Machine learning for the prediction of railway fares

Maschinelles Lernen zur Vorhersage von Bahnpreisen
Bachelor-Thesis von Fabian Hirschmann
31. Oktober 2013

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Department of Computer Science
Knowledge Engineering Group
Algorithm Engineering Group

Machine learning for the prediction of railway fares
Maschinelles Lernen zur Vorhersage von Bahnpreisen

Vorgelegte Bachelor-Thesis von Fabian Hirschmann

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Frederik Janssen, Dr. Mathias Schnee, Daniel Mäurer

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 31. Oktober 2013

_____

(Fabian Hirschmann)

Abstract

The aim of this thesis is to study and compare a variety of machine learning algorithms that can be used to predict the cost of a train ride on the German railway system according to attributes such as the distance traveled on different types of trains. This prediction is to be implemented in an existing system (MOTIS), which can be used to identify the lowest possible fare for any given journey. The comparison will also include an existing method that is currently used to predict railway fares, and we were able to produce models that perform better than the existing method. Furthermore, the thesis describes a sampling method that was developed as a means of producing data samples of train journeys that come close to train journeys booked by actual humans.

The thesis starts by providing a short introduction to the field of machine learning before progressing to describe the techniques utilized in this thesis, which include simple linear regression, multivariate adaptive regression splines, decision tree learners, support vector machines and neural networks. The thesis then continues with the development of a sampling method and suggests a probability-based sampling approach that favors routes that are planned to and from train stations that have a high number of incoming and outgoing trains. Finally, different machine learning algorithms are evaluated and ranked using 10-fold cross-validation. The outcomes reveal that a sophisticated tree ensemble learner (cubist) yields the best results, followed by support vector machines. Multivariate adaptive regression splines also pose a viable option and are especially noteworthy due to the simplicity of the model produced, which is particularly interesting because simple models have a lower prediction time than more complex ones.

Zusammenfassung

Ziel dieser Arbeit ist das Studieren und Vergleichen einer Vielzahl von Algorithmen aus dem Bereich des maschinellen Lernens zur Vorhersage des Beförderungsentgelt im Schienenverkehr anhand von Attributen wie der Distanz einer Zugfahrt in verschiedenen Zugkategorien. Die entwickelte Methode zur Vorhersage soll in ein bestehendes System (MOTIS) integriert werden. Dieses System kann zur Identifizierung der günstigsten Zugfahrt verwendet werden. Der Vergleich beinhaltet außerdem eine bestehende Methode, die zur Vorhersage von Bahnpreisen benutzt wird. Uns war es möglich ein Modell hervorzubringen, das besser als die bestehende Methode abschnitt. Außerdem wird ein Samplingverfahren zur Produktion von Dateninstanzen von Zugfahrten vorgeschlagen, das realitätsnahe Zugverbindungen generiert.

Diese Arbeit beginnt mit einer kurzen Einführung in das Feld des maschinellen Lernens, bevor die benutzten Algorithmen beschrieben werden. Zu diesen Algorithmen zählen lineare Regression, Entscheidungsbaumlerner, Multivariate Adaptive Regression Splines, Support Vector Maschinen und neurale Netzwerke. Fortgesetzt wird diese Arbeit mit der Beschreibung einer wahrscheinlichkeitsbasierten Samplingmethode, die Bahnhöfe mit einer großen Anzahl an ein- und ausgehenden Verbindungen bevorzugt. Zum Abschluss werden verschiedene Methoden des maschinellen Lernens mittels zehnfacher Kreuzvalidierung evaluiert. Die Ergebnisse zeigen, dass ein ausgereifter Ensemble-Entscheidungsbaumlerner (cubist) die besten Resultate liefert, gefolgt von Support Vector Maschinen. Multivariate Adaptive Regression Splines stellen außerdem eine praktikable Lösung dar und sind besonders aufgrund ihrer Einfachheit interessant, da einfache Modelle eine geringere Vorhersagezeit als komplexere Modelle besitzen.

Contents

1 Introduction

At present, the majority of timetable information systems that are provided from public transportation services communicate information about travel time only and disregard other relevant criteria, specifically the ticket cost. Studies that have addressed the potential to include supplementary information in addition to travel time have been carried out by a number of researchers (Müller-Hannemann and Schnee 2005; Müller-Hannemann and Schnee 2007), and a Multi-Objective Traffic Information System (MOTIS) has been proposed. This system allows service providers to optimize for multiple criteria, including the ticket cost. The MOTIS algorithm is based on a highly modified version of Dijkstra's algorithm that allows optimization for multiple criteria. It uses a price estimation for the distance already traveled as well as a lower bound for the distance yet to be traveled. In order to optimize for the overall ticket cost, a fast numeric prediction of the price is required.

The tariff systems employed by railway companies, specifically those of Deutsche Bahn[1], are very complex and the ticket cost is not necessarily proportional to the distance traveled. Many parameters contribute to the actual final ticket cost including, but not limited to, the type of train used (i.e. express trains or inter-city trains) and existing levels of competition from low-cost airlines. In 2008, Deutsche Bahn granted Müller and Schnee access to the Black-Box Pricing Component (BPC) that they utilize to calculate railway fares (excluding contingent-based tickets or special offers). Unfortunately, this component was too slow to be used in the algorithm utilized by MOTIS. Hence, this thesis focuses on finding a method that emulates the behavior of the BPC using machine learning techniques. This method will be compared with another technique for fare prediction that is currently used by MOTIS and discussed in Chapter 2. Ideally, the models we produce provide better fare predictions than the current method. Of course, in addition to providing a reliable estimation of the ticket cost, the model created needs to be able to provide predictions fast. It is important to note that all research and conclusions presented in this thesis focus on the then-current BPC. Any modifications to the pricing structure of the German railway system that have happened in the meantime are disregarded as a result of the lack of current data. Imaginable posterior adaptation of the results of this thesis include the multiplication of the predicted price with a constant factor that accounts for the inflation that has occurred between the year 2008 and today.

1.1 The German Railway System

This thesis relies heavily on vocabulary that is mostly used in the domain of the German railway system. It is therefore imperative that a short introduction to the different passenger trains that are in operation in Germany and the price structure thereof is provided. The majority of passenger trains in Germany are operated by Deutsche Bahn, a private German joint-stock company (AG) that has the federal government as its majority shareholder. The German railway system can be broken down into four different train categories:

- InterCity-Express (ICE): long-distance and high-speed trains that connect major cities.
- InterCity (IC) and the EuroCity (EC): slower express trains that connect smaller (European) cities.
- InterRegio (IR) trains: regional trains that are now very scarce in Germany.
- RegionalBahn (RB) or RegionalExpress (RE): semi-fast trains that connect different regions.

[1] http://www.bahn.de

Each of the train categories is assigned a number using background knowledge described in Subsection 4.1.2. This effectively means that ICE, IC/EC, and regional trains form classes 0, 1 and 3 respectively. InterRegio, Class 2 trains, can be neglected because of their scarceness in our data set. As mentioned, according to the tariff system employed by Deutsche Bahn, the ticket price is not necessarily proportional to the distance traveled. Instead, fare costs are based on so-called relational prices for each train category and corridors. A corridor is the area between two points which a train must not leave, and determines the price. For ICE trains, the ticket cost between any two points is based on the corridors through with a train has to travel, but can be influenced by factors such as train comfort and journey duration. It is assumed that the distance between any two tariff points in a corridor does play a crucial role in the price calculation.

The case is quite different with InterCity and EuroCity trains. As we will see in Section 4.2, the ticket cost for IC/EC trains linearly depends on the distance traveled. Once a certain cut-off distance has been reached, a cost degression is applied. Do note that the IC/EC pricing scheme is rumored to have changed recently such that it is modeled in a similar manner to that of the ICE trains. As a result of the lack of current data samples, this change cannot be taken into account in this thesis.

1.2 Thesis Structure

We will discuss related work concerning fare cost prediction in Chapter 2, and after a short introduction to machine learning in Chapter 3, this thesis will continue with the actual machine learning process. It is important to note that this process is not a linear one. Learning from data involves many iterations, often going back to the start and changing something fundamental (e.g. the sampling method). Figure 1.1 visualizes the machine learning work flow. This is also known as the Cross Industry Standard Process for Data Mining (CRISP-DM) and is described by Shearer (2000).

It all starts with data. In our case, we generate data using MOTIS in Section 4.1. We then analyze and try to understand the data in Section 4.2. After we have gained some knowledge of the structure and the character of the data, we will prepare it in order for it to be fed to several machine learning algorithms. This preparation mostly involves the removal of attributes that serve no purpose in our learning schemes. In Chapter 6, we describe how our models were tuned, and in Chapter 7 the resulting trained models are validated using 10-fold cross validation (a method which is explained in Subsection 3.3.2). There are many back pointers in this process. For example, after the first sampling method was put in place, it was deemed required to go back to work on the data generation method because the then-current method was not realistic enough and needed improvements. Nonetheless, this thesis presents the process of machine learning in a linear fashion, i.e. first sampling data, then analyzing it, and finally learning from it. Where appropriate, we describe the results of previous iterations and why it was seen necessary to go back in the learning process.

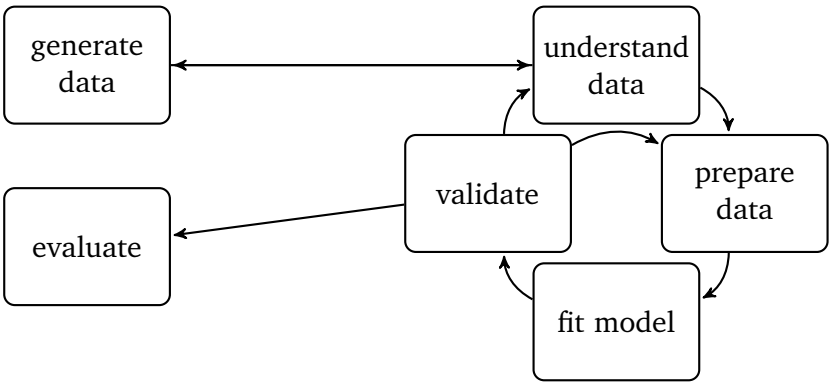

Figure 1.1.: The machine learning process. This process is not linear and often involves going back and forth between the individual steps.

2 Related Work

As a means of predicting railway fares in the German railway system, Müller-Hannemann and Schnee (2005) describe a ticket price estimation system that operates through multiplying the distance to be traveled between two train stations by a constant factor which, in this case, is price per kilometre (€ 0.14 per kilometre). Additionally, their method includes the application of surcharges, which vary according to the type of train involved (€ 12 for an ICE train or € 7 for an IC or EC train). Because this method is no longer available in MOTIS, it has been recreated in Section 5.3 for comparison. This estimation is very rough, but models the regional train travel pricing structure used by Deutsche Bahn quite precisely. However, when anything other than regional trains is involved, models for predicting railway fares are not as precise.

Harnisch and Nuhn (2010) analyzed the structure of railway fares through a lab project at Technische Universität Darmstadt. They treated the distance traveled on each train category as separate paradigms and based their final estimation on the cumulative prediction of three distinct models. The first model they produced predicted the ticket price for express trains (class 0) using a second-order polynomial regression scheme. The second model estimated the price of the distance traveled on inter-city trains (class 1) and also employed a second-order polynomial. For regional trains (class 3), the researchers adopted the original method proposed by Müller-Hannemann and Schnee (2005), because they were unable to identify a linear model for regression that offered any improvement on this model. The models for express and inter-city trains already had a very high reproducible explanatory value, with the correlation coefficient (R^2) being 0.9469 for express trains, and 0.9906 for inter-city trains – if applied to journeys consisting of the respective train class alone. Their report did not contain any evaluation of journeys involving different train classes. In addition, they proposed a ticket price ceiling of € 122. We have successfully verified the results presented in their report. The relationship between the distance traveled and the ticket price for each train category is discussed in Subsection 4.2.2, the model itself is given in Subsection 7.1.6 and the cross-validation results for this model compared to other models can also be obtained from Table 7.4

Etzioni et al. (2003) explored the possibility of predicting the prices of airline tickets using a data set of 12000 price observations. Their pilot study was limited to round-trip ticket prices for Los Angeles to Boston, and Seattle to Washington over a 41-day period. The data mining methods investigated in this study included the rule learning algorithm Ripper (Cohen 1995), the popular reinforcement learning approach Q-learning (Sutton and Barto 1998) and time series analysis (Diebold 2000; Granger 1980). The algorithm that emerged from the study combined the models produced by the algorithms mentioned previously and was able to save an overall of 4.4% on the ticket price when tested on a data set of 4,480 simulated journeys. The airfare data was collected directly from major travel web sites. However, although this research is of interest, it involves a completely different domain and focused on the prediction and time-series analysis of just two routes; as such, it varies from the scope of our research and is not aligned with our goal to predict the ticket prices for *all* train journeys.

3 Foundations of Machine Learning

Generally speaking, a machine learning problem consists of $n \in \mathbb{N}$ training samples of D-dimensional feature vectors $\mathbf{x} = (x_1, \ldots, x_D)$. Training samples are also called *instances*, and each dimension of the D-dimensional instance is a *feature* or an *attribute*.

Definition 1 *An **attribute** or a **feature** is an aspect of a system such as the price, distance traveled, or number of transfers.*

Definition 2 *A **predictor** is an attribute that plays a role in the prediction of the target attribute.*

Definition 3 *An **instance** is an observed set of values of attributes, including the value to be predicted.*

Definition 4 *A **data set** $\mathcal{D}$ is a set of instances.*

Consider a 3-dimensional space of 4 feature vectors where the first dimension is the attribute `distance`, the second dimension is the attribute `stop count`, and the third dimension is the attribute `price`. The training instances are as follows:

$$\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$$
$$\mathbf{x}_1 = (100, 10, 30)^\top, \quad \mathbf{x}_2 = (50, 3, 20)^\top, \quad \mathbf{x}_3 = (140, 9, 50)^\top, \quad \mathbf{x}_4 = (200, 20, 55)^\top$$

Where machine learning (aka predictive modelling) comes in handy is when we would like to predict one or more target attributes using only predictors. Consider the scenario where you would like to predict the attribute `price` when you only have a vector of predictors $(130, 15, ?)^\top$. Here, the only known attributes are `distance` and `stopcount`. Using the techniques of machine learning, you can now try to find a `model` that predicts the `price`.

Machine learning algorithms can basically be separated into a few categories:

- Supervised learning, in which we try to predict a target value by learning a model from previous training data. This category can be further divided into:
  - Classification, where training instances belong to two or more classes (a set of categories or nominal values such as car brands) and we are trying to classify new data by identifying to which class it belongs to.
  - Regression, where it is assumed that the training data is made up of one or more continuous (numeric) feature that we are trying to predict. The case of predicting more than one continuous feature is called multi-target regression.
- Unsupervised learning, where we are not trying to predict specific features and instead would like to group similar instances together.

Figure 3.1 depicts a subset of the fields machine learning is concerned with. Do note that many of the algorithms, such as decision trees and support vector machines, can be adapted in order to be usable for both regression and classification problems (dual use). Given the nature of the problem at hand, this thesis focuses on regression models that produced promising results.

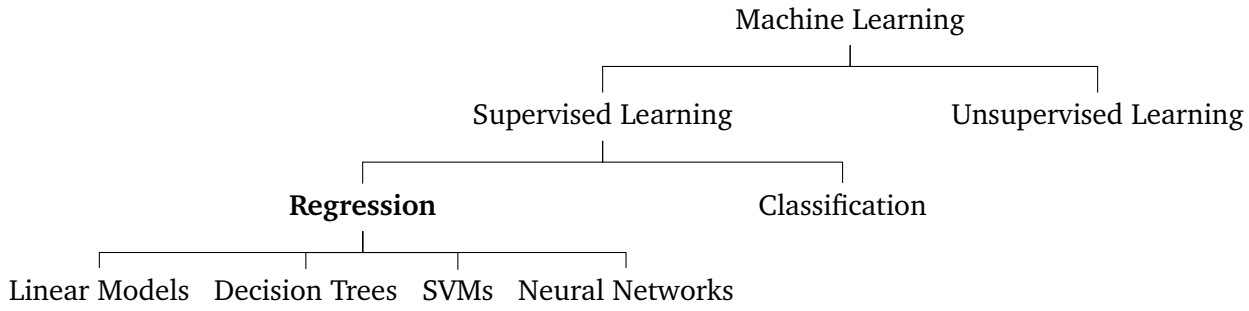

Figure 3.1.: An incomplete subset of the fields and algorithms of Machine Learning. The subfield of regression is marked because the problem at hand is a regression problem.

3.1 Models for Prediction

We are going to loosely introduce some machine learning algorithms in this section. First, we will start with a simple model: linear regression. Second, a short introduction to Multivariate Adaptive Regression Splines (MARS) will be provided. Then, we will take a closer look at so-called decision trees. To complete this section, we will present some more advanced modeling techniques, namely Support Vector Machines (SVMs) and neural networks.

Having no prior knowledge of the performance of any machine learning algorithm applied to the problem at hand, we tried to choose a selection of algorithms that are representative for the various approaches to machine learning. Although it is possible to use classification methods together with *Discretization*, a method that converts numeric attributes into nominal attributes by putting numeric attributes into a number of bins (Fayyad and Irani 1993), we have focused on algorithms that have support for regression built-in. The reason for the choice we have made is clear: time constraints. Adding new algorithms to the list of methods to evaluate is simple, but actually training a model and performing the evaluation takes time. For this reason, we tried to include standard schemes such as SVMs, neural networks, and tree learners in addition to not-so-common algorithms that provided interesting results like MARS.

3.2 Overfitting

In machine learning, overfitting is a very common problem. It can occur when an algorithm is trained with a training set that contains some regularities that do not appear in new data we are trying to predict. In other words, a method may adapt to features in the training set that have no causal relation to the function approximation. When this situation occurs, the learner performs well when predicting the training data, but fails to predict previously unseen data well.

There are several approaches to prevent overfitting, depending on the algorithm in use. For this reason, we will briefly discuss ways to do so in the respective sections, specifically for decision tree learners and the MARS algorithm.

3.2.1 Linear Regression

Linear regression comes to mind naturally when dealing with problems that have mostly numeric attributes. Let $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}, \mathbf{x}_n\}$ be a set of n observations along with their target values x_D (the last attribute in each feature vector). Linear regression outputs a prediction for the target value using the function y

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_{D-1} x_{D-1} \tag{3.1}$$

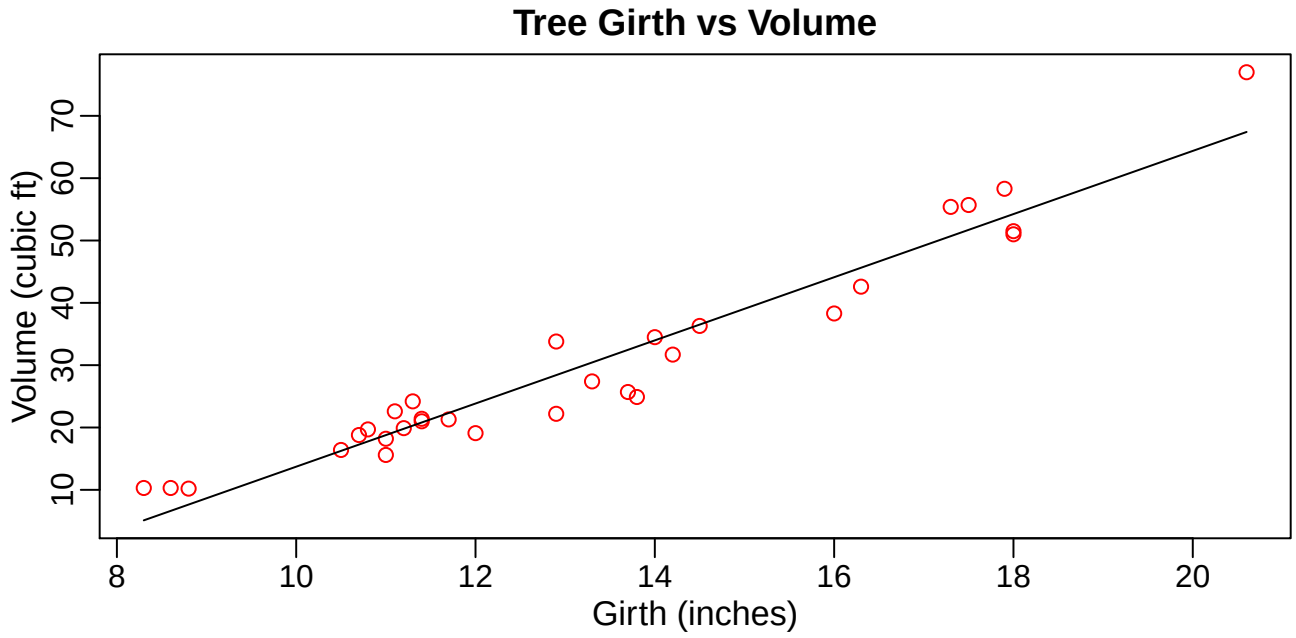

Figure 3.2.: Scatter plot with fitted regression line. The data points at the extremes suggest that the relationship between the two variables may be non-linear.

in a $D-1$-dimensional input space where $\mathbf{x} = (x_1, \ldots, x_{D-1})^\top$ is a vector of attribute values (excluding the attribute to be predicted), and $\mathbf{w}$ is a vector of attribute weights learned from previous observations. This regression is a linear function of the input variables x_i. The coefficients or `weights` $w_0, \ldots, w_D$ are calculated from training data. There are many ways to do so, but the most common one is called ordinary least squares (OLS). The idea behind OLS is to minimize the sum of squared *residuals* (the difference between the actual and the estimated value of an instance).

