. It is a very popular method for constructing decision trees. TDIDT stands for Top-Down Induction of Decision Trees. Below is the pseudocode of this algorithm.

Algorithm 1: TDIDT – Basic Algorithm

```
if all the instances in the training set belong to the same class then
    | return the value of the class;
else
    | (1) select an attribute A to split on (never select an attribute twice in the
    |   same branch);
    | (2) sort the instances in the training set into subsets, one for each value of
    |   attribute A;
    | (3) return a tree with one branch for each non-empty subset, each branch
    |   having a descendant subtree or a class value produced by applying the
    |   algorithm recursively;
end
```

This method produces decision rules in the implicit form of a decision tree. These trees are generated by repeatedly splitting on the values of attributes, which is also known as recursive partitioning.

At each non-leaf node an attribute is chosen for splitting. Selection of this attribute is arbitrary, except that the same attribute must not be chosen twice or more times in

the same branch. Obviously, each split on the value of an attribute extends the length of the corresponding branch by one, but the maximum possible length for a branch is H terms, if there are H attributes. So the algorithm is guaranteed to terminate.

TDIDT algorithm has one important condition, which must be met, and that is the consistency of the training data. That means that no two instances with the same values of all the attributes may belong to different classes. Under this condition the algorithm is guaranteed to terminate and any selection of attributes (even random) will produce a decision tree, provided that an attribute is never selected twice in the same branch [5].

An implementation of TDIDT algorithm in Weka package exposes three major hyperparameters to fine tune the results. First hyperparameter (M) determines the minimum number of instances per leaf in a decision tree. This guarantees that the path from the root to the leaf of a decision tree describes at least M instances of a respective class. The two other hyperparameters are related to the method for adjusting a decision tree called pruning.

Pruning means generating a tree with fewer branches than would otherwise be the case (known as pre-pruning) or removing parts of a tree that has already been generated (known as post-pruning). This will give a smaller and simpler tree, which may be able to predict the correct classification more accurately for unseen data. On the other side this decision tree is unlikely to be able to predict correctly the classification of some of the instances in the training set. As we already know what those values are, this is of little or no importance.

The first hyperparameter related to pruning is just a simple boolean variable, which determines whether the pruning will be used or not. The second hyperparameter is considered only if the pruning is used. This parameter is called pruning confidence factor. The value of this hyperparameter affects how pessimistically the tree is pruned back. That means how big error do we accept on the training data because of pruning. The value of this hyperparameter can acquire any real value from interval $[0, 1]$. Smaller confidence factors result in further tree pruning.

2.4 Support vector machines

Support vector machines (SVMs) is a classification method for linear, but also non-linear data. SVM algorithm works with two classes, but it is easily extensible to classify multiple classes using the approach described in Section 2.2. SVM uses a

nonlinear mapping for transforming a training data into a higher dimension. Within this dimension, it searches for the linear separating hyperplane.

A hyperplane is a subspace one dimension smaller than its surrounding space. Hyperplane for 2-dimensional space is a line, for 3-dimensional space it is a 2D plane, and so on.

This hyperplane can be interpreted as a decision boundary, which separates the tuples of one class from another. It is important to note that with a sufficiently high dimension, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using support vectors (important data points from a training set) and margins, which are defined by the support vectors. The algorithm is described in detail below.

2.4.1 Linearly separable data

Lets look at the case when the data are linearly separable first. A dataset D is defined as follows

$$D = \{(X_1, y_1), (X_2, y_2), \dots, (X_{|D|}, y_{|D|})\}$$

where X_i is the set of training tuples and y_i are associated class labels. Each y_i can represent one of the two values, either 1 or -1 , corresponding to the class of the tuple X_i . Let's consider an example dataset with two attributes, namely A_1 and A_2 . These two attributes define the class of each instance as shown on Figure 2.5. Gray dots represent instances which belong to a first class and white dots mark the instances of a second class.

This data are linearly separable, because a straight line can be drawn to separate all the tuples of one class from the other. Dashed lines represent possible placement of a separating hyperplanes. There are possibly an infinite number of them.

Our goal is to find the best separating hyperplane. That is the one that will have the minimum classification error on previously unseen instances. SVM algorithm solves this problem by searching for the maximum marginal hyperplane.

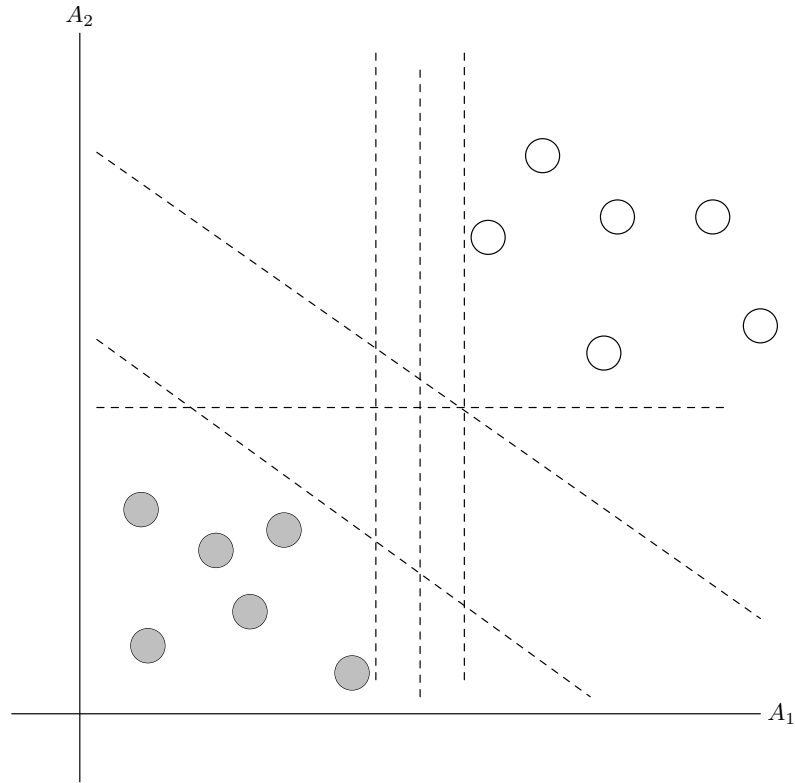


Figure 2.5: Example of a linearly separable dataset.

Consider Figure 2.6, which illustrates two different separating hyperplanes and their associated margins. Both hyperplanes correctly classify all the data instances, however, we expect that the hyperplane with the larger margin can classify future instances more accurately than the hyperplane with much smaller margin. The hyperplane with the largest margin is called the maximum marginal hyperplane (MMH). Margin associated with this hyperplane gives the largest distance between these two classes.

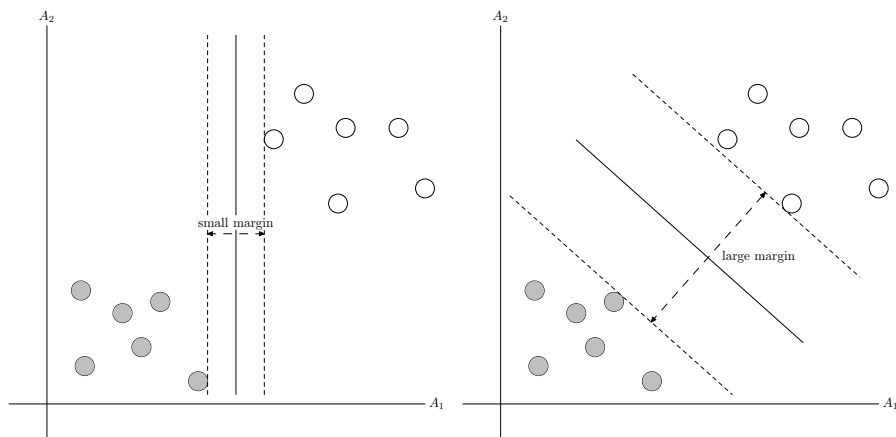


Figure 2.6: Possible separating hyperplanes and margins associated for them.

A separating hyperplane can be defined as follows

$$W \cdot X + b = 0, \quad (2.10)$$

where W is a weight vector for attributes, X are training tuples and b is a scalar called bias. Considering our example with two attributes A_1 and A_2 . We can rewrite equation (2.10) as

$$w_0 + w_1x_1 + w_2x_2 = 0, \quad (2.11)$$

where w_0 is a bias and x_1 and x_2 are the values of attributes A_1 and A_2 respectively. Therefore, we know that any point that lies above the separating hyperplane satisfies equation (2.12)

$$w_0 + w_1x_1 + w_2x_2 > 0 \quad (2.12)$$

and point that lies below the separating hyperplane satisfies equation (2.13)

$$w_0 + w_1x_1 + w_2x_2 < 0. \quad (2.13)$$

We can define the sides of the margin by adjusting the weights in equations (2.12) and (2.13) respectively

$$H_1 : w_0 + w_1x_1 + w_2x_2 \geq 1, \quad (2.14)$$

$$H_2 : w_0 + w_1x_1 + w_2x_2 \leq -1. \quad (2.15)$$

Combining these two inequalities we get

$$H : y_i (w_0 + w_1x_1 + w_2x_2) \geq 1, \quad (2.16)$$

where $y_i \in [-1, 1]$ are class labels. All training data instances that satisfy equation (2.16) are called **support vectors**. Support vectors are training tuples which are equally close to the maximum marginal hyperplane.

To find the support vectors and the MMH, SVM algorithm transforms the equation (2.16) to convex quadratic optimization problem. More details are beyond the scope of this thesis, but this problem is well known and many algorithms exists for solving it. The equation (2.16) can be redefined as follows

$$d(X^T) = \sum_{i=1}^l y_i \alpha_i X_i X_T + b_0 \quad (2.17)$$

where l is the number of support vectors, X_i is a support vector, X^T is a test tuple and α_i and b_0 are numeric constants. For an unseen instance X_T we substitute the values into equation (2.17) and check the sign of the result. If it is positive, then the instance X_T lies on or above the MMH and its class is 1. If the sign is negative, then X_T lies on or below the MMH and its class is -1 .

2.4.2 Linearly inseparable data

Now we discuss the case, when the data are not linearly separable. Figure 2.7 shows an example of linearly inseparable dataset. It is clear, that it is not possible to draw a single line that completely separates the classes. Now we describe the approach of extending the linear SVMs to nonlinear SVMs, which can classify linearly inseparable data instances.

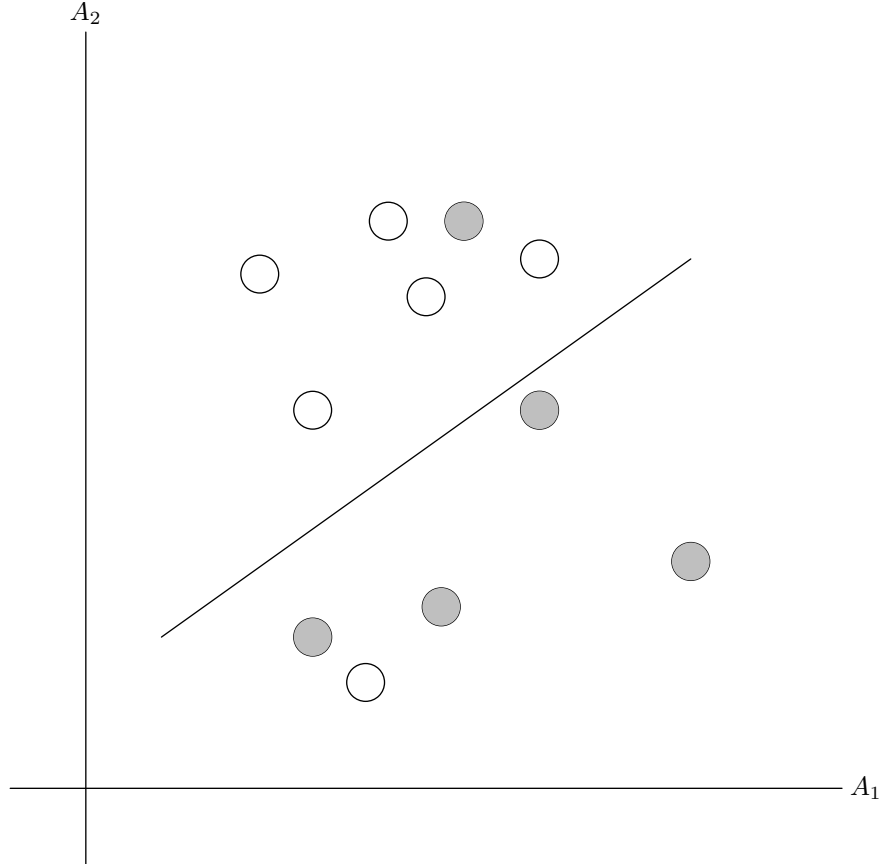


Figure 2.7: Example of a linearly inseparable data.

Firstly, we have to transform the original data instances into a space with a higher

dimension, where they can be separated by a hyperplane. This is done by nonlinear mapping. Lets assume, that we have n -dimensional vector X for training tuples. We use nonlinear transformation of X to m -dimensional space Z ($n < m$) using the transformation ϕ . A separating hyperplane in the new space is

$$d(Z) = WZ + b \quad (2.18)$$

where W and Z are vectors. The equation (2.18) is linear, so we can solve it and transform back to the original space. The separating hyperplane in a transformed space corresponds to a nonlinear polynomial in the original space. Transformed space can be much bigger than the original space, hence the computation of dot products in this space can be very expensive. For solving the quadratic optimization problem, the SVM algorithm uses a kernel function. Thanks to a kernel function, we are able to provide computations in the original space that are mathematically equivalent to the computations in the transformed space. Kernel function looks like this

$$K(X_i, X_j) = \phi(X_i) \cdot \phi(X_j), \quad (2.19)$$

where $\phi(X)$ is a nonlinear function applied to training tuples. Because the training tuples appears everywhere in a form of dot products, we can substitute them with $K(X_i, X_j)$ and make all calculations in the original space. In addition, we do not have to know how the transformation ϕ looks like.

