Niedermair Thomas, BSc

# Predictive Analytics & Customer Segmentation to Support Invoice Collection and Decision Making Processes

**MASTER'S THESIS**

to achieve the university degree of

Diplom-IngenieurIn

Master's degree programme: Computer Science

submitted to

**Graz University of Technology**

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.techn. Helic Denis

Institute of Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, May 2018

## Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

## Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum

Unterschrift

*...to my beloved parents, Meinrad and Helga,*
*for their endless love, support, and encouragement.*

## Abstract

Payment forecasts of the receipt of payments within their invoices lead time can be considered as a valuable analytical strategy to support companies throughout various invoice collection steps. Especially in small and medium-sized companies, such forecasts may contribute supportive advice towards quick responses on possible payment outages for overcoming own liquidity issues. In this thesis, we investigated various machine learning techniques to determine a suitable strategy for identifying such late paying customers. The considered classification models focused thereby mainly on the past customer's payment behaviors while trying to reveal hidden patterns within companies collected records. Moreover, we reviewed in this thesis all required processing steps which we conducted for analyzing, preparing, training and evaluating our different datasets, strategies, and models. Results demonstrated that the performance for identifying such payment outages got strongly influenced by the underlying structure and amount of used invoice records. Nevertheless, our experiments exposed that a reasonable amount of historical invoice records can enhance a proper customer segmentation and early identification of late payments with additional classification interpretability while choosing the right model.

# Kurzfassung

Zahlungsprognosen über den Eingang von Zahlungen innerhalb der Vorlaufzeit von Rechnungen können als eine wertvolle analytische Strategie angesehen werden, um Unternehmen bei verschiedenen Schritten der Rechnungseinholung zu unterstützen. Vor allem in Klein- und Mittelunternehmen können solche Prognosen unterstützende Ratschläge für schnelle Reaktionen auf mögliche Zahlungsausfälle zur Überwindung eigener Liquiditätsprobleme liefern. In dieser Masterarbeit haben wir dabei verschiedene Techniken des Maschinellen Lernens untersucht, um eine geeignete Strategie zur Identifizierung solcher spät zahlender Kunden zu ermitteln. Die betrachteten Klassifikationsmodelle konzentrierten sich dabei hauptsächlich auf das vergangene Zahlungsverhalten der entsprechenden Kunden und versuchten dabei, verborgene Muster innerhalb der gesammelten Datensätze aufzudecken. Darüber hinaus haben wir in dieser Arbeit alle erforderlichen Verarbeitungsschritte untersucht, welche wir zur Analyse, Vorbereitung, Training, und Evaluierung unserer verschiedenen Datensätze, Strategien und Modelle durchgeführt haben. Die Ergebnisse unserer Arbeit zeigten, dass die Performance zur Identifizierung solcher Zahlungsausfälle stark von der zugrunde liegenden Struktur und Menge der verwendeten Rechnungsdaten beeinflusst wurde. Dennoch haben unsere Experimente gezeigt, dass durch eine angemessene Menge an historischen Rechnungsdatensätzen eine entsprechend gute Kundensegmentierung und frühe Identifizierung von Zahlungsverzögerungen inklusive Interpretierbarkeit bei der Auswahl der richtigen Modelle erreicht werden kann.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In-depth data analytics of different business processes in enterprise and finance sectors is often a crucial and complex task to obtain insightful knowledge about complex systems. Especially when various external and internal factors influence these diverse economic structures, an interpretable and in practice applicable solution is often sought to understand and control these processes. Throughout this broad field of challenging tasks, a core area which influences undoubtedly every firm's financial position, rests in the invoice and collection management. Thereby, especially the transformation of outstanding invoices into overdue invoices needs to be minimized and preferably avoided. However, identifying such invoices which are likely to turn into problematical ones can be considered as a challenging task for itself, because every customer exposes a different behavior towards his or her willingness to pay an invoice on-time. Furthermore, many companies struggle with a proper overview of their outstanding invoices and consequently lack in taking preliminary actions towards controlling these issues. Whereby especially those invoices which are already past their *pay term due* need to be appropriately handled. The intent of this thesis is consequently to transform various business information into an asset such that these problems can be properly tackled. An increased internal efficiency in the overall payment collection process could thereby be gained by the use of Machine Learning (ML) techniques.

This chapter will help to understand the basic background behind late payments and underlines the motivation following this thesis. Furthermore, we will explain the related problems with which companies have to deal during their invoice collection process, and what the goals and contributions of this thesis are. Followed by the limitations which we had to make to evaluate our findings properly and to create concise boundary conditions for this work. Last but not least, we presented a short outline of this thesis.

## 1.1 Background and Motivation

It is commonly known that late paying customers can drive a company into significant organizational problems and bottlenecks considering a companies liquidity. This is espe-

cially true for small and medium-size companies which can only handle a certain amount of payment outages. The reason for this is that in contrary to non-store online retailers, many traditional ones do not get immediately paid for the purchased service or goods by their customers. The so-called *invoice payment terms and conditions* are therefore usually an essential part of every invoice. Besides late payment consequences (e.g. extra charge of overdue fees) and the preferred payment method, this section of an invoice explicitly states a defined range of days within which the retailer expects the invoice payment. In other words, this range of days can be seen as a period within which the company provides a sort of short-term credit to their customers. If a customer does not pay up his debts within this defined range of days, the invoice is considered as late payment, and further actions in the invoice collection process need to be taken. In general, there exist many different varieties of payment terms that are widely spread among several retail sectors which may differ from company to company e.g.: ”Payment 30 days after invoice date”, ”Payment 10 days after invoice date”, or ”Payment upon receipt”. Whereby the first two terms would mean that the customer is asked to pay the total invoice amount within 30 or rather 10 days starting from the invoice issue date. Contrary to ”Payment upon receipt”, which intends to clarify immediate payment consideration after the receipt of the invoice.

However, as two separate studies from leading accounting software systems by Xero[1] and FreshBooks[2] demonstrate, not stating a specific amount of days may leave too much room for interpretation by the debtor; actual payment considerations may thus stretch from immediate payment up to commonly used 30-day terms [Lim15; Inc16].

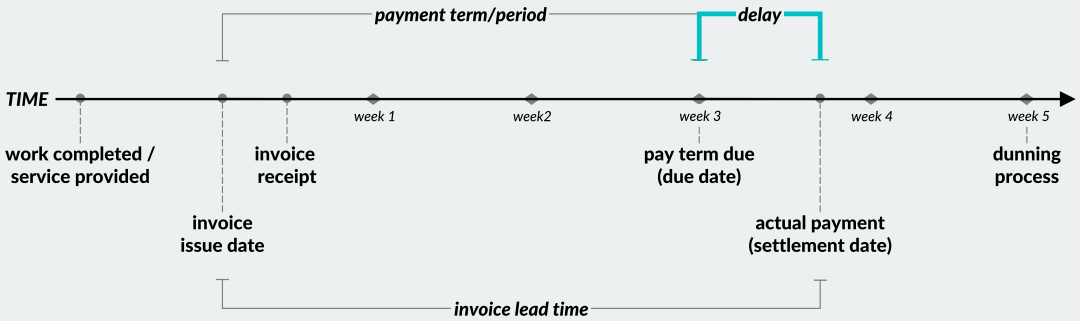


Figure 1.1: **Generalized invoice workflow**, whereby the payment period is defined with three weeks (21 days) and an additional waiting period past the due date of two weeks (14 days) is considered for demonstrating purposes. The period which starts with the invoice issue date up to the defined pay term due represents the payment term/period. Consequently, everything past this pay term due is considered as the actual invoice payment delay. The total time-frame where a short-term credit is provided to the customer (payment term/period + delay) is referred to as invoice lead time.

To demonstrate the typical workflow and to enhance the understanding of the basic terminology, Figure 1.1 visualizes the generalized procedure that an invoice usually passes through. As a starting point we assume that some goods or service has been provided to

[1] https://www.xero.com
[2] https://www.freshbooks.com

the customer and consequently the invoice is being created. The invoice creation date, which is further referred to as *invoice issue date*, is usually the date on which the invoice is sent to the customer; if this is not done electronically, the customer will receive the invoice after a short period by mail or postal delivery. The customer now has a defined time-frame to pay for the provided service or purchased goods. This period is referred to as *payment term* or *payment period* and is commonly clearly marked on the invoice. In our example, the defined payment term is precisely three weeks. If the customer pays the invoice within this period, the invoice is considered as an on-time payment, otherwise, if no payment has been received until the *pay term due* (last day of the defined payment term), the invoice is considered as a late payment. The date on which an invoice is actually paid is called *settlement date*, and the period between pay term due and settlement date is seen as the total measure of invoice *delay*. The period starting from invoice issue date until the actual settlement date represents the *invoice lead time*. In other words, this reflects the total amount of time for which the short-term credit was provided to the customer. Once even the invoice due date is exceeded, an additional waiting period is commonly provided to the customer (in our example two extra weeks). The invoice is nonetheless already considered as a late payment, and the invoice issuer may charge additional fees. As soon as this final deadline has been reached, the customer is considered as insolvent and consequently the *dunning process* starts, which triggers further actions in the invoice collection process [You+14; ZS12; Zen+08]. Note that from this point forward, we do not track the invoice collection process anymore and consequently no further steps are considered in this thesis due to the set limitations.

In practice it can be very challenging to evaluate in advance whether a customer will pay an invoice on-time or not. Either long-term experience with established customers, or additional background information from third-party credit management agencies like Schufa[3] are typically needed to perform a valid decision making process. In general one can say that the sooner a company can gauge the probability of late payments, the earlier the company can try to take preventive actions like sending out notifications or calling the related debtors. In many cases a simply overlooked invoice or other human errors might often be the most straightforward reason for late payments, but of course also software errors, company holidays, or even product defects might be imaginable. The primary impulse behind this thesis was therefore to think of the payment prediction process as a customer ranking which is based on the related late payment probabilities. One of the simplest solutions would therefore be to send out payment reminders several days after the due date, or even some days before the due date without an incoming payment. We think nevertheless that this strategy might be a primary cause of annoyed or bothered clients which could even result in increased customer churns. A more valuable strategy was consequently needed to sort the outstanding invoices by their probabilities of late payments so that prevention strategies can be followed in a smart order. With this setup in mind, additional consideration of individual cases regarding national holidays or the country of origin allowed us for example to outtake the chance of contacting customers who are on vacations and who would pay on-time anyway once returning to regular business. To neatly summarize the focus which we set in this context, we illustrate in Figure 1.2 a high-

[3] https://www.schufa.de

level *order-to-cash process* which companies usually follow. We consequently highlighted the two main sections on which this thesis builds upon, namely: Collection Management and Deductions/Dispute Management.

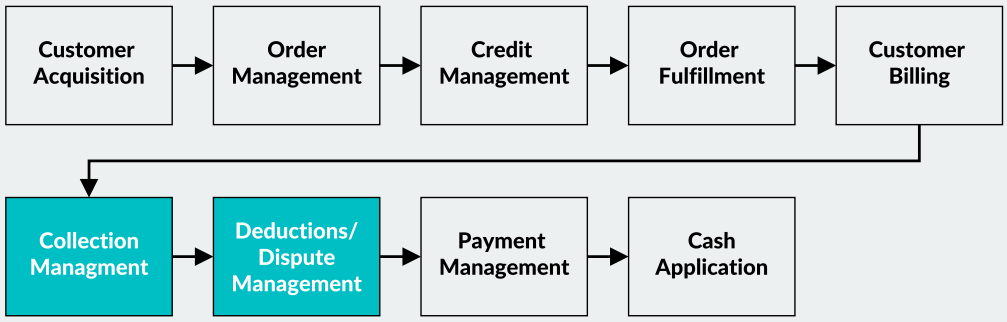


Figure 1.2: **Generalized high-level *order-to-cash process*** that represents the typical process steps which a company encounters during order processing. Starting from customer acquisition over order fulfillment, and billing to the final cash application. Considering invoice payment predictions, we mainly focused on the collection management and deductions/dispute management processes. Illustration adopted from Zeng et al. [Zen+08].

Overall we can conclude that the needed knowledge about customer payment behaviors could possibly be reduced with the help of an automated invoice payment ranking system. The related invoice payments could thereby be automatically reported according to their probability of late payments. We think that this may be especially important for small and medium-sized companies who have to react quickly to possible payment outages so that they can overcome financial bottlenecks.

The objective of this thesis is thereby to perform predictive analytics on the invoice payment behavior of customers, which might consequently support the invoice collection and decision-making process in companies. In this context, the main idea was to focus on historical payment behaviors and to predict the probability of late payments, while considering the specified invoice payment terms. The intention was thereby to reduce the manual effort during the invoice collection process and to demonstrate a suitable attempt for an early warning system which identifies those invoices which are most likely to get overdue.

## 1.2 Problem Statement

Unfortunately, collecting unpaid invoices is part of almost every companies daily business. This collection process faces thereby mainly the problem of when to start contacting a customer to remind him or her about a missing payment. While waiting too long might result in own payment bottlenecks, reacting too early can lead to annoyed and bothered customers. In addition, firms tend to have problems while manually gauging the creditworthiness of new or not so established customers. This thesis aims to find a reliable ranking strategy for customers which reflects the late payment probability right after or

during the invoice creation process. Along with this ranking, we try to identify the most significant features which show correlated influences on customers late payment behaviors. The resulting findings might consequently be used as a supportive method for assessing the creditworthiness of a companies clients. In more detail, the research questions which this thesis addresses can be summarized as follows:

- **RQ 1:** Which ML classifiers are most suited for the problem of invoice payment classification into ”on-time payments” and ”late payments”?
- **RQ 2:** Which invoice and customer features are the most significant ones to predict whether an invoice will be paid on-time or late?
- **RQ 3:** Would such invoice and customer features differ drastically while constructing ML classifiers for different companies?

## 1.3 Limitations

Although this thesis reached its goal in focusing on the narrowed down topic of payment predictions on companies invoices, there were further limitations which we had to consider.

First, due to the difficult task of finding usable datasets that include business and accounting documents with features considering invoices or purchase orders, we had to limit our research on two rather small datasets. To generalize the results of this thesis, we would need to consider bigger datasets of various companies (preferably from companies of similar sectors or regions). We also did not have access to any background information on the corresponding customers for which invoices have been issued. Valuable customer information like a companies tax returns, age, natural or legal person information etc. might further strongly influence the performance of our predictions.

Secondly, we focused our research only on outgoing invoices. This limitation was necessary due to our set requirement of tackling the problem from the seller’s point of view. Thereby, a company is mainly interested in knowing whether its customers are paying on time or not, and less in its own payment behavior. Consideration of incoming invoices would thereby reflect the companies payment performance to third parties. However, we note that this might influence the final customer’s willingness to pay as well, especially when considering business to business relations.

Last but not least, we worked with only outgoing invoices from returning customers, which means that we discarded invoices from one-time customers. Whereby we mainly performed this step due to the low volume of available one-time customers and the lack of significant features. None of the available one-time customer features would strongly outperform simple random guessing under the available knowledge that customers did not have any correlations within the used datasets, nor did we have any external background information about them.

Despite these limitations, we think that predicting the payment behavior of customers with

an abolition of the mentioned restrictions would most likely result in an improved performance rather than a worse one. This assumption is argued based on the increased available information which would additionally be gained by abolishing these limitations.

## 1.4 Outline

This thesis is consequently structured into seven chapters. The Introduction is followed by Chapter 2 which provides a brief overview of related work which has been conducted in the specific field of invoice payment predictions and the related topic of credit scoring and insolvency modeling. Chapter 3 will then continue to provide a stable groundwork for the methodology which focuses on the applied ML classifiers and reviews also further dataset preprocessing techniques and strategies. The experimental setup consists of the conducted dataset analyzing and preprocessing steps, as well as model training, evaluation, and testing which will be reviewed in Chapter 4. Chapter 5 reflects the performance evaluations of the invoice payment classification results on the used ML classifiers in this thesis. A related discussion which focuses on answering the stated research questions is consequently found in Chapter 6. Finally, Chapter 7 concludes this thesis by reviewing the main findings and presenting further fields of interest which could be considered in future work.

# Chapter 2

# Literature Review

Predictive analytics and the application of ML is an established concept and widely used tool in business and finance sectors to support decision makers in their daily business. Besides that, many different measurement metrics are commonly in use to reflect the overall evaluation of a companies performance in various aspects. This section will thereby provide an insight into two basic metrics which are used to measure the performance of invoice collection processes, followed by a brief overview of related work in the field of late payment predictions and associated customer rankings. Due to limited work which has been conducted in this specific field, we will further investigate studies which focus on related topics of data mining and predictive analytics, namely, customer credit scoring and insolvency models.

## 2.1 Business Analytics and Metrics

Measuring and analyzing companies potential weaknesses and strengths as for example in terms of supply chain performances or decision-making processes has become an essential part of competitiveness in business and financial environments [Trk+10; GK07]. All of these so-called Key Performance Indicators (KPIs) aim to evaluate and reevaluate a businesses performance in a variety of aspects [BS07], or as Chae [Cha09] describes it in one of his works: "Monitoring KPIs reveals the gap between plan and execution and helps to identify and correct potential problems and issues".

A commonly used measurement to track the performance of invoice collection processes is thereby the *Days Sales Outstanding (DSO)* indicator. This metric expresses the average time in days an invoice remains in the invoice lead time period, it's defined as:

$$DSO = \frac{Outstanding\ Receivables}{Total\ Credit\ Sales^{1}} * Number\ of\ Days \tag{2.1}$$

Apparently, the object is to keep this value as low as possible (optimally zero), considering the fact that it reflects the average time in days short-term credits are provided to

[1] **Credit Sales** refers to all purchases which are made by customers which do not require immediate payment considerations after their acquisition.

customers. At the same time, the DSO measurement could be evaluated against the defined invoice payment term which serves than as an indicator of the overall customer willingness to pay in time.

The frequency of DSO evaluations can vary from company to company e.g.: weekly, monthly, quarterly etc. - whereby the appropriate frequency selection always depends on the pursued objective. Furthermore, even though this metric is widely used, it suffers from high sales volatility and seasonality [LL11]. Such seasonality may for instance be observed in the hospitality sector where different sale peaks throughout the year distort a yearly DSO measurement. A simplified example in Table 2.1 demonstrates this behaviour.

| | **Q1** | **Q2** | **Q3** | **Q4** | **DSO (Yearly)** |
|---|---|---|---|---|---|
| **Company A** | €25.000 | €25.000 | €25.000 | €25.000 | $\frac{10.000}{100.000}$ * 365 = 36,5 days |
| **Company B** | €40.000 | €10.000 | €40.000 | €10.000 | $\frac{10.000}{100.000}$ * 365 = 36,5 days |

Table 2.1: **An example of seasonality distortion using yearly DSO**. *Company A* represents a business with regular sales, and *Company B* represents a business with seasonally increased sales (see *Q1* and *Q4*). Both companies made €100.000 in sales on credit and their outstanding invoice amount at the end of the year is €10.000. By applying the DSO formula, we see that both companies measure a yearly DSO of 36,5 days. In other words, it seams as it takes both companies on average 36,5 days to collect an invoices of their credit sales. However, in reality, *Company A* could outperform *Company B* concerning the actual invoice collection time. The DSO measurements artificially decreases the needed time to collect an invoice due to it's known seasonality problem.

To overcome this dilemma, one might consider using the so-called *Collection Effectiveness Index (CEI)*. Much like DSO, this metric is used to measure the performance of the overall invoice collection process. However, CEI compares the actual collected invoice amount to the highest possible invoice amount which could have been collected in a given period. Consequently, it overcomes the problem with seasonal sale peaks. The metric is defined as:

$$CEI = \frac{Start\,Out.\,Receivables + Credit\,Sales - End\,Total\,Out.\,Receivables}{Start\,Out.\,Receivables + Credit\,Sales - End\,Current\,Out.\,Receivables} * 100 \qquad (2.2)$$

whereby in contrary to DSO, the objective is to reach the highest score as possible - get as close as possible to 100%. Furthermore, the CEI metric is found to be a useful tool for trend analysis. In more detail, this means that an increasing CEI indicates most likely a respectively good invoice collection processes while a decreasing CEI might point out cash flow problems. Nevertheless, as with the DOS metric, the frequency of its evaluation depends on the pursuit analyzation goals and intentions. An example of a quarterly CEI measurement evaluation is presented in Table 2.2.