Given a data set that provides measurements of the girth (tree diameter), height and volume of timber in 31 felled black cherry trees, we will now give an example of the application of linear regression. Figure 3.2 shows a scatter plot and a regression line calculated using linear regression. The data points on the extremes suggest that simple linear regression might not be a good fit for the problem at hand, because the relationship between girth and volume is possibly nonlinear.

Fortunately, the model can be extended to allow linear combinations of non-linear functions of the input variables

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) \tag{3.2}$$

where $\phi_j(\mathbf{x})$ are the *basis functions* and $M-1$ the total number of parameters. Figure 3.3 demonstrates linear regression with a 2nd order polynomial basis function ϕ_j. Some authors prefer to call this *polynomial regression*. The resulting curve is a better fit for the problem at hand.

For a thorough introduction to Linear Regression for Machine Learning please see Bishop (2006, chap. 3), or for a gentler introduction Witten, Frank, and Hall (2011).

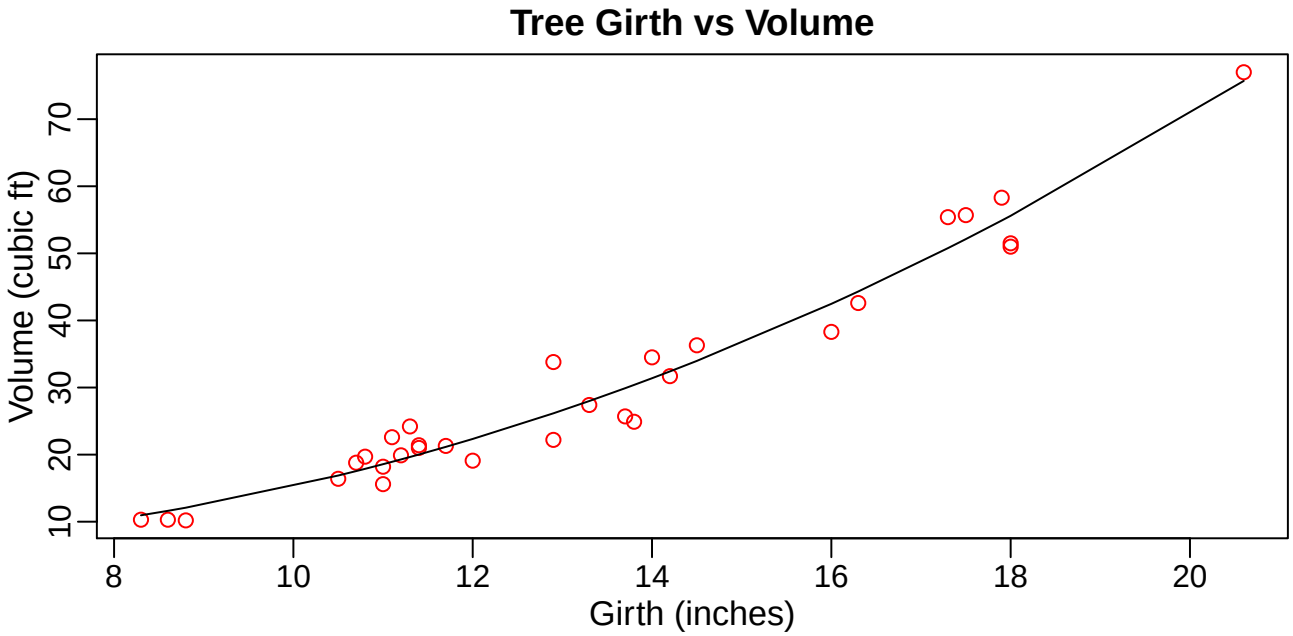

Figure 3.3.: Scatter plot with fitted polynomial regression line. Nonlinearities between the two variables are modeled more appropriately using polynomial regression.

3.2.2 Multivariate Adaptive Regression Splines

Multivariate Adaptive Regression Splines (MARS) were first invented by Friedman (1991) and are an extension of linear models as introduced in Subsection 3.2.1. The MARS algorithm automatically models nonlinearities and the interaction between predictors. Like neural networks (see Subsection 3.2.5), MARS uses surrogate features instead of original predictors. It uses *hinge functions* to take into account nonlinearities. Hinge functions can be written as

$$h(x) := \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{3.3}$$

where $\max(a, b)$ is a if $a > b$ else b and $x \in \mathbb{R}$. Do note that for $x \leq 0$ $h(x)$ is always 0. A model produced by the MARS algorithm has the form

$$y(\mathbf{x}) = \sum_{i=1}^{k} w_i \phi_i(\mathbf{x}) \tag{3.4}$$

where w_i are constant coefficients similar to the ones introduced for linear regression in Subsection 3.2.1, and ϕ_i is the basis function which can take one of the following forms:

- a constant 1
- a hinge function h
- a product of two or more hinge functions.

A MARS model is built in two steps:

- The forward pass: MARS starts with a model that is just the mean of the value to be predicted. It then keeps adding two basis functions $h(c - x)$ and $h(x - c)$ such that the residual sum of squares is minimized. MARS stays in this phase until the improvement become too minor or until the maximum number of terms is reached.

- The backward pass: MARS prunes the model to avoid overfitting (see Section 3.2), i.e. it reduces the size of the model by decreasing the number of terms it contains. This is done by calculating the cross-validation results after removing the terms one by one, where the term with the lowest negative effect on the validation results wins.

Using the tree data set (as introduced in Subsection 3.2.1), MARS will build the model shown in Figure 3.4, which is described by the following equation:

$$\text{Volume} = 28.3 + 6.5 \cdot h(\text{Girth} - 13.7) - 3.4 \cdot h(13.7 - \text{Girth}) \tag{3.5}$$

with Girth $= 13.7$ being the cutoff point produced by a pair of hinge functions. For more details on MARS, please see Kuhn and Johnson (2013, sec. 7.2), and the original paper by Friedman (1991).

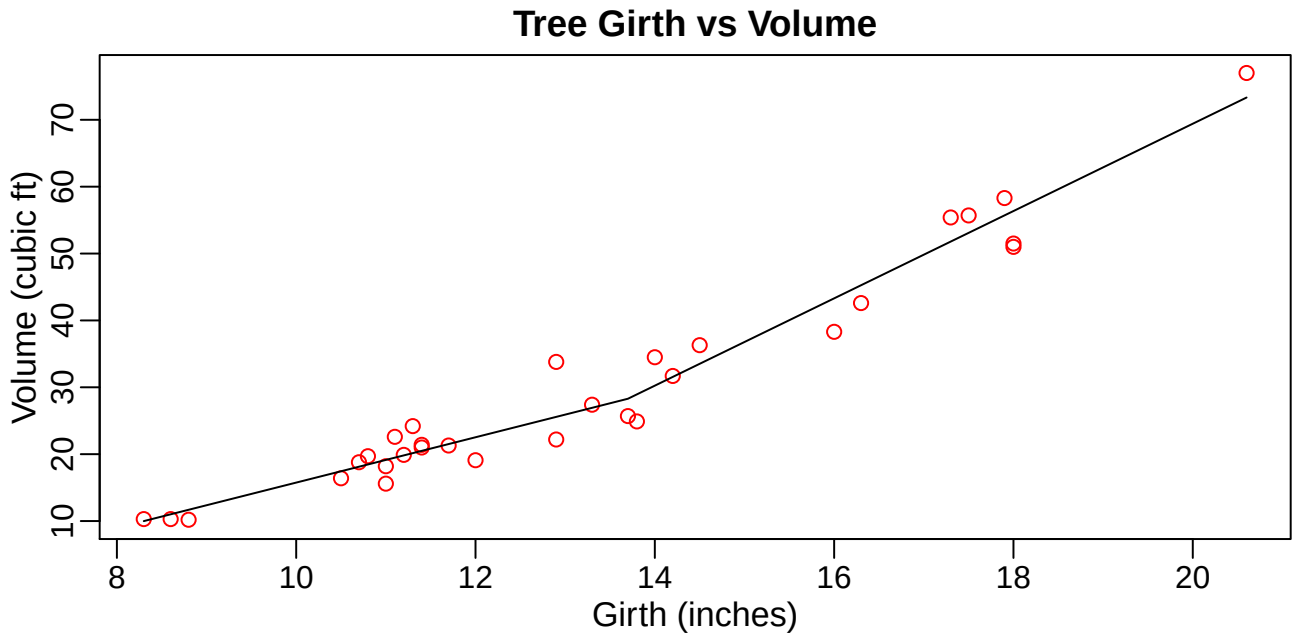

Figure 3.4.: MARS algorithm applied to the tree problem. The knit at Girth $= 13.7$ is produced by the hinge function h and automatically models nonlinearities.

3.2.3 Decision Trees

Decision trees are a popular way for both classification and regression. These trees use a tree-like graph of decisions and their possible consequences. One of the first tree learners include the Classification and Regression Tree (CART) algorithm by Breiman (1993), and Iterative Dichotomiser 3 (ID3) by Quinlan (1986) which has roots in a concept developed by Hunt (1962). In this thesis, we will focus on the Iterative Dichotomiser 3 (ID3) algorithm and its descendants, because they produced results that were much more promising compared to the CART algorithm.

In order to explain the ID3 algorithm and its descendants, we will now introduce a sample data set from Quinlan (1986), the *weather* data. The weather data is a toy data set that illustrates under what weather conditions one is likely to play an unspecified game. Table 3.1 presents this data set in a tabular fashion. All of the attributes are nominal in nature, i.e. the attribute `Play` can either be `yes` or `no`. Figure 3.5 depicts an exemplary decision tree constructed from the weather data. In this figure we can infer that one is able to play a game if the `Outlook` is `rainy` and it is not `windy`. The ID3 algorithm as shown in Listing 1 can be used to construct such a tree.

	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

Table 3.1.: The weather data set.

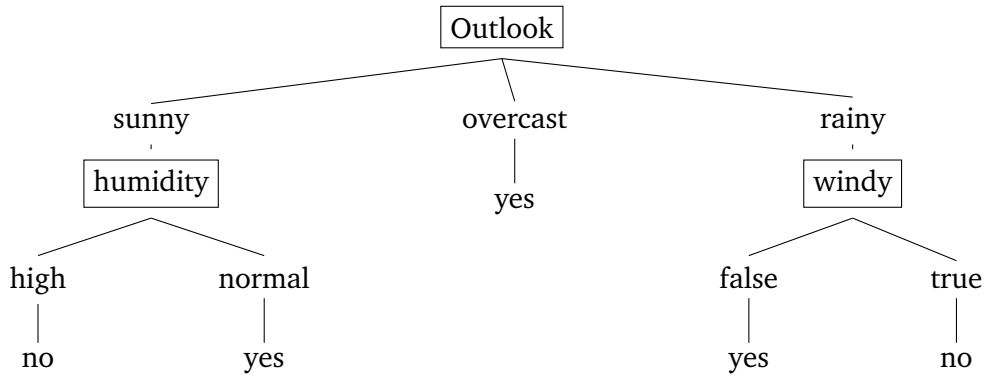

Figure 3.5.: Exemplary Decision Tree constructed from the weather data.

Still, the question on what attribute to split on remains. The entropy $E(\mathcal{D})$ is a measure of disorder or impurity and describes the amount of uncertainty for an attribute in a data set. Let $\mathcal{D}$ be a set of instances, let $p_\oplus$ be the fraction of positive training samples (`play=yes`) and $p_\ominus$ the fraction of negative training samples (`play=no`). Then, the entropy is given by

$$E(\mathcal{D}) := -p_\oplus log_2 p_\oplus - p_\ominus log_2 p_\ominus \tag{3.6}$$

The entropy can be generalized from boolean-valued target functions such as our example weather data to discrete-valued target functions. Let $\mathcal{D}$ be a set of instances, and let p_i be the fraction of instances in S with output value i. Then the entropy is given by

$$E(\mathcal{D}) := -\sum_i p_i \log_2 p_i \tag{3.7}$$

If we want to calculate the entropy for `Outlook = sunny`, we get

$$E(\text{Outlook} = \text{sunny}) = -\frac{2}{5} log_2 \frac{2}{5} - \frac{3}{5} log_2 \frac{3}{5} = 0.971 \tag{3.8}$$

```
ID3 (Examples, Target_Attribute, Attributes)
    Create a root node for the tree
    If all examples are positive, Return the single-node tree Root, with label = +.
    If all examples are negative, Return the single-node tree Root, with label = -.
    If number of predicting attributes is empty, then Return the single node tree Root,
    with label = most common value of the target attribute in the examples.
    Otherwise Begin
        A ← the attribute that best classifies examples.
        Decision Tree attribute for Root ← A.
        For each possible value, v_i, of A,
            Add a new tree branch below Root, corresponding to the test A = v_i.
            Let Examples(v_i) be the subset of examples that have the value v_i for A
            If Examples(v_i) is empty
                Then below this new branch add a leaf node with label = most common
                target value in the examples
                Else below this new branch add the subtree
                   ID3(Examples(v_i), Target_Attribute, Attributes - {A})
    End
    Return Root
```

Listing 1: Pseudo code for the ID3 algorithm from Mitchell (1997, p. 56)

because three positive outcomes can be distinguished from two negative outcomes when the `Outlook` is `sunny`.

After having calculated the entropy for each attribute, we can now use this measure during the application of the ID3 algorithm in order to decide which attribute to use for the split. As an alternative measure, the Kullback-Leibler divergence (Kullback and Leibler 1951) which, in the context of machine learning, is also known as Information Gain IG(A), can be used. This heuristic usually prefers an attribute with high mutual information over other attributes and is defined as

$$\mathrm{IG}(\mathcal{D},A) = E(\mathcal{D}) - \sum_i \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \cdot E(\mathcal{D}_i) \tag{3.9}$$

The *C4.5* algorithm is an extension of the ID3 algorithm and was also invented by Quinlan (1993a). It can deal with missing attributes by evaluating the gain for an attribute by considering only instances where that attribute is defined. In addition, the C4.5 algorithm can also deal with continuous attributes by creating a threshold value that splits the data set into two sets whose attribute values are either above, below, or equal to the threshold value. Another novel feature of C4.5 constitutes the pruning of a decision tree. Pruning is done by replacing branches (subtrees) by a leaf node. The algorithm prunes a subtree if the expected error rate in that subtree is greater than in the single leaf.

Still, in some real-world learning tasks, we are trying to predict a continuous numeric value (regression). *Discretization* methods such as the MDL method proposed by Fayyad and Irani (1993) can be utilized to convert numeric attributes to nominal attributes. The MDL method puts numeric values into bins, hence producing a nominal class. However, in this thesis we chose to focus on tree learners that have support for numeric attributes built-in.

The *M5* system, also created by Quinlan (1992), has support for numeric predicted values built-in. This system uses recursive partitioning to build a piecewise linear model, i.e. terminal leaves contain linear regression models. These linear regression models are based on predictors used in previous splits. This is explained in more detail by Quinlan (1992). Wang and Witten (1997) proposed a similar model that is implemented in the *Weka* machine learning framework (Hall et al. 2009) as *M5P*.

C5.0 (Quinlan 2013b) and *Cubist* (Quinlan 2013a) resemble the latest incarnation of the extended ID3 algorithm, with the former being applicable to classification and the latter being applicable to regression problems. *Cubist* models build on the concepts mentioned previously and add a couple of novel features.

Most notably, a cubist model can use a boosting-like scheme called *committees* where iterative model trees (trees whose leaves contain linear expressions) are created in sequence. Using this technique, many trees are constructed. The first tree is produced as usual. Any subsequent trees are however created such that they are based on an adjusted version of the outcome in the training set. If the outcome is overpredicted (e.g. the price for a train ticket is generally predicted too high), the target value of the training data fed to the subsequent tree learner is adjusted downward. The final prediction is simply the arithmetic mean of the predictions from each model tree. This effectively means that the prediction needs to be calculated for each tree, which has negative effects on the overall prediction time of the cubist algorithm. Additionally, the models constructed by the Cubist tree learner are deduced to simple if-then-else rules (Quinlan 1987; Quinlan 2013a). Whenever a case satisfies all the conditions, the linear formula is used for predicting the target value. Unlike other rule-based models, the average is taken for the final prediction in case two or more rules apply. Another feature in Cubist is the application of the k-nearest neighbor algorithm (Quinlan 1993b) to adjust the prediction from the model. First, a value is predicted as usual. Then cubist finds the k most similar cases in the training set and averages these outcomes.

3.2.4 Support Vector Machines

A Support Vector Machine (SVM) is a learning concept that has become very popular in recent years. Kernel-based methods such as SVMs use an implicit mapping from the input data into high dimensional kernel space defined by a kernel function. All learning is then done in this feature space. The way of mapping instances from a data set $\mathcal{D}$ into an inner product space V occurs without having to compute the mapping explicitly due to the instances gaining meaningful structure. This is called the *kernel trick* and is explained by Schölkopf and Smola (2001). The trick here is to use a learning algorithm that only requires dot products between vectors in the product space.

Figure 3.6a demonstrates a binary classification problem. The three bold lines represent possible separator candidates, and all of them clearly separate the black points from the white points. The question here is if these three lines are equally good. The lowest of the three lines comes very close to the black data points and may hence not be a good separator after all. SVMs address this issue by attempting to minimize generalization loss, i.e. by choosing the separator line that is farthest away from the data points. This separator is called the maximum margin separator and is shown in Figure 3.6b.

The question that arises here is what to do when the data is not linearly separable like in Figure 3.6b. Fortunately, when data is mapped into a space of sufficiently high dimension, then it will almost always be linearly separable (Russell and Norvig 2009, p. 746). Figure 3.7a depicts an input space where the black dots are not linearly separable from the white dots. The decision boundary, a circle, is also shown. Figure 3.7b shows this input space mapped into a higher-dimensinal space, where the data becomes linearly separable.

Additional literature on SVMs can be found in Vapnik (1995), Burges (1998), Cristianini and Shawe-Taylor (2000), Müller et al. (2001), and Bishop (2006, chap. 7). Originally, SVMs only supported classification problems, but methods have been developed so that they are also applicable to regression tasks (Smola and Schölkopf 2004).

3.2.5 Neural Networks

Another type of learning technique is called a Neural Network. This model is inspired by biological neural networks and has its origins in mathematical representations of information processing in biological systems (McCulloch and Pitts 1943; Rosenblatt 1962; Rumelhart, Hintont, and Williams 1986) and consists of interconnected artificial neurons.

Figure 3.8 depicts a mathematical model of a neuron as invented by McCulloch and Pitts (1943). The neuron fires when linear combinations of its inputs exceed a threshold. This linear combination is usually

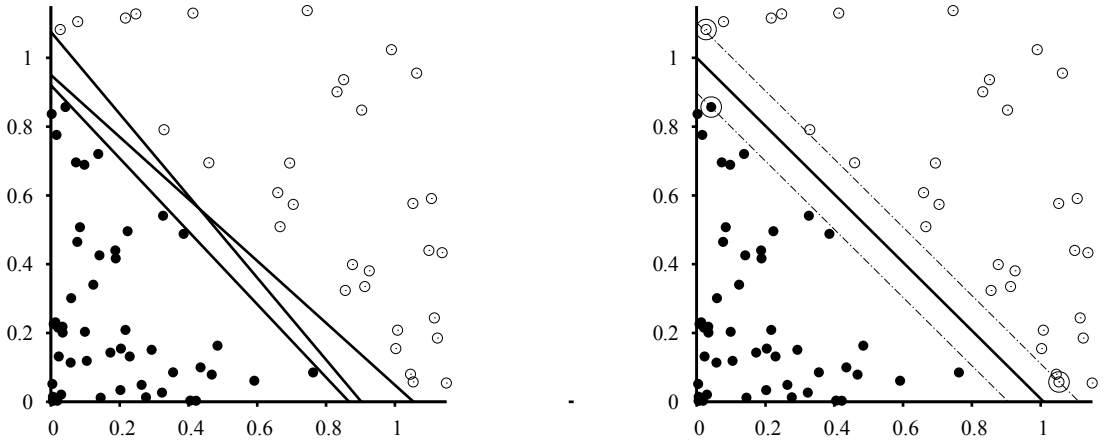

Figure 3.6.: Support Vector Machine classification as depicted by Russell and Norvig (2009, p. 745). In (a) there are three candidate separator lines that separate the two classes of points. In (b) the points are separated by the maximum margin separator which is at the midpoint of the area constructed by the two dashed lines. The circled points are the points that are closest to the separator and are called support vectors.

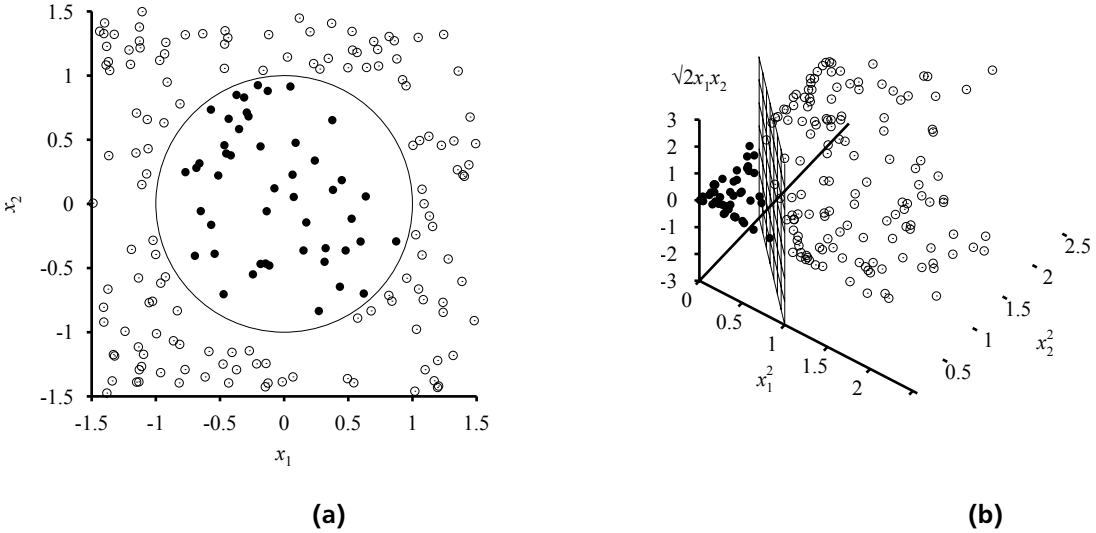

Figure 3.7.: Support Vector Machine classification as depicted by Russell and Norvig (2009, p. 747) for data that is not linearly separable. In (a) the true decision boundary between the positive and negative examples is a circle (which is non-linear). In (b) the data is mapped into a three-dimensional space. The circular boundary from (a) becomes linear.