There are many different kernel types. For example

- Linear kernel: $K(X_i, X_j) = X_i \cdot X_j$
- Sigmoid kernel: $K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta)$
- Polynomial kernel of degree h : $K(X_i, X_j) = (X_i \cdot X_j + 1)^h$
- Gaussian radial basis function (RBF) kernel: $K(X_i, X_j) = e^{\gamma \|X_i - X_j\|^2}$.

Last step is to find the maximum marginal hyperplane. This is similar to the approach described above. The only difference is that a user has to specify the upper bound constant c (complexity constant) for Lagrange multipliers α_i manually. The MMH found in the transformed space corresponds to a nonlinear separating hyper-surface in the original space.

SVMs are highly accurate, because of their ability to model complex nonlinear decision boundaries. They are also much less susceptible to overfitting. On the other

side, the training time of SVMs can be extremely slow comparing to the other methods we described in this chapter [2].

An implementation of SVM algorithm in LibSVM package exposes several hyperparameters to improve the results. We describe three of them, which we used in our experiments. First hyperparameter determines the kernel function used in the algorithm (one of mentioned above).

Hyperparameter c stands for a complexity constant mentioned above. This parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of parameter c , the optimization will choose a smaller margin decision hyperplane, if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of c will cause the optimizer to look for a larger margin decision hyperplane, even if that hyperplane misclassifies more points.

The last parameter is γ . It is considered when the SVM algorithm uses RBF kernel. In the equation for RBF kernel function gamma has a value of

$$\gamma = \frac{1}{2\sigma^2},$$

where σ is a free parameter defined by the user.

Chapter 3

Model recommendation

In this chapter we formally define Combined selection and hyperparameter optimization (CASH) problem. Then we present several solutions for selection and hyperparameter optimization of algorithms. They vary in time complexity and accuracy depending on the algorithm used. At the beginning we describe simple approach used in Gridsearch algorithm and then some more sophisticated solutions which include Meta-learning features. At the end of this chapter we present our own algorithm designed for the best time/accuracy ratio.

3.1 Definitions

Combined selection and hyperparameter optimization problem can be defined as follows: given a dataset, simultaneously choose a learning algorithm and set its hyperparameters to optimize empirical performance. At present, high attention is given to this concept [6].

The goal of model selection is to determine the algorithm $A^* \in A$ with optimal generalization performance for given set of learning algorithms A and training data $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$. Model selection can be written as

$$A^* \in \underset{A \in \mathcal{A}}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^k \mathcal{L} \left(A, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)} \right), \quad (3.1)$$

where $\mathcal{D}_{train}^{(i)}$ and $\mathcal{D}_{valid}^{(i)}$ are disjoint training and validation sets acquired by splitting training data D . Loss function $\mathcal{L} \left(A, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)} \right)$ represents the misclassification

rate achieved by algorithm A when trained on $\mathcal{D}_{train}^{(i)}$ and evaluated on $\mathcal{D}_{valid}^{(i)}$. K-fold cross validation is used as a validation technique.

Hyperparameter optimization problem can be defined as follows: for given learning algorithm A find appropriate values for its hyperparameters $\lambda \in \Lambda$. This can be written as

$$\lambda^* \in \underset{\lambda \in \Lambda}{argmin} \frac{1}{k} \sum_{i=1}^k \mathcal{L} \left(A_{\lambda}, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)} \right). \quad (3.2)$$

As you can see from equations (3.1) and (3.2), these two problems are conceptually similar. The difference between them is that hyperparameter space is often high dimensional and continuous. Correlation property between hyperparameter values $\lambda_1, \lambda_2 \in \Lambda$ can be exploited during the optimization process.

CASH problem can be defined as follows: given a set $\mathcal{A} = \{A^{(1)}, \dots, A^{(k)}\}$ of algorithms with their hyperparameter spaces $\Lambda^{(1)}, \dots, \Lambda^{(k)}$, the goal is to compute

$$A^* \lambda^* \in \underset{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}}{argmin} \frac{1}{k} \sum_{i=1}^k \mathcal{L} \left(A_{\lambda}^{(j)}, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)} \right). \quad (3.3)$$

This problem can be reformulated as a single combined hierarchical hyperparameter optimization problem with $\Lambda = \Lambda^{(1)} \cup \dots \cup \Lambda^{(k)} \cup \{\lambda_r\}$, where λ_r is a new top level hyperparameter which represents the choice of learning algorithm.

3.2 Gridsearch

The most simple approach of solving CASH problem is using Gridsearch algorithm. This algorithm simply searches whole hyperparameter space of a given data mining algorithm. We describe the principle of the algorithm on its sample source code listed below.

```
/**
 * @param data - dataset with data instances
 * @return best hyperparameter values for given data
 *         mining method and dataset
 */
public SearchResult gridSearch(DataInstances data) {
    double minError = Double.MAX_VALUE;
```

```

    double bestValueForParam1;
    int bestValueForParam2;

    for (double param1 = LOWER_BOUND_FOR_PARAM_1; param1 <
        UPPER_BOUND_FOR_PARAM_1; param1+= STEP_FOR_PARAM_1)
    {
        for (int param2 = LOWER_BOUND_FOR_PARAM_2; param2
            < UPPER_BOUND_FOR_PARAM_2; param2++) {
            double error = dataMiningAlgorithm(data,
                param1, param2);
            if (error < minError) {
                minError = error;
                bestValueForParam1 = param1;
                bestValueForParam2 = param2;
            }
        }
    }

    SearchResult bestResult = new SearchResult(param1,
        param2);
    return bestResult;
}

```

Code of above-mentioned `gridSearch()` method finds the best hyperparameter settings for concrete data mining algorithm represented by `dataMiningAlgorithm()` method whose arguments are specific values of its hyperparameters and also dataset analyzed by this algorithm. In this concrete case our data mining algorithm has two hyperparameters, namely `param1` and `param2`, for which the grid search algorithm searches the hyperparameter space. Both hyperparameters have their lower and upper bounds, which restrict the space of their reasonable values. As you can see above, these parameters can be of various types. Because it is not always possible to search all hyperparameter space value by value, such as the space of real numbers, in this case the algorithm uses a constant value, which increases a value of hyperparameter in every iteration of the algorithm. The value of such constant should be small enough to not worsen the quality of result too much, but large enough to finish in a reasonable

computation time.

Results are compared according to the error (eg. root mean square error), which reaches a data mining algorithm on a given dataset with given hyperparameters. Obviously, smaller error means the better result. During the execution of the Gridsearch algorithm, error of the best result so far is stored in `minError` variable. Hyperparameter values pertaining to that result are stored in `bestValueForParam1` and `bestValueForParam2` variables. These values are returned by the `gridSearch()` method at the end of the algorithm run.

The advantage of this approach is that you will get very accurate results for hyperparameter setting. It is also very easy to extend this algorithm to recommend not just hyperparameter setting but a data mining algorithms as well. You can run Gridsearch algorithm for every data mining algorithm you want to recommend and return the best result overall.

The main disadvantage is that you can not use this approach, if you want real time recommendation for CASH problem. Hyperparameter space can be very large for data mining algorithms with more than one or two hyperparameters, even if you have just tens or hundreds values for a single parameter. If you multiply five or more of such parameters you get hundreds of thousands possible combinations and so as many calculations. Therefore, such an algorithm can run for several hours or days. For the user of our system it is not feasible to wait so long.

Nevertheless, we decided to use this approach for background analysis which is completely independent from real-time interaction with a user. This analysis can run for very long time, because the user will interact with results from our Meta-learning approach. The data from Gridsearch analysis will serve us as a background knowledge for our Meta-learning algorithm in the future experiments.

3.3 Meta-learning

Meta-learning as a subfield of Machine learning is a discipline where metadata from previous experiments are used to improve experiments in the future. In our case we can define Meta-learning task as a task of recommending an algorithm and its hyperparameters for new dataset based on the recommendations for other datasets in the past. These Meta-learning algorithms uses various metadata information about datasets. They can be divided into several groups as follows [7]:

- **Dataset Characterization**

The central idea is that high-quality dataset characteristics or meta-features provide some information to differentiate the performance of a set of given learning strategies.

- **Statistical and Information-Theoretic Characterization**

Much work in dataset characterization has been concentrated on extracting statistical and information-theoretic parameters estimated from the training set. Measures include number of classes, number of features, ratio of examples to features, degree of correlation between features and target concept, average class entropy and class-conditional entropy, skewness, kurtosis, signal-to-noise ratio, etc.

- **Model-Based Characterization**

In addition to statistical measures, a different form of dataset characterization exploits properties of the induced hypothesis as a form of representing the dataset itself. As an example, one can build a decision tree from a dataset and collect properties of the tree (e.g. nodes per feature, maximum tree depth, shape, tree imbalance, etc.), as a means to characterize the dataset.

- **Landmarking**

Another source of characterization falls within the concept of landmarking. The idea is to exploit information obtained from the performance of a set of simple learners (i.e. learning systems with low capacity) that exhibit significant differences in their learning mechanism. The accuracy (or error rate) of these landmarks is used to characterize a dataset and identify areas where each of the simple learners can be regarded as an expert. Another idea related to landmarking is to exploit information obtained on simplified versions of the data (e.g. samples). Accuracy results on these samples serve to characterise individual datasets and are referred to as sub-sampling landmarks. This information is subsequently used to select an appropriate learning algorithm.

We have decided to study further Meta-learning algorithms for solving CASH problem described in this Chapter. One such algorithm was presented by Kazík et al on 11th International Conference on Machine Learning and Applications [8]. Further we describe this approach, because it will serve us as a baseline. Later in this chapter we compare the success rate of our design with this algorithm. The main

idea behind both algorithms is that for very similar datasets perform well same data mining methods. A difference between these two algorithms is in the methodology of computing the similarity of datasets.

3.3.1 Statistical characterization approach

Algorithm presented by Kazík recommends data mining method and its hyperparameter settings based on the comparison of datasets by their statistical characteristics. Therefore it requires certain background knowledge. This knowledge consists of a set of entries for previously analyzed datasets. Any such entry contains values of observed dataset metadata and the recommended algorithm and its hyperparameter settings. For determining the compatibility of datasets authors decided to use these meta information about dataset

- **number of attributes** – attributes that specify data characteristics
- **number of instances** – number of records in a dataset
- **data type** – refers to all values of all attributes in the dataset. Possible values are *integer*, *real*, *categorical* or *multivariate*
- **default task** – type of a task that is connected with the data. Algorithm distinguishes between the classification and regression types of tasks, but in our experiments we focused only on classification tasks so this meta attribute does not affect the results. Therefore we omitted it.
- **missing values** – number of missing values in a dataset.

The distance between the two datasets is defined by a metadata metric. If all the meta attribute values are the same, the distance equals to zero and it is the largest when the two datasets differ the most. The exact distance is computed as follows

$$d(m_1, m_2) = \sum_{i=1}^n w_i \cdot d_i(m_1[i], m_2[i]), \quad (3.4)$$

where m_1 and m_2 are the two compared datasets, n is the number of observed meta attributes, i stands for the particular meta attribute, w_i is the weight for the particular meta attribute and d_i is the distance of the two values of the i th meta attribute.

The distance function d_i of the two meta attributes values differs according to the type of the meta attribute. In case of the categorical and boolean meta attribute type, where the value is one of the given set of values, the following formula is used

$$d_i(v_1, v_2) = \begin{cases} 0 & \text{if } v_1 = v_2; \\ 1 & \text{otherwise,} \end{cases} \quad (3.5)$$

where v_1 and v_2 are the the particular values of i th meta attribute. For numerical attributes, the following formula is used

$$d_i(v_1, v_2) = \frac{|v_1 - v_2|}{\max(v)_{v \in V[i]} - \min(v)_{v \in V[i]}}, \quad (3.6)$$

where $V[i]$ stands for all possible values of a meta attribute i .

The proposed metric is similar to metric in other Meta-learning systems like MetaL [9]. This metric also exposes hyperparameters w_i to further optimize recommendations by setting weights to every particular meta attribute.

The algorithm calculates the distance of the new dataset from datasets from background knowledge and chooses the closest one. The recommended model for the new dataset is a method and its hyperparameters which performed the best on that closest dataset. Therefore, basic knowledge have to contain the recommended models for individual datasets.

This algorithm is much faster comparing to a Gridsearch since it performs only simple statistical calculations, not a sophisticated data-mining computations like Gridsearch approach. On the other hand this algorithm does not search all possible combinations so the results are less accurate.

3.3.2 Landmarking approach

Landmarking as a subfield of Meta-learning exploits the information obtained from the performance of data mining algorithms on previously analyzed datasets. Our motivation for studying this approach was the idea that we will achieve more accurate results when we focus on comparing datasets according to their performance on data mining algorithms instead of comparing their statistical characteristics.

In this section we describe our own algorithm design which is based on a Meta-learning approach with usage of Landmarking features. The main goal of our algorithm is to significantly speed up the process of selection and hyperparameter

optimization while maintaining the reliability of results within reasonable limits at the same time.

The idea behind this algorithm is similar to the algorithm described above, but instead of statistical characteristics we used landmarking features for computing dataset distances. We think that the datasets that behave similarly for predetermined models with given hyperparameters also behave similarly in the rest of hyperparameter space for these models. We compute the exact distance between datasets m_1 and m_2 as follows

$$d(m_1, m_2) = \sum_{a \in A} w_a \cdot d_a(m_1, m_2), \quad (3.7)$$

where A is a set of considered data-mining methods with predetermined hyperparameter settings, w_a is weight of individual model and d_a is a distance between datasets m_1 and m_2 for particular model $a \in A$ defined by equation (3.8)

$$d_a(m_1, m_2) = |err_a(m_1) - err_a(m_2)|, \quad (3.8)$$

where err_a is an arbitrary error function of a given model. To obtain the best performance, the computation of error function for a set of considered models is done just for a new dataset. For datasets in background knowledge, one can preserve this information as a part of this knowledge. So when a new dataset is added to a background knowledge, this information is saved with it.