Generally speaking, selecting the right metrics (including the appropriate evaluation frequency) to measure business performances can be very challenging when looking at the vast amount of KPIs which can be analyzed and looked into. Nevertheless, we found that literature and practice highly recommend the proper use of KPIs in business and finance environments [Par15]. However, it's important to note that the application of the right measurement metric always depends on the defined focus and circumstances within a company.

| | Outstanding Start of Quarter | Credit Sales | Outstanding End of Quarter | On-Time Payments | CEI Metric |
|---|---|---|---|---|---|
| **Q1** | €50.000 | €60.000 | €65.000 | €40.000 | 64% |
| **Q2** | €65.000 | €50.000 | €60.000 | €35.000 | 69% |
| **Q3** | €60.000 | €40.000 | €50.000 | €30.000 | 71% |
| **Q4** | €50.000 | €30.000 | €40.000 | €30.000 | 80% |

Table 2.2: **An example of CEI measurements on a quarterly frequency**. The measurements of quarterly credit sales which were made during one year of sale have been reported including the corresponding outstanding payment calculations in the beginning and end of each quarter. Additionally, on-time invoice payments and the corresponding CEI measurements are tracked to evaluate invoice collection processes. At the end of the year, an overall loos in credit sales is clearly observable and counts as a difficulty which needs to be addressed. Nevertheless, the trend in CEI is still rising, which indicates that incoming payments are still going and the collection process is constantly increasing. Therefore, one could conclude that the company should not run into financial problems any time soon despite its loss in credit sales.

## 2.2 Closely Related Work

Predictive models applied to the specific domain of invoice payment forecasts and related invoice collection improvement strategies have been used and studied so far in a modest manner.

One work from Kim and Kang [KK16] focuses on a late payment prediction model with customer ranking for call centers. Their main idea was to increase the quality of customer contact lists so that a proper distribution among customers who pay on time and those who do not, is fairly allocated among call center agents. To obtain the payment likelihood of every client, they applied five different classification models, i.e.: single Classification Tree, Random Forests (RF), Artificial Neural Network (NN), Support Vector Machine (SVM) and a combining hybrid approach using a combination of the former four models. Their results show that the proposed strategy of using predictive models in combination with customer scoring rules would help call center agents to develop individual collection strategy plans with an improved overall satisfaction of a fair customer list allocation among them.

Younes et al. [You+14] are focusing on identifying and ranking bottlenecks in invoice processing. In their research, they suggest using Markov Chains as a simulation tool for decision improvements on the invoice lead time. Whereby an invoice is seen as a product which is moving through a supply chain from initial receipt to payment. The results of their case study demonstrate that although their approach cannot be used to improve invoice processing time itself, it is a valuable tool to analyze, identify and prioritize opportunities for an efficient improvement in the invoice processing system. In other words, the method can be used to identify major bottlenecks during invoice processing steps i.e.: invoice approvement, data entry, or invoice checking etc..

The main study on which this thesis builds upon is the work of Zeng et al. [Zen+08]. In their study, they use supervised learning methods to predict on a customers basis whether

an individual invoice will be paid on time or not. In particular, the use: Decision Tree (DT), Naive Bayes (NB), Logistic Regression (LR), Boosting Decision Stumps (one level Decision Tree), and the PART algorithm (rule learner approach). Besides the use of one-time invoice data and historical invoice information, they further include customer background information as for example credit limits or region of origin to generate a more accurate prediction model for first-time customers where no invoice history is available. Zeng et al. further conducted experiments in creating a unified model which combines the available information from different companies which lead to an even more sophisticating accuracy due to hidden commonality patterns in the customer behavior across different company domains [Zen+08]. Finally, they conclude in their work that it may be advantageous concerning time-saving, to predict the magnitude of the payment delay so that a related ranking along this quantity can be performed. In other words, one should set the focus for preemptive actions on those invoices with the highest predicted payment delay.

## 2.3 Loosely Related Work

Due to limited work which has been conducted in the field of late payment prediction, we decided to review also some related topics such as credit risk or customer scoring and insolvency modeling. These fields tackle similar problems as in invoice payment forecasts when it comes to strategic decision making considering a customers willingness or ability to pay up financial debts. Consequently, this subsection will briefly review some prior studies which have been conducted in these associated fields.

### 2.3.1 Insolvency Models

As discussed in Section 1.1 insolvency prediction refers to the process of identifying customers who won't or can't pay up for used services or purchased goods after the invoice due date [Das+03; ZS12; Che+13]. Even if the focus of this domain slightly differs from our defined goals, the context of customer classification on payment debt collection remains the same.

Chen et al. [Che+13] proposed a late payment prediction framework for insolvency customers of a telecommunication provider. By using association rules and expert domain knowledge, they started to create clusters of behavioral models from customers who tend to show insolvency. Based on these clusters, highly accurate rules were extracted from various DT models (constructed by numerous attributes) which were finally used to identify the related problematic customers. Their evaluation results showed an accuracy boost of 13% to previously established methods of their telecommunication provider who conducted this evaluation over six months.

Zabkowski and Szczesny [ZS12] also attempted to predict customers insolvency, but in their approach, they focus mainly on the use of NN and DT models. Especially the use of NN models is widely used in many applications and reveals its strengths also in the evaluation process of their thesis. However, even though NN models outperform DT models in the respective prediction performance, the authors conclude that its major

drawbacks still lies in their lack of explainability. Considering this statement, the authors further conclude that insolvency probability predictions can successfully and efficiently be performed by using ML techniques, but when selecting a ML algorithm, one has to consider the tradeoff between performance and interpretability.

### 2.3.2 Credit Risk and Customer Scoring

Another related field where the use of ML algorithms is well established is the domain of customer credit risk and credit scoring forecasts. As mentioned in Section 1.1, we can compare the discussed invoice lead time to a period where a sort of short-term credit is provided to the customer by the invoice issuing company. The fundamental concept behind credit risk and credit scoring is thereby to help lending institutions to evaluate the risk of providing such credits to borrowers. In other words, a ranking of customers creditworthiness is respectively evaluated which reflects the probabilities that customers can pay back their credits within a specified timeframe [HCW07; Har15]. To do this, as Khandani et al. [KKL10] points out in one of their works: ”a good understanding of consumer choice and early warning signs of over-heating in consumer finance are essential to effective macroprudential risk managed policies”.

Khandani et al. [KKL10] elaborated thereby a detailed analysis of forecasting models for customer credit risk by using generalized classification and regression trees (CART). As a conclusion statement, they mention that ML forecasts used in credit risk or customer scoring can positively contribute a valuable information gain in the context of risk management. Whereby ML algorithms, in general, are ideally suited for those large sample sized and complex data sectors as we find in banking and finance.

Huang et al. [HCW07] applied a hybrid SVM-based credit scoring model to evaluate the creditworthiness of customers credit. In their work, they evaluate three different SVM strategies in combination with genetic algorithms which all achieved similar accuracies to established Genetic Programming, Back-Propagation NN, and DT approaches. Nevertheless, they conclude that even tough their SVM-based hybrid approach seems to be a good alternative, the drawback of their credit scoring model is the long training time and its black-box nature.

Harris [Har15] tried to tackle the drawback of this long training time with SMVs by using the Clustered Support Vector Machine (CSVM) approach which he applied again for credit scoring. His research showed that by splitting the data into several clusters before training, a local weighting of the classifier is possible which finally results in a faster overall classification process. His evaluations confirm that CSVM produces comparable results to nonlinear SVM techniques which consequently represents a suitable substitution for them.

Additionally, recent studies in the field of credit risk and customer scoring focus on the use of ensemble methods [Les+15a; AA16; Xia+17].

Xia et al. [Xia+17] present an accurate process of a credit scoring models using Extreme Gradient Boosting (XGBoost). The evaluation of the models are compared to established baseline models i.e.: LR, SVM, NN, DT, and further ensemble methods like Gradient Boosting and Adaptive Boosting (AdaBoost). The experimental results show that the presented XGBoost-based models perform very well on imbalanced datasets and provides good performance concerning their accuracy. Moreover, the proposed model outperforms baseline models and is comparable to other state-of-the-art parallel ensemble methods. Nevertheless, the authors state that "the experimental results also indicate that LR remains a competitor in credit scoring because of its simplicity and efficiency." [Xia+17].

# Chapter 3

# Methodology

In this chapter, we will cover the essential aspects of ML algorithms and techniques of which we made use of in this thesis. The focus lies thereby on providing a stable groundwork of ML knowledge to enable a supportive understanding in the follow-up chapters of our experimental setups and results. In the first section *Supervised Learning Classification*, we will thereby start with an overview of state-of-the-art supervised learning classification models which are commonly in use in literature and practice. Note that we evaluated all of these presented methods for our defined task invoice payment classifications. Followed by *Ensemble Learning: Boosting and Bagging* where we explore the methods which we used to combine the individual classification models to construct more robust and performance-driven ensemble classifiers. In the section of *Performance Measures & Evaluation Methods*, we will cover the used methods and measurements for evaluating the performance of our ML models on the defined classification task. Last but not least, we briefly explain the well-known problem of data imbalance and demonstrate several sampling strategies which were used to overcome this problem in the section of *Sampling Methods*.

## 3.1 Supervised Learning Classification

### 3.1.1 Naive Bayes

Naive Bayes (NB) is one of the simplistic supervised classification models which is based on the Bayes Theorem. Considering its simplicity, it thereby assumes general independence among all features. This means that given a class with $X$ features, each of these features are considered to have no dependency on each other regarding their presence related to the class. Even though this assumption is somewhat optimistic and generally not true, NB it is known as a very effective classification model due to the ease in its applicability and to its stunning performance results while it is often outperforming more sophisticated methods [BB12]. The Bayes Theorem is thereby defined as:

$$P(c_m|x_n) = \frac{P(x_n|c_m)P(c_m)}{P(x_n)} \tag{3.1}$$

whereby it is used to calculate the *posterior probability* $P(c_m|x_n)$ where $x_n \in X$ (*Features*) and $c_m \in C$ (*Classes*). Its individual terms are thereby defined as:

- $P(c|x)$ - the probability of class $c$ given feature $x \Rightarrow$ **posterior probability**.
- $P(c)$ - the **probability of class** $c$.
- $P(x|c)$ - the probability of feature $x$ given class $c \Rightarrow$ **likelihood**.
- $P(x)$ - the **probability of feature** $x$.

To better understand the use of NB as a classification model, the following example will serve as an illustrative instance. Let us assume that we have a set of invoices with a limited set of features and we want to classify whether an invoice payment arrives on-time or not. This setup includes that we have two classes $c_1, c_2 \in C$ ("on-time", "late") and some features $X$: *"invoice amount less than €5.000"* and *"due date month"* as demonstrated in Table 3.1. For illustrative purposes we further assume that we want to classify a new invoice for the month *December* with an invoice *amount less than €5.000* (see Invoice #9 in Table 3.1). By using the observations from the collected invoices and their correspondingly derived frequency tables, we can calculate the respective probabilities for late payment and on-time payment with the help of the previously stated Bayes Theorem. In the end, the class with the highest posterior probability will finally be the one which the NB classifier will predict for the new invoice.

| Collected Invoices | Amount (less €5.000) | Due Date (Month) | Class |
|---|---|---|---|
| **Invoice #1** | €12.000 ⇒ *No* | 15.11.2018 ⇒ *November* | ***on-time*** |
| **Invoice #2** | €4.000 ⇒ *Yes* | 16.11.2018 ⇒ *November* | ***on-time*** |
| **Invoice #3** | €18.000 ⇒ *No* | 20.11.2018 ⇒ *November* | ***late*** |
| **Invoice #4** | €2.000 ⇒ *Yes* | 21.11.2018 ⇒ *November* | ***on-time*** |
| **Invoice #5** | €4.000 ⇒ *Yes* | 28.11.2018 ⇒ *November* | ***on-time*** |
| **Invoice #6** | €13.000 ⇒ *No* | 12.12.2018 ⇒ *December* | ***on-time*** |
| **Invoice #7** | €16.000 ⇒ *No* | 24.12.2018 ⇒ *December* | ***late*** |
| **Invoice #8** | €2.000 ⇒ *Yes* | 25.12.2018 ⇒ *December* | ***late*** |
| **Invoice #9 (PREDICT)** | **€3.000 ⇒ *Yes*** | **28.12.2018 ⇒ *December*** | ***on-time*** |

**Likelihood Table (Amount)**

| Amount | on-time | late | |
|---|---|---|---|
| **< €5.000** | 3 | 1 | $\frac{4}{8} = 0.50$ |
| **≥ €5.000** | 2 | 2 | $\frac{4}{8} = 0.50$ |

**Likelihood Table (Month)**

| Month | on-time | late | |
|---|---|---|---|
| **Nov.** | 4 | 1 | $\frac{5}{8} \approx 0.63$ |
| **Dec.** | 1 | 2 | $\frac{3}{8} \approx 0.37$ |

Table 3.1: **Late payment classification with the NB classifier.** For illustrative purposes, eight invoices serve as small sample dataset composed by only two features: the first one is a binary indicator representing whether the invoice amount is less then €5.000 and the second feature represents the related invoice due date month. The last invoice (Invoice #9) is consequently the one which we want to predict. Note that the class label in the table corresponds to the actual ground-truth of each collected invoice sample. To clarify how the NB classifier uses the probabilities, we further plotted the likelihood tables for the available features.

$$\textit{Probability for } \textbf{\textit{on-time payment}} = P(\textit{on-time} \mid \textit{amount less } €5.000, \textit{ December})$$

$$= \frac{P(\textit{amount less } €5.000 \mid \textit{on-time})\ P(\textit{December} \mid \textit{on-time})\ P(\textit{on-time})}{P(\textit{amount less } €5.000)\ P(\textit{December})}$$

$$= \frac{3/8\ *\ 1/8\ *\ 5/8}{4/8\ *\ 3/8} = 5/32 \approx \textbf{\textit{0.156}}$$

$$\textit{Probability for } \textbf{\textit{late payment}} = P(\textit{late} \mid \textit{amount less } €5.000, \textit{ December})$$

$$= \frac{P(\textit{amount less } €5.000 \mid \textit{late})\ P(\textit{December} \mid \textit{late})\ P(\textit{late})}{P(\textit{amount less } €5.000)\ P(\textit{December})}$$

$$= \frac{1/8\ *\ 2/8\ *\ 3/8}{4/8\ *\ 3/8} = 1/16 \approx \textbf{\textit{0.062}}$$

As we see from the results, NB would classify this new invoice as a corresponding **on-time payment**. Due to the simplification of NB we are able to merely assume feature independency which allowed us to naturally multiply the corresponding feature probabilities.

Overall we concluded that this classification method is rather easy to use and as demonstrated in the example above, also suitable for small datasets. Many studies have shown that NB is able to outperform even more sophisticated alternatives despite its drastically assumptions about feature independency [BB12]. Nevertheless, NB is also known for its weaknesses especially when it comes to class specific probabilistic output results. Additionally, it is worth to note that the simplified assumptions cause somehow a loss in the possible reachable precision because in real-world scenarios it is rather unlikely to come along problems whose features do not have any dependencies among each other.

### 3.1.2 Logistic Regression

The Logistic Regression (LR) model is another popular method which is commonly used for linear and binary classification problems. It is basically a classification algorithm which estimates the class probabilities via linear functions under consideration of (linear) feature dependencies. The main idea behind LR is thereby to predict for a specific sample $x$ the actual probability of its belonging to a specific class $c$ while at the same time representing the probability of not belonging to this specific class.

To understand how LR works, we had to consider a few things. First, like in the NB classifier, $P(x)$ denotes the probability of a sample $x \in X$ to occur or to be part of a specific class. Second, we need to consider the odds of an event. This is simply defined as the outcome of an event divided by all other possible outcomes e.g.: the probability of observing a dice roll of 1 is $P(1) = 1/6$, whereby the $odds(1) = 1/5$.

The odds are thereby defined as:

$$odds(x) = \frac{P(x)}{1 - P(x)} \tag{3.2}$$

Next, the *logit*-function, which is simply the logarithm of the *odds*-function, is used to map the probability ranges (from 0 to 1) into the entire range of real numbers, it is defined as:

$$logit(P(x)) = \ln\left(\frac{P(x)}{1-P(x)}\right) = -\ln\left(\frac{1}{P(x)} - 1\right) \tag{3.3}$$

This transformation is than further used to model the linear relationship between feature values and its corresponding weights which can be rewritten to:

$$logit(P(y=1|x)) = w_0x_0 + w_1x_1 + \cdots + w_nx_n = \sum_{i=0}^{n} w_ix_i = w^Tx \tag{3.4}$$

whereby $P(y=1|x)$ refers to the probability of a sample belonging to class 1 given its features $x$. Finally, the characteristic *sigmoid*-function is received by taking the inverse of the *logit*-function, called *logistic*-function:

$$logit^{-1}(P(x)) = \frac{e^x}{e^x+1} = \frac{1}{1+e^{-x}} \tag{3.5}$$

$$sigmoid(z) = \frac{1}{1+e^{-z}} \tag{3.6}$$

whereby $z$ refers again to the linear relationship between feature values and the corresponding weights, as $z = w^Tx$. As can we can see in Figure 3.1, the *sigmoid*-function mapped each real value into the range between 0 and 1, whereby larger values resulted close to 1 and smaller ones appeared close to 0. We can further see this behavior as a sort of activation function which decides whether a sample with features $x$ and weights $w$ belongs to a class or not. As mentioned before, the output of the *sigmoid*-function represents thereby the probability of a sample belonging to a class. This means if the output of $sigmoid(z) = P(y=1|x;w)$ was for example 0.7, it would indicate that the LR model classified this sample with 70% in belonging to class $y$ and with 30% in not belonging to this class.

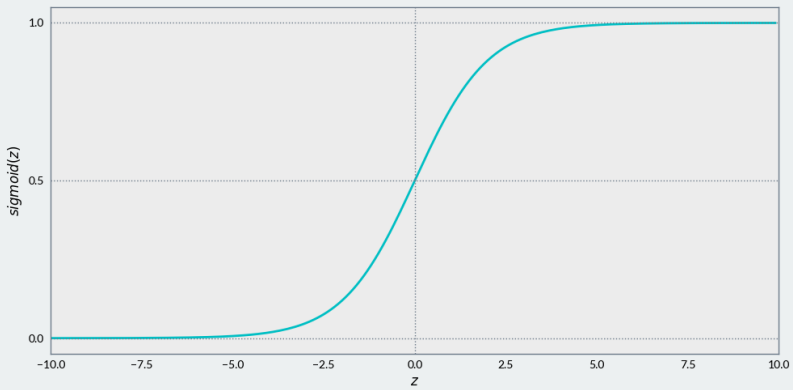


Figure 3.1: **The logistic curve representing the** *sigmoid*-**function** in the range of $-10$ *to* 10. Its typical characteristics can be directly read from the plotted curve: when $z$ goes to $+\infty$ we obtain a value close to 1, while when $z$ goes to $-\infty$ we get a value close to 0. Note that its inflection point lies exactly at 0.5 which can be seen as the default decision boundary when associating a sample either to be part of a class or not.

In terms of binary classification, we can further define a so-called *step-function* which returns the predicted classes. Assuming, that we again want to predict whether an invoice payment will be classified as late payment or as an on-time payment, such a *step-function* could look like this:

$$\text{predicted class} = \begin{cases} \textbf{late} \text{ payment} & \textbf{if } sigmoid(z) \geq \epsilon \\ \textbf{on-time} \text{ payment} & otherwise \end{cases}$$

whereby $\epsilon$ represents the threshold which is seen as a decision boundary on deciding whether to classify a sample as a positive one or not. In other words, it represents how much confidence the classifier needs to have until we can positively classify it as late payment.

We concluded that LR is a rather simple and robust approach which performs very well on linear and binary classification tasks as well as on multi-class classification problems (also with small datasets). Its probabilistic output is easy to interpret and allows tuning modifications concerning its threshold to receive more sophisticated classification results. Additionally, this model outstands also for its possible use as a feature analyzation tool. As used like that, the corresponding coefficients for each feature have to be analyzed and reported. The most promising features are thereby favoring a positive class prediction and can be told apart from the ones who contribute the most in not belonging to a class. Moreover, it is possible to identify also feature which are irrelevant due to their non-contribution in neither direction during the prediction process. Despite these advantages, the LR model comes also with some limitations. In general, LR requires for instance that each sample is independent of each other. That means if there exists some dependency among samples, the results may be overfitted due to an overestimation of feature weights. Besides that, the model may further be vulnerable to sampling biases because it relies on the underlying *logit*-function which can amplify a predictions accuracy.

### 3.1.3 Decision Tree

The Decision Tree (DT) classifier is another established classification model which especially scores with its easy interpretability considering the outcome of its classification result. As the name suggests, it can be seen as a simple decision rule tree, composed of many nodes where every one of them (besides the leaf-nodes) is considered as a decision boundary.