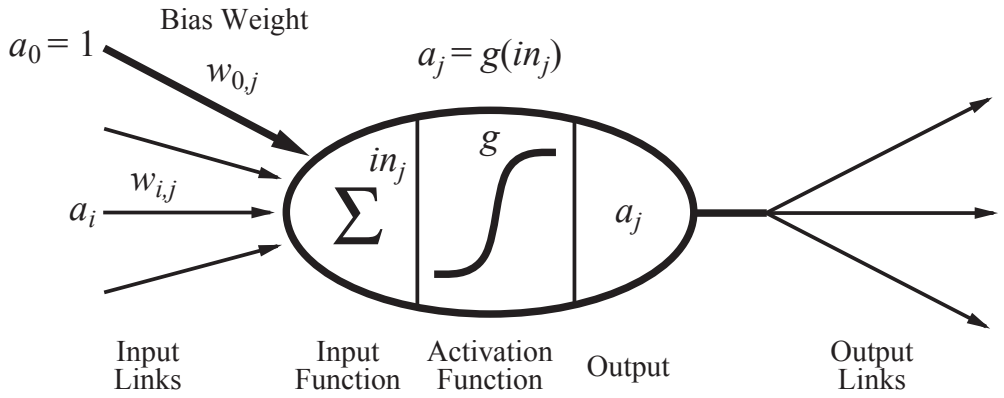

Figure 3.8.: A mathematical model for a neuron as depicted by Russell and Norvig (2009, p. 728). The output activation for this unit is given by $a_j = g(\sum_{i=0}^{n} wi_j a_i)$, where a_i is the output activation of the unit i and $w_{i,j}$ is the weight associated with the link from unit i to this unit.

transformed by a nonlinear function g. When connected together, these hidden *units* are called a neural network, connected by socalled *links*. Each link has a *weight* $w_{i,j}$, which determines the strength and sign of the connection. For the algorithm studied in this thesis, the activation function g is a hard threshold, in which case the unit is called a perceptron. In addition, the Multilayer Perceptron (MLP) is a feed-forward network, which means it has connections only in one direction and forms a directed acyclic graph.

Please see Bishop (2006, chap. 5), Russell and Norvig (2009, p. 727 – 736), Bishop (1995), and Reed and Marks (1998) for more details on the inner workings of neural networks.

3.3 Validation

After having trained models using the algorithms we have introduced in the last section, it is natural to ask how these techniques compare. What we are actually interested in is how these methods will perform on new data. Of course, we could just validate our model using the data it was trained with. But, that would be a methodical error, because the model will not be used to predict the data it was trained with – it will be used to predict data it has never seen before. This is especially important because we do not want to put algorithms at advantage that overfit the training data (see Section 3.2). When testing only on the training set, algorithms such as the k-nearest-neighbor algorithm (which encodes the training set) will rank highly because they yield excellent results on the training set. However, they will possibly perform very poor on new data due to overfitting.

Hence, we are interested in the performance of the algorithm when it is applied to data that was not part of the training set. It is usually assumed that there are separate test sets available, but this is not always the case. For this reason, we will now introduce some commonly-used methods for splitting an original data set $\mathcal{D}$ into two disjoint sets $\mathcal{D}_{\text{train}}$ for training and $\mathcal{D}_{\text{validate}}$ for validation. The process of validating a model using subsets of the original data is called *resampling*.

3.3.1 Train-Test Split

Given a data set $\mathcal{D}$ we split this into two disjoint sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{validate}}$ used for training and validation, respectively. The validation set is usually chosen to be smaller than the training set (e.g. 30% for the validation set and 70% for the training set). While this might work in some situations, we could just be unlucky and select a set that is not representative of the population it was sampled from.

The train-test method can be further improved by utilizing *stratified* random sampling. For numeric target values, this probabilistic method splits the data set $\mathcal{D}$ into n strata, i.e. segments based on n

percentiles. Then $\mathcal{D}_{i,\text{train}}$ and $\mathcal{D}_{i,\text{validate}}$ for $i \in \{1,\ldots,n\}$ is constructed in each segment using random sampling. The final train and validation set is given by the union of the respective sets in each of the n percentiles:

$$\mathcal{D}_{\text{train}} = \bigcup_{i}^{n} \mathcal{D}_{i,\text{train}} \tag{3.10}$$

$$\mathcal{D}_{\text{validate}} = \bigcup_{i}^{n} \mathcal{D}_{i,\text{validate}} \tag{3.11}$$

Another approach to stratified sampling can be utilized when the outcome of training samples highly correlates to some categorical attribute. For example, given a data set that contains school performance, we may want to split schools into rural, urban, and suburban and apply random sampling to each strata individually.

3.3.2 Cross-Validation

Cross-Validation (CV) describes a more sophisticated validation method and is very popular for machine learning. The original data set $\mathcal{D}$ is split into $k \in \mathbb{N}$ disjoint folds $\mathcal{D}_i$ for $i \in \{1,\ldots,k\}$ of the same size. Then, for each i, a learning algorithm is trained using $\mathcal{D} \setminus \mathcal{D}_i$, where $\setminus$ denotes the set difference. The instances not used for training are now in $\mathcal{D}_i$ and are used for validation. The final error is usually chosen to be the mean or median of the error metric used in each fold. This process is depicted in Figure 3.9. This is also the validation method we have chosen in this thesis. Tenfold cross-validation has become the standard method in machine learning, and extensive tests on numerous different datasets have shown that 10 is the right number of folds in order to get the best estimate of error (Witten, Frank, and Hall 2011, pp. 152 – 154). Kohavi (1995) also recommends tenfold cross-validation for model selection. Some authors prefer to repeat this process using a different seed in order to yield different folds in each iteration. Hence the name *repeated cross-validation*. There is also a special case of cross-validation called the *leave-one-out cross validation* (LOOCV). As the name implies, this is equal to cross validation using $k = |S|$, where $|S|$ is the size of the original data set. While this method can be used for small data sets, it is not applicable to larger data set such as ours due to performance reasons, i.e. it would require to train a model for each instance in the data set.

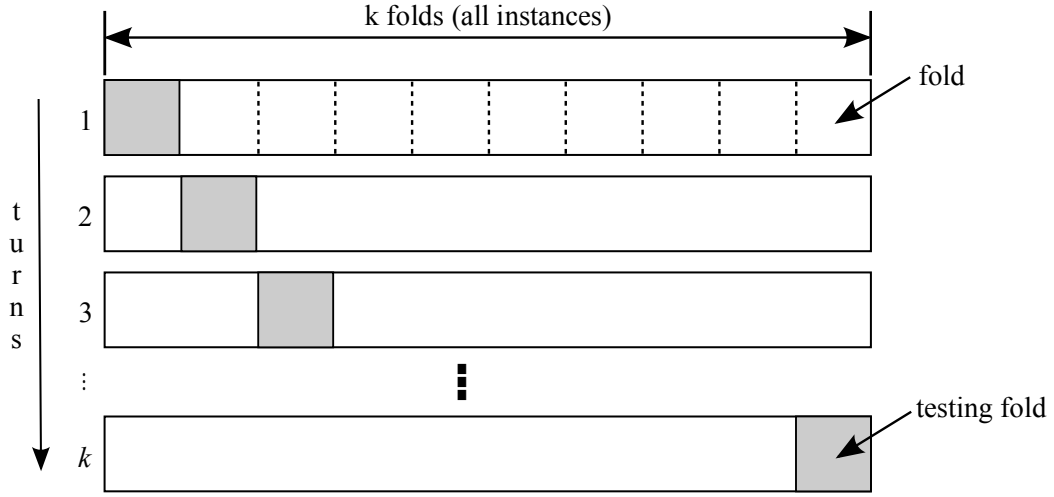

Figure 3.9.: Cross-Validation as depicted by Borovicka et al. (2012). The figure visualizes how the k folds are constructed in order to be used for validation. In each turn, a sample of data is partitioned into complementary subsets: the training set (white), and the test set (grey). These sets are then used for training and validation, respectively. By taking the mean of the results produced in each turn, an overall quality estimate can be provided.

3.4 Evaluation

In order to choose the model that best fits the problem at hand, different evaluation metrics can be utilized. Witten, Frank, and Hall (2011, p. 180) provide a very good overview of evaluation measures for regression problems. In the following, p refers to the predicted values, and a refers to the actual (observed) values.

The mean-squared error (MSE) is the most common error measure in machine learning settings, and is defined as:

$$\text{mse}(p,a) := \frac{(p_1 - a_1)^2 + \ldots + (p_n - a_n)^2}{n} \tag{3.12}$$

In order to give the mean-squared error (MSE) the same dimension as the predicted value, the square root can be taken, thus yielding the root mean-squared error (RMSE):

$$\text{rmse}(p,a) := \sqrt{\text{mse}(p,a)} \tag{3.13}$$

The mean-absolute error (MAE) works similarly:

$$\text{mae}(p,a) := \frac{|p_1 - a_1| + \ldots + |p_n - a_n|}{n} \tag{3.14}$$

The relative-squared error (RSE) describes something quite different. It puts the squared-error into relation to the error produced by just using the arithmetic average as prediction:

$$\text{rse}(p,a) := \frac{(p_1 - a_1)^2 + \ldots + (p_n - a_n)^2}{(a_1 - \overline{a})^2 + \ldots + (a_n - \overline{a})^2} \tag{3.15}$$

Analogously to the root mean-squared error (RMSE), the root relative-squared error (RRSE) just takes the square root of the RSE:

$$\text{rrse}(p,a) := \sqrt{\text{rse}(p,a)} \tag{3.16}$$

The relative-absolute error (RAE) is just the normalized total absolute error.

$$\text{mae}(p,a) := \frac{|p_1 - a_1| + \ldots + |p_n - a_n|}{|a_1 - \overline{a}| + \ldots + |a_n - \overline{a}|} \tag{3.17}$$

Another measure often used to assess how good a model fits a problem is the correlation coefficient and is described in Equation (B.6).

The RMSE is probably the most common error in machine learning. Compared to the MAE, it amplifies and severely punishes large errors, i.e. being off by 10 cents is *more* than twice as bad as being off by 5 cents. Unlike the correlation coefficient (R^2) the MAE and the RMSE are fairly easy to explain. Furthermore, because the MAE is a linear measure its meaning is more intuitive. The correlation coefficient measures the statistical correlation between the predicted and the actual values and is not necessarily a good option when the MAE and RMSE are available, because the error produced is a much better assessment as far as ranking different machine learning methods for the prediction of railway fares is concerned. In this thesis we have opted to use all error metrics except the non-normalized metrics (MSE and RSE). However, in Section 3.4, the MAE is used for the final ranking of the algorithms, because we did not want to punish outliers.

4 Data Preparation

Before training various machine learning algorithms, a data set is needed. For the problem at hand, no existing data set was available. Since machine learning is an iterative process, we have taken more than one iterations as far as sampling is concerned.

In the first iteration of the learning process, random connections where the source and destination train station had an equal chance of being chosen over a time span of one month were sampled. This resulted in connections that were very unlikely to be planned by actually humans (i.e. connections from a station in the middle of nowhere to a station in the middle of nowhere). In the second iteration, we tried to sample only a week worth of data, because our research has shown that the week number has little influence on the ticket price. Still, the problem that this approach resulted in a data set that was out of touch with reality remained. In the third and final iteration, we proposed a sampling method that prefers train stations with a high number of incoming and outgoing connections. This method of sampling is described in Section 4.1 and is the basis for the data set S_1. This data set is supplemented the data set S_2, which was sampled using a recording of train journeys booked by actual humans.

4.1 Sampling

Generally speaking, sampling describes the act of taking a subset of instances from a given population. In our case, we generate connections (train journeys from source to destination train stations) using MOTIS and the BPC. Because sampling plays a crucial part in the machine learning process, we need to think about *what* to sample. Fortunately, we are in a position to sample as much data as we like. But this does not come without risks. It is quite easy to sample data that is a poor fit for the problem at hand. After all, we need to sample railway connections that are at least close to train journeys humans would like to endeavor in reality, because MOTIS exists for exactly that purpose. Thus, the optimal situation would be to mimic a human that utilizes a system to plan train rides. Of course, this is harder than it sounds. As stated in the Chapter 1, predictive modeling is not a linear process. This is especially true for sampling.

Our final solution to this problem assigns each tration station a normalized probability of being chosen as source or destination train station in the sampling process. Because we are able to make use of background knowledge that describes the importance of a train station in terms of incoming and outgoing connections for each station, probability sampling can be used. These samples are selected in such a way as to be representative of what would be chosen by actual humans, therefore providing credible results.

Each train station s_i, $i \in \{1, ..., n\}$ is assigned the value w_i of the application of a weight function

$$\text{weight} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N} \tag{4.1}$$

$$\text{weight}(c_0, c_1, c_2, c_{\text{rbre}}) := 6c_0 + 5c_1 + 4c_2 + 1c_{\text{rbre}} \tag{4.2}$$

where c_0, c_1, c_2 and c_3 describes the number of incoming and outgoing train connections for express trains (class 0), inter-city trains (class 1), and inter-regio trains (class 2), respectively. The last argument corresponds to the number of connections for regional trains (RB and RE). Do note the absence of class 3. The number for this class does not contribute to the likelihood of being chosen because MOTIS currently does not handle commuter railway systems like the German Tram. The factors associated with each c were chosen such that connections for faster trains are more important than those from slower trains, i.e. the factor 6 for ICE trains makes this type of train more important than a RB train with factor 1. After we

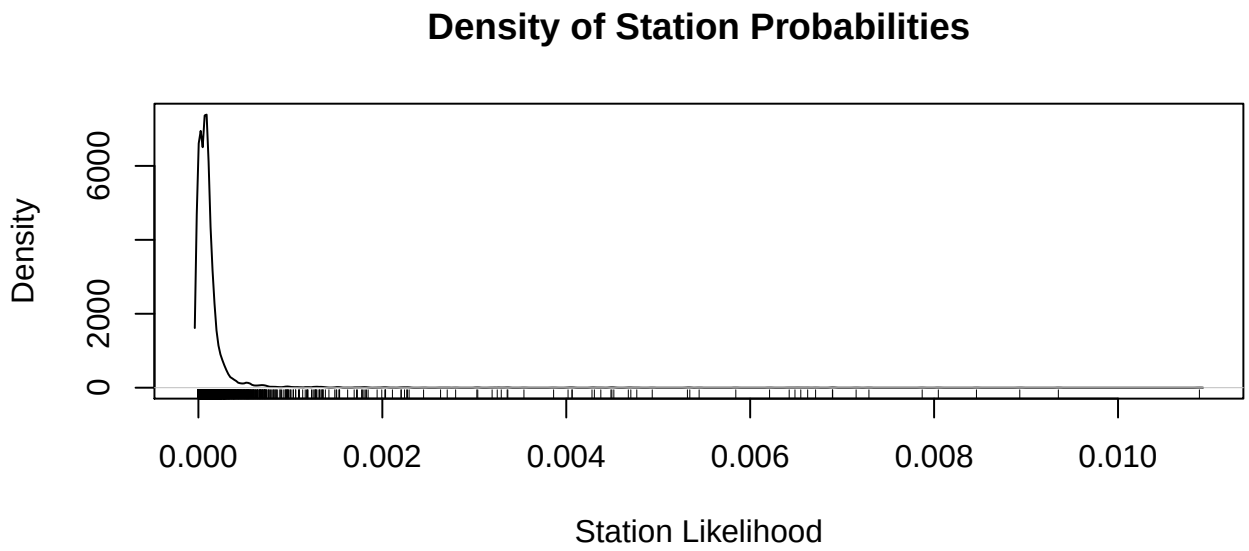

Figure 4.1.: Density and rug plot of station probabilities for the probability-based sampling technique. The plot shows two peaks, one at 0 and one just above 0, representing negligible and slightly insignificant stations, respectively.

have calculated the weight w_i for each station, the probability P_i can be inferred by the application of the probability function

$$P_i : \mathbb{N} \to [0, 1] \tag{4.3}$$

$$P_i(w) := \frac{w}{\sum_{i}^{n} w_i}, \quad \text{with} \sum_{i=1}^{n} P_i = 1 \tag{4.4}$$

where w denotes the weight of the station in question. The denominator corresponds to the sum of the weights of all stations. Figure 4.1 visualizes the computed kernel density estimate, a non-parametric way to estimate the probability density function of a random variable. Upon examination of this plot, it becomes clear that there are two peaks: One is at 0 and correlates to stations with a weight of 0. We do not want to plan routes from or to these stations because they serve only commuter trains (Trams) that MOTIS does not handle. The second peak is just a little above 0. These are insignificant stations with a very low number of regional trains. The rug plot at the bottom reveals that there are indeed a couple of very important stations that should not be neglected when taking samples.

Table 4.1 ranks the 25 most important railway stations according probablity-based technique, sorted by their weights and probabilities. This confirms that the resulting data will indeed include the usual suspects, i.e. major central railway stations (Hbf). Analogously, Table 4.2 ranks 10 of the least important stations with a probability above 0. The data set produced by this sampling method will henceforth be named S_1 in this thesis.

In addition to this probability-based technique, another form of background knowledge was integrated in a later stage. This background knowledge, which was provided by Deutsche Bahn, consists of a live recording of bookings done by humans in one single day. The data set built upon this information is named S_2 in this thesis. As a result, the final data set $S_3 = S_1 \cup S_2$ consists of 7000 connections sampled using the probability-based technique, and 7000 connections sampled using a live recording of bookings done by actual humans. The resulting data set contains many connections from one major train station to another, in addition to some not-so-common routes, and this is indeed a very realistic scenario. The departure time and date were chosen at random within one single day for the construction of the data sets S_1 and S_2.

Rank	Name	Weight	Probability
1	Hannover Hbf	5691	0.010885
2	Köln Hbf	4890	0.009353
3	Frankfurt(Main)Hbf	4670	0.008932
4	Düsseldorf Hbf	4424	0.008462
5	Hamburg Hbf	4207	0.008047
6	Duisburg Hbf	4114	0.007869
7	Mannheim Hbf	3811	0.007289
8	Berlin-Spandau	3740	0.007153
9	Dortmund Hbf	3607	0.006899
10	Nürnberg Hbf	3605	0.006895
11	F-Flughafen Fernbf.	3509	0.006712
12	Würzburg Hbf	3463	0.006624
13	Göttingen	3424	0.006549
14	Kassel-Wilhelmshöhe	3392	0.006488
15	Fulda	3358	0.006423
16	Hamburg Dammtor	3248	0.006212
17	Essen Hbf	3055	0.005843
18	Hamm(Westf)	2847	0.005445
19	Bochum Hbf	2794	0.005344
20	Hamburg-Harburg	2780	0.005317
21	Augsburg Hbf	2579	0.004933
22	Stuttgart Hbf	2492	0.004766
23	Mainz Hbf	2458	0.004701
24	Bielefeld Hbf	2444	0.004675
25	München Hbf	2361	0.004516

Table 4.1.: Weight and probability of the 25 most important stations for the probability-based sampling technique. The probability corresponds to the probability of being chosen as source or destination train station during sampling.

Rank	Name	Weight	Probability
5422	Freudenstadt Schulen	1	0.000002
5422	Rheinzabern Römerst	1	0.000002
5422	Rheinzabern Rappeng	1	0.000002
5422	Bilfingen	1	0.000002
5422	Hamburg Diebsteich	1	0.000002
5422	Zwingenberg(Baden)	1	0.000002
5422	Ersingen	1	0.000002
5422	Wüsting	1	0.000002
5422	Bremen-Mahndorf	1	0.000002
5422	Marienberg(Sachs)	1	0.000002

Table 4.2.: Weight and probability of 10 of the least important stations for the probability-based sampling technique.

4.1.1 Black Box Communication

The Multi-Objective Traffic Information System (MOTIS) can be communicated with using simple network sockets and XML requests. In order to determine the fare cost for a train journey from a source train station to a destination train station, an XML query must be sent to MOTIS.

An exemplary price query is depicted in Listing 2. In this query, the time in which the first train in a connection departs is requested to be between 10am (line 3) and noon (line 4) and specified in ISO 8601 format [1]. The source (line 8) and destination (line 12) stations are specified in the `PathDescription` tree using unique ID numbers.

Listing 3 shows a trimmed down version of the response given by MOTIS with irrelevant fragments such as debugging data or error codes removed. The response contains exactly one connection, and lists a number of intermediate stops between the source and destination train stations. MOTIS usually provides a number of alternative connections, all of which are recorded and put into the data set in our sampling approach.

```
1  <Query>
2    <Interval definitionFor="departure">
3      <Begin dateTime="2013-04-23T10:00"/>
4      <End dateTime="2013-04-23T12:00"/>
5    </Interval>
6    <PathDescription>
7      <Via number="0"/>
8      <Station EvaNo="8000068"/>
9      <Section categories="0">
10       <AttributeList/>
11     </Section>
12     <Station EvaNo="8000152"/>
13   </PathDescription>
14   <Customer BahnCard="none"/>
15   <AdditionalOptionList/>
16 </Query>
```

Listing 2: Example XML Request to be sent to MOTIS. In this query, we request all connections from Darmstadt Hbf (8000068) to Hannover Hbf (8000152) between 10:00 and 12:00.