We would like to note, that our algorithm has the same property as the previous one, that is, if datasets are the same, then they behave equally for every data-mining model and their distance is therefore equal to zero. The more different the behaviour of datasets is, the greater is their mutual distance.

Presented algorithm is slower than the algorithm based on statistical characteristics regarding to the speed of computation, because it requires to run several data-mining methods to determine dataset distances. On the other hand, it is still significantly faster than Gridsearch, because it does not use backtrack approach. We will present the accuracy of this approach later in this chapter.

3.4 Hybrid approach

We also combined two approaches described above and designed a hybrid algorithm which uses both, statistical characteristics and landmarking features. It uses both

types of background knowledge from datasets analyzed in the past to make recommendations for new dataset. This algorithm recommends a data-mining method and hyperparameter settings using the metadata about dataset as well as its performance in data-mining models. It finds a dataset from a set of already analyzed datasets that is most similar (with the smallest distance) to a new one and recommends the method which performed best on that closest dataset. The advantage of this algorithm is that an imprecision at the computed distance of one method may be balanced by a more precise distance of the second method which results in a higher overall accuracy.

The dataset distance between datasets m_1 and m_2 is a sum of distances defined by equation (3.4) and equation (3.7)

$$d(m_1, m_2) = \sum_{i=1}^n w_i \cdot d_i(m_1[i], m_2[i]) + \sum_{a \in A} w_a \cdot d_a(m_1, m_2), \quad (3.9)$$

where n stands for the number of observed meta attributes, i is the particular meta attribute, w_i is the weight for the particular meta attribute and d_i is a distance of two values of the i th meta attribute defined by equation (3.5) for categorical meta attribute and equation (3.6) for numerical attribute, A is a set of observed data-mining methods with its hyperparameter settings, w_a is weight of individual model and d_a is a dataset distance for particular model $a \in A$ defined by equation (3.8).

A diversity of distance measurement metrics in this equation can be compensated by setting the appropriate weights w_i and w_a . The bigger the weight is, the higher is an impact of observed meta parameter.

With almost no additional computational cost (statistical calculations are much faster than complex data-mining computations) compared to our original design, which uses solely landmarking features, this algorithm has a higher accuracy as we present in next section.

3.5 Experiments

To verify the accuracy of our algorithms we performed several experiments that we describe in this section. We compared four presented algorithms, namely Gridsearch, Meta-learning algorithm presented by Kazík [8], our custom Landmarking algorithm and a hybrid algorithm which combines both Meta-learning approaches. We focused on the accuracy of the last three algorithms because their time complexity is approximately the same and they can recommend a data-mining method in a few seconds.

This is an acceptable waiting time for our user unlike the running time of Gridsearch algorithm, which can run for multiple hours.

We use results from Gridsearch for the measurement of accuracy. That means, if an algorithm recommends particular data-mining method for a dataset, we look at the results from Gridsearch algorithm for this dataset and count the number of models which performed better than this recommendation. Thus, we can roughly determine how accurate the recommendation is. Because the Gridsearch searches the entire hyperparameter space evenly, the results can be regarded as authoritative.

3.5.1 Initial setup

All tested algorithms require a certain background knowledge. This knowledge consists of a set of meta information about datasets, which these algorithms use for recommendation. We used datasets from UCI Machine Learning Repository [19]. This repository contains real, publicly available datasets from various areas. We chose these 52 datasets intended to classification

- adult+stretch.data
- adult-stretch.data
- agaricus-lepiota.data
- anneal.data
- audiology.standardized.data
- auto-mpg.data-original
- bezdekIris.data
- bridges.data.version2
- car.data
- crx.data
- ecoli.data
- flag.data
- flare.data1
- german.data
- horse-colic.data
- house-votes-84.data
- housing.data
- hungarian.data
- ionosphere.data
- iris.data
- kinship.data
- kr-vs-kp.data
- lenses.data
- long-beach-va.data
- nursery.data
- o-ring-erosion-only.data

- o-ring-erosion-or-blowby.data
- post-operative.data
- reprocessed.hungarian.data
- sensor_readings_2.data
- sensor_readings_24.data
- sensor_readings_4.data
- soybean-small.data
- sponge.data
- switzerland.data
- synthetic_control.data
- test1.data
- test2.data
- test3.data
- test4.data
- test5.data
- tic-tac-toe.data
- train2.data
- train3.data
- train4.data
- train5.data
- train6.data
- vowel-context.data
- yacht_hydrodynamics.data
- yeast.data
- yellow-small+adult-stretch.data
- yellow-small.data.

For each dataset we calculated its statistical characteristics required by Kazík's algorithm. These characteristics consist of number of attributes, number of instances, data type and number of missing values. Also it is necessary to store the method with hyperparameters, which performed best for particular dataset. We use this model, which performed best for a closest dataset as a recommendation for a new one. We measured the accuracy of data-mining models by a root mean squared error function, which is defined as

$$rmse_{err} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}, \quad (3.10)$$

where y_i is the real value of a given data point and \hat{y}_i is the predicted one. We also use root mean squared error as an error function in our Landmarking algorithm. Also, to avoid overfitting, we evaluated the results using the K -fold cross validation. It is a well known technique in which a dataset is divided into K equally large parts. One part is considered as a test set and the other $K - 1$ parts are used for training

a model. Then the error rate is stated based on the performance on a test set. This step is repeated for every selection of a test and train sets and the average of these error rates is returned as a final result.

To obtain the best method with its hyperparameter settings for a particular dataset, the Gridsearch algorithm is used. In fact, that algorithm does not give the best possible result, but it searches the whole hyperparameter space equally, so the returned value is very close to optimum. This gives us a complex view of the accuracy of the algorithm for given dataset. Since our system recommends one of four different algorithms, we ran Gridsearch for each one. Therefore we considered different hyperparameter spaces.

For k-nearest neighbor algorithm the hyperparameter space is quite simple. We just iterated the k value from 1 to 100 and ran the algorithm.

Decision tree has three important hyperparameters. Gridsearch iterated the hyperparameter value for minimum number of instances per leaf from 0 to 1000 with iteration step of 5. For pruning confidence hyperparameter we tried all values from 0 to 0.5 with a step of 0.05. Pruning hyperparameter has only two possible values *true* and *false*, so we tried both of them.

Logistic regression function exposes two major hyperparameters. First one is the maximum number of iterations. We decided not to limit it and give to algorithm enough time to converge to result. The ridge hyperparameter can obtain values from 0 to 1. Gridsearch iterated hyperparameter value within those limits with a step of 0.05.

SVM has the most complex hyperparameter space from these four algorithms. As a kernel function we considered a Radial bases function and a Linear function. For complexity constant, Gridsearch algorithm iterated the values from 0 to 10 with a step of 0.1. Hyperparameter γ is the only one, which has multiplicative iteration step instead of additive. We iterated its value from 10^{-5} to 10 with multiplicative step of 10.

This will add up to several thousands of different calculations for each dataset. The considered hyperparameter values were selected to cover as much space of feasible results as possible. The last part of the background knowledge is a set A of considered data-mining methods with its hyperparameter settings for our Landmarking algorithm. We decided to use each of the four data-mining methods once with its default hyperparameter settings. Default values perform the best in general cases. It would be interesting to try also another hyperparameter settings (best, worst, ...),

but it is beyond the scope of this thesis. In our case, the set A consists of these models

1. k-nearest neighbor
 - k : 3
2. Decision tree
 - minimum number of instances per leaf: 2
 - pruning: *true*
 - pruning confidence: 0.25
3. Logistic regression
 - maximum number of iterations: *not limited*
 - ridge: 0.05
4. SVM
 - kernel: *RBF*
 - complexity constant: 0.1
 - γ : 0.1

All of the information above forms the background knowledge for our experiments.

3.5.2 Performance

As it was stated above, the performance of data-mining methods is measured by their root mean squared error. To measure the accuracy of recommendations, we decided to use different methodology, since the root mean squared error does not give us comparable results for different datasets. For example, best root mean squared error possible for one dataset with our four methods is 0.15, but for another dataset you can not get better than 0.4. Therefore, the accuracy of recommendation can not be measured by the value of root mean squared error. To estimate the correctness of the recommendation we used the results from Gridsearch.

Firstly the root mean squared error of the recommended model is calculated and then it is compared to the errors of models from Gridsearch. The recommendation error is defined by a number of models with lower root mean squared error divided by the total number of models in Gridsearch for given dataset. This defines the position

of our recommendation in the model space. For example, if Gridsearch tried 1000 different models and 50 of them have better accuracy than our recommendation, then the recommendation error is $\frac{50}{1000} = \frac{1}{20}$.

One experiment consists of recommendations for each of 52 datasets. It is important to note that Gridsearch results for the currently analyzed dataset were hidden from our recommendation methods. For each dataset, its distance to the rest 51 datasets was calculated and the model was recommended based on the closest one. Finally we calculated the recommendation error as was defined above.

To compare our Landmarking approach with the Statistical approach we had to estimate the weights in both algorithms to optimize their performance. Therefore, we defined the general error of an experiment as follows

$$err = \frac{\sum_{d \in D} better(m_d, d)}{\sum_{d \in D} total(d)}, \quad (3.11)$$

where D is a set of our 52 considered datasets, d is one particular dataset, function $better(m_d, d)$ represents a number of models in Gridsearch results with smaller error than the recommended model m_d for dataset d and $total(d)$ is the total number of Gridsearch results for dataset d . For example, if we have two datasets where recommended model for the first one is on the 11th place from 1000 and recommended model for a second one is on the 142nd from 1500 places, then the general error of recommendation algorithm for this experiment is

$$\frac{10 + 141}{1000 + 1500} = \frac{151}{2500}.$$

We used this function for performance comparison of recommendation methods with various weight settings. We performed an experiment as was described above for each recommendation method with different weight settings and chose the best one. We iterated every particular weight from 0.0 to 1.0 with an iteration step of 0.05. Moreover we considered only those weight settings of which the sum of weights is equal to 1. That is, because it does not have any sense to consider all weights equal to 1 in one experiment and equal to 0.25 in another. The ratio is the same and so are the results. We have examined 1680 different weight settings for Kazík's algorithm (four weights), 1680 different weight settings for our Landmarking algorithm (four weights) and 15075 possibilities for the combined approach with eight different weights. All the weight settings and their accuracy are attached with this thesis.

For algorithm of Kazík, these weights performed the best

- data type: 0.65
- number of instances: 0.15
- number of attributes: 0.2
- number of missing values: 0.0

For our Landmarking design these weights were selected

- Decision tree: 0.35
- Logistic regression: 0.3
- SVM: 0.25
- k-nearest neighbor: 0.1 .

These two approaches are compared in experiment shown in Table 3.1. First column represents a dataset for which the recommendation was made. Second column is for dataset with the smallest distance to currently analyzed dataset according to our recommendation algorithm based on Landmarking features. Third column represents a dataset which is closest to analyzed dataset according to Kazík’s algorithm based on statistical approach. Next three columns represent the root mean squared error (RMSE) values. First of them contains the best value overall for analyzed dataset from Gridsearch. Second one contains the RMSE of recommended method applied to analyzed dataset, where the recommendation was made by our Landmarking approach. The last one is the same as previous, but the recommendation was made by statistical approach of Kazík’s algorithm. Last two columns contain the recommendation errors of Landmarking and statistical approaches for analyzed datasets respectively.