To learn such decision boundaries, the DT algorithm tries to identify features with the highest information gain. This means that the features with the most informative characteristics are used to create nodes (introducing a split in the tree) while constructing the tree from top to bottom. The algorithm proceeds in creating more and more nodes while creating new decision boundaries on its way down the tree. This procedure continues until each child node only contains samples of one single class or other limiting conditions are met - the final nodes at the bottom of the tree are thereby known as leaf nodes. In practice, this strategy might however very quickly lead to extremely deep trees which are no longer interpretable due to their high complexity. To overcome this behavior, a conventional method is to prune the DT which refers merely to setting a maximal limiting condition for its depth.

As a measure for the informative characteristic of a feature, commonly the Gini Impurity or the concept of Entropy is used as an objective function to construct the decision boundaries in the DT. It is however worth to note that in literature and practice both methods commonly return rather similar results [Ras15; HTF09]. The **Entropy** is thereby defined as:

$$H(X) = -\sum_{x} p(x)\, log\, p(x) \tag{3.7}$$

whereby the algorithm tries to optimize (maximize) the mutual information for deciding on which features to split.On the other hand, the **Gini Impurity** is defined as:

$$I_G(p) = 1 - \sum_{i} {p_i}^2 \tag{3.8}$$

whereby the algorithm aims to minimizes the probability of misclassification.

In Figure 3.2 we provided an illustrative example to support the understanding of how a DT classifies samples; we try again to classify whether an invoice will be paid on-time or not. Once the DT algorithm constructed all decision boundaries (nodes), the classification process starts at the top root note and advance down the tree nodes while answering the questions which arise in the corresponding nodes. At each node which we face, the tree branches into $n$ subnodes and depending on the provided answer, the algorithm follows the path down the tree until a final leaf node is reached. Every node in the DT belongs thereby to a specific type of class. Meaning that when we end up in a final leaf node, we just have to look at its assigned class and we know how the DT classifies a sample. It is worth to mention at this point that the DT can handle categorical features as well as numerical ones. These features are thereby packed into decision boundaries which will be learned during the training process of the model. In the end, each decision boundary refers to a specific feature in the samples, whereby the more important a feature seems to be in identifying whether a sample belongs to a specific class, the sooner it will occur in the DT. Therefore, more important features arise at the top of the tree, while the deeper we dive into the tree structure, the less important the features can be considered. Note that this does not mean that each feature can only be considered once, it can happen that multiple nodes are build upon different boundary decision of the same features - resulting in a more detailed classification.

As a conclusion, we argued that the DT model is very well suited for classification tasks and also rather simple considering its underlying objective functions. One of the main advantages of this model is that humans can very easily interpret the results of it by just looking at the corresponding decision boundaries which were created by the algorithm. Moreover, it is also possible to distinguish between features which are more and less significant to classify samples for certain classes - features with the highest informative content will be at the top of the tree. Nevertheless, also this model comes with some downsides. First of all, it is important to note that it needs some careful parameter tuning. This tuning process is especially important when choosing an appropriate maximal

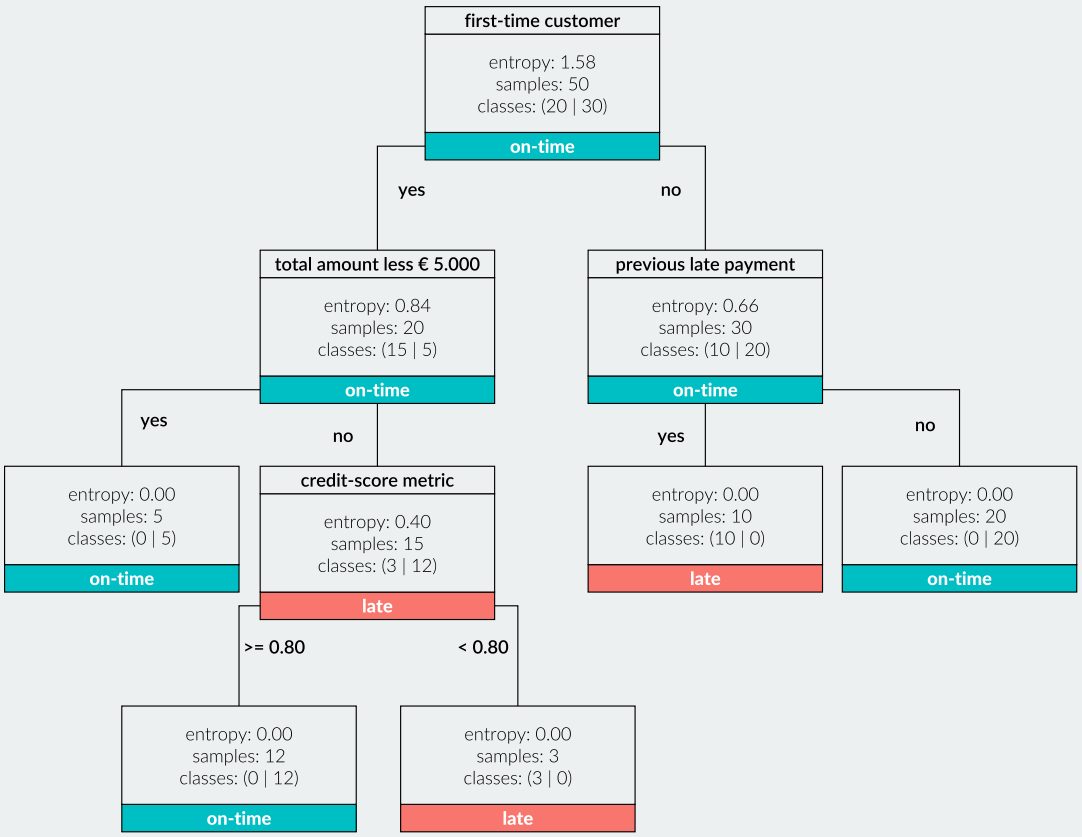


Figure 3.2: **Illustration of a constructed decision tree by the DT model** where we want to classify whether an invoice will be paid on-time or not. From the constructed DT (which gets learning during the model training process) we can derive that the feature which handles whether the invoice reflects a *first-time customer*, is the most important one. Followed by *total invoice amount* considerations and *previous late payments*. Starting at the top root node and sequentially answering boundary conditions (illustrated by branching nodes) allows a manually retracing process and supports the understanding why certain classification decisions were made. Consequently, an evaluation of the learned model can easily be performed by looking at the final DT.

tree depth. A rather high tree depth may thereby quickly lead to an increased gain of complexity when diving deeper into the tree structure which can consequently result in overfitting and loss of interpretability. At the same time, a shallow tree depth may results in poor classification results while not enough boundary conditions may be created. Besides that, the DT model usually suffers from a high variance. Meaning that already some minor changes in the data might lead to invalid classification results. Therefore, it is essential that the training set covers a suitable variance, but at the same time it should not contain outliers as well - an adequate bias-variance-tradeoff needs to be determined.

### 3.1.4 Random Forest

A more advanced classification method, which builds upon the previously mentioned DT, is the so-called Random Forest (RF) model. This method can be seen as a simple ensemble

strategy which combines multiple DT classifiers and therefore inherits the benefits from the good classification results. At the same time, this model aims to overcome the ease of overfitting which may occur when a single DT is not appropriately pruned (more insight into ensemble method strategies will be covered in Section 3.2). Even though this method is more advanced concerning the combination of multiple DT classifiers, it can still be considered as a rather simple approach which it is easy to handle regarding its parameters. At the same time, it is important to note that this model is commonly known to outperform many other supervised learning classifiers (also with rather small datasets) and therefore gains huge attention in literature and practice [Sha+17; Les+15b].

Before classifying samples with a RF model, a particular number $k$, which specifies the amount of ensembled DT classifiers, has to be specified. A trade-off between computational expenses and classification performance needs thereby to be considered. This means that the more DT classifiers are in use, the better the final classification results will be, but also the higher the computational costs. Once this number $k$ is specified, the RF algorithms start by selecting $n$ samples at random (with replacement) from the provided training dataset - these samples further serve as the basis for the training process. Based on this training set, one DT classifier can be constructed as stated in Subsection 3.1.3. However, in the approach with RF not all available features are considered to optimize the objective function, but rather only a specified amount of $m$ features which are randomly selected (without replacement) from the set of the available ones. Consequently, the construction of decision trees will be repeated $k$-times whereby every time a new DT is constructed, also a new subset of features (on which the respective DT split is performed) will be selected at random. Once all DT classifiers have been constructed, the RF classifier is ready to use. To classify a data sample with the trained RF model, a classification process on every DT classifier is performed individually with a subsequent evaluation and report of the individual classification results. Finally, the class which was reported by the majority of the ensembled DT classifiers will be used as the classification result of the respective RF classifier [L+02; Ras15].

Considering the advantages of a RF model, we concluded that it commonly results in an outstanding classification performance and outperforms many alternative classification algorithms. Furthermore, it can also handle noise from unpruned DT classifiers in a robust manner which leads to a major benefit such that the individual DT parameters do not need as much attention [Ras15]. Nevertheless, RF comes also with a central disadvantage, namely the loss of interpretability. Due to the use of multiple DT classifiers, one can no longer represent their full structure in a meaningful manner. Furthermore, the method needs special caution when choosing its parameter values for the number of used DT classifiers (performance vs. computational cost tradeoff). Moreover, the number of used features $m$ and the number of drawn samples $n$ needs also to be chosen with caution due to possible bias-variance tradeoff problems.

### 3.1.5 K-Nearest Neighbors

The K-Nearest Neighbors (KNN) model is another classification strategy which can be pursued when trying to classify samples. Contrary to the other presented methods in this thesis, this method is especially interesting due to the non-existence of a training

procedure. This means that KNN does not need any training in advance to perform a successful classification process. It will instead memorize a set of defined data points and classify samples based on this stored knowledge.

The fundamental principle behind KNN is therefore straightforward, which is also reflected by its parameter settings. In general, we could observe that the algorithm consisted only of two main parameters. The first one is the number of neighbors $k$ which are considered when classifying a sample, and the second parameter is the preferred distance metric used to measure the distance between our samples. Consequently, as these two parameters intuitively already suggest, a new sample will be classified by looking at its $k$ nearest neighbors (determined by the measurement metric), and based on the majority class of these neighbors, the corresponding class will be assigned - see Figure 3.3.

Some common measurement methods to identify a samples $k$-nearest neighbors are thereby, the Euclidean distance or the Manhatten distance. The **Euclidean Distance** is defined as:

$$euclidean\_dist(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{3.9}$$

whereby it represents the length of the unique shortest line connecting the two points $x$ and $y$ in an n-dimensional space. On the other hand, the **Manhatten Distance** is defined as:

$$manhatten\_dist(x, y) = \sum_{i=1}^{n}|x_i - y_i| \tag{3.10}$$

representing the shortest rectilinear distance between two vectors $x$ and $y$.

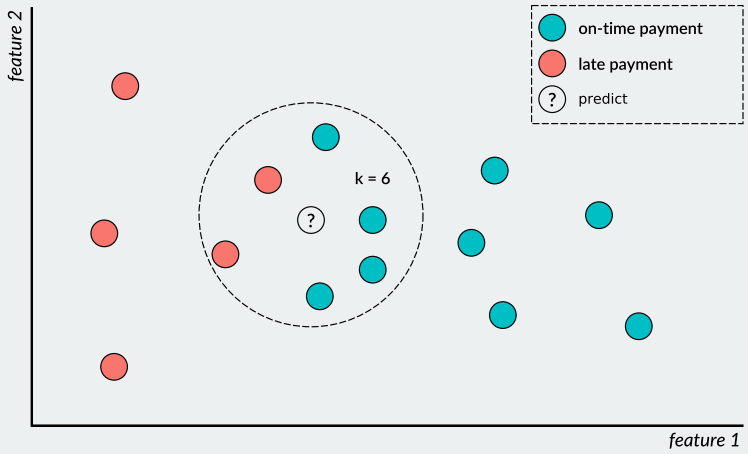


Figure 3.3: **Illustration of a classification with a KNN model** using the Euclidean distance as a measurement metric and $k = 6$ as a parameter setup. Based on these parameters, the algorithm looks for the six closest neighbors around the sample which should be classified (indicated by question mark). As a result, we see that four of these nearest neighbors are included in the "on-time payment" class, and two are within the "late payment" class. Consequently, the prediction results would be "on-time payment" due to the determined majority class.

A significant aspect which we want to underline at this points is that the dataset samples require to be standardized in order to enable a valid and correct distance comparison among them (see Subsection 3.3.1).

For a conclusion, we want to point out again that this algorithm does not need any training in advance which can be very beneficial due to its adaptive behavior. Nevertheless, the more samples we need to consider while computing the corresponding $k$-nearest neighbors, the more complex the classification process gets. This is especially true when we have to consider a high dimensional feature space and consequently need to perform also many distance calculations in this high dimensional space. Although there exist different methods for dimensionality reduction (e.g. Principal Component Analysis), which might solve this problem, another difficulty considering storage problems arises too. The fact that we have to save all the respective samples from our training set implies that especially bigger datasets may lead to significant storage bottlenecks.

### 3.1.6 Support Vector Machine

The last classification algorithm which we discuss in this section is the Support Vector Machine (SVM). Same as the other presented classification methods it is also widely used and known for its good classification performance, but besides that, it can additionally be used for regression problems too. For classification it generally uses a function which maps the given sample features into a higher dimensional space, followed by the task of finding an optimal hyperplane which separates those samples the best. Due to our set focus on classification methods, we will further concentrate this review on the SVM model with non-linear kernels for classification purposes only. To better understand what the goals of SVM are, and how it works, we will support this review by illustrating a small example with two classes (which are linearly separable) in Figure 3.4.

As already mentioned, the SVM classifier tries to find the optimal decision boundary which separates the sample features the best. The objective is thereby to find an optimal hyperplane (decision boundary) which maximizes the margin between the samples which are the closest to the hyperplane itself. These points which are considered to be the closest one are thereby called support vectors - see illustration below. The main idea behind this concept is to find a hyperplane which provides the largest possible margin. A SVM with a respectively bigger margin tends thereby to have a better generalization performance and is less likely to show overfitting problems in contrary to a SVM with a hyperplane with a fewer margin [Ras15]. As we found this optimal hyperplane, a samples class is determined by checked on which side of the hyperplane the sample is located; the sample gets classified based on its position to the hyperplane.

The objective function to find this best hyperplane considers thereby the minimization of the following function:

$$\frac{1}{2}w^T w + C\left(\sum_i^N \xi_i\right) \tag{3.11}$$

whereby $w$ is related to the feature weights, and $N$ to the number of available classes.

Besides that, $\xi$ corresponds to the so-called slack-variable which enables better handling of non-linear separable classes in a relaxed manner. Furthermore, $C$ represents the capacity constant which can be seen as the responsible parameter for controlling the misclassification penalty. It is important to note that this parameter needs to be picked with caution due to possible overfitting problems. As can be seen in the illustration below, a lower value for $C$ allows the SVM classifier to make more misclassification errors while a higher one restricts that behavior by penalizing the SVM [Ras15].

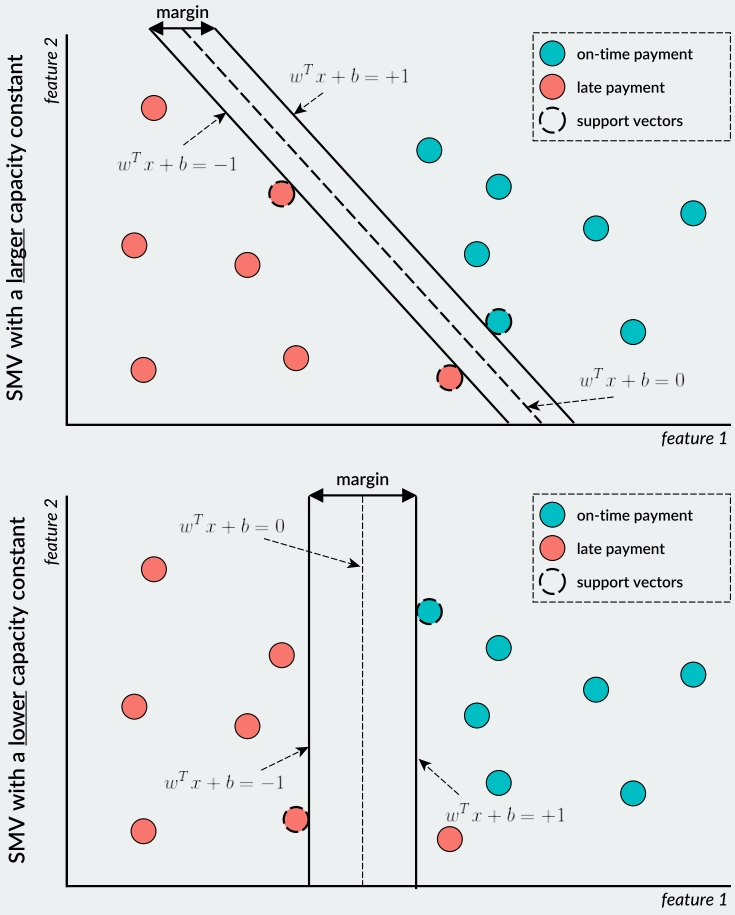


Figure 3.4: **Illustration of a classification with a SVM model** using a high and low value for the respective capacity constant. In both cases, the SVM classifier tried to find the best hyperplane which separated both classes most appropriately while considering to maximize the margin to the support vectors. In the top figure, we were using a high value for the capacity constant, whereby the hyperplane is constructed considering three support vectors, allowing no error. In the figure at the bottom, the same example was plotted with a lower value for the capacity constant. Thereby, only two support vectors were considered while additionally some error in finding the best boundary condition is overlooked, resulting in a overall bigger margin. The hyperplane (decision boundary) was generally defined by the equation $y = w^T x + b$ whereby $w$ and $b$ were parameters of the hyperplane itself. Moreover, the outer border lines which limit the width of the decision boundary were defined in way such that they resolved to +1 and -1.

Last but not least, we briefly review some of the basic *kernel*-functions in SVM classifiers which are used by the model to map feature values into a higher dimensional feature space. This specific concept allows us to find more suitable decision boundaries of non-linear separable samples by exploring regions in higher dimensions. Some of the most common *kernel*-functions are thereby:

$$K(X_i, X_j) = \begin{cases} X_i \cdot X_j & \textbf{\textit{Linear} kernel} \\ \left(\gamma\, X_i \cdot X_j + C\right)^d & \textbf{\textit{Polynomial} kernel} \\ \exp\left(-\gamma \left\| X_i - X_j \right\|^2\right) & \textbf{\textit{Radial Basis Function (RBF)} kernel} \\ \tanh\left(\gamma\, X_i \cdot X_j + C\right) & \textbf{\textit{Sigmoid} kernel} \end{cases}$$

whereby $\gamma$ refers to a parameter for the respective *kernel*-functions which needs to be optimized, and $K(X_i, X_j) = \phi(X_i)^T \cdot \phi(X_j)$ represents the *kernel*-function which is the dot product between two points transformed into the higher dimensional feature space by the function $\phi$ [HTF09].

To summarize the SVM classifiers, we concluded they are well suited for classification problems whereby a particular attention is set on samples next to the decision boundary (support vectors). A benefit of SVM classifiers is that they are capable of handling also very complex relationships among features which is sometimes not possible with other classifiers. Nevertheless, training such a SVM classifier can be rather expensive, especially while considering the respectively used *kernel*-function and computational costs of the minimization in the objective function to maximize the margin. Although the classification results are most of the time very good, the SVM model can be seen as a sort of black-box which can not be interpreted as easily. This refers primarily to the fact that they are not very interpretable in the sense of how the SVM model came to a concluding classification result considering individual samples [HTF09].

## 3.2 Ensemble Learning: Bagging and Boosting

Ensemble learning is a widely spread method in the field of ML which is commonly used to combine the strengths of multiple ML algorithms. In the case of supervised classification, data samples get classified by combining multiple supervised classification algorithms whereby the results get determined by an aggregation process among the individual results. This means, that instead of creating an entirely new classifier, we independently classify a sample on multiple types of classifiers and combine the final result by either smart decision boundary conditions or majority voting evaluations. In the previous subsection, we already reviewed one such ensemble classifier, namely the RF model. As already discussed, this strategy refers to the majority vote classification process where multiple outputs of $k$ different DT classifiers get combined and aggregated to one specific outcome. In this subsection, we will further present two of the most commonly used techniques when it comes to ensemble learning, called Bagging and Boosting.