4.1.2 Attribute Identification and Computation

In this subsection, we will identify the attributes that can be extracted directly from the response depicted in Listing 3. As many useful features as possible are extracted or computed – even if they are unusable by machine learning algorithms. This is especially useful for debugging purposes (i.e. source and destination train station names for a particular connection). Some of these attributes are removed in Section 4.2 prior to feeding different machine learning algorithms. The following features can be extracted directly from a response:

- The departure **date and time** (line 5) is stored in ISO 8601 format and allows to infer information such as the weekday and time of the day on the fly.
- The journey **duration** (line 1) in minutes describes the time spent traveling between the source and destination station.

[1] ISO 8601:2004, Information interchange – Representation of dates and times

```
1  <Connection connectionId="2" Duration="167" InitialWaitingTime="0" Transfers="1"
2              Date="23.04.2013" SleepTime="0" PQOps="0" PQSize="0" NumOfQueryPQOps="0">
3    <StopList>
4      <Stop evaNo="8000068" name="Darmstadt Hbf">
5        <Departure dateTime="2013-04-23T11:30" platform="7"/>
6      </Stop>
7      <Stop evaNo="8003523" name="Langen(Hess)">
8        <Arrival dateTime="2013-04-23T11:38" platform="1" minStanding="1"/>
9        <Departure dateTime="2013-04-23T11:39" platform="1"/>
10     </Stop>
11     <Stop evaNo="8000105" name="Frankfurt(Main)Hbf">
12       <Arrival dateTime="2013-04-23T11:48" platform="12"/>
13       <Departure dateTime="2013-04-23T11:58" platform="8"/>
14       <InterchangeInfo needed="8" buffer="2" waiting="0" security="68"/>
15     </Stop>
16     <Stop evaNo="8003200" name="Kassel-Wilhelmshöhe">
17       <Arrival dateTime="2013-04-23T13:20" platform="3" minStanding="2"/>
18       <Departure dateTime="2013-04-23T13:22" platform="3"/>
19     </Stop>
20     <Stop evaNo="8000128" name="Göttingen">
21       <Arrival dateTime="2013-04-23T13:41" platform="9" minStanding="2"/>
22       <Departure dateTime="2013-04-23T13:43" platform="9"/>
23     </Stop>
24     <Stop evaNo="8000152" name="Hannover Hbf">
25       <Arrival dateTime="2013-04-23T14:17" platform="7"/>
26     </Stop>
27   </StopList>
28   <JourneyInfo>
29     <Transport name="RB 15352" categoryID="12" categoryName="RB" number="15352"
30                from="0" to="2" distance="30.1678"/>
31     <Transport name="ICE 76" categoryID="1" categoryName="ICE" number="76"
32                from="2" to="5" distance="156.084"/>
33   </JourneyInfo>
34   <Price PAErrorCode="0" cost="7900" estimate="7546"/>
35 </Connection>
```

Listing 3: Example XML Response produced by MOTIS in response to the query listed in Listing 2. The response can contain multiple connections between the requested train stations, but is reduced to only one connection here.

- The **transfer count** (line 1) describes how many times a train change is required.
- The **price** in cents (line 34) is the attribute to be learned.
- The **estimated price** in cents (line 34) is current price prediction. Please consult Chapter 2 for details.
- The first stop in the `Stop` tree (line 4) describes the **ID** of the **source** train station.
- Similarly, the last stop in the `Stop` tree (line 24) describes the **ID** of the **destination** train station.

Additionally, the following attributes can be computed from a black box response:

- The number of **stops**; calculated according to the number of `Stop` children.
- The value of the **distance** attribute for each of the `Transport` children (line 30 and 32) describe the distance traveled in the respective train category. Using background knowledge, `categoryID` is

mapped to a category class. For example, in Listing 3 `categoryID` 12 is mapped to category class 3 and `categoryID` 1 is mapped to category class 0. Table 4.4 provides a mapping from category IDs to classes and, for some IDs, the corresponding train type in parentheses. The distance traveled for each category class is then summed up. Thus the attributes **dist_0** to **dist_8** describe the cumulative distance traveled in the respective class. It is assumed that trains belonging to the same category class have a similar effect on the final price of the journey.

Using background knowledge (i.e. the name and coordinates of a station), the following attributes can also be inferred.

- The **name** of the **source** train station. This serves only debugging purposes.
- Analogously, the **name** of the **destination** train station. This also serves only debugging purposes.
- The **linear distance** between the source and destination train station. Given coordinates in the geographic coordinate system (longitude and latitude), the great-circle distance is calculated as follows: Let (ϕ_s, λ_s) and (ϕ_d, λ_d) be the source and destination train station's latitude and longitude, respectively. The formula known as the haversine formula (Robusto 1957) is used to calculate the great-circle distance:

$$\Delta\widehat{\sigma} = 2\arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_s \cos\phi_f \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{4.5}$$

  The distance is then converted to kilometres, hence making it comparable to the values of the distance attributes.

  This is the standard way of computing distances on the surface of earth, and this attribute was initially calculated because the distance attributes provided by MOTIS were incorrect. When we added this attribute to our data set, the models learned produced much better results. When the problem in MOTIS was fixed by its authors, we decided to keep this attribute because the removal led to slightly worse results. This behavior is discussed in Chapter 7 for some algorithms.

Table 4.3 summarizes all of the attributes presented above in the order they appear in the database. Do note that not all attributes are used for the prediction of railway fares.

4.1.3 Missing Class Values

The sampling method introduced in Section 4.1 does produce very rare cases of missing class values. For example, MOTIS fails to provide the correct price for a connection under certain circumstances, i.e. a connection that consists solely of commuter trains operated by third party train operators. For the problem at hand, we have chosen to drop instances that contain missing class values altogether. Instances without prices serve no purpose for either training or validation, and is the only viable option here. As far as missing values are concerned, no special treatment is necessary because they simply do not occur.

Name	Description
estimate	the current estimate
duration	total journey duration
transfers	number of transfers
stops	number of stops
dt	the date and time of the departure from the source station
source_eva	the id number of the source station
source_name	the name of the source station
dest_eva	the id number of the destination station
dest_name	the name of the destination station
dist_0	sum of distances traveled on train class 0
⋮	⋮
dist_8	sum of distances traveled on train class 8
dist	sum of **dist_0** to **dist_8**
lindist	great-circle distance between source and dest station
estimate	journey price estimate in cents
price	journey price in cents

Table 4.3.: Summary of all identified or computed attributes.

Class	Category IDs
0	1 (ICE), 2, 29, 30, 51, 52, 58, 63, 64
1	3 (EC), 4 (IC), 5, 8, 46, 49, 61, 62
2	6 (D), 7, 9 (IR), 18, 26, 27, 28, 31, 32, 40, 43, 56, 59
3	10, 11, 12 (RB), 13 (RE), 14, 16, 21, 24, 42, 44, 45, 47, 48, 53, 54, 55, 57
4	15, 19, 20, 23 (Bus), 25, 33, 34, 35, 39, 50
5	22, 36, 60, 65
6	17, 37, 38, 41, 66

Table 4.4.: Mapping from category ID to category class.

4.2 Data Analysis

Before comparing and ranking different machine learning algorithms, it is always a good idea to get to know your data. Descriptive statistics can be used to quantitatively describe the main features of the data generated in Section 4.1. Table 4.5 provides an overview of commonly used statistical measures applied to the numeric attributes of the generated data set. `min` and `max` represent the minimum and maximum values in the data set, respectively. The `mean` (formula B.1), `median` (formula B.2) and standard deviation `sd` (B.4) are common metrics in statistics. The formulae of all these metrics can be found in the appendix.

The second column in this table (n) corresponds to the number of valid values for each attribute. The fact that it is 14000 for each attribute means that there are no invalid values in our data set (e.g. missing values). In the first iteration in our learning process, the median for the linear distance `lindist` was well below the accumulated sum `dist`. This is of course impossible given that the linear distance is the shortest distance between two points, but it does emphasize the importance of a simple analysis. It turned out to be an error in the calculation done by MOTIS and was promptly fixed, henceforth causing our predictive models to yield much better results.

Aside from that, particularly striking is the fact that the standard deviation (s) of `dist_4` to `dist_8` is equal to 0. This suggests that the values of all instances for these attributes are 0. We will see in the next section that these attributes can be removed altogether.

	n	min	max	mean	median	sd
duration	14000	3	1316	316	288	182
transfers	14000	0	7	2	2	1
stops	14000	2	88	20	18	12
dist_0	14000	0	995	174	77	213
dist_1	14000	0	1318	112	0	173
dist_2	14000	0	822	3	0	36
dist_3	14000	0	731	115	86	111
dist_4	14000	0	0	0	0	0
dist_5	14000	0	0	0	0	0
dist_6	14000	0	0	0	0	0
dist_7	14000	0	0	0	0	0
dist_8	14000	0	0	0	0	0
dist	14000	0	1641	405	387	217
lindist	14000	2	830	275	261	145
estimate	14000	39	15880	7019	7089	3364
price	14000	130	13460	6892	7000	3240

Table 4.5.: Descriptive Statistics of the continuous attributes in our data set.

In addition to simple descriptive statistics, we will now take a look at the character of the data in question, particularly at the price attribute. Figure 4.2 shows a kernel density plot (a non-parametric way to estimate the probability density function of a random variable) in red superimposed on a histogram. The plot suggests that there is some kind of maximum price after a certain criterion (e.g. a maximum distance or cut-off price) has been met. Interestingly, there are exceptions to this rule, because the histogram contains a tiny bar at the right hand side. Upon inspection of this behavior, it becomes clear that this exception is due to connections which include trains of privately held train companies. For example, A journey from Ulm to Süderdeich includes riding a train owned by NBE (Nordbahn Eisenbahngesellschaft mbH). In addition, the price attribute in the data set S_3 has 744 unique values. This indicates that the problem at hand is truly a regression problem – as opposed to a classification problems with only a handful of classes.

4.2.1 Zero and Near-Zero Variance Predictors

As we have seen in the previous section, our data generation mechanism creates predictors that have a single unique value only. These predictors are called *zero-variance predictors*. Similarly, there may be predictors that have values that occur with very low frequency. There is a pretty good chance that these *near-zero-variance predictors* become zero-variance predictors when our data set is split into sub-samples for cross-validation (see Subsection 3.3.2 for an introduction to validation methods). With the problem at hand, we do not want these predictors to be fed to machine learning algorithm, because they cannot be used to differentiate instances. In order to identify and then remove these attributes from our data set, the following two metrics are calculated (Kuhn 2013, p. 62):

- **Frequency Ratio**: The frequency of the most prevalent value over the second most frequent value. Well-behaved predictors have a frequency ratio near 1, while highly-unbalanced predictors have large ratios.

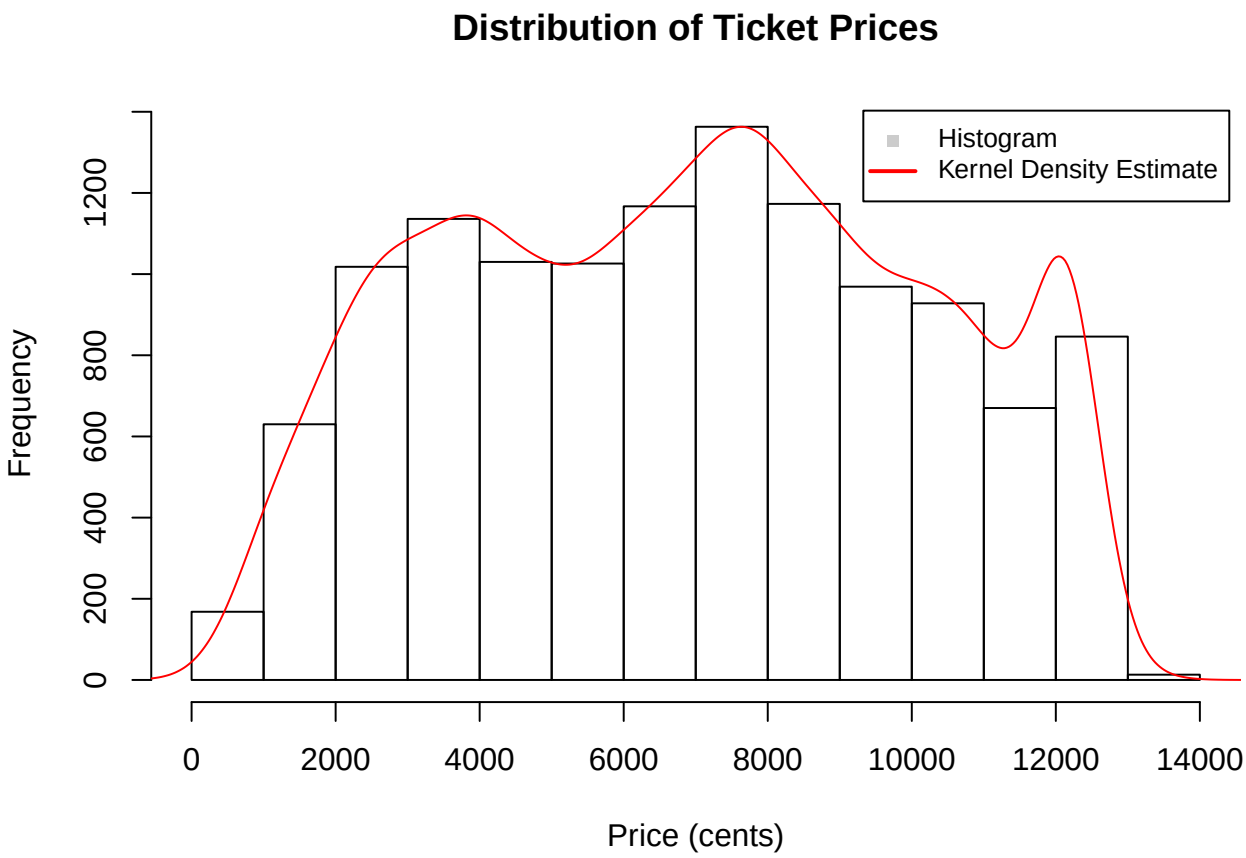

Figure 4.2.: A kernel density plot (red) superimposed on a histogram. The density estimate suggests the existence of a maximum price at around € 120.

- **Percent Unique**: The number of unique values divided by the number of instances times 100.

In order to be classified a zero-variance predictor, the value for a predictor must be the same across all instances. To be flagged a near-zero-variance predictor, the frequency ratio and the unique percentage must be above `freqCut` and below `uniqueCut`, respectively.

In this thesis, we have used freqCut := 20 and uniqueCut := 5. We have chosen these values because we did not want to lose either dist_0 or dist_1, which have a frequency ratio just above 20. This method was merely used to identify attributes that have potential for removal, and the parameters were specified such that dist_2 and dist_4 to dist_8 are removed. In the former case, there are hardly any trains of category 2 present, and in the latter case, there are no trains present. Table 4.6 demonstrates this technique as it is applied to our database. Predictors that are `TRUE` for either case are removed automatically.

4.2.2 Attribute Correlation

Another interesting aspect of our data is the analysis of how different attributes correlate to each other. For this purpose, the Pearson product-momentum correlation coefficient r is utilized. This coefficient is a measure that assesses the degree of linear relationship between two variables, and yields a value in $[-1, 1]$, where $|r| = 1$ implies a perfect linear relationship between two variables and $r = 0$ no linear relationship. A positive value suggests a positive relationship, whereas a negative value denotes a negative relationship. For details on how to calculate this measure see formula B.6.

Figure 4.3 contains the correlation coefficients for all numeric attributes for our data set in the upper half. In the lower half, these coefficients are visualized using colored ellipse-shaped glyphs for each entry. The level of smoothing corresponds to the level of correlation, as does the color. Blue and red mean a

	Frequency Ratio	Percent Unique	Zero Variance	Near-Zero Variance
transfers	1.05	0.07	FALSE	FALSE
stops	1.05	0.64	FALSE	FALSE
dt	1.11	27.49	FALSE	FALSE
dist_0	76.42	21.98	FALSE	FALSE
dist_1	150.46	20.37	FALSE	FALSE
dist_2	1000.08	0.42	FALSE	TRUE
dist_3	75.05	65.72	FALSE	FALSE
dist_4	0.00	0.01	TRUE	TRUE
dist_5	0.00	0.01	TRUE	TRUE
dist_6	0.00	0.01	TRUE	TRUE
dist_7	0.00	0.01	TRUE	TRUE
dist_8	0.00	0.01	TRUE	TRUE
dist	1.14	91.83	FALSE	FALSE
lindist	1.00	34.92	FALSE	FALSE
estimate	1.23	59.74	FALSE	FALSE
price	4.07	4.33	FALSE	FALSE

Table 4.6.: Zero And Near-Zero Variance Predictors. The attributes distance 4 to 8 are zero variance predictors because none of our samples contain connections which include trains of these categories (MOTIS does not support these). Distance 2 is a near-zero variance predictor due to trains being very scarce in this category nowadays. It is hence safe to remove these predictors before applying machine learning algorithms.

positive and negative correlation, respectively. Darker colors correspond to a higher relationship and are closer to $|c| = 1$.

Using these two items, there are indeed some traits that are worth exploring. The high correlation (0.90) between `dist` and `price` means that the higher the actual distance traveled, the higher the price. This could of course be deemed obvious, but this conclusion depends highly on background knowledge. Not so obvious traits include the positive relationship between `price` and `dist_0` as well as `price` and `dist_3`. These relationships suggest that it is more expensive to travel on category class 0 than it is on category class 3 (for details on train categories see Table 4.4).

Figure 4.4 shows four scatter plots that draw the distance traveled on the horizontal axis and the ticket price on the vertical axis using the data set S_3. The plots are smoothed using a kernel density estimate, and darker regions correspond to a greater number of data points. The first scatter plot contains all connections, i.e. connections that are composed of trains of different types. The other three plots contain only connections that consist solely of one type of train category, i.e. ICE, IC/EC, and regional trains. Harnisch and Nuhn (2010) have already explored the possibility to split up the data into subgroups, each containing only connections for one train category. Especially interesting is the fact that for IC/EC-only connections, a polynomial regression scheme is likely to fit the data quite reliably with only a small number of outliers. Outliers in the IC/EC plot include connections from Wuppertal Hbf to Berlin Südkreuz, and from Weimar to Mannheim. The reason for their existence is unclear. The case is quite similar with connections that consist solely of regional trains (RegionalBahn and RegionalExpress). However, the reason for the existence of outliers is quite clear here. The outliers are due to the involvement of third party train operators such as AKN Eisenbahn, or the Usedomer Bäderbahn (UBB), a train operator that serves the Baltic Sea island Usedom. This also explains the fact that outliers appear mostly above a potential polynomial function, because the involvement of a third party train operator increases the total ticket price. As far as ICE connections are concerned, a simple regression scheme is very likely to produce subpar results, because the scatter plot cannot be described by a simple function.

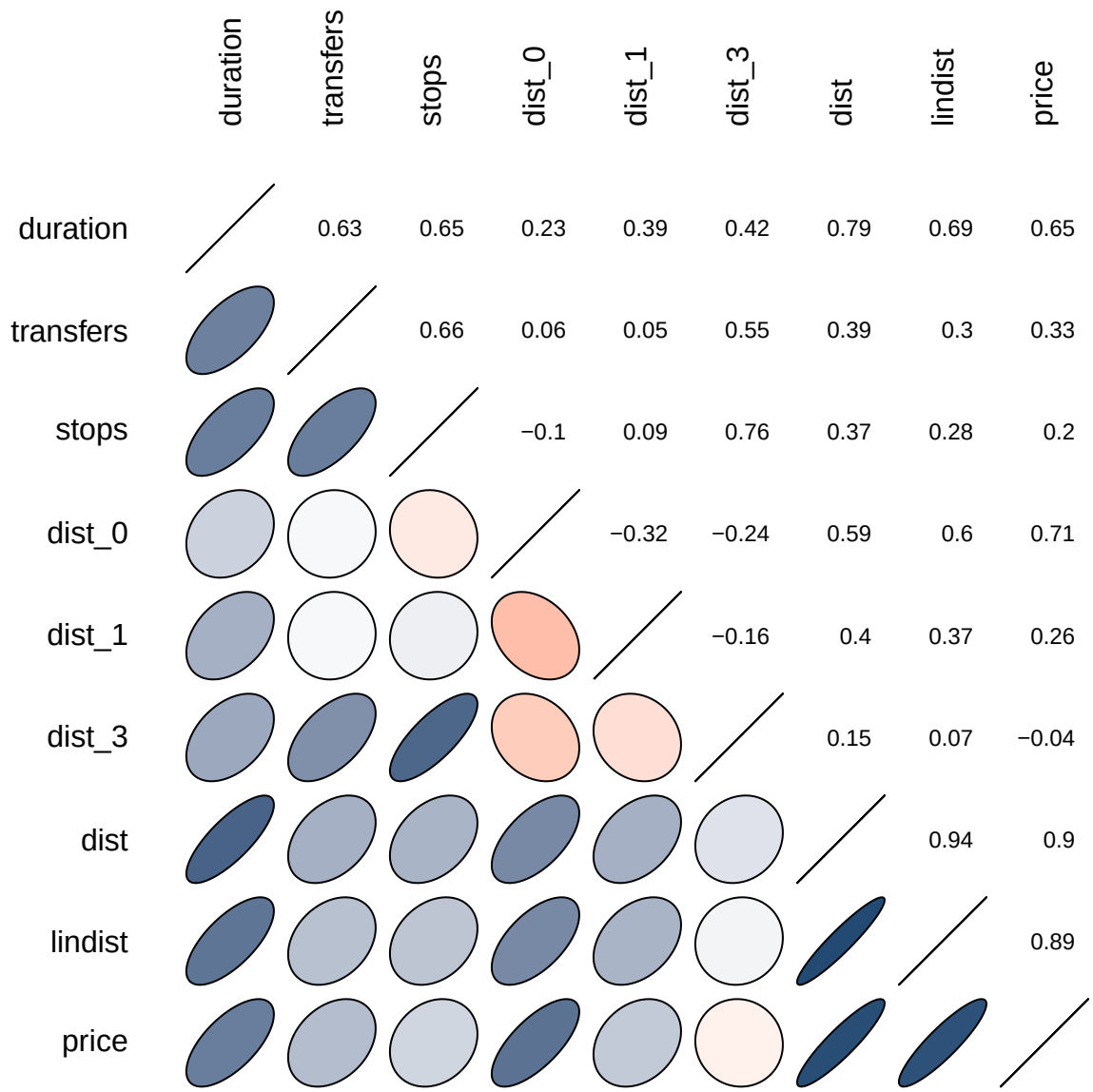

Figure 4.3.: Correlation coefficients for all continuous attributes. The upper half contains the coefficients, where $|c| = 1$ implies that a linear equation describes the relationship perfectly. The lower half contains a visualization of these coefficients.