Dataset	Landmarking	Metadata	Best rmse	Landm. rmse	Meta. rmse	Landm. r.e.	Meta. r.e.
car.data	bezdekIris.data	tic-tac-toe.data	0.06133	0.27690	0.16128	1398/3520	204 /3520
lenses.data	o-ring-erosion-or-blowby.data	o-ring-erosion-or-blowby.data	0.21300	0.21300	0.21300	0/3560	0/3560
adult+stretch.data	yellow-small.data	adult-stretch.data	0.44472	x	0.44472	3554/3554	152 /3554
german.data	vowel-context.data	vowel-context.data	0.03165	0.03165	0.03165	0/3560	0/3560
horse-colic.data	reprocessed.hungarian.data	reprocessed.hungarian.data	0.05793	0.05793	0.05793	0/3560	0/3560
flare.data1	reprocessed.hungarian.data	ecoli.data	0.07280	0.07280	0.07280	0/3560	0/3560
long-beach-va.data	yeast.data	train5.data	0.02625	0.02625	0.02625	0/3560	0/3560
auto-mpg.data-original	yacht_hydrodynamics.data	kinship.data	0.05662	0.05662	0.05662	0/3558	0/3558
bridges.data.version2	flag.data	post-operative.data	0.26162	0.27725	0.32136	74 /3549	190/3549
kr-vs-kp.data	car.data	anneal.data	0.06359	0.12255	0.07030	133/3356	6 /3356
hungarian.data	long-beach-va.data	train2.data	0.02312	0.02312	0.02312	0/3560	0/3560
iris.data	bezdekIris.data	bezdekIris.data	0.06689	0.06689	0.06689	0/3542	0/3542
adult-stretch.data	adult+stretch.data	adult+stretch.data	0.45640	0.45640	0.45640	55/3554	55/3554
kinship.data	o-ring-erosion-or-blowby.data	auto-mpg.data-original	0.20029	0.20029	0.20029	0/3555	0/3555
bezdekIris.data	iris.data	iris.data	0.06689	0.06689	0.06689	0/3542	0/3542
housing.data	test1.data	synthetic_control.data	0.04454	0.04454	0.04454	0/3560	0/3560
house-votes-84.data	post-operative.data	crx.data	0.34425	0.34763	0.39019	75 /3547	172/3547
anneal.data	audiology.standardized.data	ionosphere.data	0.13579	0.13775	0.26300	1 /3513	436/3513
crx.data	house-votes-84.data	house-votes-84.data	0.32073	0.33280	0.33280	111/3493	111/3493
flag.data	bridges.data.version2	ionosphere.data	0.26284	0.28257	0.39974	127 /3538	3024/3538
agaricus-lepiota.data	o-ring-erosion-or-blowby.data	sensor_readings_24.data	0.22288	0.22544	0.22332	180/3356	61 /3356
audiology.standardized.data	anneal.data	sponge.data	0.11255	0.11258	0.15942	26 /3542	803/3542
ecoli.data	auto-mpg.data-original	flare.data1	0.05472	0.05472	0.05472	0/3560	0/3560
ionosphere.data	flag.data	soybean-small.data	0.22645	0.34620	0.29005	590/3524	323 /3524
o-ring-erosion-or-blowby.data	o-ring-erosion-only.data	o-ring-erosion-only.data	0.21797	0.21797	0.21797	0/3560	0/3560
o-ring-erosion-only.data	o-ring-erosion-or-blowby.data	o-ring-erosion-or-blowby.data	0.21797	0.21797	0.21797	0/3560	0/3560
reprocessed.hungarian.data	horse-colic.data	horse-colic.data	0.05852	0.05852	0.05852	0/3560	0/3560
post-operative.data	house-votes-84.data	bridges.data.version2	0.33141	0.33193	0.33604	2016 /3553	2050/3553
nursery.data	kr-vs-kp.data	agaricus-lepiota.data	0.01756	0.09674	0.09102	207/3356	195 /3356
sponge.data	lenses.data	anneal.data	0.13245	0.22435	0.18497	960/3552	797 /3552
test4.data	test2.data	test5.data	0.02722	0.02722	0.02722	0/3560	0/3560
test2.data	test5.data	test3.data	0.02717	0.02717	0.02717	0/3560	0/3560
test3.data	test4.data	test5.data	0.02729	0.02729	0.02729	0/3560	0/3560
soybean-small.data	flare.data1	ionosphere.data	0.10314	0.26976	0.10314	202/3551	29 /3551
sensor_readings_24.data	kr-vs-kp.data	agaricus-lepiota.data	0.03828	0.03946	0.03946	2 /3356	8/3356
sensor_readings_4.data	sensor_readings_2.data	sensor_readings_2.data	0.02324	0.02324	0.02324	0/3356	0/3356
test1.data	housing.data	yeast.data	0.04320	0.04320	0.04320	0/3560	0/3560
synthetic_control.data	test1.data	housing.data	0.04089	0.04089	0.04089	0/3560	0/3560
sensor_readings_2.data	sensor_readings_4.data	sensor_readings_4.data	0.02324	0.02324	0.02324	0/3356	0/3356
switzerland.data	vowel-context.data	german.data	0.03337	0.03337	0.03337	0/3560	0/3560

train2.data	yeast.data	hungarian.data	0.02660	0.02660	0.02660	0/3560	0/3560
train6.data	train4.data	train4.data	0.03008	0.03008	0.03008	0/3560	0/3560
test5.data	test2.data	test3.data	0.02714	0.02714	0.02714	0/3560	0/3560
train3.data	train4.data	train4.data	0.02980	0.02980	0.02980	0/3560	0/3560
vowel-context.data	german.data	german.data	0.03181	0.03181	0.03181	0/3560	0/3560
train5.data	train6.data	train6.data	0.03058	0.03058	0.03058	0/3560	0/3560
tic-tac-toe.data	flag.data	post-operative.data	0.32495	0.23145	0.43240	20 /3493	391/3493
train4.data	train6.data	train6.data	0.03004	0.03004	0.03004	0/3560	0/3560
yellow-small+adult-stretch.data	crx.data	adult+stretch.data	0.42572	0.50206	0.66234	118 /3554	3120/3554
yellow-small.data	adult+stretch.data	adult+stretch.data	0.25234	0.45052	0.45052	154/3554	154/3554
yacht_hydrodynamics.data	auto-mpg.data-original	horse-colic.data	0.05717	0.05717	0.05717	0/3560	0/3560
yeast.data	long-beach-v4.data	test1.data	0.02617	0.02617	0.02617	0/3560	0/3560
						10003 /183484	12281 /183484

Table 3.1: Performance comparison of Meta-learning algorithm presented by Kazík with our custom Landmarking algorithm.

It may be observed that the total number of Gridsearch results are different for each dataset. This is because the data have different character and for some hyperparameter settings it was not possible to analyze the dataset. This is why for dataset *adult+stretch.data* our algorithm has value x in the RMSE field. Our algorithm performed bad for this dataset and recommended a model, which could not be applied to this dataset. Therefore we set the error for that recommendation to maximum. In 18 cases both algorithms identified as the most similar same dataset so the recommended model was the same and so were the results. In 17 cases the algorithms identified different datasets as most similar, but the recommended models performed equally well. Different datasets were identified as most similar in remaining 17 cases. That table rows are highlighted. In these cases also the recommended models performed differently. Our approach overcame the statistical approach in 9 cases and it was worse in 8 ones. Our presented algorithm thus successfully competed with Kazík's algorithm. Moreover, if we look at the comparison of their general errors, the accuracy of our algorithm was nearly by 20% higher with general error of $\frac{10003}{183484}$ compared to $\frac{12281}{183484}$. With minimal increase in computational time, we were able to improve the accuracy of the recommendations.

Next we compared these two approaches with a hybrid algorithm which combines both, statistical and Landmarking features for recommendation. This combined algorithm performed best with these weight settings

- number of instances: 0.5
- Logistic regression: 0.3
- k-nearest neighbor: 0.1
- data type: 0.1
- Decision tree: 0.0
- SVM: 0.0
- number of attributes: 0.0
- number of missing values: 0.0 .

The data above shows, that the greatest emphasis is on the number of instances and a Logistic regression model. Also k-nearest neighbor algorithm and data type are

considered. On the other hand, Decision tree and SVM are excluded which reduces the overall computation time of recommendation, because these two methods may not be considered.

Table 3.2 shows the results of this experiment compared to previous two methods. First column contains analyzed dataset as in the previous table. Next three columns contain the closest dataset according to Landmarking, statistical and combined approach, respectively. Last three columns represent the recommendation error of these three models.

In most cases, the hybrid model chose the same dataset as one of first two methods. From 17 datasets where were differences in recommendations between models, the combined approach chose 8 times better or equal model than other two methods and 9 times worse one. However, this model performed better in general. Its general error is just $\frac{6829}{183484}$. This is more than 30% accuracy increase compared to Landmarking approach and almost 45% compared to pure statistical approach. Moreover, the computational complexity of this model with adduced weight settings is about the half compared to pure Landmarking approach. In most cases, all three recommendation models calculated results in a few seconds depending on dataset size.

The experiments have shown, that our design of recommendation algorithm for solving CASH problem is competitive to other existing solutions. Moreover the hybrid approach outperformed them by almost 45% in accuracy while remaining the computational time on the acceptable level.

Dataset	Landmarking	Metadata	Combined	Landm. r.e.	Meta. r.e.	Comb. r.e.
car.data	bezdekIris.data	tic-tac-toe.data	tic-tac-toe.data	1398/3520	204 /3520	204 /3520
lenses.data	o-ring-erosion-or-blowby.data	o-ring-erosion-or-blowby.data	o-ring-erosion-or-blowby.data	0/3560	0/3560	0/3560
adult+stretch.data	yellow-small.data	adult-stretch.data	adult-stretch.data	3554/3554	152 /3554	152 /3554
german.data	vowel-context.data	vowel-context.data	vowel-context.data	0/3560	0/3560	0/3560
horse-colic.data	reprocessed.hungarian.data	reprocessed.hungarian.data	reprocessed.hungarian.data	0/3560	0/3560	0/3560
flare.data1	reprocessed.hungarian.data	ecoli.data	reprocessed.hungarian.data	0/3560	0/3560	0/3560
long-beach-va.data	yeast.data	train5.data	train5.data	0/3560	0/3560	0/3560
auto-mpg.data-original	yacht_hydrodynamics.data	kinship.data	ecoli.data	0/3558	0/3558	0/3558
bridges.data.version2	flag.data	post-operative.data	flag.data	74 /3549	190/3549	74 /3549
kr-vs-kp.data	car.data	anneal.data	car.data	133/3356	6 /3356	133/3356
hungarian.data	long-beach-va.data	train2.data	train2.data	0/3560	0/3560	0/3560
iris.data	bezdekIris.data	bezdekIris.data	bezdekIris.data	0/3542	0/3542	0/3542
adult-stretch.data	adult+stretch.data	adult+stretch.data	adult+stretch.data	55/3554	55/3554	55/3554
kinship.data	o-ring-erosion-or-blowby.data	auto-mpg.data-original	o-ring-erosion-or-blowby.data	0/3555	0/3555	0/3555
bezdekIris.data	iris.data	iris.data	iris.data	0/3542	0/3542	0/3542
housing.data	test1.data	synthetic_control.data	synthetic_control.data	0/3560	0/3560	0/3560
house-votes-84.data	post-operative.data	crx.data	crx.data	75 /3547	172/3547	172/3547
anneal.data	audiology.standardized.data	ionosphere.data	tic-tac-toe.data	1 /3513	436/3513	3/3513
crx.data	house-votes-84.data	house-votes-84.data	house-votes-84.data	111/3493	111/3493	111/3493
flag.data	bridges.data.version2	ionosphere.data	bridges.data.version2	127 /3538	3024/3538	127 /3538
agaricus-lepiota.data	o-ring-erosion-or-blowby.data	sensor_readings_24.data	sensor_readings_24.data	180/3356	61 /3356	61 /3356
audiology.standardized.data	anneal.data	sponge.data	iris.data	26 /3542	803/3542	754/3542
ecoli.data	auto-mpg.data-original	flare.data1	yacht_hydrodynamics.data	0/3560	0/3560	0/3560
ionosphere.data	flag.data	soybean-small.data	flag.data	590/3524	323 /3524	590/3524
o-ring-erosion-or-blowby.data	o-ring-erosion-only.data	o-ring-erosion-only.data	o-ring-erosion-only.data	0/3560	0/3560	0/3560
o-ring-erosion-only.data	o-ring-erosion-or-blowby.data	o-ring-erosion-or-blowby.data	o-ring-erosion-or-blowby.data	0/3560	0/3560	0/3560
reprocessed.hungarian.data	horse-colic.data	horse-colic.data	horse-colic.data	0/3560	0/3560	0/3560
post-operative.data	house-votes-84.data	bridges.data.version2	house-votes-84.data	2016 /3553	2050/3553	2016 /3553
nursery.data	kr-vs-kp.data	agaricus-lepiota.data	agaricus-lepiota.data	207/3356	195 /3356	195 /3356
sponge.data	lenses.data	anneal.data	lenses.data	960/3552	797 /3552	960/3552
test4.data	test2.data	test5.data	test5.data	0/3560	0/3560	0/3560
test2.data	test5.data	test3.data	test3.data	0/3560	0/3560	0/3560
test3.data	test4.data	test5.data	test5.data	0/3560	0/3560	0/3560
soybean-small.data	flare.data1	ionosphere.data	reprocessed.hungarian.data	202/3551	29 /3551	340/3551
sensor_readings_24.data	kr-vs-kp.data	agaricus-lepiota.data	sensor_readings_4.data	2 /3356	8/3356	594/3356
sensor_readings_4.data	sensor_readings_2.data	sensor_readings_2.data	sensor_readings_2.data	0/3356	0/3356	0/3356
test1.data	housing.data	yeast.data	yeast.data	0/3560	0/3560	0/3560
synthetic_control.data	test1.data	housing.data	housing.data	0/3560	0/3560	0/3560
sensor_readings_2.data	sensor_readings_4.data	sensor_readings_4.data	sensor_readings_4.data	0/3356	0/3356	0/3356
switzerland.data	vowel-context.data	german.data	german.data	0/3560	0/3560	0/3560

train2.data	yeast.data	hungarian.data	hungarian.data	0/3560	0/3560	0/3560
train6.data	train4.data	train4.data	train4.data	0/3560	0/3560	0/3560
test5.data	test2.data	test3.data	test3.data	0/3560	0/3560	0/3560
train3.data	train4.data	train4.data	train4.data	0/3560	0/3560	0/3560
vowel-context.data	german.data	german.data	german.data	0/3560	0/3560	0/3560
train5.data	train6.data	train6.data	train6.data	0/3560	0/3560	0/3560
tic-tac-toe.data	flag.data	post-operative.data	anneal.data	20 /3493	391/3493	83/3493
train4.data	train6.data	train6.data	train6.data	0/3560	0/3560	0/3560
yellow-small+adult-stretch.data	crx.data	adult+stretch.data	post-operative.data	118/3554	3120/3554	31 /3554
yellow-small.data	adult+stretch.data	adult+stretch.data	adult-stretch.data	154/3554	154/3554	154/3554
yacht_hydrodynamics.data	auto-mpg.data-original	horse-colic.data	horse-colic.data	0/3560	0/3560	0/3560
yeast.data	long-beach-va.data	test1.data	test1.data	0/3560	0/3560	0/3560
				10003 /183484	12281 /183484	6829 /183484

Table 3.2: Performance comparison of Meta-learning algorithm presented by Kazík, our custom Landmarking algorithm and a hybrid algorithm, which combines both Meta-learning approaches.

Chapter 4

Design and implementation

Next we present the general overview of our project and its functionality. Our goal is to design a prototype system for managing data mining projects that is simple and user-friendly even for people, who are not familiar with such methods and data mining terminology. In data-mining, a CRISP-DM methodology is widely considered as a standard for data analysis. In our system, we will stick to this methodology so we briefly describe it.