In general, the application of ensemble learning strategies comes to use once an already robust model of a set of classification algorithms has been developed. This means in particular that first of all one should concentrate on the development of multiple single classification methods. As these models show suitable evaluation performances, a boosting or bagging approach could be considered to push the classification performance towards an optimal model. However, there are of course exceptions, as for example the use of the RF classifier. At this point, it is important to note that even if the RF model can be considered as an ensemble strategy for its own, this does not restrict the further use and combination with additional ensemble strategies like Bagging and Boosting.

### 3.2.1 Bagging: Voting Classifier

In literature, the strategy of Bagging was first introduced by Breiman [Bre96] who proposed it to be a valuable strategy which is able to gain a significant amount of accuracy by combining various classification results. Besides this aggregation process, which generally refers to collecting and combining different algorithm results, the sample bootstrapping process is a key aspect of this ensemble strategy. With the help of bootstrapping it is thereby possible to create multiple training subsets out of only one unique training dataset. It is thereby essential to note that each of these subsets differs in its represented bias-variance tradeoff which counts as a beneficial advantage. During the creation process of such a subset, different sample instances are randomly drawn (with replacement) from one central training set. This new subset is consequently used for the actual training process of a specific classifier. Depending on the number of classifiers which one wants to combine with this ensemble method, this bootstrapping and training process will be repeated $n$-times. In the end, especially this usage of multiple classifiers and different dataset variations enables the strategy to minimizes the bias within the overall model. For a better understanding, we illustrated the fundamental procedure of the described bagging process in Figure 3.5.

As mentioned before, the final prediction process is performed through a voting procedure which generally can be categorized into a *hard-voting* or a *soft-voting* strategy. In the case of *hard-voting*, the simplistic approach of the majority class over all different classifier predictions is considered to be the final result. With *soft-voting*, the respective *argmax* of the sums of predicted probabilities gets calculated, on which basis the corresponding predicted class gets inherited (a prerequisite is thereby the support of a probabilistic classifier output).

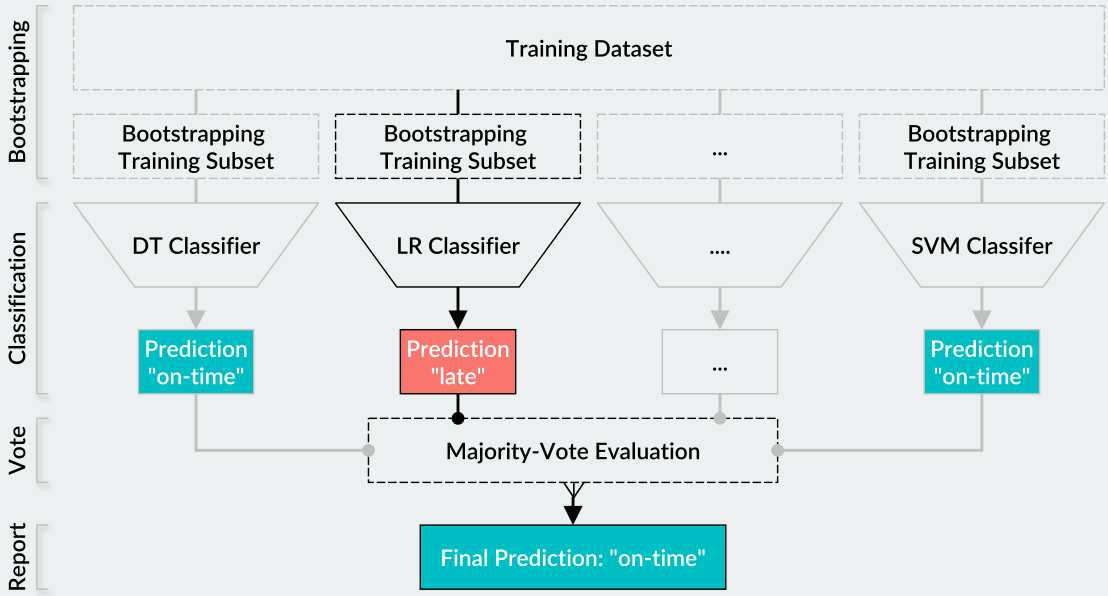


Figure 3.5: **Illustration of a classification with a Voting classifier**. In the training process of a Voting classifier, we start with a set of classifiers (also possible to have multiple classifiers of the same type with different hyperparameter setup) and train those on different training subsets which are determined in a bootstrapping process. Once all classifiers are trained, the classification process of a new sample is performed by classifying it on each classifier independently with a subsequent aggregation of the results. As illustrated above, even though the LR model classified the invoice in our example as "late payment", the majority of the classifiers does not, which is why the final result of the Voting classifiers is "on-time payment" (used hard-voting aggregation).

### 3.2.2 Boosting: AdaBoost

The second ensemble method which is commonly in use is Boosting. Generally, this strategy can be seen as a sort of variation or expansion of the previously presented Bagging strategy. In comparison to Bagging, this method focuses thereby primarily on the bootstrapping process which is responsible for the data subset sampling process. One of the most common implementations of Boosting is Adaptive Boosting (AdaBoost) which was first presented by Schapire and Freund [F+96].

The basic concept behind AdaBoost is to learn sequentially from prediction mistakes (e.g. misclassification errors) and to improve the algorithm performances step by step over time. In more detail, this concept can be summarized as follows. As a first step, the algorithm starts with a subset of randomly drawn samples (without replacement) from a central training set, and classifies them with the help of a defined classification method. Each sample in the training dataset holds thereby a corresponding weight which is uniformly initialized. Once the classification process is completed, the data samples get validated and reweighted. This means that each sample which was classified correctly receives an updated lower weight while those who were classified incorrectly get a higher weight. Next, a new subset of samples is drawn (without replacement) from the training set, but

this time an additional focus is set on those samples which have a higher weight (samples with a higher weight are more likely to get picked for the training subset). Additionally, the classification process will further concentrate on the correct classification of those samples with a higher weight. This procedure of classification and reweighting of samples gets now repeated $n$-times, depending on the specified repetitions/rounds. The strategy implies thereby that samples which got misclassified in a prior classification step, are more likely to get correctly classified in a new processing step - the algorithm improves itself over time. In the end (after $n$ classification and reweighting reruns), the results of each different classification outcome on the differently weighted training subsets get combined by a majority vote process, and the final prediction result gets reported [Ras15].

As a conclusion, we argued that the use of ensemble strategies can be beneficial concerning the increase of performances in different classification methods while combining their strengths and lowering the overall bias of the independent classifiers. However, the computational costs which come with the use of these ensemble strategies need to be considered as well. Especially in practice it is known to be difficult in finding the right trade-off between these computational expenses and the growth of a classifiers performance.

## 3.3 Performance Measures & Evaluation Methods

Evaluating and judging the output quality of prediction or classification tasks is a well-studied area in the field of ML [SL09; Ste+10]. Moreover, also various techniques for correct hyperparameter tuning and sample splitting have been studied and reviewed. This section will consequently review some common statistical concepts, key metrics for model evaluations, and optimization techniques which we used in this thesis to handle the specific tasks of model evaluations, data splitting, and hyperparameter tuning.

### 3.3.1 Relevant Statistical Concepts

Throughout our upcoming feature analyzation and preprocessing steps we made use of several statistic related concepts which we further briefly describe in this subsection to provide a solid groundwork and to clear out possible misunderstandings.

The **Mean** refers to the simple use case of calculating the average value of a finite set of numerical values. It is defined as:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{3.12}$$

whereby $n$ refers to the number of values in a defined set $X$.

The **Median** is a basic concept to determine the value which lies exactly in the middle of an in ascending order sorted value range. This means that the median value exactly separates the whole range of values into an upper and lower half. It is defined as:

$$\widetilde{X} = \begin{cases} x_{\frac{n+1}{2}} & \textbf{if } n \text{ is odd} \\ \left(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}\right)\frac{1}{2} & \textit{otherwise} \end{cases} \tag{3.13}$$

whereby $n$ corresponds again to the number of elements in a defined set $X$. It is however important to note that for certain use cases where the set of values is even, it is not practical to determine the average between the two numbers $x_{\frac{n}{2}}$ and $x_{\frac{n}{2}+1}$ which is why we simply use the later one as the corresponding median value. This is especially important when considering a list of given values where we want to pick the median *index* which corresponds to a specific element in a list.

The **Z-Score Standardization** is a well known concept which refers to the feature normalization process and is generally defined as:

$$z = \frac{x - \mu}{\sigma} \tag{3.14}$$

whereby $\mu$ refers to the mean value and $\sigma$ to the corresponding standard deviation of the mean. This concept can thereby be simply summarized as a strategy to scale all the available values into a common range. Applied on feature scaling, we end up with the common setup of rescaled features with a mean of 1 and a standard deviation of 0. In this thesis, we use however a slightly modified version of this z-score standardization process whereby we replace the mean with the **median** value. This setup comes with the additional advantage of providing a more stable groundwork and considers at the same time potential outliers in the data.

The **Min-Max Scaling** is an alternative strategy for normalizing values and is defined as:

$$\hat{X} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.15}$$

whereby $X_{max}$ refers to the highest value in $X$ and similar $X_{min}$ to the lowest value in $X$. Employment of this strategy maps each value of a given set of values into the range between 0 and 1 whereby the highest value will be mapped to 1 and the lowest one to 0.

The **Correlation Matrix** is a conventional technique to represent the correlation coefficients among different variables (features) and thereby represents a common tool to test and visually inspect the relationship among variables in a statistical context. These feature correlation coefficients are thereby defined in a range from -1 to +1, whereby -1 represents a negative relationship and +1 a positive relationship between different variables. Furthermore, a coefficient which corresponding to a value of 0 means that no relationship between two variables exists. In more detail, a positive correlation implies that if variable $a$ increases, also variable $b$ increases and on the other hand, a negative correlation implies that if variable $a$ increases, variable $b$ decreases. The value of these relationship reflects consequently

the strength of the relationship between two variables. To calculate these coefficient values, the standard Pearson correlation coefficient is commonly in use, which is defined as:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{3.16}$$

whereby $\sigma$ refers to the standard deviation and $cov$ to the covariance between two variables.

### 3.3.2 Confusion Matrix & Key Metrics

An accessible way to evaluate the results of a supervised learning task is commonly based on four measurements, namely: True Positive, True Negative, False Positive, and False Negative. All of those measurements can be seen as a simple collection of observations whereby each predicted sample is compared, counted and reported.

- **True Positive (TN)**: The number of samples which were predicted as a specific class and comparison to related ground-truth class is positive.
- **True Negative (TN)**: The number of samples which were predicted not to belong to a specific class and the comparison to the related ground-truth class is positive.
- **False Positive (FP)**: The number of samples which were predicted as a specific class and comparison to related ground-truth class is negative.
- **False Negative (FN)**: The number of samples which were predicted not to belong to a specific class and the comparison to the related ground-truth class is negative.

As demonstrated above, the comparing process refers to the result when matching a samples ground truth class to its predicted class. As an example, we can look at the classification of invoice payments where we have two classes: ”payment arrived on-time” and ”payment arrived late”. First, each invoice gets label according to whether the invoice was paid on-time or not (by looking a the due date and the actual settled date). Next, all invoices get classified by a specific supervised classification model. Finally, each prediction result can be compared to its previously defined ground-truth, followed by the assignment to one of the four presented measurement groups.

Especially in binary classification tasks, where we have only two classes: positive and negative, the presented measurement groups are a standard procedure to judge the performance of a classifier. A popular way to represent the outcome of these measurements is thereby in the form of a so-called **Confusion Matrix** which is a simple $2\times2$ matrix that summarizes the gathered results as illustrated in Table 3.2.

Note that for multi-class problems with $n$ classes, one can simply extend the confusion matrix to an $n\times n$ matrix. Same as in the simple $2\times2$ confusion matrix, the rows reflect thereby the actual ground-truth and the columns correspond to the predicted classes.

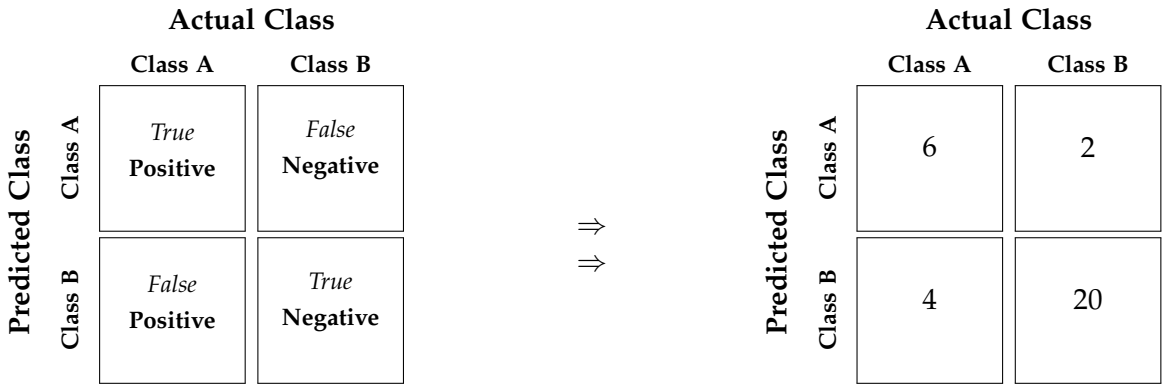


Table 3.2: **Example of a confusion matrix on a binary classification task**. On the left side, the basic structure of a confusion matrix is illustrated, and on the right, the illustrated table is filled with example results from a classifier. In total 26 samples were correctly classified as samples affiliated to class A or respectively class B. The other six samples were incorrectly classified as can be seen in the top right and bottom left corner of the table.

Besides the presented confusion matrix, supplementary key metrics can be derived from the stated measurements to evaluate a supervised learning model. Commonly used metrics in literature and practice are thereby Precision, Recall, Accuracy, and the F1-Score.

The **Precision** indicates the percentage of correctly classified samples among all the samples which were classified as a specific class. Meaning that we divide the correctly classified positive predictions by the total amount of predictions made:

$$Precission = \frac{TP}{TP + FP} \tag{3.17}$$

The **Recall** indicates the percentage of correctly classified samples while considering the total ground-truth of all samples in the corresponding class:

$$Recall = \frac{TP}{TP + FN} \tag{3.18}$$

The **Accuracy** represents the percentage of correctly classified samples. This means that it can be simply calculated by dividing the correctly classified samples by the number of all considered classifications:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.19}$$

The **F$\beta$-Score** can be seen as a weighted average of the precision and the recall measurements. The choice of $\beta$ depends thereby on how much weight one wants to give the precision value over the recall value. Note that $\beta = 1$ weights both measurements equally (known as F1-Score) and simply returns the harmonic mean of precision and recall:

$$F\beta\text{-}Score = (1 + \beta^2)\frac{Precission * Recall}{(\beta^2 * Precission) + Recall} \tag{3.20}$$

$$F1\text{-}Score = \frac{2TP}{2TP + FP + FN} \tag{3.21}$$

As a conclusion, on the presented measurement metrics, we want to point out the importance of selecting a measurement one wants to focus on before starting with an evaluation process. Thereby, one has to admit a trade-off among the presented metrics and further decide thoughtfully on which one wants to use and optimize during the performance evaluation of a model. Changing between measurement optimization strategies e.g. focusing on accuracy and recall optimization in the first place and then later trying to optimize for precision, will result in problems due to the competing focus of these measurements. Furthermore, focusing the attention on one measurement strategy only, as for example accuracy, may also be a critical strategy due to misleading interpretation problems. When we have for example an imbalanced dataset which contains 99 samples of class A and 1 sample of class B, the final accuracy measurement of a naive majority class predictor would be 99% - clearly not a valuable strategy to evaluate the quality of this classifier. Consequently, the choice of suitable measurement strategies is always problem specific and implies skeptical thinking while applying them.

### 3.3.3 Receiver Operating Characteristic

Another popular method to evaluate the performance of a classification model is the use the so-called Receiver Operating Characteristic (ROC). This method is a simple tool to visualize the tradeoff between recall and precision while considering all possible thresholds calibrations which can be set to classify a sample as a positive one (threshold values reach from 0 to 1). This threshold values are thereby responsible for setting the limit from where on a sample is affiliated to a class. As an example, we can think of a classifier which outputs the probability of sample $x$ and two classes $A$ and $B$. When the classifier predicts the probability of sample $x$ to be with 0.70 affiliated with class $A$ (consequently 0.30 to class $B$), a defined threshold of 0.75 would still force the classifier to affiliate the sample $x$ to class $B$. Thus, this parameter is responsible for determining the strength of the decision boundary in each classifier whereby it needs to be accordingly optimized to obtain a suitable classification result.

Note that the described ROC visualization requires thereby a classifier which is able to produce a probabilistic output e.g.: Logistic Regression, Decision Trees, SVM etc.. This requirement results from the way the corresponding output measurements for the ROC (**True Positive Rate** and **False Positive Rate**) are computed:

$$True\ Positive\ Rate = \frac{TP}{TP + FP} = Recall \tag{3.22}$$

$$False\ Positive\ Rate = \frac{FP}{FP + TN} \tag{3.23}$$

Once these two metrics have been measured for all possible thresholds values of one classifier, it is possible to plot the so-called ROC-Curve as illustrated in Figure 3.6. To further assess a classifier's performance, the calculation of the respective Area Under

the Curve (AUC) is a commonly applied measurement metric. This measurement value reaches from 0.00 to 1.00 and represents the percentage of the total amount of the area which lies under the calculated ROC-Curve of a specific classifier. A higher AUC value indicates thereby better classification results than one close to 0.50. This scenario can also be observed in the Figure 3.6 where the optimal ROC-Curve holds an AUC of 1.00 and the random classifier only an AUC of 0.50. Consequently, everything below 0.50 means that we have trained a classifier which performs worst than pure random guessing, which most likely indicates a problem during in the implementation process itself. Therefore, only AUC measurements between 0.50 and 1.00 can be seen as valuable and comparable results which are useful to evaluate a classifier's performance.

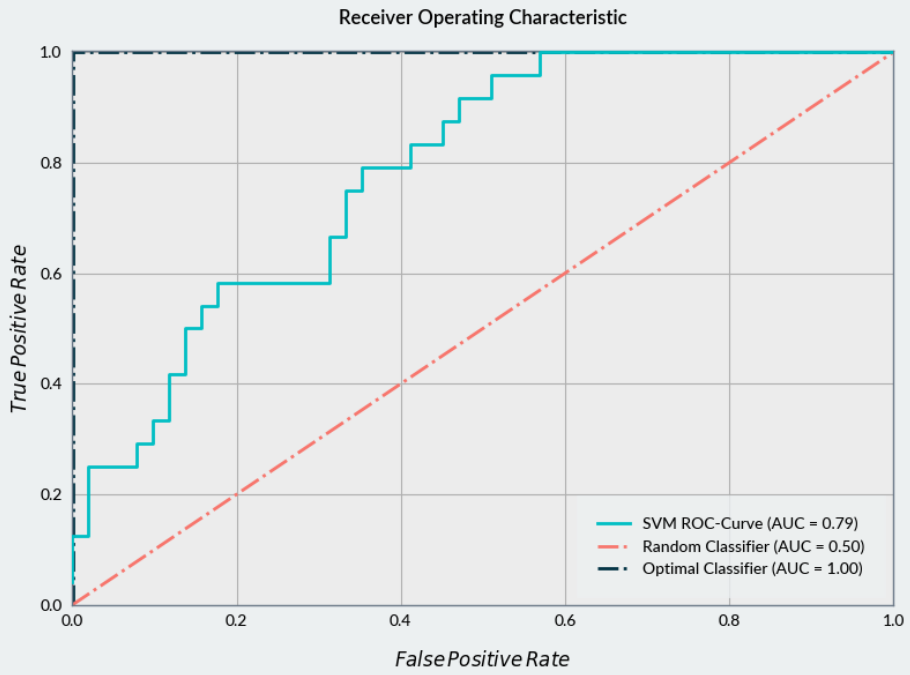


Figure 3.6: **Illustration of a ROC-Curve for a SVM classifier**. The performance of the SVM classifier is thereby plotted for all possible threshold values between 0 and 1 (see green line). Furthermore, the performance of a random classifier is visualized by the red dashed line holding an AUC of 0.50. The most optimal result which one aims to get as close as possible to is visualized by the blue line, holding an AUC of 1.00. Note that the possible performance result of the SVC classifier can be adjusted by setting the threshold value between 0 and 1, whereby the lowest threshold value corresponds to the bottom left of the line and the highest to the top right.