4.3 Example Data

In order to get an adequate feeling of the actual data, we will now look at a subset of the data in question. Table 4.7 shows a random subsample of the data we have generated and examined in the last section. Some of these attributes, such as the source and destination name, serve only debugging purposes. The `dt` attribute will be transformed to represent only the day of the week and the hour of the day. The attribute `estimate` is the estimate generated by the current prediction method (see Chapter 2), and will be removed from the data set prior to the application of learning algorithms. As expected, important train stations such as central train stations (Hauptbahnhof) are overrepresented in this random subsample.

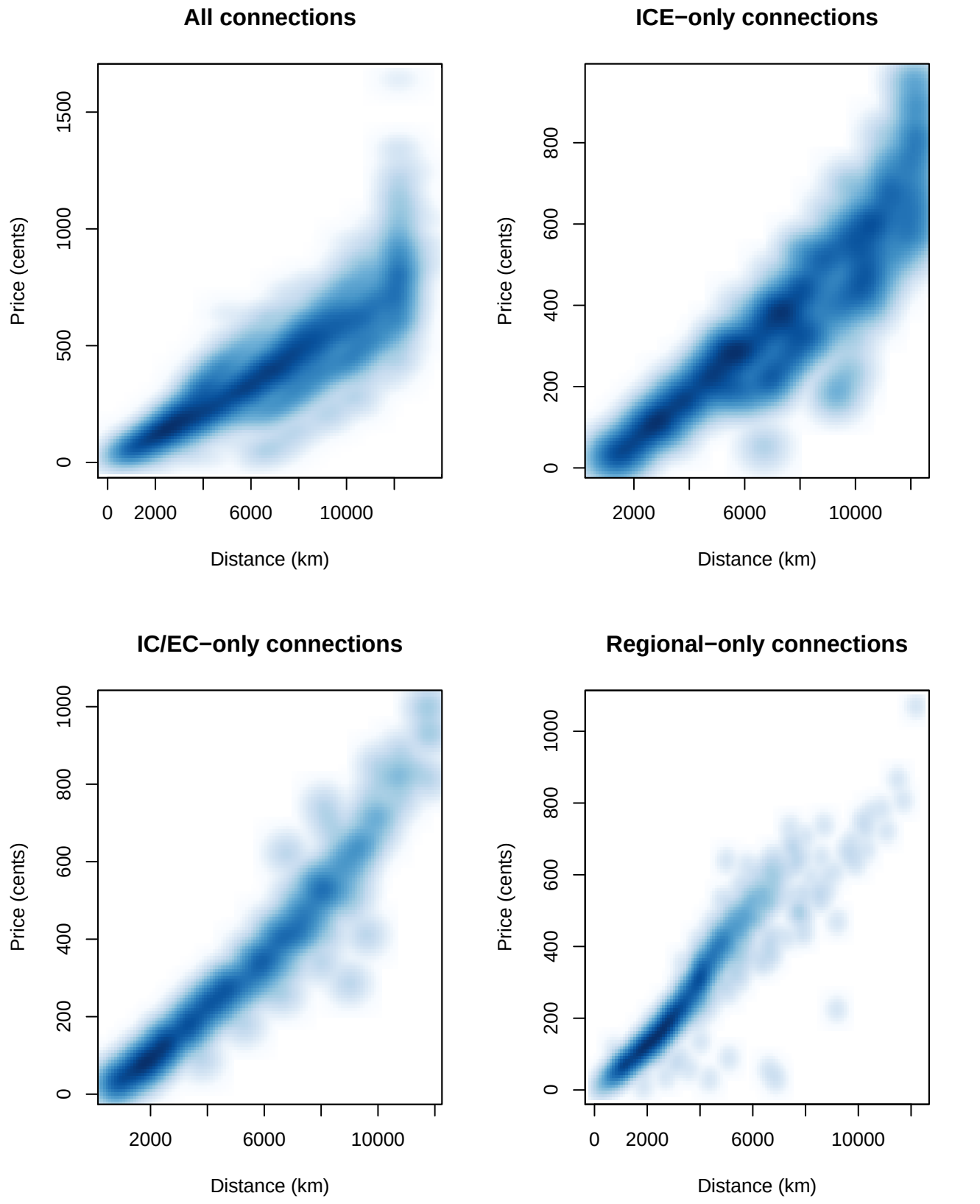

Figure 4.4.: Scatter plots that show the relationship between the distance and ticket price. The plots are smoothed using a kernel density estimate.

duration	transfers	stops	dt	dist_0	dist_1	dist_3	dist	lindist	estimate	price	source_name	dest_name
340	3	31	2013-04-25T07:07:00	303	0	245	547	381	9539	9400	Hersbruck(r Pegnitz)	Münster(Westf)Hbf
196	2	21	2013-04-26T17:28:00	0	0	279	279	191	3892	3600	Bremerhaven Hbf	Wolfsburg Hbf
379	4	40	2013-04-28T10:51:00	0	0	235	235	168	3271	3330	Ellwangen	Nidderau-Eichen
314	4	20	2013-04-28T16:10:00	112	0	245	356	213	6210	6700	Redwitz(Rodach)	Herborn(Dillkr)
356	3	25	2013-04-29T02:07:00	290	0	41	331	247	7040	7160	Finkenkrug	Ellerau
556	4	25	2013-04-22T20:13:00	0	517	15	532	290	8274	7700	Frankfurt Hbf (tief)	Barnstorf(Han)
266	2	13	2013-04-29T06:09:00	0	304	20	324	282	5838	5800	Hamburg-Harburg	Kablow
533	4	38	2013-04-25T07:16:00	0	0	431	431	340	6005	5370	Rohrbach(Oberbay)	Leipzig Hbf
87	1	9	2013-04-23T09:21:00	0	0	131	131	97	1824	2070	Hagen Hbf	Paderborn Hbf
223	1	15	2013-04-28T19:12:00	0	235	64	299	241	5001	5400	Stuttgart Hbf	Rosenheim
466	3	22	2013-04-25T03:52:00	180	206	166	552	387	10644	9800	Frankfurt(Main)Hbf	Passau Hbf
176	1	33	2013-04-27T09:07:00	0	0	156	156	116	2238	2510	Kassel-Wilhelmshöhe	Erfurt Nord
329	3	27	2013-04-26T11:35:00	43	317	86	446	332	7716	7700	Darmstadt Hbf	Dessau Süd
375	3	42	2013-04-28T02:57:00	0	270	112	382	210	6284	6300	Mannheim Hbf	Garbeck
317	3	21	2013-04-25T12:28:00	0	466	78	543	412	8369	7800	Berlin Friedrichstr	Haren(Ems)
546	3	16	2013-04-27T22:17:00	200	0	37	477	404	8731	10100	München Harras	Hattorf
354	1	13	2013-04-24T10:45:00	548	0	0	548	490	10443	10500	Köln Messe/Deutz	Ahrensfelde Friedhof
357	3	13	2013-04-26T00:00:00	94	0	80	175	121	3889	3900	Augsburg Hbf	Nürnberg Hbf
315	3	28	2013-04-26T14:15:00	0	341	55	397	317	6477	7200	Solingen Hbf	Stuttgart Flugh/Mess
428	2	23	2013-04-27T06:28:00	467	0	100	567	400	10251	9000	Dortmund Hbf	Hennersdorf(Sachs)
363	2	16	2013-04-25T09:01:00	0	586	17	603	396	9029	8900	Hamm(Westf)	Emmendingen
335	3	16	2013-04-25T17:20:00	511	0	82	593	375	11183	12100	Biberach(Riß)	Plettenberg
598	3	30	2013-04-28T05:48:00	745	0	218	963	661	13884	12200	Göppingen	Bredstedt
341	3	33	2013-04-26T09:34:00	0	163	272	435	303	6504	7200	Köln Hbf	Bamberg
158	1	21	2013-04-28T12:12:00	0	0	173	173	125	2410	2640	Frankfurt(Main)Hbf	Rotenburg a.d. Fulda
353	3	23	2013-04-28T14:12:00	448	0	212	659	502	10883	11400	Frankfurt(Main)Hbf	Rostock Hbf
586	4	34	2013-04-28T00:05:00	0	229	256	484	378	8010	7500	Briesen(Mark)	Bremen Hbf
580	3	17	2013-04-24T00:19:00	366	89	15	470	328	9578	8800	Stuttgart Hbf (tief)	Paderborn Hbf
526	4	33	2013-04-27T00:47:00	0	0	312	312	184	4354	4010	Köthen	Baunatal-Guntershaus
477	3	32	2013-04-23T07:33:00	0	463	80	542	375	8318	7900	Wüstenfelde	Niederwiesa
533	5	42	2013-04-29T05:34:00	0	0	459	459	333	6412	5760	Böhlen Werke	Urbach(b Schorndorf)
184	1	13	2013-04-24T16:27:00	285	0	34	319	229	6837	8100	Düsseldorf Hbf	Neustadt(Weinstr)Hbf
409	3	23	2013-04-25T09:53:00	363	0	166	530	372	9487	8600	Finnentrop	Plüschow
700	4	24	2013-04-29T18:27:00	177	0	139	758	423	11176	10200	Mersch(Westf)	Herbertshofen
48	0	7	2013-04-27T18:24:00	0	0	67	67	52	935	1090	Braunschweig Hbf	Mieste
615	4	44	2013-04-25T07:03:00	0	584	181	765	539	11111	10500	Rheydt Hbf	Bad Schandau
434	4	32	2013-04-25T03:35:00	0	230	143	374	220	6224	5600	Hannover Hbf	Caputh-Geltow
167	1	16	2013-04-27T06:54:00	0	0	215	215	157	3004	2960	Delmenhorst	Braunschweig Hbf

Table 4.7.: Random data sample. Some attributes serve only debugging purposes.

5 Implementation Details

This chapter deals with the implementation of the experiments we have conducted in this thesis. In Section 5.1, we will discuss the different formats as far as data storing is concerned, and in Section 5.2 we will describe the R programming language and environment that was used in the machine larning process. In Section 5.3 we will talk about how the current and old fare predictions (see Chapter 2) were integrated.

5.1 Database

There are many choices when it comes to storing data. A popular format is the attribute-relation file format (ARFF) described by Witten, Frank, and Hall (2011). This format is used extensively in the `Weka` data mining framework (Hall et al. 2009). However, in this thesis, we opted to use a simple comma-separated values format with minimal quoting, i.e. only quoting objects which contain special characters such as the delimiter (`,`) or quote character (`"`). The ARFF format is essentially equal to simple CSV, plus data type declarations (e.g. numeric or nominal). Since we have used the `R` programming language, making an attribute nominal is just a matter of applying the `as.factor` function, or for the other way around, the `as.numeric` function.

5.2 The R Programming Language

The R programming language (R Core Team 2012) is a programming language and an interactive environment for statistical computing, graphics, and through the use of third party packages from the Comprehensive R Archive Network[1] (CRAN), for machine learning. Many excellent packages have been used for this thesis. First one foremost, the `caret` package (Kuhn 2008; Kuhn 2013) unifies many machine learning algorithms into one common framework. This package builds the bridge between `Cubist` (Kuhn, Weston, et al. 2013) for cubist regression, `kernlab` (Karatzoglou et al. 2004) for Support Vector Machines, `RSNNS` (Bergmeir and Benítez 2012) for an artificial neural networks, the Multilayer Perceptron (MLP), and `RWeka` (Hornik, Buchta, and Zeileis 2009) for `M5`.

To use R interactively, the `R` command has to be issued. Listing 4 depicts an exemplary R session where input lines are prefixed with "R>", and all other lines resemble output produced by R. First, a small data set is loaded into the environment (line 3). Then, the attribute names and instance count is printed using the `names` and `nrows` commands, respectively. Using the technique described in Subsection 4.2.1, (Near)-Zero Variance attributes are removed from the data set, resulting in a new data set named `data2` (line 14). Additional attributes that should not be fed to machine learning algorithms are removed in line 20. In line 26, the validation method is set to cross-validation. Finally, the `train` function of the `caret` package is called, yielding a tuned cubic model for regression. Do note that the first argument to `train` is an `R` formula and means that the attribute `price` is predicted using all the other attributes in the supplied data frame. The `caret` package supports many parameters, including a user-defined matrix for the selection of tuning parameters (see Chapter 6).

[1] http://cran.r-project.org/

```
R> library(caret)
R>
R> data <- read.csv("data/small.csv")
R>
R> names(data)
 [1] "duration"    "transfers"   "stops"       "dt"          "source_eva"
 [6] "source_name" "dest_eva"    "dest_name"   "dist_0"      "dist_1"
[11] "dist_2"      "dist_3"      "dist_4"      "dist_5"      "dist_6"
[16] "dist_7"      "dist_8"      "dist"        "lindist"     "estimate"
[21] "price"
R> nrow(data)
[1] 202
R>
R> data2 <- data[, -nearZeroVar(data, freqCut=20, uniqueCut=5)]
R> names(data2)
 [1] "duration"    "transfers"   "stops"       "dt"          "source_eva"
 [6] "source_name" "dest_eva"    "dest_name"   "dist_0"      "dist_1"
[11] "dist_3"      "dist"        "lindist"     "estimate"    "price"
R>
R> data3 <- data2[!names(data2) %in% c("source_eva", "source_name", "dest_eva", "dest_name")]
R> names(data3)
 [1] "duration"  "transfers" "stops"     "dt"        "dist_0"    "dist_1"
 [7] "dist_3"    "dist"      "lindist"   "estimate"  "price"
R>
R> set.seed(42)  # for reproducibility
R> ctrl <- trainControl(method = "cv")
R> fit <- train(price ~ ., data = data3, method = "cubist", trControl = ctrl)
R> fit
202 samples
 10 predictors

No pre-processing
Resampling: Cross-Validation (10 fold)

Summary of sample sizes: 182, 183, 181, 182, 181, 181, ...

Resampling results across tuning parameters:

  committees  neighbors  RMSE  Rsquared  RMSE SD  Rsquared SD
  1           0          691   0.94      129      0.0264
  1           5          615   0.951     136      0.0241
  1           9          630   0.949     132      0.0248
  10          0          619   0.952     105      0.0133
  10          5          495   0.969     81.3     0.00829
  10          9          526   0.965     81.5     0.00929
  20          0          639   0.949     153      0.0196
  20          5          521   0.965     136      0.0153
  20          9          547   0.962     137      0.0161

RMSE was used to select the optimal model using  the smallest value.
The final values used for the model were committees = 10 and neighbors = 5.
```

Listing 4: Example R session.

5.3 Integration of the Current and Old Estimations

One goal of this thesis is to find a method that does better than the current estimation method described in Chapter 2. It is therefore vital to postulate that the original estimation was indeed integrated into the evaluation process in a fair manner. In the following, we differentiate between the *old* method (Müller-Hannemann and Schnee 2005) and the *current* method (Harnisch and Nuhn 2010), both of which are described in Chapter 2.

The current estimator was treated no differently than any other model. In fact, it was integrated into the `train` function of the `caret` package so that it even runs through the same cross-validation process like any other predictive model. This is important to note because the other option as far as integration is concerned is to calculate several error metrics on the data set manually (the predicted price for the current method is part of the data set), and this could mean that we end up with incorrent results for this method when the data set or the validation method changes in case we forget to update the results for the current method accordingly.

Listing 5 indicates how the current estimation was integrated into the machine learning workflow. Line 1 and 2 describe methods that "predict" an attribute by just returning the value of another attribute of the input data. Line 3 supports the claim that the estimation is treated like any other method by using the same control parameters (e.g. parameters that set the validation method to 10-fold cross-validation). Lines 4 through 7 show that the attribute named `estimate` is used as prediction. This attribute is of course not fed to any other learning methods.

```
1 curModelFunc <- function(data, parameter, levels, last, ...) list(fit=parameter$.colname)
2 curPredFunc <- function(object, newdata) newdata[[levels(object$fit)[1]]]
3 ctrlCur <- ctrl
4 ctrlCur$custom <- list(model=curModelFunc, prediction=curPredFunc, probability=NULL,
5                        sortFunc=function(x) x, parameters=data.frame(.colname=c("estimate")))
6 trainers$current <- function(form, ...)
7     train(form, method="custom", trControl=ctrlCur, ...)
```

Listing 5: Integration of the current estimation. The default column name of the attribute to use as prediction is set to *estimate* in our data set (line 5).

The old estimator is treated quite similarly. However, because MOTIS cannot provide estimates from two estimators at the same time, the old estimation is calculated on-the-fly instead of being extracted from the response given by MOTIS. Listing 6 shows a port of the original C++ code to the R programming language in order to be used with the `caret` package.

```
1  oldModelFunc <- function(data, parameter, levels, last, ...)
2      list(fit=parameter$.factor)
3
4  oldPredFunc <- function(object, newdata) {
5      adply(newdata, 1, function(x) {
6            res <- x$dist * object$fit
7            if (x$dist_0 > 0) res = res + 1200
8            else if (x$dist_1 > 0) res = res + 700
9            res
10     })$V1}
```

Listing 6: Integration of the old estimation. The factor to be multiplied with the distance (line 2) is set to 14 cents. A surcharge is added to the final ticket price according to the highest train class involved (lines 7 and 8).

6 Tuning

Most machine learning techniques support the configuration of one or more tuning parameters. For example, cubist trees (Subsection 3.2.3) require the parameters `committees` and `neighbors` to be specified. Because the performance of a machine learning technique greatly depends on the selection of appropriate parameters, this chapter is devoted to tuning these parameters.

In order to find the best possible parameter combination, each algorithm is trained and validated using a tuning grid. A tuning grid is constructed by taking the Cartesian product of the tuning parameter sets. For example, let the two tuning parameter sets for cubist be $\mathrm{N} := \{1,2,3,4\}$ for the parameter `neighbors` and $\mathrm{C} := \{1,2,3\}$ for the parameter `committee`, then the tuning grid is constructed as

$$G := N \times C = \{(n,c) | n \in N \wedge c \in C\} \tag{6.1}$$

where G is the tuning grid set to be used. The complete tuning grid for the example above is displayed in a tabular manner in Table 6.1. For each element $g \in G$, a model is trained and validated using 10-fold cross validation (see Subsection 3.3.2). The best g is then chosen according to the lowest MAE. The MAE was chosen because we did not want to punish large errors. An explanation of this circumstance can be found in Section 3.4.

Neighbors	Committees
1	1
2	1
3	1
4	1
1	2
2	2
3	2
4	2
1	3
2	3
3	3
4	3

Table 6.1.: Exemplary expanded tuning parameter grid with the tuning parameters committee and neighbors. Four values for the neighbors parameter and three values for the committees parameter are expanded to a total of 12 parameter pairs.

It is important to note that the lowest MAE does not necessarily correspond to the model that should be used as final model. Depending on the use case, it may be more appropriate to choose an algorithm with tuning parameters that yield a slightly worse result in order to decrease the complexity and thus the prediction time of the model. This is especially true for the cubist algorithm, the result of which we will discuss in the next chapter.

7 Evaluation

This chapter deals with the performance evaluation for the different learning algorithms studied within this thesis. The performance is determined using 10-fold cross-validation (see Subsection 3.3.2) with equal folds for all algorithms, accomplished by setting a constant seed.

We will first study the performance for each algorithm individually using the data set S_3, which forms the union of the data sets S_1 (generated using the probability-based sampling technique described in Section 4.1) and S_2 (built using a live recording of humans booking train journeys). The best parameters for each algorithm are selected such that the MAE (see Section 3.4) is minimized when the algorithm is trained using a tuning grid. We have chosen the MAE as metric to minimize because we did not want to punish outliers (see Chapter 6). The data set S_3 now contains a total of 14,000 instances. Originally, this data set contained 50,000 instances. However, it was necessary to reduce the size of this data set to 14,000 instances in order to reduce the training time for SVMs. We would like to have have worked with a larger data set, but were unable to do so due to time constraints. Next, we will examine and compare how the different techniques behave when they are applied to our three data sets individually in Section 7.2. Finally, we determine whether the evaluated methods overpredict or underpredict the actual price in Section 7.3, and continue to measure the prediction time for each model in Section 7.4.

7.1 Method Comparison

In this section, we will evaluate and interpret the results for the machine learning methods utilized in this thesis individually. A description for each algorithm can be found in Section 3.1. All experiments here are based on the data set S_3.

7.1.1 Decision Trees

Figure 7.1 shows the effect of the tuning parameters on the evaluation results for the cubist algorithm. The detailed results can be found in Table A.1. The parameter `committees` refers to the number of decision trees to create, and the `neighbors` parameter refers to k in the k-nearest-neighbor algorithm. The maximum permissible values for these parameters are 100 for `neighbors` and 9 for `committees` in Quinlan's implementation of cubist.