4.1 The CRISP-DM methodology

Cross Industry Standard Process for Data Mining (CRISP-DM) is a standard methodology used by data mining experts. This methodology describes steps of solving data-mining tasks. These are

- **Business understanding**

This is the initial phase of the whole process. It focuses on understanding the objectives and requirements of the project from a business perspective. These knowledge is then transformed to data mining problems and objectives.

- **Data understanding**

This phase deals with data collection and understanding of the data with respect to domain knowledge.

- **Data preparation**

Data preparation is closely related to data understanding. It covers activities

such as attribute selection, transformations, cleaning data, ... The goal is to construct final dataset from initial data.

- **Modelling**

In modelling phase appropriate data-mining techniques and methods are selected and applied to dataset. Various parameters of these techniques are adjusted to suit the particular dataset.

- **Evaluation**

This step evaluates the quality of model from business objectives. Before deployment is necessary to review the whole process and determine next steps.

- **Deployment**

The knowledge gained during the process is presented to customer in a way that the customer can understand and work with it.

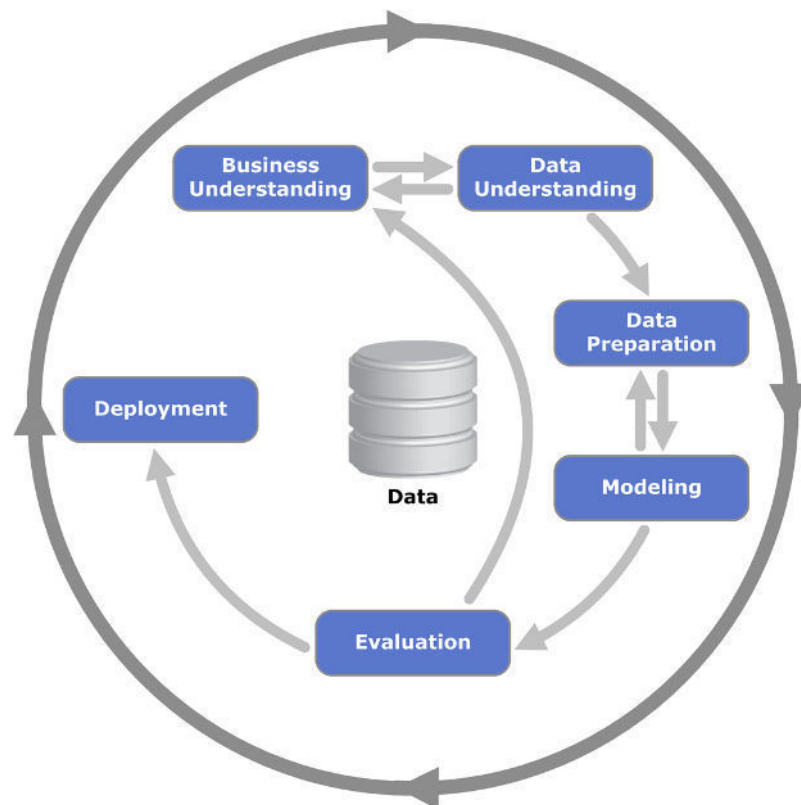


Figure 4.8: Relationship between the individual steps of the CRISP-DM methodology [18].

As Figure 4.8 shows, tasks are not independent. They are tightly linked together. Even the sequence of the phases is not strict and moving back and forth between different phases is always required [18].

4.2 Project design

Our system should guide a user through every step of CRISP-DM process in a way that is simple to understand.

The goal of business and data understanding phase is to determine the background of the data. This is the most challenging phase, because it requires a close cooperation with user. Firstly we need to know the user objectives. Whether he wants to perform classification, regression, pattern mining, ... This can be achieved by simple conversational system like one we designed for data preprocessing phase or by explaining to user what are the objectives of these methods and simply ask him what he wants to do. Since our project is just a prototype we focused only on classification problems. However, the project is implemented in a way which allows easy addition of models in future. Also, there is a problem of understanding data domain, because next phases are dependent on this knowledge. In order to preserve our system as general as possible, we could not specialize only on one data domain (e.g. medical data). Because each domain is different and there are almost infinitely many of them, we could not use a conversational system with predefined questions to determine domain details. We omitted the step of determining the domain from user and proposed a solution, which is more convenient to the user.

Data preparation step is crucial for model selection, its performance and thus the overall result. The goal of this process is to use the knowledge gained in previous phases, determine the best data-mining model and prepare data for it. Since we do not have any specific domain knowledge about data and also no data-mining model specified so far, we designed a simple conversational system. This system can generate questions based on user's dataset characteristics. As an example we implemented a system, which analyzes a number of different values in data attributes and when an attribute has less than 5% of such values (considering total number of instances), our assumption is that this attribute data are independent (without ordering). These values can represent some code or flag and therefore they should be binarized. To confirm our assumption, the system will generate a question to determine, if an attribute should be binarized or not. Answer can be either yes or no. If user selects yes, then the attribute is binarized, in the other case not.

An example of such generated form is shown in Figure 4.9. Our system determined, that three attributes have just few different values and could be binarized, so our tool generated questions for user to confirm the assumption. User selected one of these

attributes for binarization, so this attribute will be binarized and the other two will not.

Attribute **year** has only 12 different values. Are they independent and do not have any order?

☐ yes

☒ no

Attribute **age** has only 19 different values. Are they independent and do not have any order?

☐ yes

☒ no

Attribute **marst** has only 6 different values. Are they independent and do not have any order?

☒ yes

☐ no

✓ Finish

Figure 4.9: An example of form generated by a conversational system for determining whether an attribute should be binarized.

Data are preprocessed as generally as possible to cover wide variety of possible models in next step. First of all the missing values are handled. For discrete attribute, the missing values are replaced with major value of that attribute. Numeric attributes has two options for handling missing values. First one replaces the values with major value of attribute. The second one replaces missing value with attribute mean. So if dataset contains a numeric attribute with missing values, two different datasets are created. One with values replaced with mean and one with major value. Naturally, other attributes are copied. Missing values in boolean attribute are replaced just with its major value.

Datasets generated during the process of removing the missing values continue to transformation phase. In this phase, the attributes marked for binarization are binarized. These attributes remain untouched for the rest of this process. Numeric attributes are normalized and discretized. For every numeric attribute, the dataset is multiplied four times. On first copy, the z-score normalization is applied to given attribute. Second copy uses the zero-one normalization for a given attribute. Last two datasets use equidistant and equipotent discretization. We also remain the original dataset as a case when the attribute should not be preprocessed. That means, that

for each numeric attribute, five new copies are created and then the preprocessing continue for next attribute in dataset.

Lets imagine that we have a dataset about cars with these attributes

- *number of doors* – numeric
- *drive* – discrete {gasoline, diesel, lpg}
- *trunk volume* – numeric
- *max speed* – numeric
- *engine volume* – numeric (with missing values)
- *buy* – boolean {true, false}

Lets say, that our conversational system asked user about the *drive* attribute and he confirmed, that it should be binarized. The preprocessing phase will look like this. From the original dataset, two new files are created in phase of handling the missing values, because there are two methods of replacing the missing values in numeric attribute (*engine volume*). Next, for each of these two datasets, the numeric attributes are preprocessed (four new datasets are created for each attribute + one original dataset). That sums up to $2 \times 5^4 = 1250$ different datasets. At the end of this phase, attribute *drive* is binarized in each dataset. These preprocessed datasets continue to the modelling phase. Having so many differently preprocessed datasets will help us to choose the right model without close user interaction, which user is not willing to make.

The main problem in modelling phase is to choose the correct model for user data in a short time. This is especially hard if we have so many datasets to process from the previous phase. We note that we do not have any knowledge about dataset which could help us to determine the right model. Normally in such situation a Gridsearch algorithm is used to solve it. This is unacceptable in our situation, because Gridsearch algorithm is very time consuming even for one dataset and we need to process hundreds of them.

Therefore we designed and implemented custom recommendation algorithm which solves this problem partially. Based on experiments we decided to use a hybrid recommendation algorithm presented in Chapter 3 which uses both, statistical and Land-marking features for model recommendation. This algorithm is fast enough to recommend a method for one dataset in a few seconds. However, it is still too slow for

hundreds of datasets. On the other side, our user would be confused if we show him thousands of results for just one analysis. The results would be difficult to understand and interpret for him. So we only show to the user top 5 data files generated from his original dataset based on their performance.

Once the analysis process starts, our algorithm runs four data-mining algorithms (described in Chapter 2) with default hyperparameters (results are used for our recommendation algorithm) to determine performance on original dataset. Next, the recommendation is made for this dataset and recommended algorithm is applied to data. These five results (four default and one recommended) are presented to the user. While the user examines the results, same process is applied to other datafiles which were generated from user's dataset. The results are available to the user immediately after elaboration on per datafile basis. The user has results available almost in real time. The more accurate results accure later.

During the evaluation phase a user can examine the results for variously preprocessed data files. If he recognizes that some file performed well during the initial analysis, he can download particular file and look how it was generated during preprocessing phase. Also he can start precise analysis for this file using Gridsearch algorithm. Because it may take some time to get results, the user can close session and he will be notified by an email once it will be done. On the other side when he recognizes that no analysis was good enough, he can repeat the process again with adjusted original dataset or just answering differently on questions at our conversational system in preprocessing phase.

Our system is tied to the CRISP-DM methodology as we described above. Now we describe a typical use case of our system to better explain the system as a whole. Next, we describe its architecture and implementation details.

4.3 Use case

This section describes a typical use case of our system. We note that it is a web application. It is compatible with all major web browsers (Firefox 36 and later, Chrome 41 and later, Safari 8 and later, Internet Explorer 11 and later).

When user opens our web application for the first time, he will see a simple welcome screen as shown on Figure 4.10. There is a big green call to action button in the center of it. When the user is logged in, this button redirects him to a dashboard. Otherwise he will be redirected to the login screen (Figure 4.11), where he has to provide his

username and password.

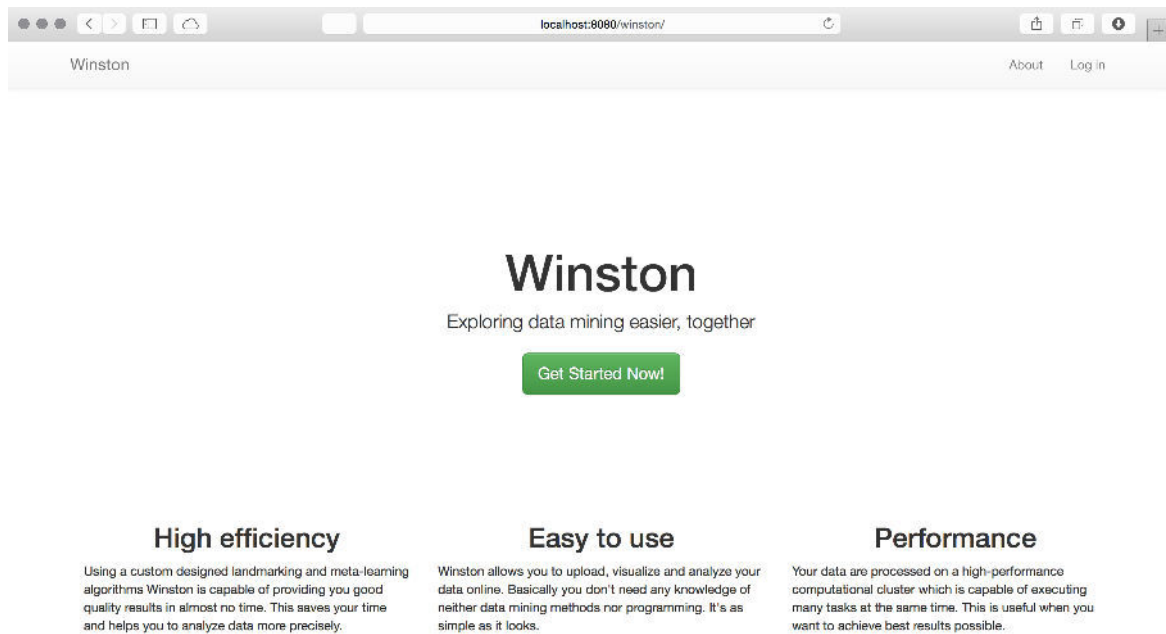


Figure 4.10: The welcome screen of our prototype called Winston.

If user is not registered, he can do so by clicking on the registration button. He has to provide his email, username and password.