The AUC measurement is thereby commonly used to gauge and compare the performances of multiple models. Nevertheless, as mentioned before, it is important to note that the threshold is a parameter which one has to pick smartly to reach the desired performance in a deployed model. This means that before applying the actual model for a classification task, one has to pick a specific threshold value which corresponds to one specific point on the ROC-Curve. Having for example a suitable model as the SVM classifier (green line in the plot below), one has to adjust the performance of it by tweaking its threshold value which consequently moves the actual model performance along the visualized ROC-Curve.

Generally speaking, decreasing the threshold moves the performance towards the top right corner while increasing it leads to the bottom left. This behavior can be argued by investigating the performances at the threshold extrema of 1.00 and 0.00. A threshold of 1.00 would thereby mean that every sample not reaching a prediction probability of 1.00 is marked as negative prediction, which is why one ends up in the lower left corner of the ROC-Curve. On the other hand, a threshold of 0.00 would predict every sample as a positive one, leading to a prediction performance in the upper right corner.

As a conclusion for the ROC-Curve and the AUC measurement metric, we concluded that it is definitely a valuable approach to judge the overall performance of a classifiers result, while it is most often more accessible than just investigating a single measurement metric on its own. Moreover, it is rather easy to compare multiple model performances while plotting their individual ROC-Curves in the same figure. Nevertheless, when it comes to deploying the model, we still need to carefully choose a suitable threshold value in a smart way which reflects our desired goal. The choice of the threshold is thereby always problem dependent and may heavily influences the actual model outcome.

### 3.3.4 K-Fold Cross-Validation

Having defined evaluation metrics in Subsection 3.3.2 and seen a brief overview of the commonly used ROC-Curve visualization in Subsection 3.3.3, we further focus now on a conventional method for dataset splitting, namely the process of **Cross-Validation** (CV). This method refers mainly to the estimation of the prediction error considering training and testing a model which allows us to properly measure the classification error while splitting the available dataset into independent training, validation and test subsets.

A standard procedure to accurately estimate a ML models performance is generally to split the available dataset into three subsets called training set, validation set, and test set. The main concept behind this idea is thereby to use these three subsets in independent processing steps. First, the model training is performed on the training set, followed by the model validation process which is performed on the validation set. An iterative process of model improvement and validation is consequently repeated until a certain level of desired performance is reached. Once the model has successfully been trained, and a certain level of performance is reached on the validation set, the final testing is performed on the corresponding test set. This test set is thereby used to simulate real-world circumstances and to generalize the model's performance on data which was never seen in the former training or validation process.

This strategy is however not always applicable due to possible dataset size restrictions. Especially in real-world scenarios, the size of used datasets to develop a ML model can sometimes be rather limited whereby a proper splitting into three data subsets is not always possible. To overcome this dilemma, one makes use of $k$-fold CV which further divides the training dataset into $k$ subsets on which the model is alternately trained and validated [HTF09]. This implies that one has to split the original dataset into two subsets only, namely training set and test set. Note that, a certain percentage of the data should always be held back to evaluate the model on a separate final test set (this data

should never be mixed up with validation or training set). An illustration of the $k$-fold CV technique (with $k = 5$) can be found in Figure 3.7.

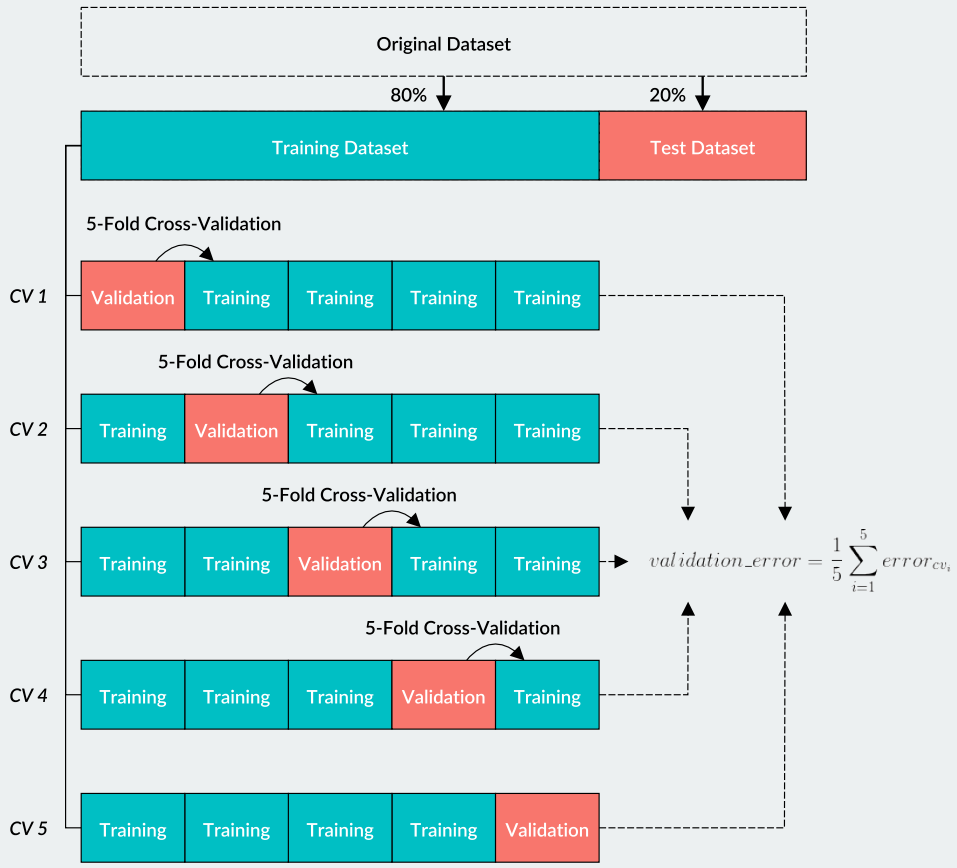


Figure 3.7: **Illustration of a $k$-Fold CV example** with $k = 5$. First, the original dataset is split into two subsets of the size 80% to 20% whereby the later one is held back as the final test set. The remaining 80% of the data (training set) is then further split into five subsets. As can be seen in the illustration, the first subset is firstly used for the model validation purposes and the other $k - 1$ subsets for training the model. By iteratively switching the validation set selection, each of these five subsets will once be used as a validation set while the rest of it serves always for model training purposes. The average validation error of this k-fold CV process is finally computed by aggregating the individual validation results of each model evaluation process.

As already stated, a certain amount of data should always be held back for final testing purposes on unseen data. The rest of the data is considered as the training set and will be split into $k$ subsets whereby it is important to note that all of these subsets hold the same class distribution as the defined training set. Having these $k$ data subsets, the CV process declares one of these sets as the validation set. Now a ML model can be trained on the remaining $k - 1$ subsets and a final validation process will be executed on the prior declared validation set. The corresponding validation error on this subsets gets consequently reported and the CV process continues to declare another subset as the validation set. This procedure gets now repeated for $k$ iterations whereby each of the subsets is declared exactly once as a validation set. The overall validation error can finally

be computed by:

$$Total\ CV\ Error = \frac{1}{k}\sum_{i=1}^{k} error_{cv_i} \tag{3.24}$$

where $k$ corresponds to the number of dataset splits in the training set. In general, one can argue that the size of this parameter $k$ influences the overall outcome of the averaged validation error, which means that a higher or lower $k$ may generally lead to bias or variance problems. Nevertheless, literature commonly recommends setting $k$ around 5 or 10 iterations to achieve a good compromise in consideration of these problems [HTF09].

Another use case is to set $k = N$, whereby $N$ is the number of available data samples (always after the test set has been removed and held back for final testing purposes). This technique is commonly known as **Leave-One-Out Cross-Validation** (LOOCV) and is especially recommended for small datasets. In this case, exactly one sample is always considered as validation set whereby the rest of the dataset is used for training the model. Due to the possible insufficiency of data variance which is reflected by the validation sets when choosing $k$ to be for example 5 or 10, this approach may expose improvements in the evaluation results when working with few data samples. However, this strategy lacks in terms of performance due to the increased amount of needed iterations. Especially when using bigger datasets, LOOCV is not recommended due to the huge performance loss and the already sufficient variance representation of the validation set when setting $k$ to 5 or 10.

For a general conclusion on the CV strategy, we concluded that the choice of $k$ strongly depends on the size of the used dataset and the corresponding task which one wants to solve. Having a rather small dataset, LOOCV is recommended due to the high bias and low variance which can be perceived when working with a small number of dataset splits. However, depending on the pursued goal, this drawback may also be considered as not too important in practice. The overall recommendation is yet to pick $k = 5$ or $k = 10$ when working with not too small datasets to obtain fairly reliable evaluation results considering variance and bias problems.

### 3.3.5 Hyperparameter Tuning and Optimization

In Section 3.1 we presented various supervised classification methods which count to state-of-the-art approaches in the domain of ML. However, all of these classification methods provide various parameters which need to be modified an tuned to obtain optimal results for the objective which wants to be achieved. This section will consequently review some strategies which are commonly in use to support this selection and tuning process. First, we will discuss the term hyperparameter which is used to describe these tunable model parameters and give a short introduction to the tuning process itself. Followed by the explanation of two a widely used technique, called Grid Search and Random search, which automatically report the best hyperparameter setup from a specified set of parameter ranges.

**Hyperparameter** is a commonly used term in the ML domain and refers to model parameters which we need to define before the actual model training process, which

implies that the model can not learn these type of parameters on its own. Consequently, they usually need to be set manually. The tuning process of such hyparameters can thereby be sometimes very challenging, but is especially needed when additional data gathering for model improvements is not possible due time limitations or simple non-existence of more data. As an example for hyperparameters, we can look at a DT classifier which typically uses the *maximal depth* of the tree itself, the number of *minimal samples* required to be at a leaf node, or the *maximal number of considered features*. The tuning process of these hyperparameters can be seen as a major key aspect of the model training process and influences heavily the performance of it. Manually selecting such hyperparameters implies most often deep knowledge of the underlying model behavior and can be rather challenging due to their quantity and the broad range of their acceptable values. To overcome this drawback, various techniques for automatical detection and selection of the most promising hyperparameters have been studied and used in literature and practice.

**Grid Search** is one of such techniques to automate the process of finding the most suitable hyperparameters. Instead of manually persuing a trial-and-error approach in executing a model *k*-times with model training, evaluation reporting and hyperparameter tuning to finally end up with an applicable parameter setup, Grid Search uses a pre-defined set of parameters (parameter-grid) and automatically reports the best combination to use. Nevertheless, this method is rather time-consuming, and we have to have some knowledge about a narrowed down parameter set which we need to present to the method (see Table 3.3).

Grid Search will consequently pick up all the defined parameters and constructs all possible hyperparameter combination with which the model is then trained and evaluated. Finally, the hyperparameter combination with the best measurement metric (minimal error) will be reported. It is important to note that even if this approach is rather simple and pleasant in its application, it can be computationally cost expensive especially when the defined value ranges for each hyperparameter are not narrowed down properly. Nevertheless, this strategy provides good results and minimizes the challenging task of manual selection and evaluation.

| Hyperparameter | Tuning Description | Range | # Parameters |
|---|---|---|---|
| **max_depth** | maximal depth of the decision tree | [1-20] | 20 |
| **max_features** | number of considered features when splitting | [2-14] | 13 |
| **min_samples_leaf** | minimal number of samples required to split | [1-20] | 20 |

$\Rightarrow$ ***20*13*20 = 5.200 possible hyperparameter combinations***

Table 3.3: **Grid-Search example on a DT classifier**. For illustration purposes, we choose to pick three hyperparameters with a defined value range for each one of them (parameter-grid-space). Grid Search will consequently train and evaluate the DT model with each possible combination (5.200) and compares the results for each the specified metric e.g. accuracy, recall, F1-score etc.. As a final result, the hyperparameter setup with the best evaluation metric will be reported. Note that the application of Grid-Search can be rather time consuming.

Another favorite method to optimize hyperparameters is **Random Search**. The basic concept behind this strategy is the same as in Grid Search, but when it comes to iterating through the parameter-grid-space, not all specified hyperparameter combinations are

evaluated but rather only a fixed number of them is picked at random as representatives. Note that a specific range of values must be still manually pre-defined for each hyperparameter. Even though this approach might at first glance seem less accurate in finding a good parameter setup, a recent study by Bergstra and Bengio [BB12] demonstrated that it outperforms Grid Search regarding its efficiency while achieving approximately the same evaluation results. This is especially true when we have many hyperparameters with many possible values to evaluate our model on. The reason for its accuracy lies thereby in the probabilistic random picking procedure while assuming that the pre-defined parameter-grid covers a decent amount of nearly optimal parameter values.

Last but not least, an interesting aspect is the combination of Random Search with a subsequential application of Grid Search, that might result in an even more substantial approach regarding an optimal hyperparameter setup in a computationally efficient time. To do this, we have to define in a first step a parameter-grid-space which covers a broader value range for each hyperparameter. Followed by applying Random Search and using its output (which should be computed rather efficiently) to create a new, more detailed, parameter-grid-space around it. Finally, applying the Grid Search strategy should result in the most optimal parameter setup whereby a considerable amount of time-saving can be assumed.

As a conclusion, we argued that the use of strategies like Grid Search, Random Search or a possible combination of both is a very well suited strategy to overcome the vast workload which comes along when manually approaching a trial-and-error strategy on hyperparameter tuning. Nevertheless, a reasonable amount of background knowledge of the underlying prediction and classification models is always needed to define the overall parameter-grid-space most efficiently.

## 3.4 Sampling Methods

Using supervised ML classifiers faces most often the rather challenging problem of data imbalance. Especially in real-world classification scenarios, this problem is commonly known and refers to situations where most of the available data represents only one majority class while a rather small amount represents one or more minority classes. If the classification process is executed on such an imbalanced dataset, the results could be easily biased by this skewed data [K+06].

In general, literature and practice differentiate between oversampling and undersampling to overcome such problems. In simple words, oversampling refers to adding samples to the minority class whereby undersampling refers to removing samples from the majority class. However, both techniques aim to balance the datasets such that an approximately even distribution amount all classes is reached. In terms of binary classification with two defined classes, the goal is thereby to get a common class distribution of about 50%. Several studies addressed the positive and negative sides of under- and oversampling techniques [Bar+04; Yap+14], as well as different combinations of both [Ram+12; SKV09].

In this section, we investigate several common sampling strategies which were used to over-

come the problem of class imbalance, namely: Random Over-Sampling (ROS), Synthetic Minority Over-Sampling Technique (SMOTE), and a hybrid version of SMOTE and Edited Nearest Neighbor (SMOTE + ENN). It is important to note that due the limited size of the used datasets in this thesis, no undersampling approach would have been useful to apply in the sense of proper invoice payment classifications. Consequently, only oversampling strategies were considered to handle the related class imbalance problems in this thesis.

### 3.4.1 Random Over-Sampling

The most naive and simplest approach to perform oversampling is the Random Over-Sampling (ROS) method. As the name suggests, it randomly picks samples (with replacement) from the minority class and duplicates them. This sample and duplication process will then be repeated until all classes are approximately equally distributed, as we illustrated in Figure 3.8.

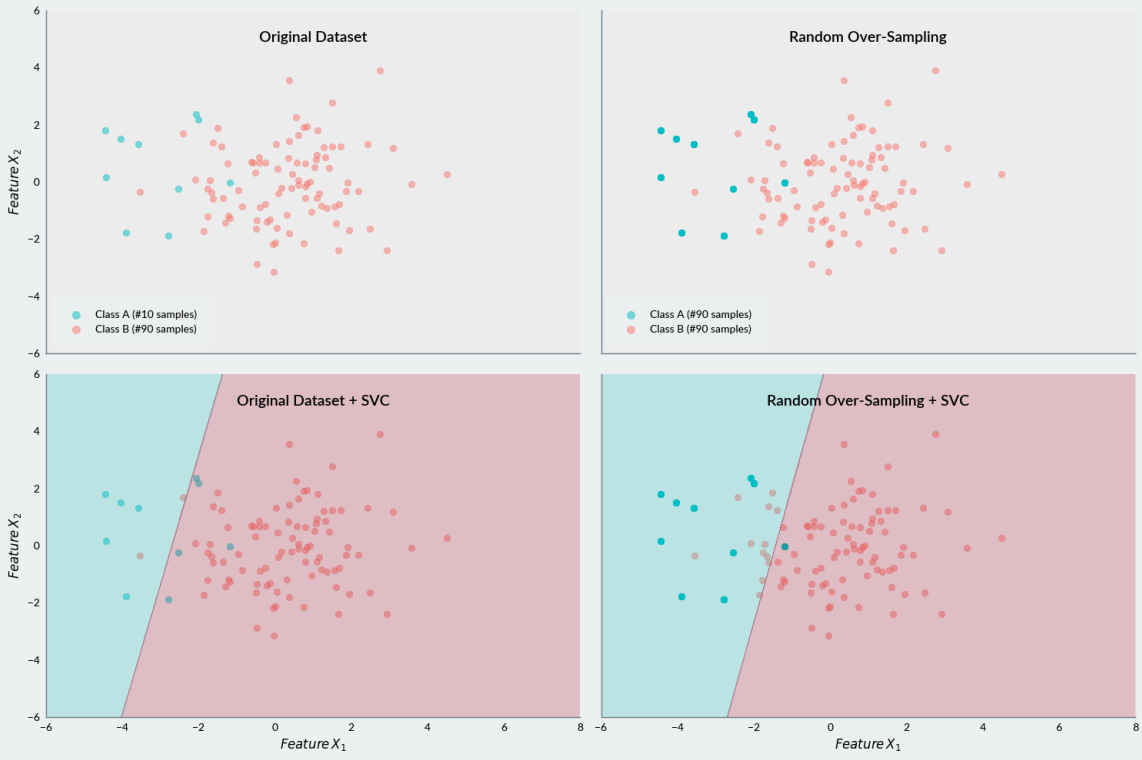


Figure 3.8: **ROS applied on an imbalanced class problem**. For illustration we scattered an imbalanced dataset with a ratio of *10% (Class A)* to *90% (Class B)* on the left-hand side of the figure. After applying the ROS method, an equal distribution among both classes was achieved, whereby corresponding sample feature values were simply duplicated (see right-hand side of the figure). To demonstrate the separability before and after the sampling method, a Support Vector Classifier (SVC) was used.

Even though this approach is relatively simple and the positive side-effect of no information loss comes to our advantage, ROS is known for its drawback of an increased likelihood in overfitting when trying to subsequently apply ML algorithms. This effect arises due to

the exact reproduction of the minority class samples whereby no modification within the features of the sample has been applied.

### 3.4.2 Synthetic Minority Over-Sampling

A more advanced approach to overcome the imbalance class problem is the Synthetic Minority Over-Sampling Technique (SMOTE) proposed by Chawla et al. [Cha+02]. As we demonstrated in Figure 3.9, SMOTE is thereby focusing on the creation of additional synthetic samples within the minority class instead of just creating exact replicas as in the rather naive approach of ROS.