For committees ≥ 30, cubist produces models with equal cross-validation results. This suggests a fix point. The best model was created using the parameters committees ≥ 30 and neighbors $= 9$. The application of the k-nearest-neighbor algorithm as part of the cubist algorithm turned out to be a reasonable step, with an improvement of 40 cents between neighbors $= 1$ (the nearest-neighbors algorithm is not applied) and neighbors $= 9$. Cubist finds the neighbors $- 1$ most similar cases in the training set and averages these k points (Quinlan 1993b). We would like to have tested the cubist algorithm with the parameter neighbors > 9, but were unable to do so due to the limitations by Quinlan's implementation of cubist.

Listing 7 shows the first two rules in the first committee. Each rule consists of one or more `if` clauses and a linear model as potential outcome. In order to determine the final prediction of a committee, the arithmetic mean is calculated based on all rules whose conditions evaluate to true. Cubist deduces a total of 2746 of these rules across 30 committees for committees $= 30$. Unfortunately, single rules are not interpretable, even if dealing with one committee only.

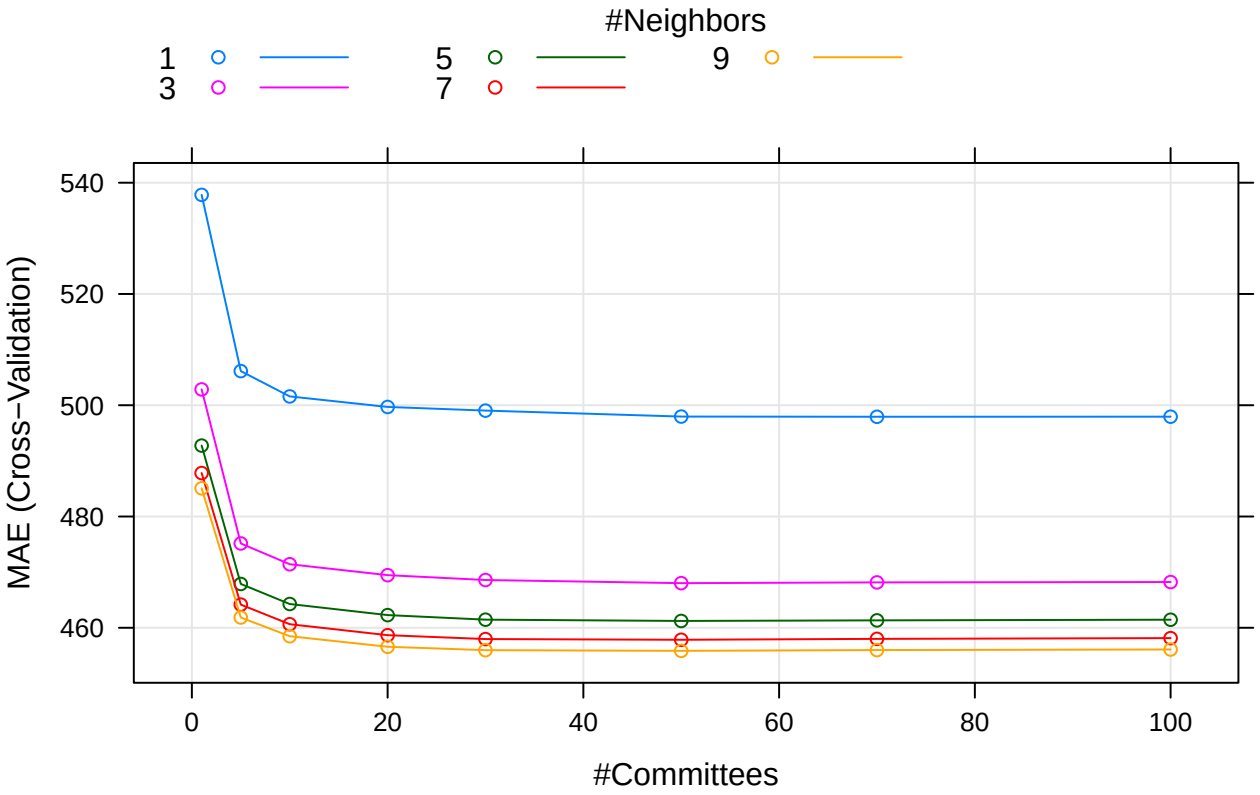

Figure 7.1.: Visualization of the evaluation results for the cubist algorithm. The error does no longer decrease after the parameter `neighbors` reaches 30.

7.1.2 Linear Regression

Linear regression, as described in Subsection 3.2.1, produces the following model described by Equation (7.1):

$$\begin{aligned}\text{price} = \\ &+ 1475.34 \\ &+ 0.97 \cdot \text{duration} \\ &+ 282.18 \cdot \text{transfers} \\ &- 6.62 \cdot \text{stops} \\ &+ 2.45 \cdot \text{dist\_0} \\ &- 0.71 \cdot \text{dist\_1} \\ &- 5.14 \cdot \text{dist\_3} \\ &+ 7.78 \cdot \text{dist} \\ &+ 5.98 \cdot \text{lindist} \\ &+ 20.24 \cdot \text{dt\_weekday} \\ &+ 3.08 \cdot \text{dt\_hour}\end{aligned} \tag{7.1}$$

Unlike other learning schemes such as MARS, linear regression uses all the predictors that were present in the data set at the time of training. When removing the parameters `dt_weekday` and `dt_hour` from the data set S_3, the cross-validation results for linear regression do not change at all and stay constant with a

```
Model 1:

  Rule 1/1: [568 cases, mean 1292.3, range 360 to 2960, est err 75.7]

    if
        duration <= 106
        dist_0 <= 1.67477
        dist_1 <= 9.4963
        dist_3 > 21.0408
    then
        outcome = 146 + 266.6 dist_1 + 128.8 dist_0 + 4.7 dist + 8.1 dist_3
                  + 1.3 duration + 1.2 lindist - 3 stops

  Rule 1/2: [41 cases, mean 1861.5, range 130 to 6800, est err 294.0]

    if
        dist_0 <= 1.67477
        dist_1 <= 9.4963
        dist_3 <= 21.0408
    then
        outcome = 3.6 + 70.9 dist_0 + 26 lindist + 16.7 dist_1 - 16.9 dist_3
                  + 78 stops - 2.7 dist
```

Listing 7: The first two rules in the first model created by the cubist algorithm. Single rules in a large ensemble method are not interpretable and are only shown for reference.

MAE of 810 cents. The same cannot be said for the attributes `stops` and `transfers`. When removing the attribute `stops`, the MAE increases by only 1 cent, and when removing the attribute `transfers`, the MAE increases by 23 cents.

Equation (7.1) represents the model learned from non-normalized data. Equation (7.2) on the other hand describes the model created when learning from normalized data. Normalization is a common data transformation, and scaling refers to dividing each value of a predictor variable by its standard deviation. Scaling coerce the values to have a common standard deviation of 1, and makes the resulting model more interpretable. Large coefficients in the linear model correspond to important predictors. Hence, in Equation (7.2) `dist_0` (the distnace traveled on ICE trains) is the most important predictor with a coefficient of 5221.32, followed by `dist` (the total distance traveled) with a coefficient of 1684.54. The attributes `stops`, `dt_weekday`, and `dt_hour` are of very little importance in this model.

$$
\begin{aligned}
\text{price} = \\
&+ 6892.04 \\
&+ 176.38 \cdot \text{duration} \\
&+ 374.98 \cdot \text{transfers} \\
&- 79.69 \cdot \text{stops} \\
&+ 5221.32 \cdot \text{dist\_0} \\
&- 123.62 \cdot \text{dist\_1} \\
&- 569.122 \cdot \text{dist\_3} \\
&+ 1684.54 \cdot \text{dist} \\
&+ 867.76 \cdot \text{lindist} \\
&+ 39.73 \cdot \text{dt\_weekday} \\
&+ 18.27 \cdot \text{dt\_hour}
\end{aligned}
\tag{7.2}
$$

The linear model can be improved greatly by choosing a polynomial basis function in addition to differentiating between the various train classes. The current estimator implemented in MOTIS, as described in Chapter 2, does exactly this and is evaluated in Subsection 7.1.6.

7.1.3 Multivariate Adaptive Regression Splines

For the MARS algorithm (see Subsection 3.2.2), the effects of the tuning parameters `degree` and `terms` on the evaluation results can be seen in Figure 7.2, and the detailed results are given in Table A.2. In this table, the correlation coefficient is not listed in some cells due to a division by zero that occurs for nprune $= 1$, in which case the MARS algorithm uses the mean as prediction.

The `nprune` parameter specifies the maximum number of permissible terms in the final pruned model, and the `degree` determines the maximum degree of interaction between the predictors, and can either be 1 or 2 for our implementation of MARS. In other words, when setting `degree` to 1, a basis function in the MARS model cannot be the product of two hinge functions. The `degree` parameter has very little impact on the final results and is therefore negligible.

For nprune ≥ 15, all results stay constant with a MAE of 593. This is due to the number of terms produced by MARS being fewer than or equal to the maximum number of permissible terms, as specified by the `nprune` parameter. This suggests that the pruning of terms as part of the MARS algorithm does not decrease the overall error for our data set.

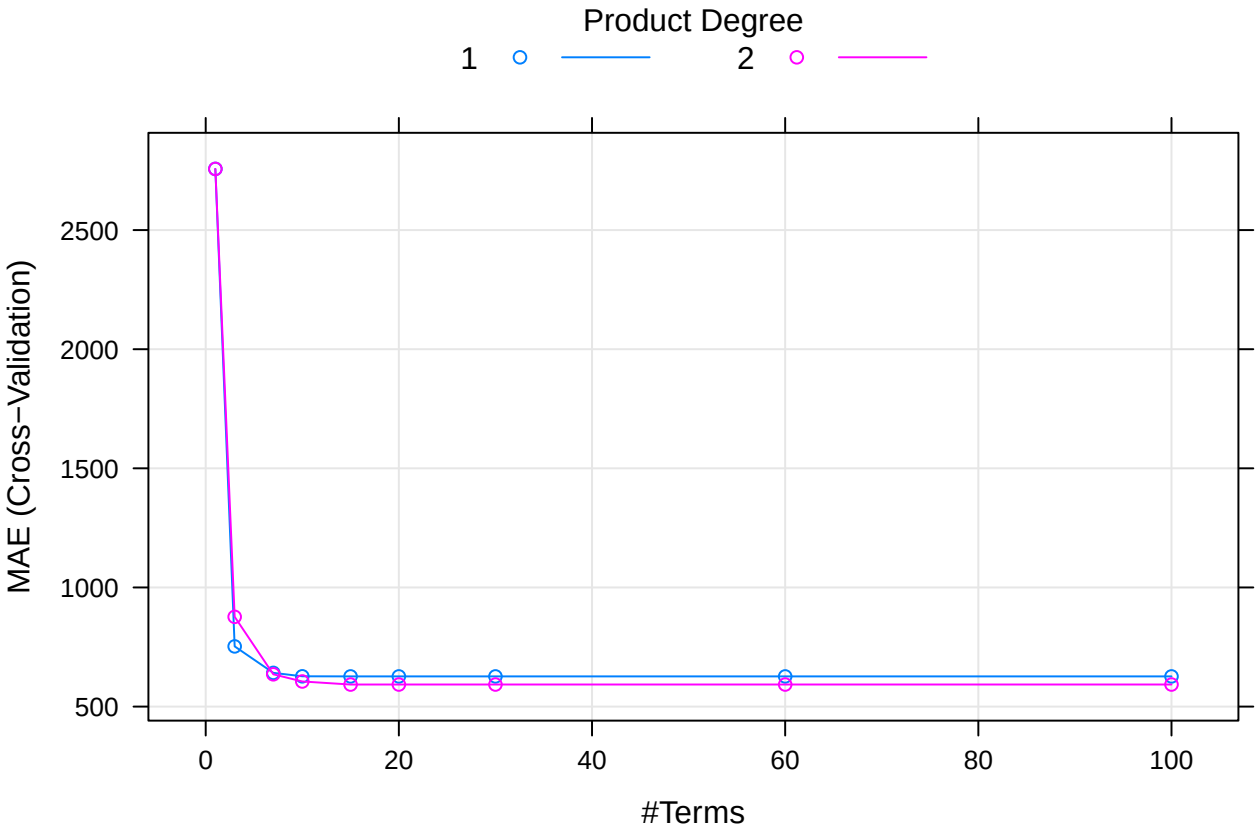

Figure 7.2.: Visualization of the evaluation results for the MARS algorithm using the mean-absolute error. The nprune parameter has little impact on the final results.

The MARS algorithm creates a model that consists of 15 terms, and 7 of these are of degree 2. This model is described by Equation (7.3):

$$
\begin{aligned}
\text{price} = \\
&+ 9507.948 \\
&+ 0.978 \cdot h(\text{dist} - 636.22) \\
&- 1.513 \cdot h(636.22 - \text{dist}) \\
&+ 1.085 \cdot h(\text{dist}_0 - 182.81) \\
&- 7.376 \cdot h(182.81 - \text{dist}_0) \\
&+ 0.003 \cdot h(182.81 - \text{dist}_0) \cdot h(\text{dist}_1 - 155.667) \\
&- 0.01 \cdot h(182.81 - \text{dist}_0) \cdot h(155.667 - \text{dist}_1) \\
&- 8.163 \cdot h(\text{lindist} - 558.684) \\
&- 7.045 \cdot h(558.684 - \text{lindist}) \\
&+ 0.037 \cdot h(\text{dist} - 340.687) \cdot h(558.684 - \text{lindist}) \\
&- 0.017 \cdot h(340.687 - \text{dist}) \cdot h(558.684 - \text{lindist}) \\
&- 0.046 \cdot h(\text{dist}_3 - 279.84) \cdot h(636.22 - \text{dist}) \\
&+ 0.007 \cdot h(279.84 - \text{dist}_3) \cdot h(636.22 - \text{dist}) \\
&- 0.596 \cdot h(3 - \text{transfers}) \cdot h(558.684 - \text{lindist}) \\
&+ 10.503 \cdot h(\text{lindist} - 378.519)
\end{aligned}
\tag{7.3}
$$

Unfortunately, this model is not as interpretable as previously thought, specifically since all the distance predictors are highly dependent on each other (i.e. the distance for ICE trains cannot be increased without increasing the overall distance as well). Interestingly, the MAE increases from 593 to 602 (using the data set S_3) when removing the `lindist` predictor (the linear distance between source and destination train station), and the number of terms decreases from 15 to 14. Another interesting aspect of the model is the fact that the predictors `duration` and `stops` are not used at all. These two predictors are likely candidates as far as removal of attributes is concerned.

7.1.4 Neural Networks

The Multilayer Perceptron (MLP), a feedforward artificial neural network, did not deliver very good results. Table 7.1 shows the expanded tuning grid for the MLP, where `size` specifies how many nodes to have in the hidden layer, and `decay` is a parameter that is used to reduce overfitting. The best results were obtained using size $= 17$ and decay $= 0.001233$. The final model produced by the MLP delivers inferior cross-validation results than the baseline predictor in terms of the MAE. This suggests that this algorithm is not a very good choice for the problem at hand.

7.1.5 Support Vector Machines

Support Vector Machines, as described in Subsection 3.2.4, produce quite promising results. Figure 7.3 visualizes the effect of the tuning parameters `C` and `degree` on the cross-validation results in terms of the MAE. The detailed results can be found in Table A.4. The parameter `degree` corresponds to the degree of the polynomial kernel, and the parameter `C` (Cost) controls the trade off between allowing training errors and forcing rigid margins. When increasing the value of C, the cost of misclassification of points increases and the model becomes more accurate. However, this also means that it may not generalize well, thus producing worse results in the end. The best results were obtained using the parameters $C = 0.5$ and degree $= 4$. The final model (constructed using the data set S_3 with $|S_3| = 14000$) consists of 7083 support vectors.

Size	Decay	MAE	RMSE	RRSE	RAE	Rsq
1	0.000000	5732	6420	1.982	2.078	0.003
3	0.100000	4390	5089	1.572	1.594	**0.012**
5	0.053367	4299	5018	1.550	1.561	0.006
7	0.028480	3412	4024	1.241	1.237	0.008
9	0.015199	3850	4475	1.382	1.398	0.008
11	0.008111	3155	3773	1.165	1.145	0.006
13	0.004329	3757	4392	1.354	1.362	0.004
15	0.002310	3426	4035	1.245	1.241	0.004
17	0.001233	**2807**	**3314**	**1.023**	**1.018**	0.003
19	0.000658	3644	4396	1.358	1.323	0.006
21	0.000351	3728	4424	1.361	1.348	
23	0.000187	3367	3986	1.232	1.219	0.003
25	0.000100	3765	4596	1.433	1.380	

Table 7.1.: Evaluation results for a Multilayer Perceptron. The best result for each metric is printed in bold.

Figure 7.4 on the other hand visualizes the effect of the tuning parameters on the validation results when using a SVM with a radial kernel, and the detailed tuning results can be obtained from Table A.3. The best results for this SVM were obtained using $C = 4$ or $C = 8$. Interestingly, this kernel produces better results than the polynomial kernel. The SVM with a radial kernel has 7018 support vectors, and the `sigma` parameter is obtained automatically by the SVM implementation used in this thesis.

We would like to have removed some attributes in order reduce the number of support vectors, but were unable to do so due to time constraints and the amount of time required for the training of Support Vector Machines. We expect the number of support vectors to drop when removing attributes such as `stops` and `duration`, and this should indeed be explored when choosing a SVM as replacement for the current method.

7.1.6 The Current Method

The current method is described in Chapter 2 and is given by the equation

$$\begin{aligned} \text{price} = & \\ & + \min(12200, \max(700, 23.917 \cdot \text{dist} - 0.0122 \cdot \text{dist}_0^2 + 622.29)) \\ & + \min(11700, \max(600, 18.433 \cdot \text{dist} - 0.0073 \cdot \text{dist}_1^2 + 334.79)) \\ & + 14 \cdot \text{dist}_3 \end{aligned} \tag{7.4}$$

where $\max(a, b)$ is a if $a > b$ else b and $\min(a, b)$ is a if $a < b$ else b. The distances dist_0, dist_1, and dist_3 correspond to the distance traveled in ICE, IC/EC, and regional trains, respectively. Noteworthy is the fact that it is one of the simplest and most interpretable model we have evaluated in this thesis. It provides fairly good results, and treats the distance traveled in each train category separately.

7.1.7 The Old Method

The old method is also described in Chapter 2 and is given by

$$\text{price} = \begin{cases} 14 \cdot \text{dist} + 1200 & \text{if } \text{dist}_0 > 0 \\ 14 \cdot \text{dist} + 700 & \text{if } \text{dist}_0 = 0 \text{ and } \text{dist}_1 > 0 \\ 14 \cdot \text{dist} & \text{otherwise} \end{cases} \tag{7.5}$$

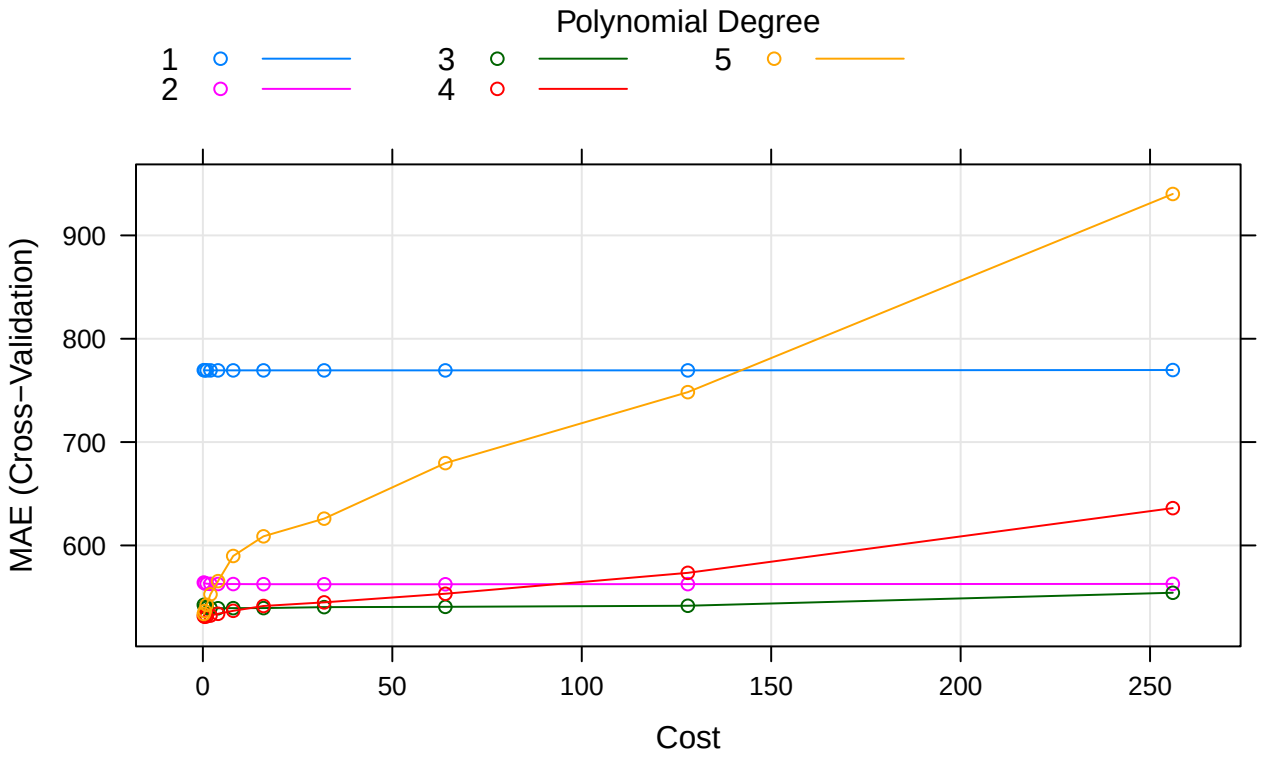

Figure 7.3.: Visualization of the evaluation results for a SVM with a polynomial kernel using the MAE. The best results were obtained using $C = 0.5$ and degree $= 4$.