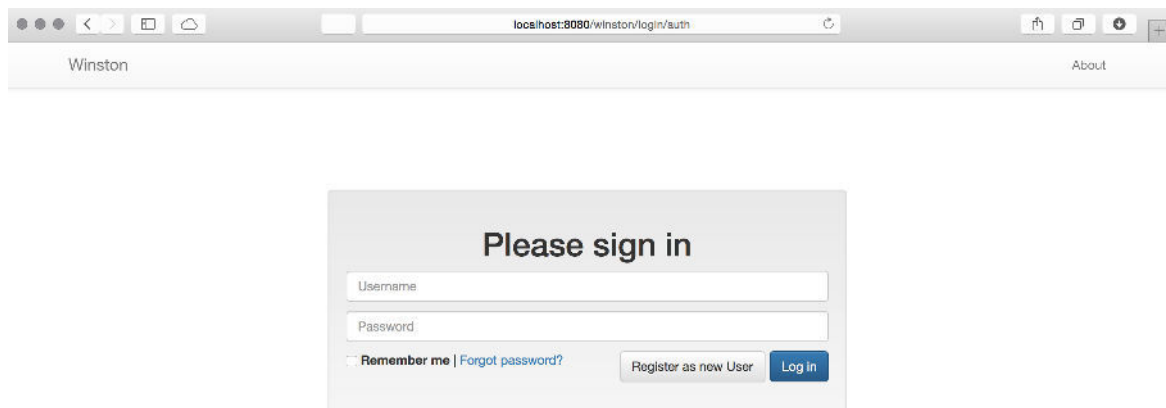
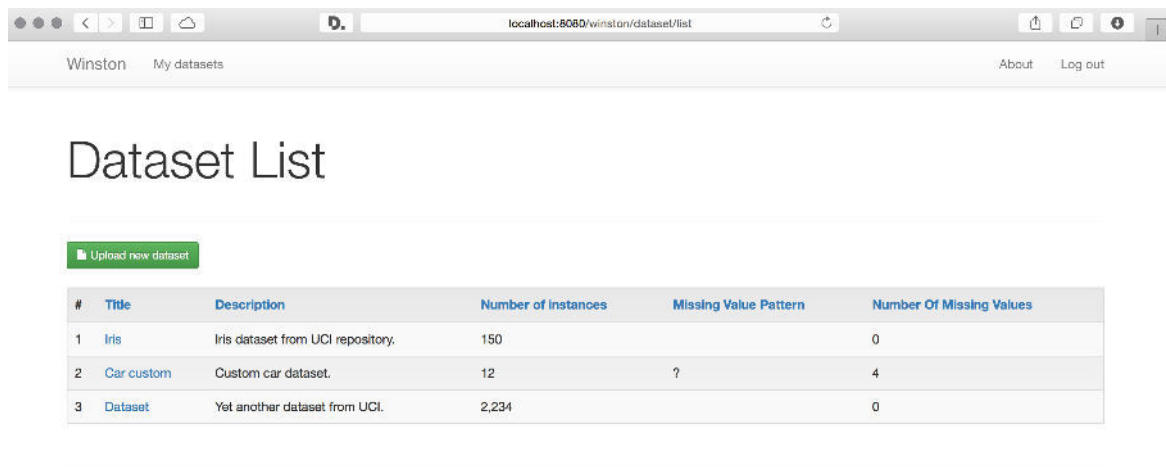


Figure 4.11: Login screen for user authentication. Also user registration is accessible from here.

A confirmation email will be sent to confirm email address. After this step the registration is complete. Login screen also contains a button for users who forgot their passwords. After successful login, user is redirected to his dashboard, where he can see an overview of his datasets. Dashboard is also always accessible from navigation menu at the top. Datasets are not shared between users, so each user can see only datasets which he uploaded. On the other side, to exploit the knowledge from new analyzes, the metadata are shared on the background. That is, if one dataset is analyzed in detail by Gridsearch algorithm, it is included into background knowledge for future recommendations for other datasets by our recommendation algorithm.



#	Title	Description	Number of instances	Missing Value Pattern	Number Of Missing Values
1	Iris	Iris dataset from UCI repository.	150		0
2	Car custom	Custom car dataset.	12	?	4
3	Dataset	Yet another dataset from UCI.	2,234		0

Figure 4.12: Dashboard with datasets, their description and statistical information.

Also, there is an option to upload new dataset in a dashboard. This form is shown on Figure 4.13. User fills the information in a form and saves the data by clicking to *Upload* button. Dataset title and description are not required by our recommendation system nor any other part of our prototype. They serve just for the user to write some notes about the data to find and understand later more easily.

Using the file picker, user can choose and upload the data file which he wants to analyze. Currently, we support data in **.csv** format. Each line of this file contains one instance and attributes are separated by comma (,) sign. First line of this file contains the attribute names also separated by comma. User can also fill in the mark for missing values, if a dataset contains missing values, which are specially marked.

This field can be blank. In that case, dataset is considered to be without missing values.

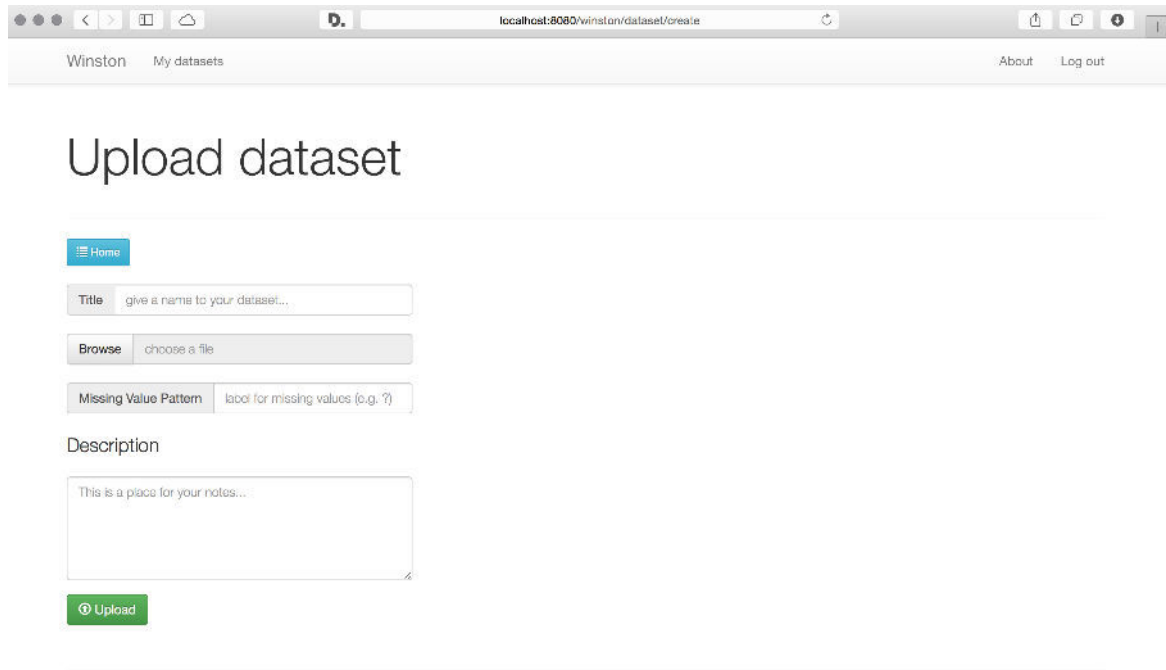


Figure 4.13: Form for uploading a new dataset.

Immediately after the form is submitted, the system quickly process the dataset. If there is an error while parsing the file, upload process is canceled and the user is informed, that the file has wrong format. Otherwise, dataset file is sent to backend server and early statistical analysis is performed. Number of instances, number of attributes and number of missing values are counted. Also, type of attributes (*numeric*, *categorical* or *boolean*) is determined. For categorical attributes, number of distinct values is calculated. For numeric attribute, statistical characteristics like minimum, maximum and average is calculated and also distinct values are counted. For boolean attribute, number of false and true values is counted.

After submitting the form user is redirected to an overview of his dataset, where all the information mentioned above is presented in a user-friendly way. One such screen is shown on Figure 4.14. Moreover, if there are analyzes for this dataset, they are shown at the bottom of this screen. It would not make any sense to present thousands of analyzes for variously preprocessed data file. Therefore only top five analyzes (based on root mean squared error) are shown.

After uploading and inicial processing of a dataset there are no analyzes for this dataset. User will create one by clicking the *Analyze* button. This process consists of

two steps.

The screenshot shows a web application interface for Winston. The browser address bar shows 'localhost:8080/winston/dataset/show/25'. The page has a header with 'Winston' and 'My datasets' on the left, and 'About' and 'Log out' on the right. The main content area is titled 'Iris'. Below the title, there are four buttons: 'Home', 'Analyze', 'Edit', and 'Delete'. The dataset details are as follows:

- Title: Iris
- Data File: iris.csv
- Number Of Instances: 150
- Description: Iris dataset from UCI repository.

Below the details, there is a section titled 'Attributes' with a table:

#	Title	Type
1	sepal length in cm	numeric
2	sepal width in cm	numeric
3	petal length in cm	numeric
4	petal width in cm	numeric
5	class	categorical

Below the attributes, there is a section titled 'Top analyzes' with a table:

#	Data File	Number Of Attributes	Best rmse
1	Iris-analysis_1058.arff	5	0.115973054487532
2	Iris-analysis_1014.arff	5	0.115973054487532
3	Iris-analysis_1050.arff	5	0.126970225698421
4	Iris-analysis_1030.arff	5	0.133129641241098
5	Iris-analysis_1062.arff	5	0.146936258710236

Figure 4.14: An overview of dataset, its attributes and list of performed analyzes.

Firstly, user has to choose target attribute for analysis. This is the attribute, for which the prediction will be made. Since he does not have any knowledge about data-mining, he would not understand what information we need from him. Therefore, we provided a quick description of a data-mining task, which will be performed on his data. User should be capable of choosing the right attribute for classification after reading these instructions. A sample screen of this step for Iris dataset is shown on Figure 4.15. When the user chooses target attribute, our system analyzes other

attributes. If there are some attributes, for which our conversational system generated questions, the user continue to the second step.

localhost:8080/winston/analysis/create/25

Winston My datasets About Log out

Analyze Iris

[< Back](#)

A method, which will analyze your data is called **classification**. It solves the problem of **assigning** a new object to a **certain class** based on its similarity to previous examples of other objects. For example, a classification model could be used to identify loan applicants as low, medium, or high credit risks based on observed data for many loan applicants over a period of time.

Before we analyze your data, you have to **choose an attribute** which represents an **outcome** for you. This target attribute has to be discrete and do not imply order. Continuous, floating-point values would indicate a numerical, rather than a categorical, target. A predictive model with a numerical target uses a regression method, not a classification method.

Choose a target attribute:

☐ sepal length in cm

☐ sepal width in cm

☐ petal length in cm

☐ petal width in cm

☒ class

[Continue >](#)

Figure 4.15: Target attribute selection screen for dataset analysis.

In second phase of analysis, user has to answer the questions about his data. These questions are dynamically generated based on attribute characteristics. An example is presented on Figure 4.16. The page includes a simple description of the importance of this step, which is related to preprocessing. The questions are formulated in a way, which should be clear even for non-technical user. When user answers the questions and submits a form, the analysis process starts.

Dataset information and user input are sent to backend server for processing. Based on that input, the original dataset file is preprocessed in many different ways. Result datasets are saved in Attribute-Relation File Format (ARFF), which is standard format for data representation in data-mining software like Weka. It is similar to CSV format and algorithms exist for transforming file from one format to another. We used this format at backend just for compatibility with data-mining libraries.

Firstly, our recommendation algorithm runs four basic models to determine the

most similar dataset from background knowledge and then a new model is recommended and applied. These five results for original dataset are saved to database and then the user is redirected to dataset overview where he can view these results. Meanwhile, the recommendation process is applied to all the data files from preprocessing step. The results are saved in database periodically, so if the user refreshes page, he can see new results immediately as they are computed on the backend. The whole process can take several minutes or hours, so user has to be notified when it finish.

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/winston/analysis/index'. The page has a header with 'Winston' and 'My datasets' on the left, and 'About' and 'Log out' on the right. The main content area has a large heading 'Analyze Dataset'. Below the heading is a blue button labeled '< Back'. The text explains that every analysis is preceded by a step called 'pre-processing' and lists various data cleaning steps. It then asks the user to help pre-process data to be 'more accurate'. Three questions follow, each with 'yes' and 'no' radio button options: 1. 'Attribute **year** has only 12 different values. Are they independent and do not have any order?' 2. 'Attribute **age** has only 19 different values. Are they independent and do not have any order?' 3. 'Attribute **marst** has only 6 different values. Are they independent and do not have any order?'. At the bottom is a green button labeled '✓ Finish'.

Winston My datasets About Log out

Analyze Dataset

< Back

Every analysis is preceded by a step called [pre-processing](#). Data goes through a series of steps during this phase. They are cleansed through processes such as replacing missing values or smoothing the noisy data. They can be normalized, aggregated, generalized, reduced or discretized to better fit into computation models.

Answering the questions below, you will help us to pre-process your data **more accurate** which leads to better results. If you are not sure about the answers, you can run the analysis again with different options selected.

Attribute **year** has only 12 different values. Are they independent and do not have any order?

☐ yes

☐ no

Attribute **age** has only 19 different values. Are they independent and do not have any order?

☐ yes

☐ no

Attribute **marst** has only 6 different values. Are they independent and do not have any order?

☐ yes

☐ no

✓ Finish

Figure 4.16: Subsidiary questions about user's data generated by conversational system.

Therefore, we will send him an email from backend server with a notification

that the analysis finished. User can view analyzes from dataset overview page. An analysis corresponds to a single file created in preprocessing phase. Multiple analyzes are created from one analysis process from the user point of view. On each analysis, multiple models are trained and evaluated. An example of such analysis is shown below.

The screenshot shows a web browser window with the URL `localhost:8080/winston/analysis/show/627`. The page title is "Car custom analysis". At the top, there are navigation links: "Winston", "My datasets", "About", and "Log out". Below the title, there are two buttons: "< Back" and "Download analysis file". The dataset information is displayed as follows:

- Dataset: [Car custom](#)
- Data File: Car custom-analysis_2.arff
- Data Type: Multivariate
- Number Of Attributes: 4

Below this, it says "Top results for this analysis by [rmse](#):" followed by a table:

Position	Method	Rmse
1.	Logistic regression	0.496
2.	Decision Tree	0.523
3.	k-NN	0.559
4.	Support vector machines	0.764

Below the table, there is a text block: "Do you consider this analysis good? Run [grid search](#) algorithm to obtain even better results. This algorithm simply tries all possible methods to describe your data and chooses the best one. Keep in mind that it can take a lot of time." At the bottom of this section is a button: "Run Grid Search".

Figure 4.17: Analysis detail with performance on default models.