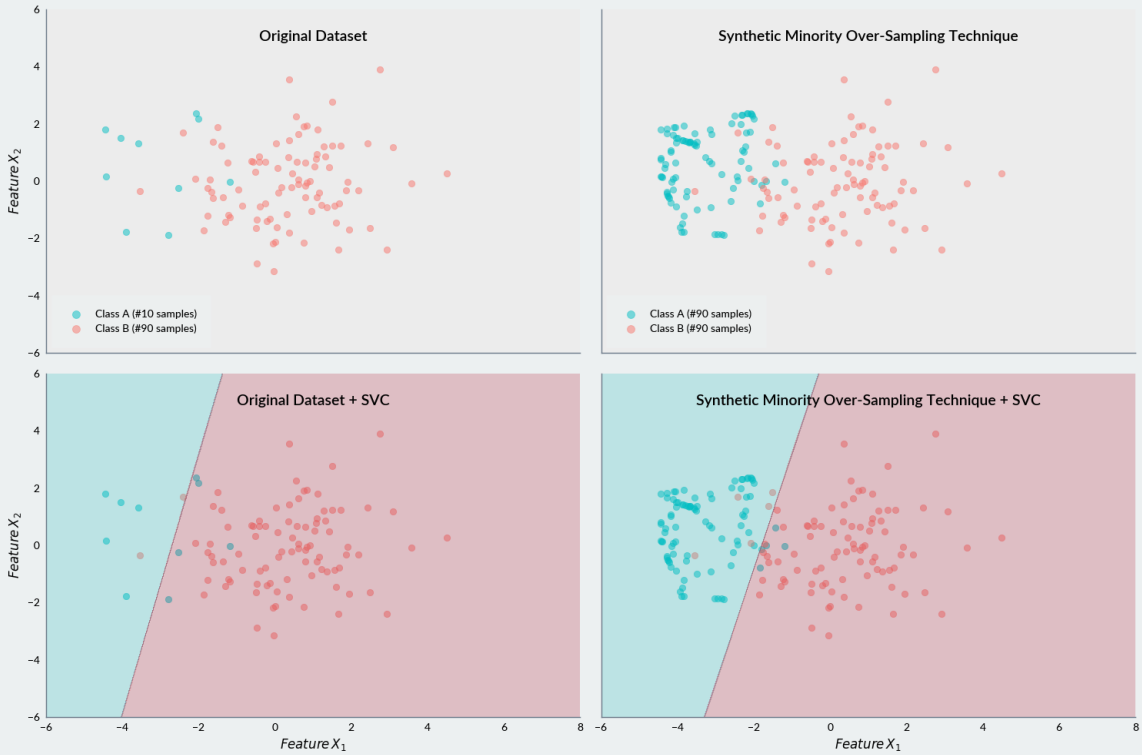


Figure 3.9: **SMOTE applied on an imbalanced class problem**. On the left-hand side we scattered an imbalanced dataset with a ratio of *10% (Class A)* to *90% (Class B)* to demonstrate the imbalanced class problem. On the right-hand side we reported the result after applying SMOTE. Besides the balanced class distribution, we could observe that synthetic samples in the minority class (Class A) were successfully created. A Support Vector Classifier (SVC) was again used to demonstrate the separability of both classes.

As mention before, this process is able to overcome the problem of overfitting which is a resulting drawback of the ROS method. To create such synthetic samples, SMOTE uses a $k$-nearest neighbors interpolation within the minority class. In more detail, this sample generation process works as we illustrated in Figure 3.10. First, a sample $x_i$, including its $k$-nearest neighbors get selected. Next, one sample of these $k$ neighbors $x_j$ gets picked at random. Finally, a new sample $x_{new}$ will be generated by interpolating between $x_i$ and $x_j$, as:

$$x_{new} = x_i + \lambda(x_j - x_i) \tag{3.25}$$

whereby $\lambda$ corresponds to a random number between 0 and 1 [Cha+02].

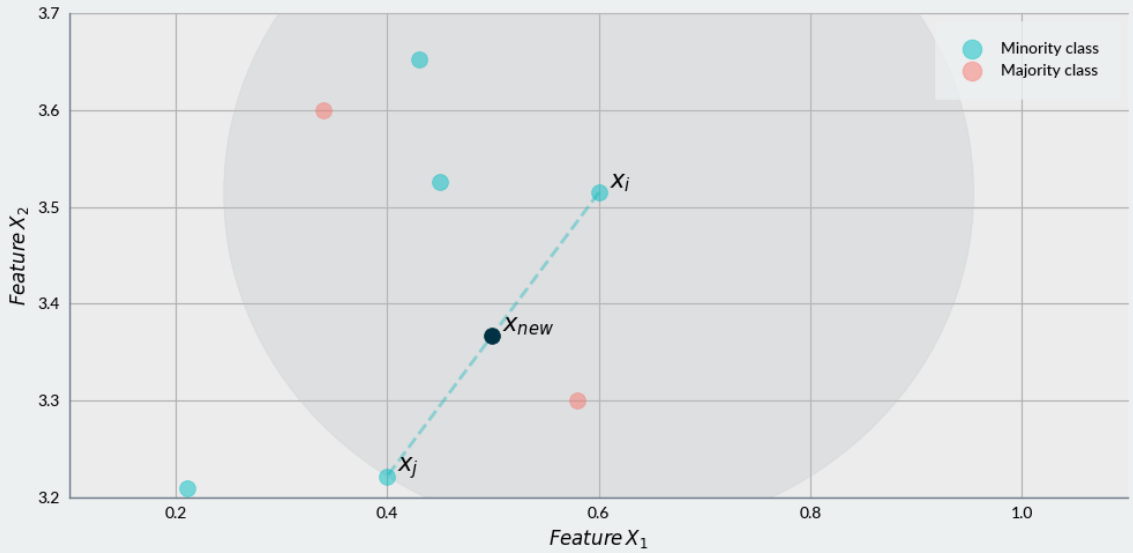


Figure 3.10: **Synthetic sample generation process in SMOTE**. New samples get generated by interpolating between $x_i$ and $x_j$. Whereby $x_j$ gets randomly selected out of the $k$-nearest neighbors of the sample $x_i$ in the minority class. The method will generate as many samples as needed to reach a desired (balanced) class distribution. Illustration was adapted from the *scikit-learn* examples website [Lem16].

Aside from regular SMOTE, there exist also different variations considering the synthetic sample generation and data cleaning process. Common hybrid methods in literature and practice are thereby: SMOTE with Tomek links [Tom76] and SMOTE with Edited Nearest Neighbor (SMOTE + ENN) [Wil72]. The motivation behind those extensions is to apply undersampling so that potential outliers get removed after the oversampling process to gain a better cluster separation and to prevent possible overfitting problems. Let us assume for example that we have two classes which are not well separable. With regular SMOTE, it could now happen that synthetic samples from the minority class expand too deeply into the majority class space. This would consequently lead to a poor separability and potential overfitting in the data. By applying under-sampling, such potential outliers and noise can subsequently be removed to get a more robust behavior [Ram+12; BPM04]. In general, SMOTE with Tomek links and SMOTE with ENN are rather similar and mainly differ only in the calculation process whether a sample should be removed or not. Nevertheless, the Tomek link method is generally known to detect rather few samples as outliers compared to ENN. The SMOTE - Tomek link method removes samples which are identified/connected as so-called "Tomek links", which is exactly the case when for two different samples $x$ and $y$ (from two different classes) no other sample $z$ exists such that:

$$dist(x,y) < dist(x,z) \text{ or } dist(x,y) < dist(y,z) \tag{3.26}$$

whereby $dist()$ corresponds to the distance between two samples [Tom76].

On the other side, ENN applies a nearest-neighbor strategy which removes samples if its nearest neighbors do not fulfill a defined criterion e.g.: remove sample $x$ if its three nearest-neighbors do not include at least one common class. An illustration of how ENN is applied on the result of SMOTE with a comparison to regular SMOTE can be found in Figure 3.11.

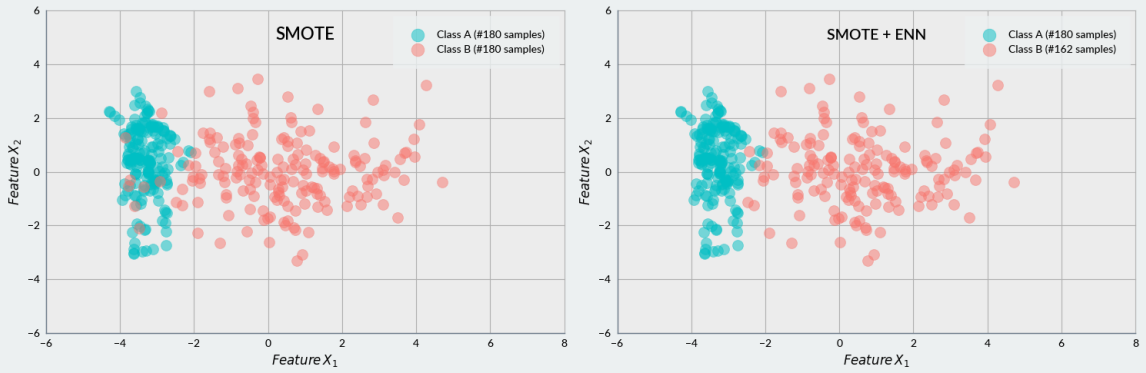


Figure 3.11: **Oversample comparison of SMOTE and SMOTE + ENN**. On the left-hand side we reported an imbalanced dataset which we oversampled by applying SMOTE. It resulted in a balanced class distribution of 50%, whereby some outliers from the majority class were appearing in the expanded minority class (noise in green sample cloud). On the right-hand side we additionally applied ENN on these results of SMOTE to undersample these produced outcome. This cleaning process lead thereby to a less noisy oversample result which may consequently boost the performance of further clustering and ML techniques.

It is important to note that neither ENN nor Tomek link restricts itself to the minority class only. The methods can be used to identify outliers and noise either in the minority class, majority class, or the total dataset [BPM04]. Furthermore, it is also relevant to state that by removing such identified samples, SMOTE can no longer guarantee to succeed in balancing the classes (see right side Figure 3.11). Nevertheless, the overall performance may still be improved due to a better overall data representation.

# Chapter 4

# Experimental Setup

In this chapter, we will focus on the conducted experiments and evaluation processes which we followed to find a possible solution for the stated problems in this thesis. Thereby we will describe the basic setup of these experiments and provide insightful information about each step in the development and evaluation process. Starting with the section *Datasets and Terminology* which covers the most important characteristics and background information about the gathered invoice datasets. Followed by the section *Data Analysis and Processing*, which handles the attempted data analyzations and processing steps, namely: data cleaning and initial data review, feature engineering and selection, in-depth data analysis, and data preprocessing. Finally, the section *Model Fitting, Selection, and Optimization* will cover the ML techniques which we used to build supervised learning classifiers including evaluation strategies and model optimization methods.

For all conducted experiments and evaluations in this thesis we used the `Python 3.6` environment with the following library setup:

- `graphviz v0.8.2`
- `imbalanced-learn v0.3.3`
- `matplotlib v2.2.2`
- `numpy v1.14.2`
- `pandas v0.22.0`
- `scikit-learn v0.19.1`
- `scipy v1.0.1`
- `seaborn v0.8.1`
- `workalendar v2.4.0`

whereby `scikit-learn` was the main library to construct the different ML models.

## 4.1 Datasets and Terminology

*"An analysis can only be as good as the data on which it builds upon."*

While considering this well-known statement in Data Science, we discovered that getting an authorized access to datasets which hold information about privacy critical business documents with features regarding invoices or credit notes is an already difficult task for itself. This section provides some background information on the used datasets and gives an insight into the basic structure while aiming to ease the understanding of the underlying

data on which this thesis is built upon. Additionally, this section will provide an overview of the data terminology and the given feature contexts concerning the provided data.

Most of the public available datasets which aim to provide invoice information are either covering governmental spendings (due to government disclosure regulations) or outgoing purchase orders from companies to third parties. However, throughout most of these datasets, we noticed a major absence of payment receipt information or a contrariwise point of view (seller vs. buyer perspective). Meaning that these datasets could not provide any value to our problems due to the missing time information of elapsed days between invoice issue date and actual payment, or due to the misleading perspective of incoming instead of outgoing invoices. Moreover, it is important to note that datasets from non-store online retailers are most likely not able to provide any useful information either. The reason for that is the immediate payment behavior of customers, which implies that no late-payment is possible (e.g. payment by credit card). Nevertheless, we were able to find two suitable datasets for our problems which considered the set limitations in this thesis. The first dataset which we found was provided by *EmcienScan*[1] and held a list of suitable invoices, including all the needed features for a proper payment classification - this dataset is further referred to as *Dataset A*. The original dataset from *EmcienScan* can be freely downloaded as a comma-separated values (CSV) file from the companies support website[2] (Accounts Receivable). Besides that particular dataset from *EmcienScan*, we got a granted access to a real-world Enterprise-Resource-Planning (ERP) system which provided us a database with outgoing invoices of two years by a small company based in Italy - this dataset is further referred to as *Dataset B*. Unfortunately, the database from the ERP system can not be published due to related privacy concerns.

| Invoice | Country | Customer | Issue | Due | Total | Settled |
|---|---|---|---|---|---|---|
| **00001** | 391 | 0000-0001 | 2012-01-03 | 2012-02-02 | 55.37 | 2012-02-16 |
| **00002** | 770 | 0000-0002 | 2012-01-03 | 2012-02-02 | 50.39 | 2012-01-23 |
| **00003** | 406 | 0000-0003 | 2012-01-03 | 2012-02-02 | 71.33 | 2012-01-30 |
| **00004** | 391 | 0000-0004 | 2012-01-03 | 2012-02-02 | 97.60 | 2012-02-25 |
| **00005** | 770 | 0000-0005 | 2012-01-03 | 2012-02-02 | 15.99 | 2012-02-15 |

Table 4.1: **Representative samples from the used datasets** which were employed to predict the invoice payment outcome of on-time and late payments. Note that both datasets included the same basic features.

To compensate the lack of availability and to get a better understanding of the underlying data, we provided an insight into the basic structure and characteristics of both datasets in Table 4.1. It is worth to note that both datasets did not include any information about the purchased goods or services, neither were any detailed background information about the respective customers included. Six features consequently composed the basic information of each provided invoice: three invoice related timestamps, two customer related fields and the total amount of the corresponding invoice.

[1] https://emcien.com
[2] https://support.emcien.com/help/sample-data-sets

In more detail these features were:

- ***Country***: representing the country to which the invoice was issued to. Assuming that this represents the main customer location. Partly no country assignment by name was possible due to data anonymizations in *Dataset A*.
- ***Customer***: representing the customer who purchased a specific service or goods. These values were respectively anonymized by replacing names with integer values, if not already provided in that format.
- ***Issue***: referring to the issue date of the invoice. Assuming that this was the date when the invoice was sent to the customer (either electronically or by postal delivery).
- ***Due***: referring to the due date of the invoice, which corresponds to the last day on which the retailer was still considering the invoice payment as on-time.
- ***Total***: representing the total amount (including vat) which was charged by the retailer for the provided service or goods.
- ***Settled***: refers to the payment receipt day when the total invoice amount was settled/paid - payment receipt on the bank account.

Further background information on a generalized invoice workflow can be found in Section 1.1. The next section will continue to provide additional insight into the feature distributions and the underlying feature engineering process on both presented datasets.

## 4.2 Data Analysis and Processing

### 4.2.1 Data Cleaning and Initial Review

An initial step which is always necessary when working with real-world data, is a data cleaning and filtering process to overcome inconsistency and bias-variance problems within the datasets. Our two datasets with which we worked in this thesis recorded thereby a very different starting situation. On the one hand, *Dataset A* was an already prepared and cleaned dataset which was intentionally constructed by *EmcienScan* for classification and prediction tasks, which did therefore not require any special cleaning or filtering. On the other hand, the real-world *Dataset B* from the Italian companies ERP system showed some common obstacles which still needed some fine-tuning. By a manual inspections process of the samples in *Dataset B*, we observed that it notably lacked concerning data cleansing and some thesis limitation contradictions. That means besides invoices from one-time customers and invoice which did not have any settlement date or due date yet, we also discovered some outliers and noise within the dataset which needed to be cleaned out before starting a more detailed data analyzation process.

**Dataset A** held out of the box 2.466 individual invoices from almost two years (January 2012 - December 2013) whereby each invoice held initially 12 features. Due to inconsistency among the presence of the features in both datasets, we had to drop six of them to end up with the same features in both datasets. Consequently, we dropped the features: *paperless date*, *invoice number*, *disputed*, *paperless bill*, *days to settle*, and *days late*. Thereby we want to note that features like *days to settle* or *days late* got reconstructed

in a later preprocessing step. Besides these interventions, no further steps were necessary to prepare the data for the initial analysis process. As an additional side note, we want to mention that the dataset did not contain any invoices from one-time customers.

**Dataset B** held a set of 531 individual invoices which were extracted from the Italians company ERP system and covered almost three years of issued bills (March 2015 - February 2018). By manually looking at the data, we discovered that some of the represented invoices were most likely still involved in an ongoing order handling process due to missing information like customer location, settlement date, or due date. Furthermore, we discovered that the feature which represented the main customer location showed some distortions regarding an inconsistency of the country names e.g. spelling or style mistakes. Consequently, we decided to drop all the invoices which were likely to be involved in an ongoing invoice creation process (containing *NaN* values in the due date or settlement date) and fixed the country name inconsistencies by replacing spelling or style difference to a common standard. Moreover, it is important to note that we further dropped invoices which were either only partially or not at all paid by the customers (due to possible insolvency problems). Besides that, we also removed invoices which were issued to one-time customers, since it referred to a contradiction to our set limitations in this thesis. Last but not least, we manually checked for noise and outliers which represented purchases way over the usual standard behavior of established customers. To do this, we filtered out invoices which showed either a purchasing behavior of a very high or very low total invoice amount. Finally, we ended up with a cleaned dataset holding the same characteristics as *Dataset A* which ultimately contained a total amount of 361 individual invoices.

Once the datasets had been cleaned and filtered, such that a reasonable amount of data variance could be assumed, the next step was to perform an initial data review which aimed to reveal the underlying data structure. Consequently, we continued to analyze the related feature frequency distributions as well as the overall invoice payment behavior distribution of on-time and late payments. Note that we performed these analyzations again on both datasets individually.

In our thesis we generally distinguished between *categorical* and *numerical* features whereby we considered the related timestamp values (which are commonly ordinal cyclic features) as part of the *categorical* features. Regarding the evaluations of the categorical features, we evaluated the statistical measurements of the maximal and minimal frequency counts as well as the unique quantity of feature values. For the numerical features, we reported the mean and the corresponding minimal and maximal values. We presented the results of these evaluations in Table 4.2 for *Dataset A* and in Table 4.3 for *Dataset B*.

Based on these measurements, we were able to gain a further understanding of the used invoices and their related feature characteristics. The results allowed us thereby to see for example how many unique customers the datasets covered or what the average amount of each invoice in the dataset corresponded to. As we looked for example on the results from *Dataset A*, we saw that each of the 100 unique customers held a minimal number of 15 invoices whereby a normal distribution considering the invoice amounts seemed to

be likely. On the other hand, the results of *Dataset B* showed that the minimal number of invoices which were guaranteed for each customer was only two. Furthermore, we observed that the invoice amounts of both datasets differed very strongly, whereby *Dataset B* covered rather high invoice amounts compared to *Dataset A*. Moreover, these statistical measurements shed a light on the frequency of issued invoices in different countries, and the related frequency of the companies invoice issuing behavior.

| | categorical | | | | | numerical | |
|---|---|---|---|---|---|---|---|
| **stats** | **country** | **customer** | **issue** | **due** | **settled** | **stats** | **total** |
| **count** | 2.466 | 2.466 | 2.466 | 2.466 | 2,466 | **count** | 2.466 |
| **unique** | 5 | 100 | 681 | 681 | 695 | **mean** | €59,89 |
| **freq_min** | 387 | 15 | 1 | 1 | 1 | **min** | €5,26 |
| **freq_max** | 616 | 36 | 10 | 10 | 10 | **max** | €128,28 |

Table 4.2: **Statistical analysis results on *Dataset A*** which show the basic characteristics of each feature in the dataset. To facilitate the representation, we separated our categorical and numerical features into two tables.

| | categorical | | | | | numerical | |
|---|---|---|---|---|---|---|---|
| **stats** | **country** | **customer** | **issue** | **due** | **settled** | **stats** | **total** |
| **count** | 361 | 361 | 361 | 361 | 361 | **count** | 361 |
| **unique** | 5 | 125 | 196 | 196 | 235 | **mean** | €25.348,61 |
| **freq_min** | 5 | 2 | 1 | 1 | 1 | **min** | €110,50 |
| **freq_max** | 338 | 15 | 7 | 7 | 7 | **max** | €610.000,00 |

Table 4.3: **Statistical analysis results on *Dataset B*** which show the basic characteristics of each feature in the dataset. To facilitate the representation, we separated our categorical and numerical features into two tables.

Next, we proceeded by analyzing the distribution of invoices which were considered as on-time payments versus invoices which were handled as late payments. To do this, we reported the elapsed days between the issue date and the settlement date for each of the invoices. If this number was less than the established payment terms, the invoice was considered as an on-time payment, while when this number exceeded these terms, the invoice was seen as a late payment. In *Dataset A*, the payment terms were given by looking at the time difference between the invoice issue date and the corresponding due date; which resulted in payment terms of 30 days. Unfortunately, in *Dataset B* we did not have the actual due date of each invoice but rather only the background information that the payment terms which were stated on each invoice were "Payment upon receipt". As already briefly discussed in Section 1.1, this type of payment term may be problematic due to often rather "freely" interpreted conditions from the customer's side [Lim15; Inc16]. Due to this lack of information, we decided to approximate the payment terms in *Dataset B* by calculating the average days it commonly took a customer to pay an invoice. The results from this experiment corresponded thereby to 19 days. Next, we continued to calculate the same measurement for *Dataset A* which resulted in 28 days. As a final revealment, we concluded that the actually considered payment terms for the *Dataset B* were thereby 21 days (3 weeks), whereby we compared the calculated number of 19 days with the calculated baseline of *Dataset A* (28 days) and its corresponding payment terms of 30 days. It is important to note that changes in these payment terms might completely distort the

presented results in this thesis. We illustrated a final summary of the on-time and late payments distributions in Figure 4.1. Note that in both datasets the class distribution was somewhat similar which reinforces a correct assumption about the calculated payment terms of 21 days in *Dataset B*.