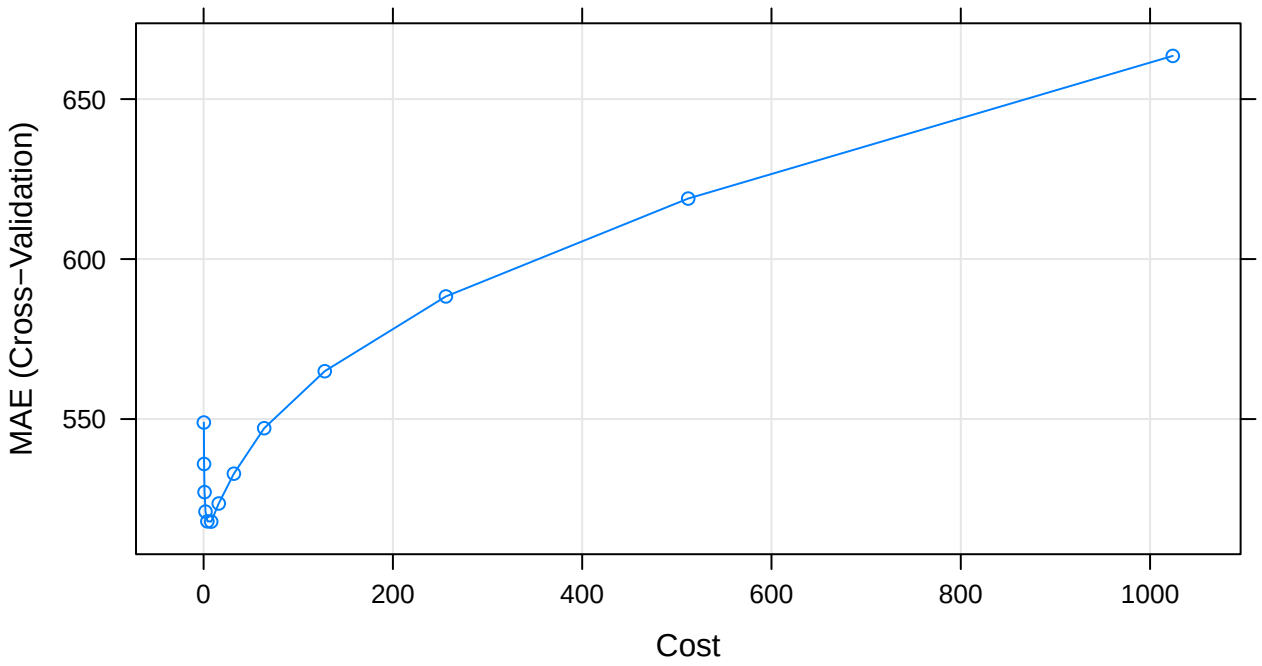

Figure 7.4.: Visualization of the evaluation results for a SVM with a radial kernel using the MAE. Setting $C = 4$ or $C = 8$ yields the best results for this SVM.

This simple model produces fair results. Unlike the current method we have evaluated in Subsection 7.1.6, it treats all distances the same. It does however add a surcharge depending on the highest train class involved.

7.2 Method Comparison

Table 7.2 ranks the machine learning algorithm applied to the data set S_1. Table 7.3 and Table 7.4 rank the same algorithms applied to the data sets S_2 and S_3, respectively. The rank corresponds to the MAE, and the algorithm with the lowest MAE is ranked first. We have chosen this metric for selecting the best model because we did not want to punish large outliers (see Section 3.4). The baseline entry corresponds to a predictor that predicts the arithmetic mean. Each table contains several error metrics in each column, including the mean-absolute error (MAE), the root mean-squared error (RMSE), the root relative-squared error (RRSE), the relative-absolute error (RAE), and R squared (Rsq). All of these metrics are discussed in Section 3.4.

Interestingly, the ranking order is identical for each data set when taking the MAE into account. This is even true when selecting the RMSE as rank-determining metric. It should be noted that since the MAE was chosen as metric to minimize in the tuning step of the machine learning process, the MAE should also be used to assess and compare the final models with each other. However, for the algorithms studied, minimizing the MAE yields a tuning parameter set that is also the best choice when taking the RMSE metrics into account. When selecting the correlation coefficient as rank-determining metric, the ranking order is quite different. The correlation coefficient measures the statistical correlation between the actual value and does not measure the error produced when predicting the ticket cost. Hence, preference should be given to the MAE as far as the ranking of techniques is concerned.

In any case, the fact that the MAE-determined ranking order is equal suggests that the probability-based sampling method used to create S_1 comes close to the actual behavior of humans when booking train journeys (S_2) and confirms the sampling approach we have taken in Section 4.1. The RRSE and the RAE are relative to what would have been if a simple predictor, the baseline, had been used. These two metrics are redundant in the result tables because these tables include an entry for the baseline, but are nevertheless a good indicator on how a specific algorithm compares to the baseline. For the baseline, the mean predictor was chosen. This predictor outputs the mean of the price attribute in the data as prediction, and is a common choice in the field of machine learning.

We will now look compare the results of the algorithm studied. Cubist models yield the best result, with a MAE of 456 when applied to the combined data set (Table 7.4). M5 from `Weka` should thoeretically be equal to cubist without the application of the kNN algorithm and committes, and produces slightly worse results than cubist. This behavior is expected since cubist is an enhanced version of `M5`. Support Vector Machines, a very popular method for predictive modelling, provide good results, and due to a lower MAE preference should be given to a machine with a radial kernel. The MARS algorithm is a little better than the current method according to the MAE. Interestingly, when taking the RMSE into account, the situation is not so clear. According to the RMSE and Table 7.4, the current method does produce results superior to those produced by MARS. This suggests that MARS produces slightly more outliers than the current method (see Section 3.4 for an explanation of the differences of these metrics). The old method and linear regression provide only fair results. The multilayer perceptron, a feedforward artificial neural network, does slightly worse than the baseline method, and should not be considered for the replacement of the current method.

	Rank	MAE	RMSE	RRSE	RAE	Rsq
Cubist Trees	1	531	758	0.251	0.210	0.937
M5	2	546	787	0.257	0.212	0.934
SVM (Radial)	3	567	813	0.265	0.220	0.930
SVM (Poly)	4	576	835	0.273	0.224	0.926
MARS	5	619	847	0.277	0.241	0.923
Current Method	6	644	843	0.275	0.250	0.935
Linear Regression	7	852	1120	0.365	0.332	0.867
Old Method	8	877	1310	0.427	0.341	0.844
Baseline (Mean)	9	2530	3020	1.000	1.000	
Neural Net (MLP)	10	2580	3070	1.000	0.999	

Table 7.2.: Cross-Validation results for the algorithms applied to the data set S_1. This data set contains instances generated using the probability-based sampling method that weights stations according to the number of incoming and outgoing connections.

	Rank	MAE	RMSE	RRSE	RAE	Rsq
Cubist Trees	1	421	642	0.190	0.144	0.964
M5	2	455	696	0.212	0.162	0.955
SVM (Radial)	3	494	737	0.225	0.175	0.950
SVM (Poly)	4	507	773	0.236	0.180	0.944
MARS	5	541	776	0.237	0.192	0.944
Current Method	6	551	748	0.228	0.196	0.951
Linear Regression	7	751	1040	0.317	0.267	0.900
Old Method	8	758	1190	0.364	0.269	0.885
Baseline (Mean)	9	2910	3370	1.000	1.000	
Neural Net (MLP)	10	3000	3570	1.090	1.060	0.008

Table 7.3.: Cross-Validation results for the algorithms applied to the data set S_2. This data set contains instances from a real-life capture of bookings done by humans in one day.

	Rank	MAE	RMSE	RRSE	RAE	Rsq
Cubist Trees	1	456	673	0.206	0.163	0.958
M5	2	487	723	0.223	0.177	0.950
SVM (Radial)	3	518	760	0.235	0.188	0.945
SVM (Poly)	4	531	781	0.241	0.193	0.942
MARS	5	593	826	0.255	0.215	0.935
Current Method	6	597	797	0.246	0.217	0.945
Linear Regression	7	810	1090	0.335	0.294	0.888
Old Method	8	817	1250	0.387	0.297	0.870
Baseline (Mean)	9	2790	3270	1.000	1.000	
Neural Net (MLP)	10	2810	3310	1.020	1.020	0.003

Table 7.4.: Cross-Validation results for the algorithms applied to the data set $S_3 = S_1 \cup S_2$. This data set forms the union of the previous two data sets.

7.3 Residual Analysis

The residual is the difference between the actual price of a train ticket and the predicted price. In order to determine whether the learning methods studied over or underpredict the actual price, we will now analyze the residuals produced by the prediction of instances in the data set S_3.

Figure 7.5 and Figure 7.6 show scatter plots of the residuals for most of the methods studied in this thesis. These plots are a smoothed color density representation of a scatter plot, obtained through a kernel density estimate. Dark colored regions represent a higher point density than light colored regions. All data points below the red line are overpredicted, while all points above the red line are underpredicted. When a point lies on the red line itself, it is predicted exactly.

Cubist predicts the price quite reliably up until € 40. For prices higher than € 40, cubist produces more outliers and shows a slight tendency to underpredict the actual values. Linear regression on the other hand has a clear tendency to overpredict less expensive train tickets and a slight tendency to underpredict medium to high-priced tickets. High-priced tickets seem to be equally over and underpredicted by linear regression. The MARS algorithm does not have a clear tendency to either over or underpredict the observations. However, for high-priced tickets, it does have a tendency to underpredict the values, and also produces quite a few outliers in the medium-prized segment. The current method is quite interesting and seems to almost exclusively underpredict the actual prices up until € 30. For prices higher than € 30, the current method has a tendency to overpredict the actual price, with some outliers on that end. The old method has a clear tendendy to underpredict the ticket price, except for high-prized tickets, where this method over and underpredicts equally. Support Vector Machines produce some underpredicted outliers for medium to high-priced tickets. The SVM with a radial kernel features a high point density around the red line, suggesting a stable prediction with a low number of large residuals.

7.4 Time Evaluation

The search for an algorithm that provides a prediction *fast* is an important goal for this thesis. For this reason, we will now measure the prediction time for each of the algorithms studied using a completely new data set consisting of 3,000 instances sampled using the probability-based sampling method described in Section 4.1. We have used a new data set because we did not want to put algorithms at advantage that work faster on previously seen data. While the training time is not an important factor in this thesis, it is still quite interesting to see what time was required to train the final model. It is important to note that in the following, training time corresponds to the time spent training the final model, i.e. a model with one set of parameters. In order to figure out the best parameter combination, several models have to be trained, and this circumstance may increase the total training substantially.

All of our tests were conducted in one single run and the CPU time was measured. Unlike the wallclock time, the CPU time is the amount of time the CPU was used for processing instructions from a single process (program). This specifically means that when the CPU context switches (stores and restores the state of a process) to another process, the measured time of the process in question will not be increased. All of our tests were conducted on dedicated machines that have only a minimum amount of processes running. Nevertheless, the prediction experiments were repeated 10 times, and the numbers in Table 7.5 constitute the averaged numbers across these experiments.

Table 7.5 lists the algorithms studied and the associated time spent for training and the prediction of 3,000 instances. It comes as no surprise that linear regression is one of the fastest models as far as training and classification time is concerned. The current method (see Subsection 7.1.6) is missing from the table because it was neither trained nor is there any informative value since the prediction is obtained using MOTIS. However, because the current method consists of three separate linear regression schemes, the training and classification time should be very close to the timings of simple linear regression. The prediction time for the old method (see Subsection 7.1.7) should even be below the time needed by linear regression, because it is based on a single attribute only. The prediction time for MARS is relatively low

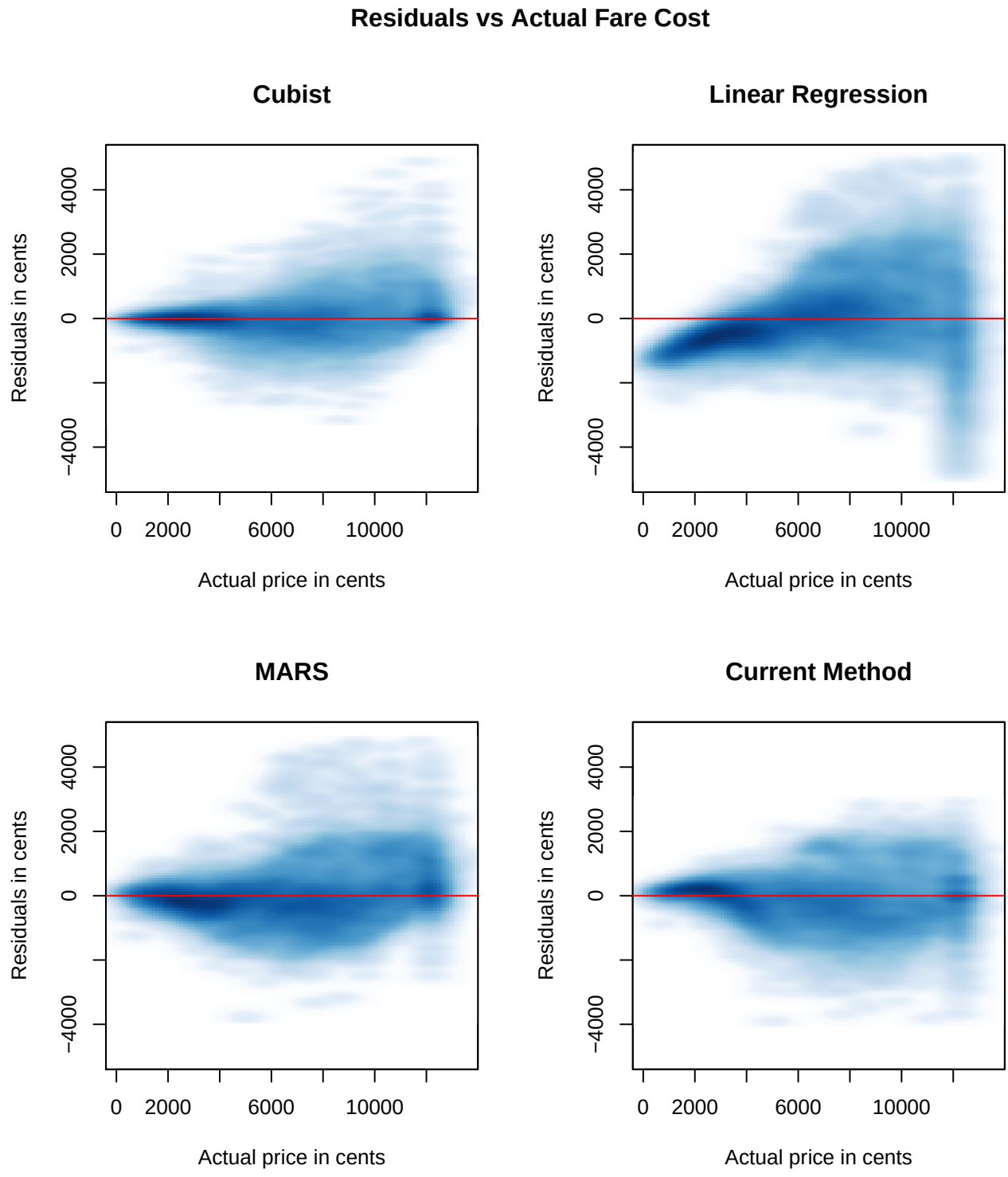

Figure 7.5.: Smoothed versions of scatter plots of the residuals. All points below the red lines are overpredicted, while all points above the red lines are underpredicted. Darker regions represent a higher density of points.

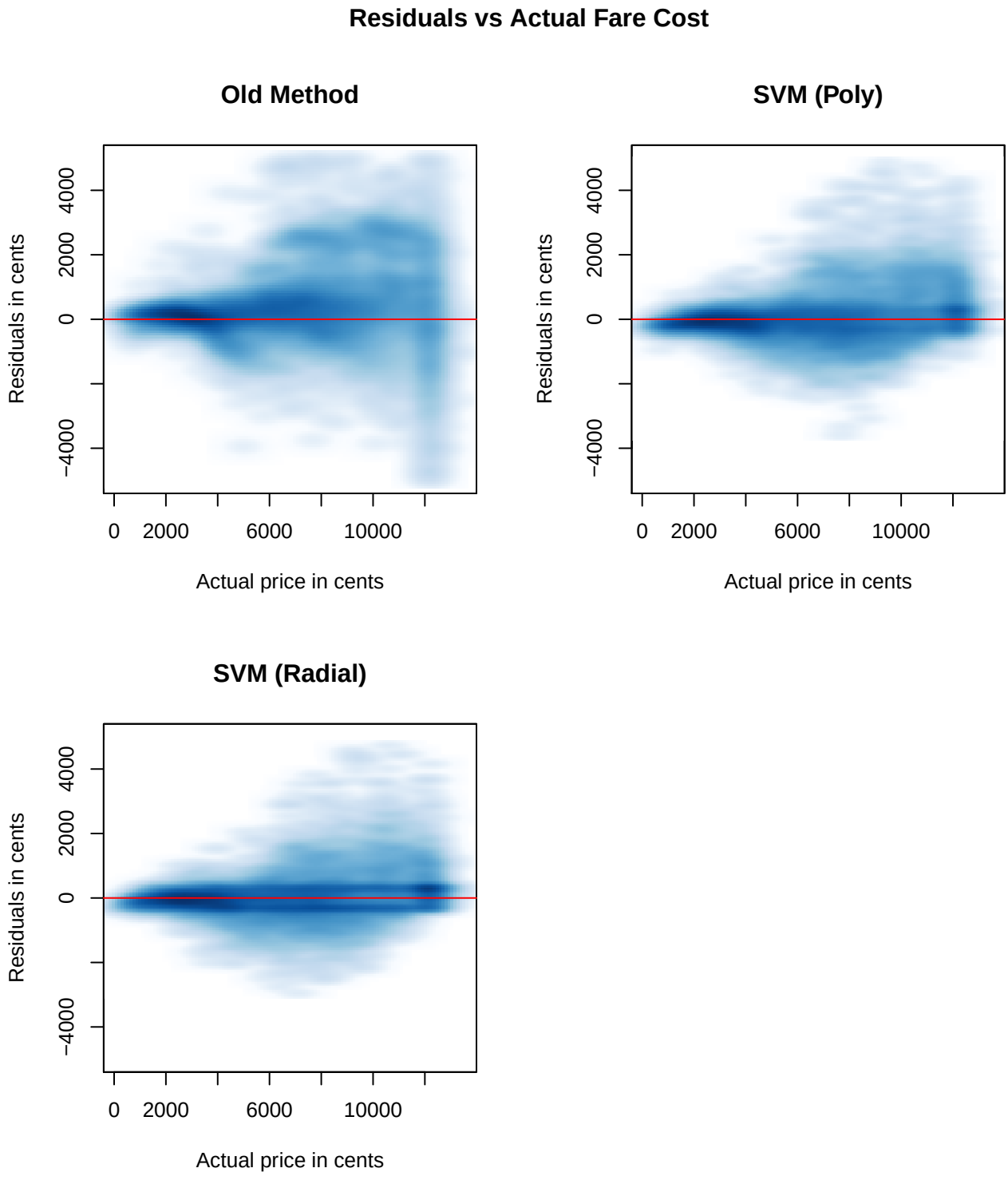

Figure 7.6.: Smoothed versions of scatter plots of the residuals. All points below the red lines are overpredicted, while all points above the red lines are underpredicted. Darker regions represent a higher density of points.

	Training Time	Prediction Time
Linear Regression	0.15	0.07
M5	3.56	0.13
MARS	2.41	0.46
Cubist Trees	76.30	3.07
SVM (Radial)	179.75	5.22
SVM (Poly)	147.43	10.66
Neural Net (MLP)	21.73	13.70

Table 7.5.: Training and classification CPU time in seconds. The training time in seconds corresponds to the time spent training the final model, while the prediction time describes the time spent for the prediction of 3,000 previously unseen data instances.

compared to the other more complex models studied. Cubist trees, despite being deduced to 2746 rules in 30 committees, need only 3.07 seconds for the prediction of 3,000 instances. The M5 algorithm from `Weka` (Hall et al. 2009), which should be as fast as the cubist algorithm without committees and the application of the kNN algorithm, is also quite fast. However, the cubist algorithm is relatively faster, given that it predicts the ticket price using 30 trees instead of just one. It is important to note that the prediction time for cubist can be decreased greatly by choosing a smaller value for the `committees` parameter. The SVM with a radial kernel is almost twice as fast as the one with a polynomial kernel as far as prediction time is concerned, and the MLP requires the most time for the prediction of our test set.

In order to optimize for the ticket price, MOTIS requires about 50,000 predictions to be made when booking a train ticket. For this reason, only the first three models (Linear Regression, MARS, and M5) are recommendable as far as integration into MOTIS is concerned. With more computing power at hand, SVMs pose an option as well. It should be noted though, that cubist tree are also advisable when fewer committees are learned. For the prediction of 50,000 ticket prices, MARS needs about 7.7 seconds on a consumer notebook and the M5 tree learner about 2.2 seconds. The current method is still the fastest method, and given that it is a model with just 3 variables, it should take about 1 second for the prediction of 50,000 instances.