Figure 4.17 shows one particular analysis for dataset *Car custom*. On this page, user can see the performance of this file on several models. Also, he can download this file (in ARFF format) and see how it was preprocessed. If he considers, that the initial results are good, he can start the deep analysis of this file using the Gridsearch algorithm. This algorithm starts on our backend server. User will be notified when the

results are computed by an email as in the previous case. The best results will be shown in a table on page with analysis overview.

User can also further examine the details about evaluated model. Model performance page (Figure 4.18) contains type of data-mining method, its hyperparameter settings and performance details such root mean squared error or mean absolute error. For non-technical users who do not understand these terms, there are indexes like number of correctly and incorrectly classified instances. For advanced users who are used to work with data-mining software like Weka, there is a complete output summary formatted in the same way like in Weka software.

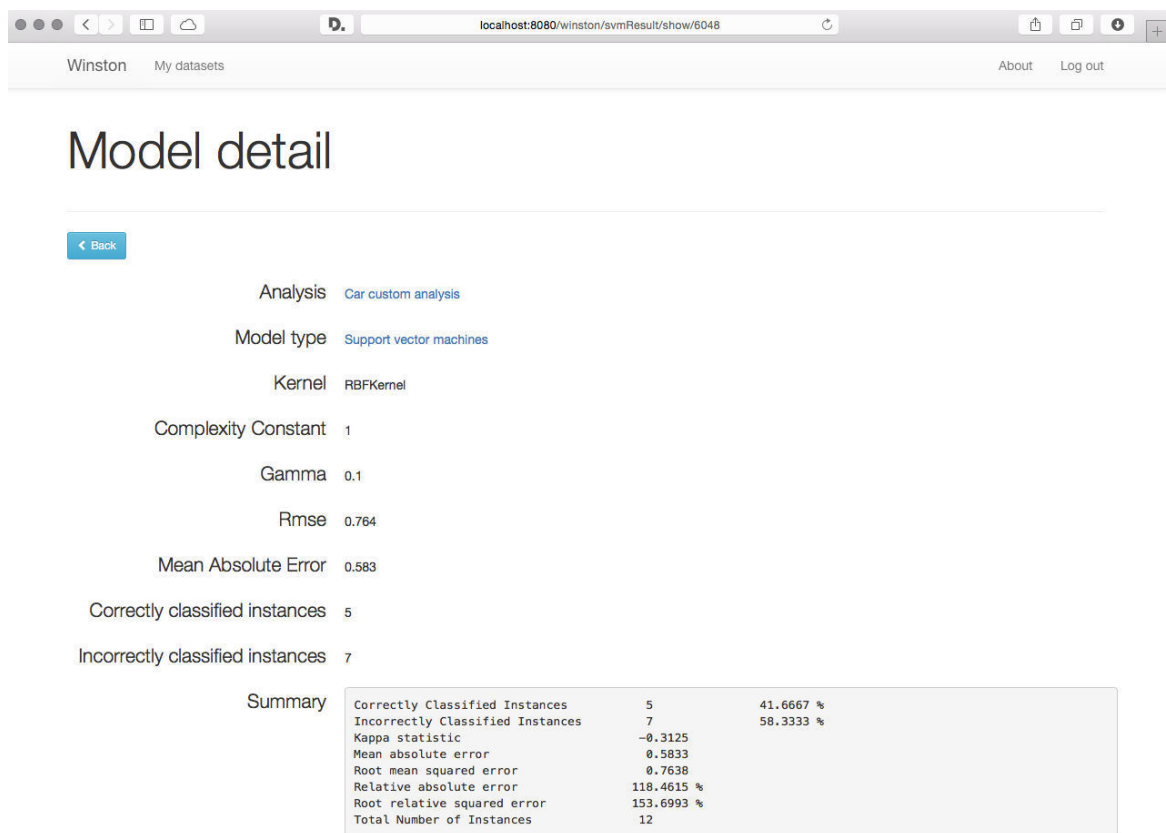


Figure 4.18: Model performance overview for a given method, hyperparameter settings and analysis.

As shown above, the user interface is simple and easy to use for everyone. Each page contains call to action element, which guides a user through all steps of data-mining process.

4.4 Implementation

In this section we provide a deep overview of system architecture and technology used. The whole system consists of three main components: web server, backend computational cluster and a database server. A general overview is described on Figure 4.19. Web server serves the user requests. For simple requests when the user wants just to display a data it connects directly to database server and load required data. Complex requests, which include data processing and modelling are sent to the computational cluster at the backend. They are processed on this cluster, data are saved into a database and the backend notifies a webserver, that it can load required data from database.

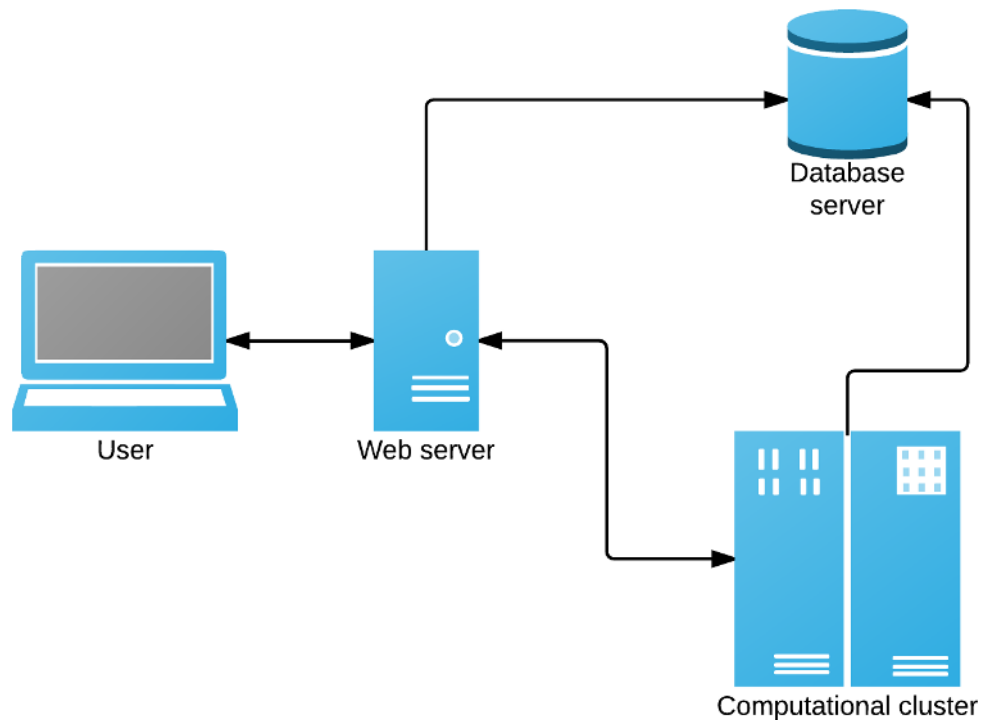


Figure 4.19: System architecture design.

This design allows us to isolate data, computational and user interface (UI) parts. This is due to the fact, that our system should be extendible. It will include more data-mining tasks and methods in the future and also UI will evolve. Using this design one can modify just one specific component (e.g. change database technology) without affecting the rest of the system. Also deployment and maintenance is easier, because the system can be distributed on several physical machines to distribute the

load. Simple tasks can run on slow webserver and complex requests are processed on high performance machine.

Next, we describe single components of our system from an implementation point of view.

4.4.1 Web application

We had a number of requirements on technology used for our web application framework. The main requirement was that it should support quick and easy prototyping, because in the beginnings of the project the whole idea was just forming and it evolved during the time. The other requirement was, that it should support a large variety of libraries, which could be used in our system. We were also looking for technology which is well documented and have a big community support. Good documentation simplify the development process and community support is useful when dealing with bugs.

We considered several JavaScript frameworks, Grails framework and also Ruby on Rails. The pros of JavaScript is that it is intended directly for web development and it has many different libraries. However, it does not support application prototyping in a way that we demanded.

Ruby on Rails uses Ruby programming language, a dynamic language with focus on simplicity and productivity. Because of this, it is easy to create a functional prototype easily. It also has many plugins available to use. On the other side, for our early prototypes, we had to be able to use data-mining libraries (mostly written in Java) directly from the web application. Ruby on Rails does not run on Java Virtual Machine (JVM), so it could not be possible to use it. There is also JRuby on Rails which gives a Java functionality to Ruby on Rails, but it is more complicated to start with than using Grails.

Grails is a Groovy-based web application framework for the JVM. Groovy is optionally typed and dynamic language, with static typing and static compilation capabilities, for the Java platform. It can be treated as a simplified Java language with some extra features. Therefore, it is very easy to learn this language for people familiar with Java.

The biggest advantage of this framework is its prototyping capabilities. A simple empty project is fully usable web application with dynamic in-memory database and dynamically created views. It is possible to start development just with local

installation of Grails. No database or development web server is required.

Also, Grails has many useful plugins, for example security plugins, which provide user authentication, plugins for sending e-mails ... Plugins can be easily integrated to the project just by adding the plugin name to the configuration file. After project compilation, plugin is installed and ready to use. Last but not the least, it has an excellent documentation.

Grails deeply uses the Model View Controller (MVC) pattern to make clean and manageable code. MVC design pattern is a software architecture that encourages separation of concerns into three main components:

- Model – handles data representation and operations. Classes that represents entities (e.g. customer, product) and business logic/functionalities (e.g. create product record), represents the model. Below is an example of Model in Grails framework.

```
class Customer {  
    Long id  
    String firstName, lastName  
    Date birthday  
    String gender  
}
```

- View – handles how data is viewed by the user. Grails views are written using HTML and GSP tags in files with .gsp extension. Below is an example code for showing a table with customer name and gender.

```
<table>  
    <tr><th>Name</th><th>Gender</th></tr>  
    <g:each in="${customerList}" var="cust">  
        <tr><td>${cust.lastName} ${cust.firstName}</td>  
        <td>${cust.gender}</td></tr>  
    </g:each>  
</table>
```

- Controller – handles the code that links user in the system and responds to his actions. Controller receives requests from the user, prepares data and invokes proper methods. Then it returns the view that represents the result of user

actions. Below is a Controller for Customer model which responds to action called list. The controller is invoked whenever the URI */customer/list* is invoked.

```
class CustomerController {  
    def list() {  
        def list = Customer.list()  
        [list:list]  
    }  
}
```

Our web application consists of fourteen domain classes (models). These are

- Dataset – stores all information about the user’s dataset like name, description, name of data file and statistical information like number of attributes, number of instances ...
- Attribute – represents a general information about attribute such title, position in data file and number of missing values. This class is a common superclass for concrete types of attributes. We consider these three types
 - BooleanAttribute – stores information about number of true and false values in particular attribute in addition to the information from general Attribute class.
 - NumericAttribute – contains additional information related to numeric values in a given attribute.
 - StringAttribute – includes also the number of distinct values. Every attribute which type is none of the two above is considered as string attribute.
- Analysis – corresponds to one single file generated during the preprocessing phase. It stores the data such a file name, data type, number of attributes ...
- AnalysisResult – stands for general result of a single data-mining model. It stores data like root mean squared error, mean absolute error, number of correctly and incorrectly classified instances and an overall performance summary. Its subclasses below contains concrete method and hyperparameter values.
 - KnnResult
 - DecisionTreeResult

- LogisticRegressionResult
- SvmResult.
- Role, User, UserRole – store all information related to user authentication. We used Spring Security plugin for Grails to provide this functionality, because Spring Security is a widely used authentication framework.

All of these models have their corresponding controllers and views which provide described functionality altogether. It is not necessary to write code for communication with database, because Grails can manage it their own way. All other functionality like communication with a computational cluster is implemented in Grails services.

4.4.2 Database

As already mentioned above, we used a MySQL database. We did not make any research about the most suitable database for our application, because it is not a crucial component of our prototype. Since it is an isolated part of our system and is independent from business logic, it can be replaced by a more suitable database technology in the future easily.

We also did not create a database schema for our prototype on our own. It was automatically generated by a Grails framework as part of its prototyping capabilities. Grails require just a configuration file with database type, address and authentication information. Schema is generated based on the domain classes and relationships defined between them. Figure 4.20 shows a generated database schema of our prototype. Tables *database* and *analysis* corresponds to the respective domain classes. Table attribute integrates all types of attributes (numeric, string and boolean) into a single table. Dataset attribute type is determined from parameter *class*. Also, modelling results are integrated in single table *analysis_result* and particular type is determined from *class* attribute. Tables *user*, *role* and *user_role* serve for user authentication and determining his rights in system. Finally, table *registration_code* stores the registration codes for users who just registered and did not confirm his email address so far.

As you can see, we do not save real dataset data into database. Instead of this we save just file names and real data are saved in a directory on computation cluster. This option reduces database load, because the database contains only meta information. We use this technique for dataset files as well as for files from preprocessing step.