Figure 4.1: **Class distributions in the used datasets**, showing that both datasets held fairly the same distribution of on-time and late payments with a ratio of 70% to 30%.

### 4.2.2 Feature Engineering and Selection

A good dataset basis is only half the story when it comes to the development process of a reliable ML classifier. Feature engineering and feature selection is generally an iterative process and counts to one of the most crucial steps when developing a proper ML classifier. In the previous section, we gained a better understanding of the used datasets and its characteristics, as well as an overview of the underlying class distributions. In this section, we review how we used this knowledge for feature engineering and thus present the constructed and selected features for our model fitting process.

Due to our set goal of predicting whether a customer will pay an invoice on-time or not, we constructed our features on an invoice basis along with the historical customer payment behaviors. That means within our feature matrix, each row represented exactly one invoice whereby the customer historical payment behaviors were considered as features of each invoice. In total, we worked thereby with 12 features which we extracted from our available invoices - consisting of 10 numerical features, 1 categorical feature, and 1 target feature. The target feature stated thereby the ground-truth class of each invoice, representing if it was an on-time or a late payment. We presented a complete list with short accompanying description of the engineered features in Table 4.4.

| Feature | Description | Type |
|---|---|---|
| **country** | main customer location | categorical |
| **total** | invoice total amount (including vat) | numerical |
| **non_business_days** | number of closed business days within invoice lead time | numerical |
| **prior_total** | number of prior paid invoices (late payments + on-time payments) | numerical |
| **prior_total_late** | number of prior late paid invoices | numerical |
| **prior_paid** | total amount of prior paid invoices (late payments + on-time payments) | numerical |
| **prior_paid_late** | total amount of prior late paid invoices | numerical |
| **ratio_prior_paid_late** | ratio of prior late paid invoices | numerical |
| **recent_paid_ontime_days** | elapsed days from last on-time payment | numerical |
| **recent_paid_late_days** | elapsed days from last late payment | numerical |
| **avg_days_late** | average days of payment delay over all passed payments | numerical |
| **payment_receipt** | ground-truth payment label: ”on-time” or ”late” | **target** |

Table 4.4: **List of used features for the payment classification process**. Composed of 12 engineered features: 10 numerical features, 1 categorical feature, and 1 ground-truth target. Each feature refers to the past payment behavior of the respectively consider customer.

To extract our set of features for each invoice, we had to iterate through all the available customers while considering each time the corresponding subset of related invoices. That means we had to pick each of the customers, including his or her related list of invoices and consequently extracted the past customer payment behaviors for each of those invoices. Some examples are thereby the number of previous late payments or the recent elapsed days from the last on-time payment. This process further implied that we needed to consider the time-related aspect of each invoice which required a previous sorting by their issue date. In the end, each invoice of our datasets corresponded to one row in our feature matrix whereby the stated features corresponded to the columns of this matrix.

The overall goal of our feature engineering process was to keep the feature dimensions as low as possible while constructing only those features which were most valuable for the invoice classification task. During the feature engineering process, we encountered thereby some challenging difficulties considering the equivalency among features and the iterative feature improving process. For example, as we investigated the time series of when invoices get issued and when the corresponding due date was set, we tried to find an essential feature in the monthly or weekly aspect which could possibly help in the later classification process. Our assumption was thereby that invoices whose due dates were set on a Saturdays or Sundays, get less likely paid on-time. However, this assumption was generally wrong as we observed from corresponding time series analyzations. However, we fine-tuned this assumption and finally came up with the feature *non_business_days* which covers the number of state holidays and weekends altogether. This feature is thereby reflecting the number of days a company has closed its business concerning the invoice lead time. As it turned out, this features was much more valuable as only the consideration of weekends. With the help of the constructed correlation matrix, we were further able to

identify features which were considered equivalent to others. As an example, we initially started with the features *ratio_prior_payed_late* and *ratio_prior_payed_ontime*. Obviously one of those features was already sufficient since the other simply reflected the counterpart of this ratio. However, with the help of the correlation matrix, such critical edge cases were identified automatically. In the end, we ended up dropping such identified features which showed strong relationships among each other. To do this, we dropped features with a rather high correlation coefficient very close to -1 or +1 (we presented an overview of these corresponding correlation matrices for both of our datasets in the next subsection).

Last but not least, the actual feature selection process was conducted manually. That means on an iterative trial-and-error basis, features were manually added and removed based on the validation results of the individual ML classifiers and the in-depth feature analyzations in the next subsection. Besides that, it is again important to note that most of the used features were only valuable under the assumption of using invoices from returning customer only (not one-time customers).

### 4.2.3 In-Depth Data Analysis

As we have constructed our features for the payment prediction process, we continued with a related feature analysis. The aim of this analysis was thereby to get a better understanding of the individual feature distributions and their relations to each other. We further used this knowledge to identify the essential features for the prediction process, and to get a feeling of the possible modeling process itself. Note that the feature analysis was thereby again performed on both datasets individually.

First of all, we started again with some standard statistical evaluations to get a better overview of the engineered features and their frequency distributions. As mentioned earlier, our features were thereby categorized into numerical and categorical features. To facilitate this analysis, we performed different statistical measurements on these two feature groups independently. For the numerical features, we reported the mean, standard deviation, and minimal and maximal values. For our categorical feature *country*, we reported the number of unique values and the corresponding maximal and minimal feature frequencies. We presented all of these evaluation results in Table 4.5 and Table 4.6.

With the help of these results, we were finally able to learn a lot about our constructed features and their distribution in the related datasets. For example, revealed that the invoices from *Dataset A* tended to delay only half as much as invoices from *Dataset B*, whereby the overall maximum payment delay was at about 60 days. At this point, we already assumed that feature like *avg_days_late* or *prior_paid_late* might especially be important for our upcoming payment classifications. By investigating the related statistical measurements, we got already a good feeling of how the whole picture in the underlying datasets looked like. A very striking difference between the two datasets could however be observed while investigating the statistical evaluations of the features *prior_total* and *prior_total_late*. These huge difference between our two datasets indicated already that these features are most likely differently ranked among the individual classifier feature importance. Furthermore, we think that a unified model would not be able to rank these features in a proper way which implies

that they could be rather useless for such a specific scenario. However, in our case on the construct of company-specific classifiers, these features might be valuable due to considerations of the established customer's payment behaviors based on particular invoice amounts.

| stats | avg_days_late | non_business_days | prior_paid | prior_paid_late | prior_total | prior_total_late |
|---|---|---|---|---|---|---|
| **mean** | 6,11 | 8,87 | 12,27 | 4,67 | 729,93 | 282,36 |
| **std** | 5,40 | 0,84 | 7,88 | 5,58 | 509,32 | 347,20 |
| **min** | 0,00 | 8,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| **max** | 31,00 | 10,00 | 35,00 | 31,00 | 2.563,22 | 1.804,78 |

| stats | ratio_prior_paid_late | recent_paid_late_days | recent_paid_ontime_days | total | stats | country |
|---|---|---|---|---|---|---|
| **mean** | 0,38 | 68,27 | 38,78 | 59,90 | **count** | 2.466 |
| **std** | 0,37 | 106,14 | 54,61 | 20,44 | **unique** | 5 |
| **min** | 0,00 | 0,00 | 0,00 | 5,26 | **min_freq** | ”391” |
| **max** | 1,00 | 641,00 | 533,00 | 128,28 | **max_freq** | ”818” |

Table 4.5: **Statistical feature analysis results on *Dataset A*** which shows their basic feature characteristics. The categorical and numerical features have thereby been split into separated tables to facilitate the analysis structure considering the different measurement criteria.

| stats | avg_days_late | non_business_days | prior_paid | prior_paid_late | prior_total | prior_total_late |
|---|---|---|---|---|---|---|
| **mean** | 4,62 | 6,01 | 1,27 | 0,42 | 93.773,06 | 26.433,93 |
| **std** | 11,34 | 0,10 | 1,94 | 1,07 | 372.022,80 | 135.841,10 |
| **min** | 0,00 | 6,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| **max** | 57,00 | 7,00 | 14,00 | 7,00 | 2.720.179,00 | 1.133.925,00 |

| stats | ratio_prior_paid_late | recent_paid_late_days | recent_paid_ontime_days | total | stats | country |
|---|---|---|---|---|---|---|
| **mean** | 0,16 | 22,45 | 58,13 | 25.348,61 | **count** | 361 |
| **std** | 0,32 | 55,87 | 96,08 | 60.392,42 | **unique** | 5 |
| **min** | 0,00 | 0,00 | 0,00 | 110,55 | **min_freq** | Austria |
| **max** | 1,00 | 344,00 | 680,00 | 610.000,28 | **max_freq** | Italy |

Table 4.6: **Statistical feature analysis results on *Dataset B*** which shows their basic feature characteristics. The categorical and numerical features have thereby been split into separated tables to facilitate the analysis structure considering the different measurement criteria.

To additionally support the analyzations for our categorical feature *country*, we further evaluated the customer payment behaviors for each country individually. Thereby we revealed that certain countries perform way better in comparison to others. As a counterpart, we also calculated the respective probabilities for invoice late payments in each country. We reported the respective frequency evaluations in Figure 4.2, and the the conditional late payment probabilities in Table 4.7.


Figure 4.2: **Payment behavior distributions by *country* for the used datasets**. The respective invoices were counted, reported, and categorized into "on-time" and "late" payments to visualize the different payment behavior distributions for each country individually.

| **Dataset A** | | **Dataset B** | |
|---|---|---|---|
| ***country*** | ***P(late \| country)*** | ***country*** | ***P(late \| country)*** |
| **391** | 25,49 % | **italy** | 31,07 % |
| **770** | 38,74 % | **austria** | 0,00 % |
| **406** | 41,53 % | **germany** | 40,00 % |
| **818** | 41,34 % | **luxembourg** | 85,71 % |
| **897** | 33,08 % | **switzerland** | 0,00 % |

Table 4.7: **Late payment probabilities for the individual countries in both datasets**. Representing the conditional probabilities that an invoice payment will arrive late under a given customer main country.

As visible from the results above, there existed some specific trend within the customer willingness to an invoice on-time concerning his or her main location. From the results in *Dataset A* we could reveal that especially customers from the country "391" had a higher tendency to pay on-time while their probability to pay late was only 25,49%. Compared to other customer locations in this dataset, this was an already valuable conclusion. Similarly, the analyses on *Dataset B* revealed that customers from Italy had a way higher tendency to pay invoices on-time than customers from other countries in this datasets. However, it is important to note that *Dataset B* represented only a tiny fraction of customers from other countries than Italy. This observation could therefore not be neglected and implied that even though the late payment probability of 31,07% in Italy provided a valuable insight, the probabilities concerning other countries in this dataset needed to be handled with caution. Consequently, we decided to drop the feature *country* from our feature matrix in *Dataset B* to prevent possible bias problems towards invoices classification difficulties which were not issued to customers based in Italy.

To visually back the frequency analysis of the numerical features, we continued by plotting the related boxplots and histograms. For clarity reasons, we discuss in this subsection only the most unique outcomes which showed the most relevant results for the upcoming invoice payment classification process. A full overview of the constructed boxplots and histograms was however provided in Figure 4.4 and Figure 4.5 for *Dataset A*, or respectively Figure 4.7 and Figure 4.8 for *Dataset B*. Moreover, we investigated how those numerical features relate to each other with the help of a scatter plot construction. Again, this subsection will only discuss the most characteristic results, whereby the full scatter plot is presented in Figure 4.3 for *Dataset A* and Figure 4.6 for *Dataset B*.

Out of our ten numerical features, three particular ones caught our attention: *avg_days_late*, *prior_paid_late*, and *ratio_prior_paid_late*. Note that all of these features reflected thereby the customer's payment behavior considering the amount of previouse late payments. Primarily, the analyzation of the constructed scatter plots has helped us thereby to identify these three features, whereby we think that they might be the most important ones to distinguish between on-time and late payments.

From these results we could for example derive that the feature *ratio_prior_paid_late* must most likely be a valuable characteristic for the upcoming classification task, as its associated payment receipt distribution showed some good separability among both classes. Consequently, we could argue that the higher the ratio of prior late payments of a particular customer, the higher also the probability of future late payments. Similarly the other two features *prior_paid_late* and *avg_days_late* could be reviewed. Contrary to *Dataset B* where our investigations revealed that the feature *prior_paid_late* did not provide such a clear separability among both classes. Nevertheless, we observed that some particular values for on-time payments seemed to incorporate with some specific use cases, which is why we thought that this feature might still be important for the upcoming classification task in *Dataset B*.

In combination with the constructed boxplots and histograms, we could further derive the distribution of our numerical features within both datasets. With the help of these results we were consequently able to construct a better picture of the overall customer late payment behavior. We reported the results of these evaluations in Figure 4.4 and Figure 4.5 for *Dataset A*, or respectively Figure 4.7 and Figure 4.8 for *Dataset B*.

To conclude our feature analysis, we constructed the respective feature correlation matrices for both datasets which we reported in Figure 4.9 for *Dataset A* and in Figure 4.10 for *Dataset B*. It is important to note that these matrices represent the relationship between all categorical and numerical features in the datasets, whereby we dropped the categorical feature *country* in *Dataset B* due previously identified possible problems during the upcoming classification task.

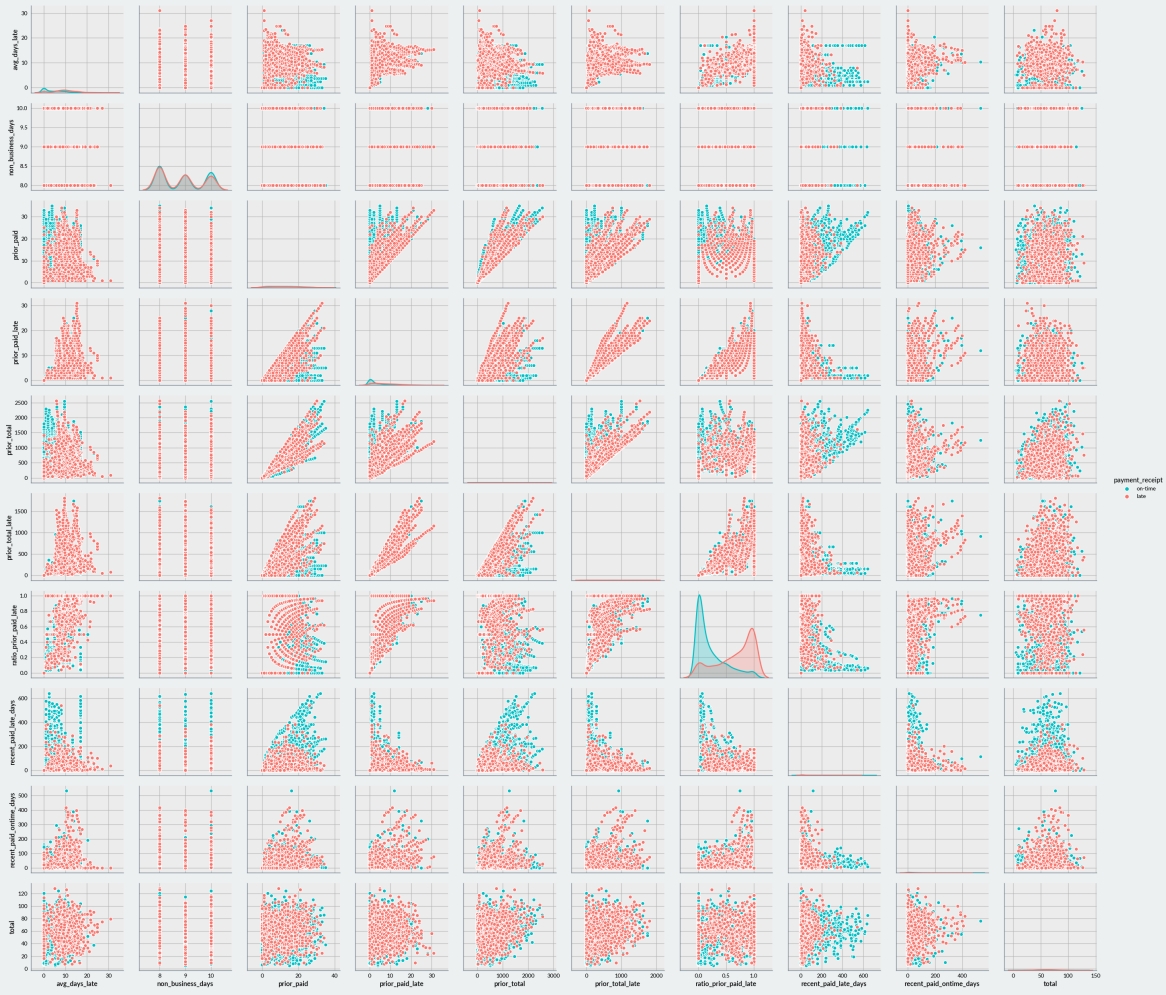


Figure 4.3: **Scatter plots of all numerical features in *Dataset A***. The individual plots represent thereby the relationship between features, and were particularly helpful to determine how each numerical feature contributed to different on-time and late payment behaviors. Especially the feature *ratio_prior_payed_late* seemed to be a suitable classification indicator due to its good separability between both classes.

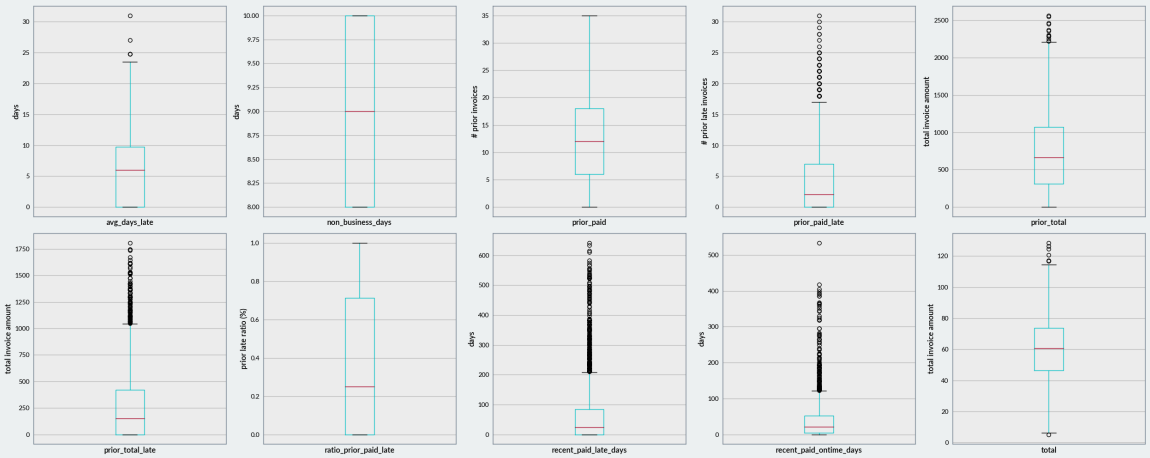


Figure 4.4: **Boxplots of the all numerical features in *Dataset A*** which demonstrate the individual feature distributions within the used dataset. The most characteristic features were thereby (i) *avg_days_late*: average days an invoice was considered overdue, (ii) *prior_paid_late*: number of previously late paid invoices, and (iii) *ratio_prior_paid_late*: the overall ratio of previously late paid invoices in proportion to all paid invoices.