8 Conclusion

Within this thesis, we have studied and evaluated several machine learning algorithms that can be used to predict railway fares, and have compared them to two existing methods, one of which is currently being used in MOTIS. The best results were obtained using an ensemble method: the decision tree learner cubist. For the best selection of parameters, this method creates an ensemble of 30 committees (trees) that are then deduced to 2746 rules. Support Vector Machines (SVMs) have also delivered an effective result and the study indicated that preference should be given to a SVM with a radial kernel due to the lower prediction time and error. In the event a technique involving SVMs is chosen to replace the current method, the removal of predictors should be considered in order to decrease the number of support vectors. Multivariate Adaptive Regression Splines (MARS) models provided good results, and were especially interesting because of the simplicity of the model produced and the negligible prediction time. The method currently employed by MOTIS, which was developed by Harnisch and Nuhn (2010), also provides good results; however, it tends to almost exclusively underpredict low-priced tickets. The current method is also by far the fastest of the techniques studied in this thesis.

No clear indication can be given as to which method represents the most suitable learning algorithm with which to replace the current method. The prediction time is acceptable for most of the methods studied and, as such, while cubist trees are a likely candidate, both SVM and MARS models also pose a viable option. However, when choosing a SVM, considerable more computing power is needed. As far as integration into MOTIS is concerned, all of these options are practically realizable. The cubist tree learner is now licensed under the General Public License, and C source code intended to read and interpret the learned models is provided[1] "as is." Several mature implementations of a SVM are in existence, including libSVM[2] and SVMlight[3]. A model produced by MARS can be programmed directly into MOTIS.

The use of a combination of some of the techniques that were examined in this study is also feasible, possibly considering the analysis of the residuals explored in Section 7.3. However, it should be noted that such an amalgamation of algorithms may increase the overall prediction time when multiple models are used for the prediction of railway fares. The residual analysis could also be used to improve the the fare prediction for the current method, specifically for less expensive tickets, but we expect the improvement to be minor.

By modelling the problem at hand in a different fashion, it may also be possible to learn the actual corridors (the area between two points which a train travels through and must not leave) that determine the final ticket price. However, no such attempts were undertaken in this study.

In addition to studying different machine learning algorithms, a sampling method that favors routes planned from and to major railway stations, determined by the total number of incoming and outgoing connections, has been proposed. The ranking of the algorithms studied when applied to this data set was identical to the ranking acquired when applying the techniques to a data set that consisted of a live recording that was performed by actual humans. This confirms our method of sampling from the BPC, because the character of the data in these two data sets does not seem to differ substantially, and our priority is to identify the method that works best for the problem at hand.

All work in this thesis is based on the then-current Black-Box Pricing Component (BPC) from 2008, and train ticket prices for IC trains in the German railway system are rumored to now follow the same fare structure as ICE trains, i.e. they are no longer simply based on the distance traveled. For this reason, all models created using an obsolete data set are not applicable to the prediction of current railway fares. If

[1] http://www.rulequest.com/download.html
[2] http://www.csie.ntu.edu.tw/c̃jlin/libsvm/
[3] http://svmlight.joachims.org/

at all possible, future work should also focus on the acquisition of a current data set, possibly using web scraping techniques. An up-to-date data set is essential when trying to minimize the current fare cost.

A Detailed Evaluation Results

Committees	Neighbors	MAE	RMSE	RRSE	RAE	Rsq
1	1	538	844	0.258	0.193	0.934
1	3	503	769	0.235	0.180	0.945
1	5	493	754	0.231	0.177	0.947
1	7	488	747	0.228	0.175	0.948
1	9	485	743	0.227	0.174	0.948
5	1	506	780	0.239	0.182	0.943
5	3	475	709	0.217	0.170	0.953
5	5	468	694	0.212	0.168	0.955
5	7	464	687	0.210	0.166	0.956
5	9	462	684	0.209	0.166	0.956
10	1	502	774	0.237	0.180	0.944
10	3	471	702	0.215	0.169	0.954
10	5	464	687	0.210	0.167	0.956
10	7	461	681	0.208	0.165	0.957
10	9	458	677	0.207	0.164	0.957
20	1	500	773	0.237	0.179	0.944
20	3	469	701	0.214	0.168	0.954
20	5	462	685	0.210	0.166	0.956
20	7	459	679	0.208	0.164	0.957
20	9	457	675	0.207	0.164	0.957
30	1	499	772	0.236	0.179	0.945
30	3	469	699	0.214	0.168	0.954
30	5	461	684	0.209	0.165	0.956
30	7	458	677	0.207	0.164	0.957
30	9	**456**	674	**0.206**	0.164	**0.958**
50	1	498	771	0.236	0.179	0.945
50	3	468	698	0.214	0.168	0.954
50	5	461	683	0.209	0.165	0.956
50	7	458	677	0.207	0.164	0.957
50	9	**456**	**673**	**0.206**	**0.163**	**0.958**
70	1	498	771	0.236	0.179	0.945
70	3	468	698	0.214	0.168	0.954
70	5	461	683	0.209	0.165	0.956
70	7	458	677	0.207	0.164	0.957
70	9	**456**	**673**	**0.206**	0.164	**0.958**
100	1	498	771	0.236	0.179	0.945
100	3	468	698	0.214	0.168	0.954
100	5	461	683	0.209	0.165	0.956
100	7	458	677	0.207	0.164	0.957
100	9	**456**	**673**	**0.206**	0.164	**0.958**

Table A.1.: Evaluation results for the cubist algorithm. The best result for each metric is printed in bold.

Degree	Nprune	MAE	RMSE	RRSE	RAE	Rsq
1	1	2757	3240	1.000	1.000	
1	3	753	1008	0.311	0.273	0.903
1	7	642	874	0.270	0.233	0.927
1	10	627	856	0.264	0.228	0.930
1	15	627	855	0.264	0.227	0.930
1	20	627	855	0.264	0.227	0.930
1	30	627	855	0.264	0.227	0.930
1	60	627	855	0.264	0.227	0.930
1	100	627	855	0.264	0.227	0.930
2	1	2757	3240	1.000	1.000	
2	3	877	1141	0.352	0.318	0.876
2	7	635	859	0.265	0.231	0.930
2	10	605	836	0.258	0.220	0.933
2	15	**593**	**826**	**0.255**	**0.215**	**0.935**
2	20	**593**	**826**	**0.255**	**0.215**	**0.935**
2	30	**593**	**826**	**0.255**	**0.215**	**0.935**
2	60	**593**	**826**	**0.255**	**0.215**	**0.935**
2	100	**593**	**826**	**0.255**	**0.215**	**0.935**

Table A.2.: Evaluation results for the MARS algorithm. The best result for each metric is printed in bold. Two cells are empty due to division by zero.

C	Sigma	MAE	RMSE	RRSE	RAE	Rsq
0.25	0.13	549	802	0.248	0.199	0.939
0.50	0.13	536	787	0.243	0.194	0.942
1.00	0.13	527	775	0.239	0.191	0.943
2.00	0.13	521	766	0.236	0.189	0.944
4.00	0.13	**518**	761	**0.235**	**0.188**	**0.945**
8.00	0.13	**518**	**760**	**0.235**	**0.188**	**0.945**
16.00	0.13	524	767	0.237	0.190	0.944
32.00	0.13	533	780	0.241	0.193	0.942
64.00	0.13	547	802	0.248	0.198	0.939
128.00	0.13	565	831	0.257	0.205	0.935
256.00	0.13	588	872	0.269	0.213	0.928
512.00	0.13	619	928	0.287	0.225	0.919
1024.00	0.13	664	1013	0.313	0.241	0.905

Table A.3.: Evaluation results for a Support Vector Machine with a radial kernel. The best result for each metric is printed in bold.

C	Degree	MAE	RMSE	RRSE	RAE	Rsq
0.25	1	770	1118	0.345	0.279	0.886
0.25	2	564	830	0.256	0.205	0.935
0.25	3	543	801	0.247	0.197	0.939
0.25	4	532	783	0.242	**0.193**	**0.942**
0.25	5	534	812	0.250	0.194	0.938
0.50	1	769	1119	0.345	0.279	0.886
0.50	2	563	829	0.256	0.204	0.935
0.50	3	541	798	0.246	0.196	0.940
0.50	4	**531**	**781**	**0.241**	**0.193**	**0.942**
0.50	5	537	815	0.252	0.195	0.937
1.00	1	769	1119	0.345	0.279	0.886
1.00	2	563	828	0.256	0.204	0.935
1.00	3	540	797	0.246	0.196	0.940
1.00	4	**531**	783	0.242	**0.193**	**0.942**
1.00	5	543	833	0.257	0.197	0.934
2.00	1	769	1119	0.345	0.279	0.886
2.00	2	563	828	0.256	0.204	0.935
2.00	3	539	796	0.246	0.196	0.940
2.00	4	532	786	0.243	**0.193**	0.941
2.00	5	553	912	0.281	0.200	0.920
4.00	1	769	1119	0.346	0.279	0.886
4.00	2	563	828	0.255	0.204	0.935
4.00	3	539	796	0.246	0.196	0.940
4.00	4	534	794	0.245	0.194	0.940
4.00	5	565	1040	0.320	0.205	0.893
8.00	1	769	1119	0.346	0.279	0.886
8.00	2	563	828	0.255	0.204	0.935
8.00	3	539	796	0.246	0.196	0.940
8.00	4	537	803	0.248	0.195	0.939
8.00	5	590	1327	0.408	0.214	0.850
16.00	1	769	1119	0.346	0.279	0.886
16.00	2	562	828	0.255	0.204	0.935
16.00	3	539	797	0.246	0.196	0.940
16.00	4	541	816	0.252	0.196	0.937
16.00	5	609	1340	0.412	0.221	0.844
32.00	1	769	1119	0.346	0.279	0.886
32.00	2	562	827	0.255	0.204	0.935
32.00	3	540	798	0.246	0.196	0.940
32.00	4	545	835	0.258	0.198	0.934
32.00	5	626	1251	0.386	0.227	0.859
64.00	1	769	1119	0.346	0.279	0.886
64.00	2	562	827	0.255	0.204	0.935
64.00	3	541	799	0.247	0.196	0.940
64.00	4	553	851	0.263	0.201	0.932
64.00	5	680	1550	0.477	0.246	0.807
128.00	1	769	1119	0.346	0.279	0.886
128.00	2	563	828	0.256	0.204	0.935
128.00	3	542	803	0.248	0.196	0.939
128.00	4	573	894	0.276	0.208	0.924
128.00	5	748	1561	0.481	0.271	0.803
256.00	1	770	1120	0.346	0.279	0.885
256.00	2	563	829	0.256	0.204	0.935
256.00	3	554	817	0.252	0.201	0.937
256.00	4	636	985	0.304	0.231	0.911
256.00	5	940	2318	0.714	0.341	0.707

Table A.4.: Evaluation results for a Support Vector Machine with a polynomial kernel. The best result for each metric is printed in bold.

B Descriptive Statistics Formulae

Let $x = \{x_1, \ldots, x_n\}$ be a discrete set of n numbers.

The arithmetic **mean** (average) is given by

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{B.1}$$

Reorder x_i so that $x_1 < x_2 < \ldots < x_n$. The statistical **median** is given by

$$\tilde{x} = \begin{cases} x_{\left(\frac{n}{2}\right)} & \text{if } n \text{ is odd} \\ x_{\left(\frac{n+1}{2}\right)} & \text{if } n \text{ is even} \end{cases} \tag{B.2}$$

The *corrected* $(n-1)$ sample **variance** is given by

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{B.3}$$

The *corrected* $(n-1)$ sample **standard deviation** is given by

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{B.4}$$

Let $x = \{x_1, \ldots, x_n\}$ and $y = \{y_1, \ldots, y_n\}$ be sets of two attributes for an observation.

The sample **covariance** is given by

$$s_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \tag{B.5}$$

The Pearson product-moment **correlation coefficient** (PPMCC) is given by

$$r_{xy} = \frac{s_{xy}}{s_x s_y} \tag{B.6}$$

C Glossary

BPC Black-Box Pricing Component. 6, 21, 51, 57

CART Classification and Regression Tree. 13, 57

CV Cross-Validation. 19, 57

EC EuroCity. 6–8, 27, 30, 43, 57

IC InterCity. 6–8, 27, 30, 43, 51, 57

ICE InterCity-Express. 6, 7, 21, 27, 30, 43, 51, 57

ID3 Iterative Dichotomiser 3. 13, 15, 57

IR InterRegio. 6, 27, 57

MAE mean-absolute error. 20, 37, 38, 40–42, 44, 45, 57, 61

MARS Multivariate Adaptive Regression Splines. 10, 12, 13, 39, 41, 45, 47, 50, 51, 57

MLP Multilayer Perceptron. 18, 34, 42, 50, 57

MOTIS Multi-Objective Traffic Information System. 2, 3, 6–8, 21, 22, 24, 26, 27, 36, 41, 50, 51, 57

MSE mean-squared error. 20, 57

NNET Neural Network. 57

RAE relative-absolute error. 20, 45, 57

RB RegionalBahn. 6, 21, 27, 57

RE RegionalExpress. 6, 21, 27, 57

RMSE root mean-squared error. 20, 45, 57

RRSE root relative-squared error. 20, 45, 57

RSE relative-squared error. 20, 57

Rsq R squared. 45, 57

SVM Support Vector Machine. 10, 16, 38, 43, 47, 50, 51, 57

Bibliography

Bergmeir, Christoph and José M. Benítez (2012). Neural Networks in R Using the Stuttgart Neural Network Simulator: RSNNS. In: *Journal of Statistical Software*, 46.7, pp. 1–26 (cited on p. 34).

Bishop, Christopher M. (1995). *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc. ISBN: 0198538642 (cited on p. 18).

– (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. ISBN: 0387310738 (cited on pp. 11, 16, 18).

Borovicka, Tomas, Marcel Jirina Jr., Pavel Kordik, and Marcel Jirina (2012). Selecting Representative Data Sets. In: *Advances in Data Mining Knowledge Discovery and Applications*. InTech (cited on p. 19).

Breiman, Leo (1993). *Classification and regression trees*. CRC Press. ISBN: 0412048418 (cited on p. 13).

Burges, Christopher J. C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. In: *Data Mining and Knowledge Discovery*, 2.2, pp. 121–167. ISSN: 13845810 (cited on p. 16).

Cohen, William (1995). Fast effective rule induction. In: *Proceedings of the 12th International Conference on Machine Learning*. Tahoe City, CA, USA: Morgan Kaufmann, pp. 115–123 (cited on p. 8).

Cristianini, Nello and John Shawe-Taylor (2000). *An introduction to Support Vector Machines and other kernel-based learning methods*. New York, NY, USA: Cambridge University Press. ISBN: 0521780195 (cited on p. 16).

Diebold, Francis X. (2000). *Elements of Forecasting*. 2nd ed. Nashville, TN, USA: South-Western College Publishing. ISBN: 0324023936 (cited on p. 8).

Etzioni, Oren, Rattapoom Tuchinda, Craig A. Knoblock, and Alexander Yates (2003). To buy or not to buy: mining airfare data to minimize ticket purchase price. In: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, pp. 119–128. ISBN: 1581137370 (cited on p. 8).

Fayyad, Usama and Keki Irani (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In: *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. Chambery, France: Morgan Kaufman, pp. 1022–1027. ISBN: 1558603000X (cited on pp. 10, 15).

Friedman, Jerome H. (1991). Multivariate adaptive regression splines. In: *The Annals of Statistics*, 19, pp. 1–67. ISSN: 00905364 (cited on pp. 12, 13).

Granger, Clive W. (1980). *Forecasting in business and economics*. Vol. 2. New York, NY, USA: Academic Press. ISBN: 9780122951800 (cited on p. 8).

Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten (2009). The WEKA data mining software: an update. In: *SIGKDD Exploration Newsletter*, 11.1, pp. 10–18. ISSN: 19310145 (cited on pp. 15, 34, 50).

Harnisch, Stefan and Helge Nuhn (2010). *Analyse und Verbesserung der Preisschätzung im MOTIS-System*. German. Tech. rep. Technische Universität Darmstadt (cited on pp. 8, 30, 36, 51).

Hornik, Kurt, Christian Buchta, and Achim Zeileis (2009). Open-Source Machine Learning: R Meets Weka. In: *Computational Statistics*, 24.2, pp. 225–232. ISSN: 09434062 (cited on p. 34).

Hunt, Earl B. (1962). *Concept learning: An information processing problem*. Wiley. ISBN: 0882751522 (cited on p. 13).

Karatzoglou, Alexandros, Alexander J. Smola, Kurt Hornik, and Achim Zeileis (2004). kernlab – An S4 Package for Kernel Methods in R. In: *Journal of Statistical Software*, 11.9, pp. 1–20 (cited on p. 34).

Kohavi, Ron (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Montreal, Quebec, Canada: Morgan Kaufmann, pp. 1137–1143. ISBN: 1558603638 (cited on p. 19).

Kuhn, Max (2008). Building Predictive Models in R Using the caret Package. In: *Journal of Statistical Software*, 28.5, pp. 1–26. ISSN: 15487660 (cited on p. 34).
– (2013). *caret: Classification and Regression Training*. R package version 5.16-04 (cited on pp. 28, 34).
Kuhn, Max and Kjell Johnson (2013). *Applied Predictive Modeling*. Springer-Verlag New York, Inc. ISBN: 9781461468486 (cited on p. 13).
Kuhn, Max, Steve Weston, Chris Keefer, and Nathan Coulter (2013). *Cubist: Rule- and Instance-Based Regression Modeling*. R package version 0.0.13 (cited on p. 34).
Kullback, Solomon and Richard A. Leibler (1951). On information and sufficiency. In: *The Annals of Mathematical Statistics*, 22.1, pp. 79–86 (cited on p. 15).
McCulloch, Warren S. and Walter Pitts (1943). A logical calculus of the ideas immanent in nervous activity. In: *The Bulletin of Mathematical Biophysics*, 5.4, pp. 115–133 (cited on p. 16).
Mitchell, Thomas M. (1997). *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc. ISBN: 0070428077 (cited on p. 15).
Müller-Hannemann, Matthias and Mathias Schnee (2005). Paying Less for Train Connections with MOTIS. In: *Proceedings of the 5th Workshop on Algorithmic Methods and Models for Optimization of Railways*. Palma de Mallorca, Spain (cited on pp. 6, 8, 36).
– (2007). Finding all attractive train connections by multi-criteria Pareto search. In: *Proceedings of the 4th Workshop on Algorithmic Methods and Models for Optimization of Railways*. Bergen, Norway, pp. 246–263 (cited on p. 6).
Müller, Klaus-Robert, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf (2001). An introduction to kernel-based learning algorithms. In: *IEEE Transactions on Neural Networks*, 12.2, pp. 181–201. ISSN: 10459227 (cited on p. 16).
Quinlan, John R. (1986). Induction of decision trees. In: *Machine Learning*, 1.1, pp. 81–106. ISSN: 08856125 (cited on p. 13).
– (1987). Generating production rules from decision trees. In: *Proceedings of the 10th International Joint Conference on Artificial Intelligence*. Vol. 1. Milan, Italy: Morgan Kaufmann, pp. 304–307 (cited on p. 16).
– (1992). Learning with continuous classes. In: *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*. Vol. 92, pp. 343–348. ISBN: 981021250X (cited on p. 15).
– (1993a). *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann. ISBN: 1558602380 (cited on p. 15).
– (1993b). Combining instance-based and model-based learning. In: *Proceedings of the 10th International Conference on Machine Learning*. Amherst, MA, USA, pp. 236–243 (cited on pp. 16, 38).
– (July 10, 2013a). *An Overview of Cubist*. URL: `http://rulequest.com/cubist-unix.html` (cited on pp. 15, 16).
– (July 10, 2013b). *C5.0: An Informal Tutorial*. URL: `http://www.rulequest.com/see5-unix.html` (cited on p. 15).
R Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. ISBN: 3900051070 (cited on p. 34).
Reed, Russell D. and Robert J. Marks (1998). *Neural smithing: supervised learning in feedforward artificial neural networks*. Cambridge, MA, USA: The MIT Press. ISBN: 0262181908 (cited on p. 18).
Robusto, C. C. (1957). The cosine-haversine formula. In: *The American Mathematical Monthly*, 64.1, pp. 38–40 (cited on p. 26).
Rosenblatt, Frank (1962). *Principles of neurodynamics*. Spartan Book (cited on p. 16).
Rumelhart, David E., Geoffrey E. Hintont, and Ronald J. Williams (1986). Learning representations by back-propagating errors. In: *Nature*, 323.6088, pp. 533–536 (cited on p. 16).
Russell, Stuart and Peter Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press. ISBN: 0136042597 (cited on pp. 16–18).

Schölkopf, Bernhard and Alexander J. Smola (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: The MIT Press. ISBN: 0262194759 (cited on p. 16).

Shearer, Colin (2000). The CRISP-DM Model: The New Blueprint for Data Mining. In: *Journal of Data Warehousing*, 5.4, pp. 13–22 (cited on p. 7).

Smola, Alexander J. and Bernhard Schölkopf (2004). A tutorial on support vector regression. In: *Statistics and Computing*, 14.3, pp. 199–222. ISSN: 09603174 (cited on p. 16).

Sutton, Richard S. and Andrew G. Barto (1998). *Introduction to Reinforcement Learning*. 1st ed. Cambridge, MA, USA: The MIT Press. ISBN: 0262193981 (cited on p. 8).

Vapnik, Vladimir N. (1995). *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc. ISBN: 0387987800 (cited on p. 16).

Wang, Yong and Ian H. Witten (1997). Inducing model trees for continuous classes. In: *Proceedings of the 9th European Conference on Machine Learning*, pp. 128–137 (cited on p. 15).

Witten, Ian H., Eibe Frank, and Mark Hall (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. 3rd ed. San Francisco, CA, USA: Morgan Kaufmann. ISBN: 0123748569 (cited on pp. 11, 19, 20, 34).

List of Figures

List of Tables