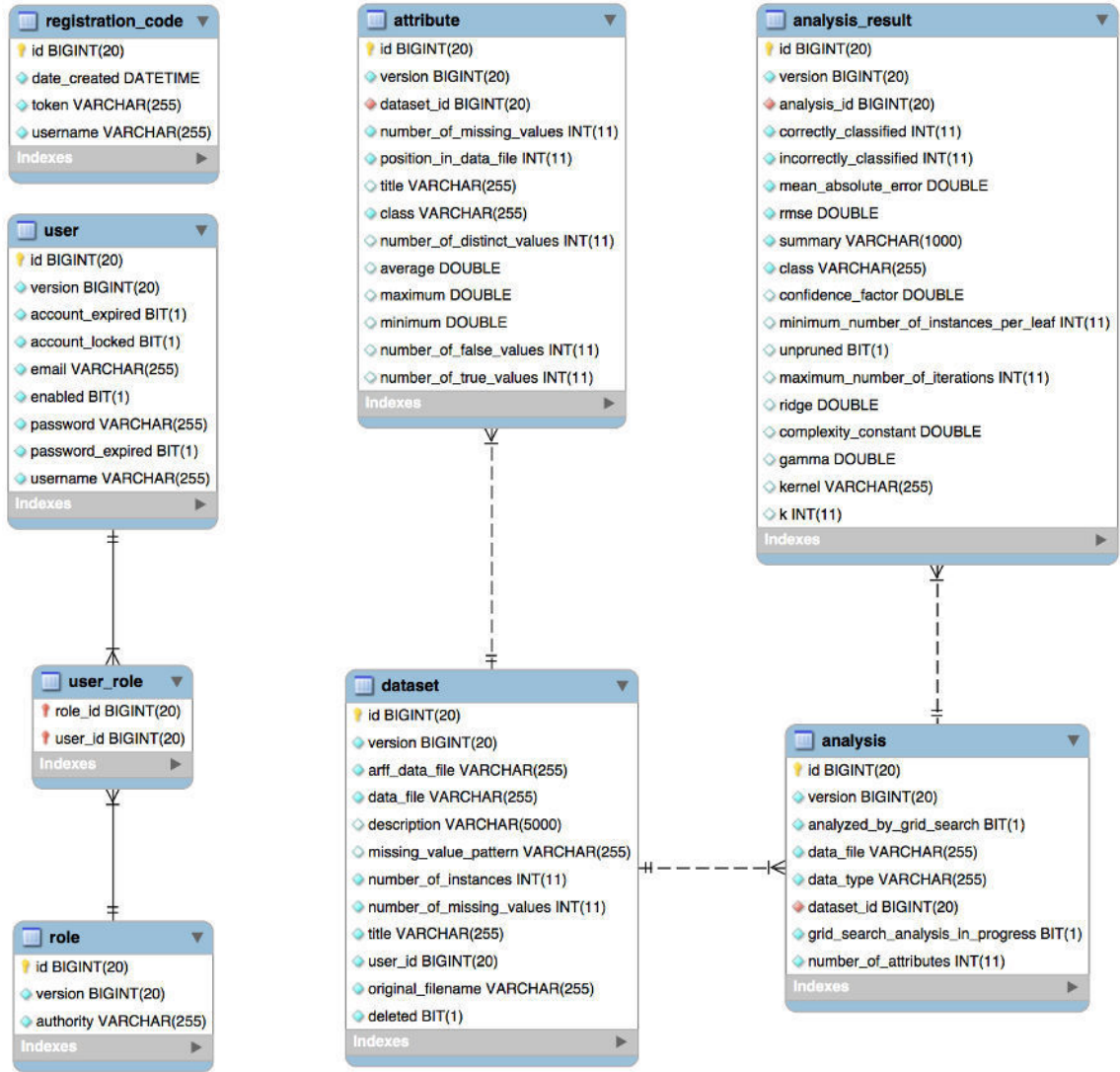


Figure 4.20: Database schema of the system.

4.4.3 Computational server

Last component of the system is a backend server for performing complex operations. These computations are difficult in terms of time and also computational resources. Because of these demands, it would not be possible to run them on standard web server. The calculations would consume all resources for a long time and the web server would fall down. In early stage of the project we performed all computations on a web server through Grails, because we needed just to test the concept as a whole. Later we detached costly calculations to separate component. Because the calculations were written in Groovy combined with pure Java, we decided to use Java language for this part as well to minimize a migration effort.

All data-mining algorithms described in Chapter 2 including their evaluation on a

given dataset are already implemented in various data-mining software libraries. So we did not implement them on our own. Instead of this we used implementations of k-nearest neighbor, Decision tree and Logistic regression algorithms from Weka library written in Java. This library is also used in data-mining software of the same name. For k-nearest neighbor algorithm we used IBk class, for Decision tree J48 class and for Logistic regression Logistic class of this library. The concrete implementation of evaluation method for Decision tree algorithm in our project is shown below

```
/**
 * @param dataInstances analysis instances
 * @param m             minimum number of instances
 *                      per leaf
 * @param c             pruning confidence
 * @param unpruned      whether tree should be
 *                      unpruned or not
 * @return evaluation
 */
public Evaluation j48DecisionTreeAnalysis(Instances
dataInstances, int m, float c, boolean unpruned) {
    J48 j48 = new J48();
    j48.setMinNumObj(m);
    j48.setConfidenceFactor(c);
    j48.setUnpruned(unpruned);
    Evaluation evaluation;
    try {
        evaluation = new Evaluation(dataInstances);
        evaluation.crossValidateModel(j48,
            dataInstances, 10, new Random(1));
    } catch (Exception e) {
        return null;
    }
    return evaluation;
}
```

This code performs j48 decision tree algorithm and evaluates results 10 times with 10-fold cross validation method. The Evaluation object for given model is returned.

Support vector machines algorithm has a faster implementation in LibSVM library written in C language, but there is also Java wrapper of this library, so we used this one.

Communication with web application is implemented through a simple web socket. Web server connects to the socket and sends a request information. Then it disconnects, if it does not require the results immediately or waits for the response. Computational server receives the request and immediately creates a new thread for calculations. Currently we support three types of request

- PREPROCESS – web server sends the dataset information and user answers from conversational system and backend server generates the preprocessed data files and performs simple analysis using the recommendation system.
- GRIDSEARCH – computational server performs deep Gridsearch analysis for a given file based on the request from web server.
- FILE_REQUEST – file for a given analysis is sent to web server from backend, because of the user request for its download.

Calculations can take a long time. So to serve requests, the system runs every calculation on a separate thread in parallel. This component runs on computational cluster with 128 processors and 128 GB of RAM, so it has enough power for our needs.

Conclusion

In this thesis, we dealt with systems for managing of data mining projects. We proposed our custom design of such tool. The main focus is on ease of use, even for users with no or minimum knowledge about data-mining. This tool guides a user through all steps of CRISP-DM methodology and performs required actions automatically based on his input. It contains a simple conversational system to gain the knowledge from user and also a recommendation system for recommending suitable data-mining models.

The whole area of study is too wide, therefore we do not take some parts into account. For example, we decided to take into account only the selected data mining methods for classification in the modelling phase. Nevertheless, system design allows us to implement them later.

We successfully implemented our proposal into a web application. This application is available online and several users have already made their first experiments. In the future we plan to analyze their feedback and try to improve our system based on their experience.

During the elaboration of this thesis we dealt with many challenges and we proposed several solutions for them. At the same time, we identified many weaknesses of these systems.

For example there are many opened questions about the user interface. Our prototype contains very simple UI, because it is comfortable for users. Can the UI be better? Is it even possible to build such a UI that is simple and user friendly on the one side and complex and comprehensive on the other side? The main subject of these questions is if there are some hidden problems after the simplicity of the UI.

There is also one very important question. Is it even possible to automate data-mining tasks and tools? Hyperparameter search and model selection tasks are very sensitive to data. Therefore, it is very hard to recommend good model and its hyperparameter settings. Our prototype performs many calculations in real time. Despite

of it, user has to wait for the complete results, because data-mining models are very complex and time consuming. Also, it requires some user interaction to gain a background knowledge. Is it possible to eliminate this interaction?

We remain these questions opened for now, because they require further study and analysis. Solving problems which we identified can definitely help to build better data-mining systems in the future.

Resumé

V tejto práci sme sa zaoberali systémami pre správu projektov v oblasti dolovania dát. Naším cieľom bolo navrhnuť a implementovať prototyp webovej aplikácie na dolovanie dát určený pre užívateľov bez pokročilých technických znalostí z tejto oblasti. Tento nástroj prevedie užívateľa všetkými krokmi CRISP-DM procesu a automaticky vykoná príslušné akcie na základe pokynov užívateľa.

V prvej a druhej kapitole sme popísali techniky a terminológiu dolovania dát použitú počas práce. V tretej kapitole sme zadefinovali problém algoritmizácie odporúčania výpočtového modelu pre konkrétny dataset. Takisto sme popísali existujúce riešenia pre tento problém. Na ich základe sme navrhli a implementovali vlastný prístup k riešeniu, ktorý sme experimentálne porovnali s publikovanými prístupmi. Z tohto experimentu vyplynulo, že naše riešenie sa vyrovná prezentovaným prístupom. Spojením týchto algoritmov sme dokázali ďalej zlepšiť vhodnosť odporúčania.

V druhej časti práce sme navrhli a implementovali prototyp webovej aplikácie. Užívateľovi umožňuje nahrať dáta vo formáte .csv, ktoré následne zanalyzuje a zobrazí mu štatistické informácie o jednotlivých atribútoch ako napríklad typ atribútu, počet chýbajúcich hodnôt ... Následne užívateľ môže nechať tieto dáta analyzovať algoritmami na dolovanie dát, pričom v súčasnosti nástroj pracuje s algoritmami k-NN, Logistic regression, Decision tree a tiež SVM. Na získanie doplnujúcich informácií od užívateľa sme implementovali jednoduchý konverzačný systém, ktorý na základe prvotnej analýzy vygeneruje otázky pre užívateľa. Ten na ne odpovie pomocou webového formulára. Na základe týchto odpovedí sa predspracujú dáta a odporučí sa najvhodnejší algoritmus spolu s konkrétnymi nastaveniami hyperparametrov. Po dobehnutí analýz sa zobrazia výsledky spolu s vybraným algoritmom a jeho hyperparametrami.

V závere sme zhrnuli nevýhody takýchto systémov, ktoré sme identifikovali počas práce na tejto aplikácii. Takýmto problémom je napríklad používateľské rozhranie. Na jednej strane musí byť čo najjednoduchšie pre používateľa, ale na druhej strane

prílišnou jednoduchosťou obmedzujeme jeho možnosti interakcie so systémom. Riešenie identifikovaných problémov si vyžaduje ďalší výskum v tejto oblasti.

Zdrojové kódy aplikácie sú k dispozícii v prílohe tejto práce.

Bibliography

- [1] BERKA, P. Dobývání znalostí z databází. Vyd. 1. Praha: Academia, 2003, 366 s. ISBN 80-200-1062-9.
- [2] JIAWEI HAN, M. K. Data mining: concepts and techniques. 3rd ed. Amsterdam: Elsevier/Morgan Kaufmann, 2012. ISBN 0123814790.
- [3] ZAKI, M. J., WAGNER, M. Data mining and analysis: fundamental concepts and algorithms. Cambridge: Cambridge University Press. 593 p. ISBN 9780521766333.
- [4] WITTEN, I., EIBE, F., HALL, A. M. Data mining: practical machine learning tools and techniques. 3rd ed. Burlington, MA: Morgan Kaufmann, 2011, 629 p. Morgan Kaufmann series in data management systems. ISBN 0123748569.
- [5] BRAMER, M. Principles of data mining. 2nd ed. London: Springer, 2013. ISBN 1447148835.
- [6] THORTON, C., HUTTER, F., HOOS, H. H., LEYTON-BROWN, K. *Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms*. Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, Chicago, Illinois, USA, 2013, p. 847–855. ISBN 978-1-4503-2174-7.
- [7] VILALTA, R., GIRUARD-CARRIER, C., BRAZDIL, P. SOARES, C. *Using Meta-Learning to Support Data Mining*. International Journal of Computer Science & Applications, Vol. I, No. 1, p. 31–45. 2004.
- [8] KAZÍK, O., PEŠKOVÁ, K., PILÁT, M., NERUDA, R. *Combining parameter space search and meta-learning for data-dependent computational agent recommendation*. 11th International Conference on Machine Learning and Applications (ICMLA 2012): Boca Raton, Florida, USA, 12-15 December 2012. 2 volumes. ISBN 9781467346511.

- [9] BRAZDIL, P., SOARES, C., DA COSTA, J. P. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results, *Machine Learning*, Vol. 50, p. 251–277, 2003.
- [10] DASU, T., JOHNSON, T. *Exploratory data mining and data cleaning*. New York: Wiley-Interscience, 2003, 203 p. ISBN 978-0-471-26851-2.
- [11] HAND, D., MANNILA, H., SMYTH P. *Principles of data mining*. Cambridge, Mass.: MIT Press, 2001, 546 p. *Adaptive computation and machine learning*. ISBN 026208290x.
- [12] GUO, Y., GROSSMAN, R. *High performance data mining: scaling algorithms, applications, and systems*. Boston: Kluwer Academic, 1999, 105 p. ISBN 0-7923-7745-1.
- [13] HORNICK, M. F., MARCADÉ, E., VENKAYALA, S. *Java data mining: strategy, standard, and practice: A practical guide for architecture, design, and implementation*. Amsterdam: Elsevier/Morgan Kaufmann, 2007, 520 p. *Morgan Kaufmann series in data management systems*. ISBN 978-0-12-370452-8.
- [14] RAJARAMAN, A., ULLMAN, J. D. *Mining of massive datasets*. Cambridge: Cambridge University Press, 2012, 315 p. ISBN 9781107015357.
- [15] SULLIVAN, R. *Introduction to data mining for the life sciences*. New York: Humana Press, 2012, 635 p. ISBN 978-1-58829-942-0.
- [16] ANDERSON, R. K. *Visual data mining: the VisMiner approach*. Hoboken, N.J.: Wiley, 2012, 196 p. ISBN 978-1-119-96754-5.
- [17] LANDER, J. P. *R for Everyone: Advanced Analytics and Graphics*. Addison-Wesley Professional. 2013, 432 p. ISBN 978-0321888037.
- [18] *Cross Industry Standard Process for Data Mining*. Available from Internet [accessed 29-May-2014]: http://en.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining/
- [19] *UCI Machine Learning Repository*. Available from Internet [accessed 28-May-2014]: <http://openml.org>
- [20] *OpenML*. Available from Internet [accessed 14-April-2015]: <http://archive.ics.uci.edu/ml/>

[21] *BigML*. Available from Internet [accessed 14-April-2015]: <https://bigml.com>

Attachments

A CD with source code of web application and backend application, several database dumps with measurement data and electronic copy of this thesis.