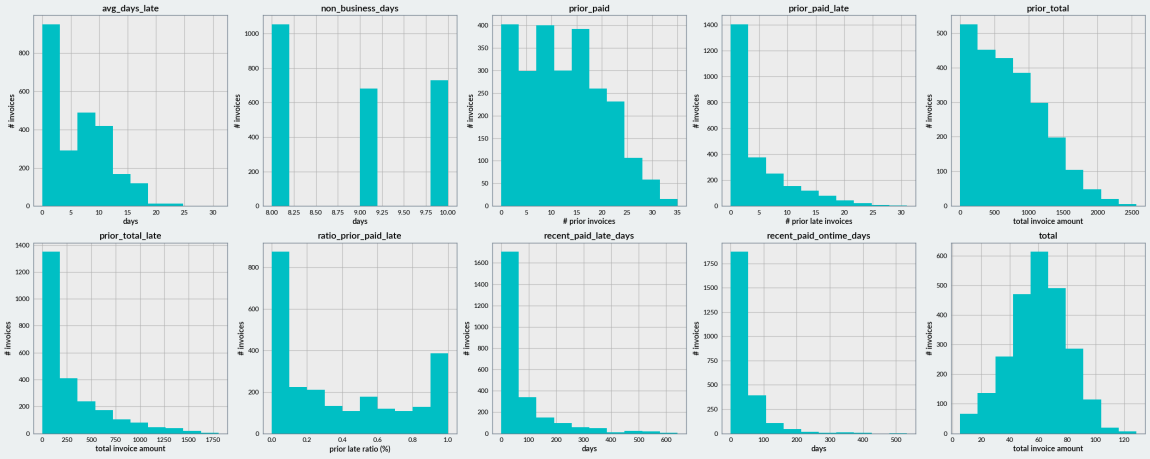


Figure 4.5: **Histograms of all numerical features in *Dataset A*** which demonstrate the individual feature distributions within the used dataset. The most characteristic features were thereby (i) *avg_days_late*: average days an invoice was considered overdue, (ii) *prior_paid_late*: number of previously late paid invoices, and (iii) *ratio_prior_paid_late*: the overall ratio of prior late paid invoices in proportion to all paid invoices.

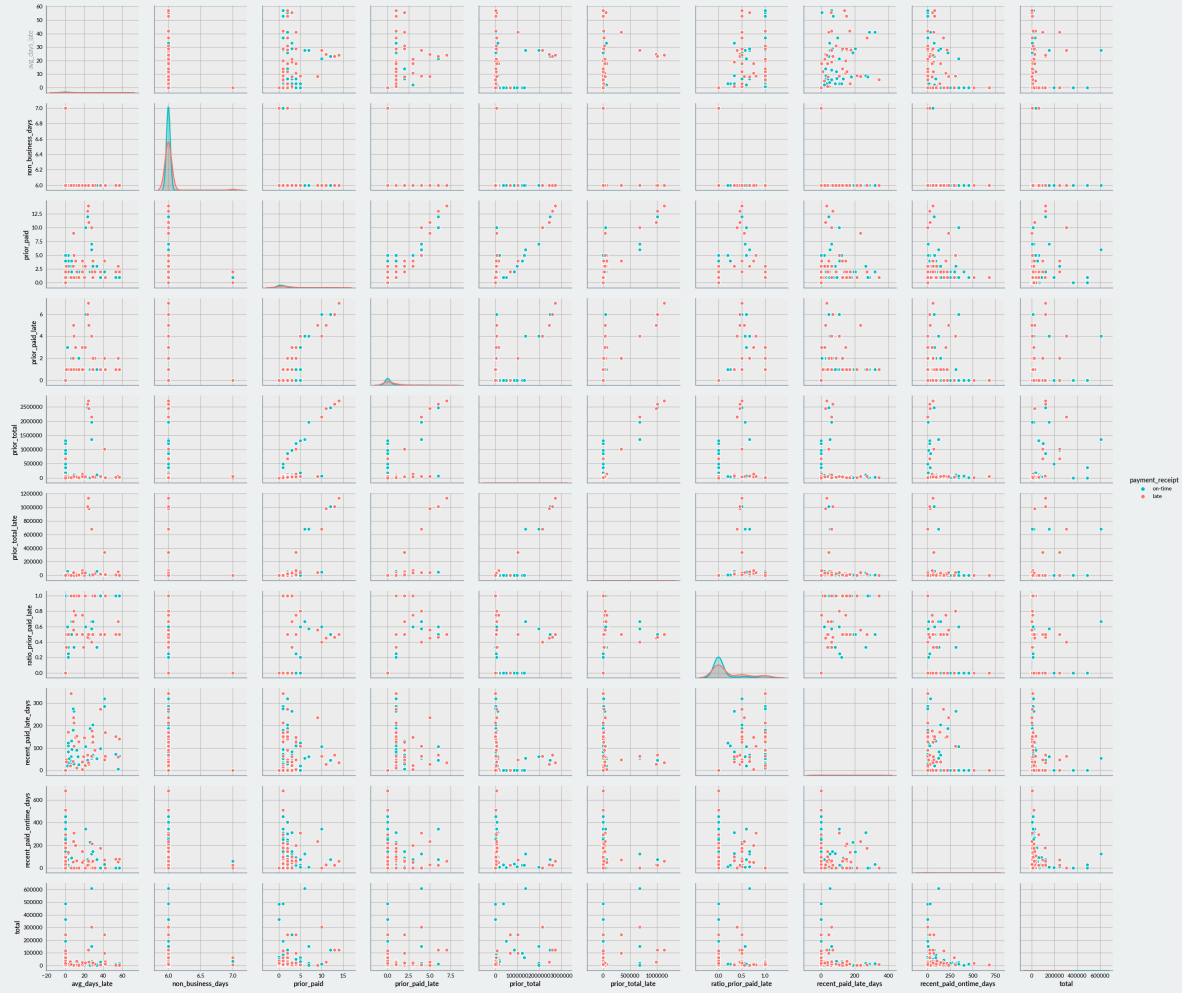


Figure 4.6: **Scatter plots of all numerical features in *Dataset B***. The individual plots represent thereby the relationship between features, and were particularly helpful to determine how each numerical feature contributed to different on-time and late payment behaviors. Especially the feature *ratio_prior_payed_late* and *prior_paid_late* seemed to be a suitable classification indicator due to its good separability between both classes.

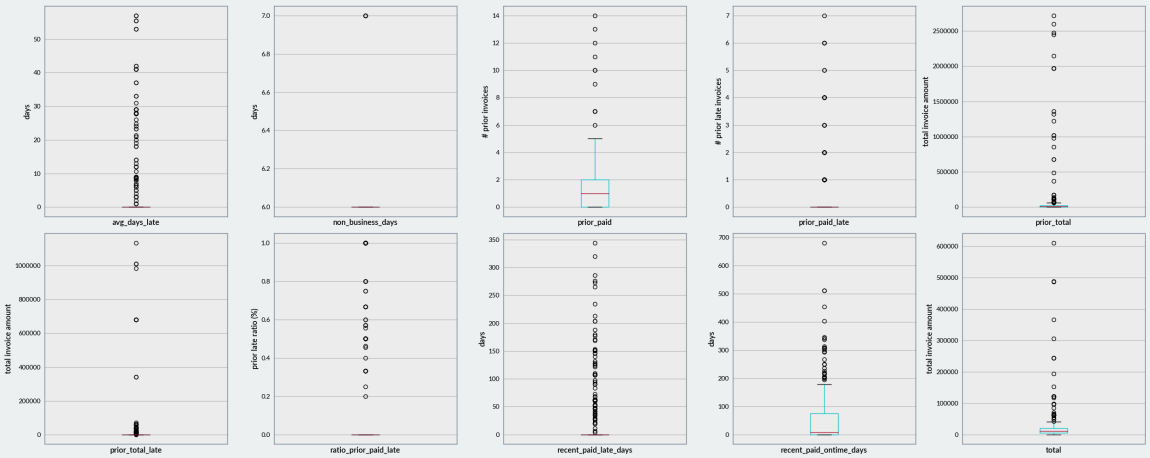


Figure 4.7: **Boxplots of the all numerical features in *Dataset B*** which demonstrate the individual feature distributions within the used dataset. The most characteristic features were thereby (i) *avg_days_late*: average days an invoice was considered overdue, (ii) *prior_paid_late*: number of previously late paid invoices, and (iii) *ratio_prior_paid_late*: the overall ratio of previously late paid invoices in proportion to all paid invoices.

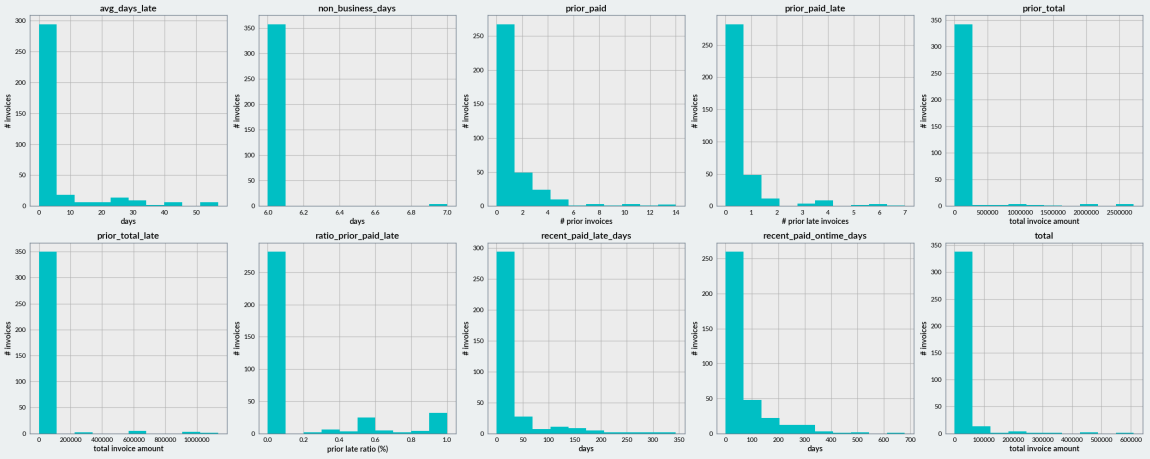


Figure 4.8: **Histograms of all numerical features in *Dataset B*** which demonstrate the individual feature distributions within the used dataset. The most characteristic features were thereby (i) *avg_days_late*: average days an invoice was considered overdue, (ii) *prior_paid_late*: number of previously late paid invoices, and (iii) *ratio_prior_paid_late*: the overall ratio of prior late paid invoices in proportion to all paid invoices.

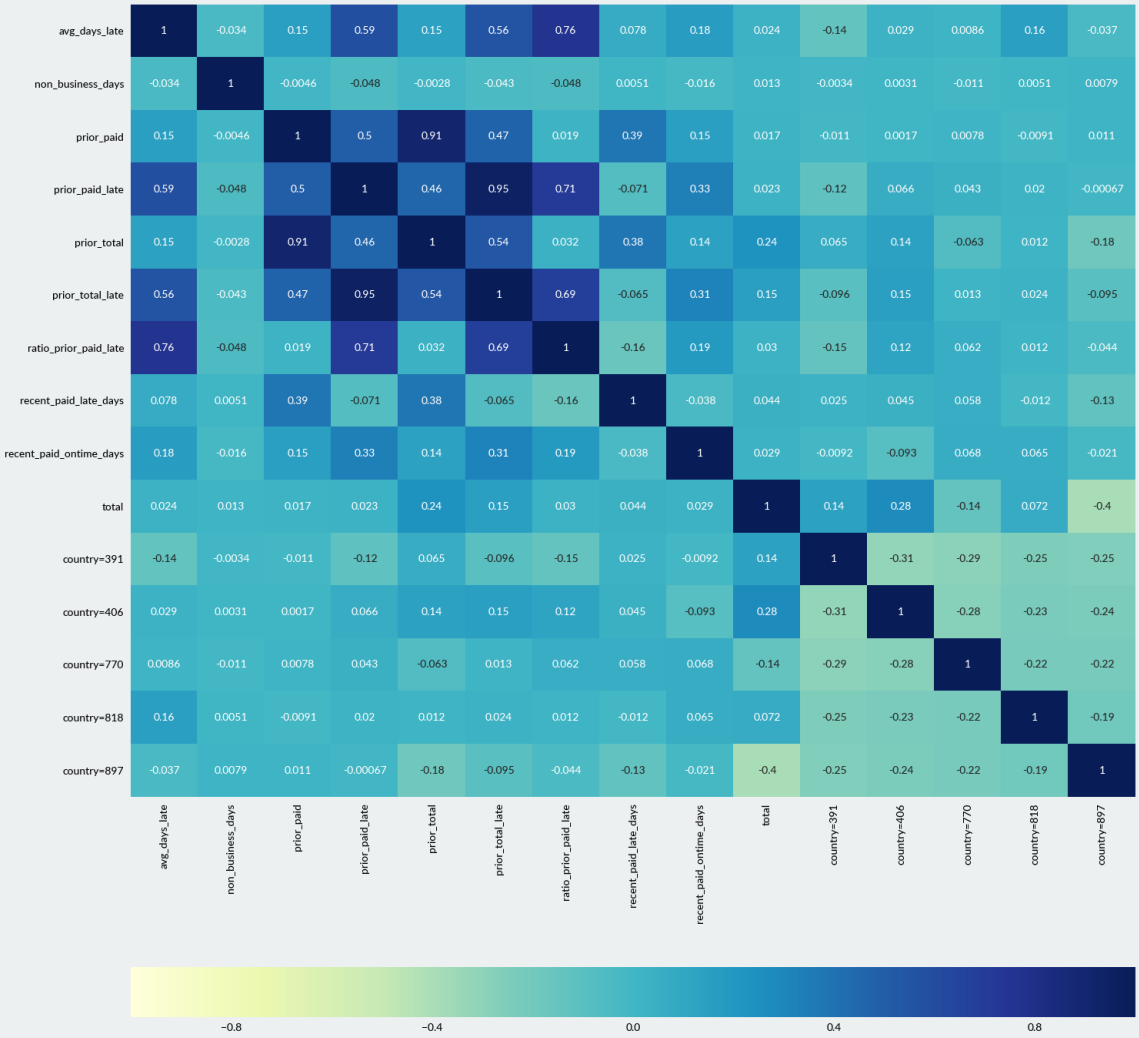


Figure 4.9: **Correlation matrix of all features in *Dataset A*** which reflects the relationship between all numerical and categorical features in the respective dataset. Note that coefficient values close to +1 and -1 represented thereby a strong relationship between features whereby a coefficient value close to 0 represented a very low relationship among features. It is important to restate that we kept the categorical feature *country* for *Dataset A* because our analysis revealed that it should not cause any problems for the upcoming classification task. Moreover, we already preprocessed these categorical features (see *one-hot encoding* in next section), which is why we finally ended up with 15 different features for *Dataset A*.

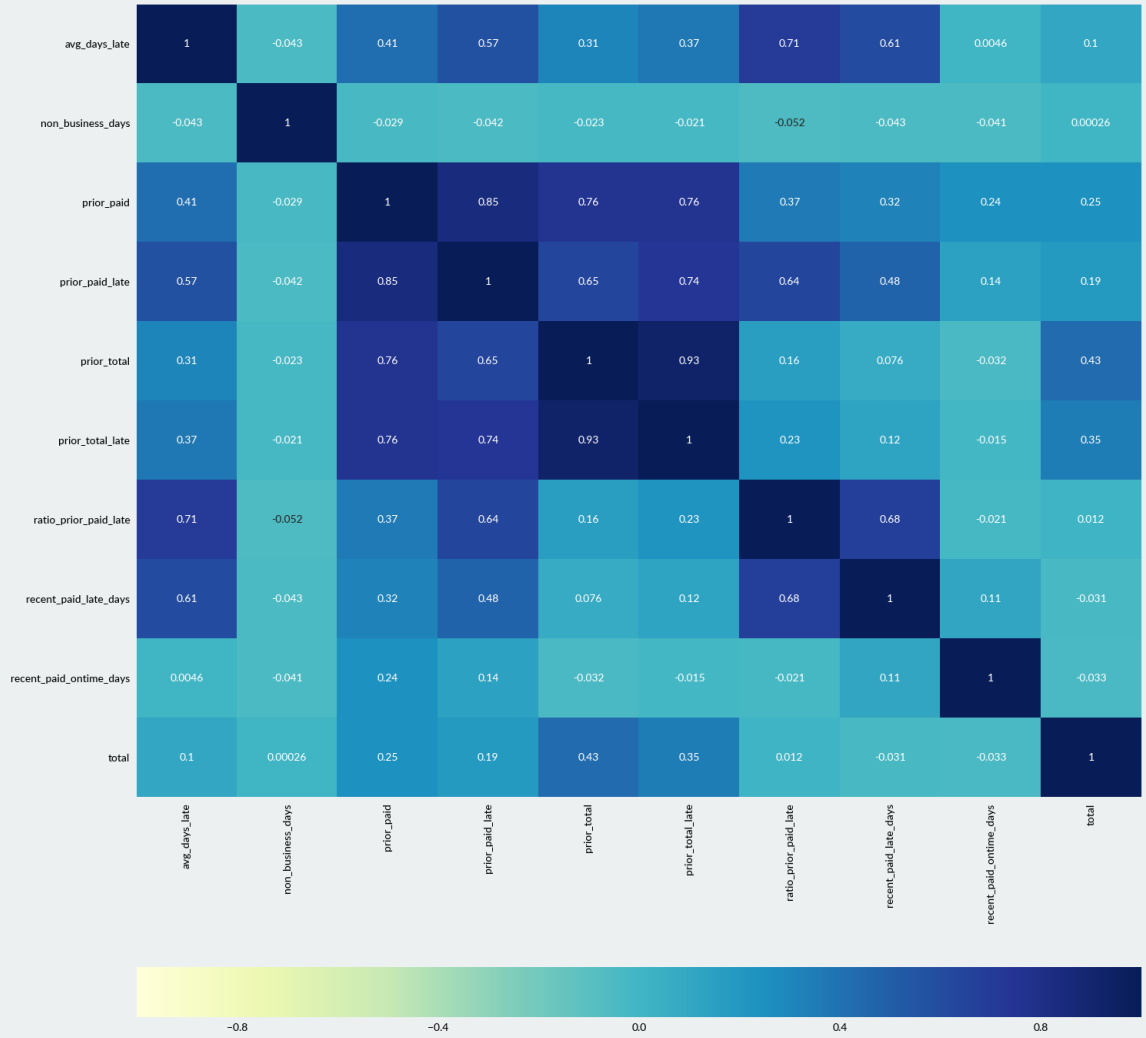


Figure 4.10: **Correlation matrix of all features in *Dataset B*** which reflects the relationship between all numerical and categorical features in the respective dataset. Note that coefficient values close to +1 and -1 represented thereby a strong relationship between features whereby a coefficient value close to 0 represented a very low relationship among features. It is important to restate that we did not include the categorical feature *country* for *Dataset B* because our analysis revealed possible problems during the upcoming classification process with invoices which were issued to other countries than Italy. This implied that we consequently ended up with 10 different features for *Dataset B*.

### 4.2.4 Data Preprocessing

Before we could start with the actual model fitting process, we needed to prepare our datasets and features. This process involved feature encoding, feature scaling or standardization, dataset oversampling, and a final dataset splitting into a training and test set. In this subsection, we will shortly review each of these processing steps and presents the results of the most characteristic outcomes. It is important to note that the process of oversampling serves thereby just as an illustrative instance since the actual oversampling process has been repeated several times during the training of each classifier.

The process of **Feature Encoding** refers to our categorical features and was the first step which we considered. It covered thereby the process of transforming our categorical features into numerical ones. This step was necessary so that all our ML models could work with these type of features. Except for DT and RF, all other proposed classifiers in this thesis do require this specific setup since they cannot operate on labeled data directly (e.g. strings, dates, etc.). In practice, two strategies are commonly used to achieve this transformation. The most simple method is *integer encoding*. This method assigns to each unique categorical value a corresponding integer number which replaces each categorical feature value. Assuming for example that we want to encode all months of a year, this method would simply assign the numbers 1 to 12 to each of the months. The use of this strategy is however task dependent due to its disadvantage of assigning integer values in a natural ordering. In specific ML models such as KNN, this may thereby lead to some unexpected side effect because higher values are considered to have stronger relations than lower ones (due to distance considerations between features). The second commonly used method, which overcomes this problem, is *one-hot encoding*. This method replaces each categorical feature with a new set of binary features whereby each new binary feature represents a unique value of the categorical feature space. Referring again to the example of encoding the months of a year, this strategy would delete the original categorical features and replaces them with 12 new binary features. Each of those features corresponds thereby to a binary value which represents the original state from the categorical features. In our thesis we concluded that the use of *one-hot encoding* would thereby be a better-suited strategy since our categorical feature *country* did not include any ordinal relationships. We provided an illustrative example of the *one-hot encoding* strategy in Table 4.8.

| **Original Feature** | | | | **One-Hot Encoded Feature** | | | | |
|---|---|---|---|---|---|---|---|---|
| | ... | **country** | | | ... | **Austria** | **Italy** | **Germany** |
| **Invoice 1** | ... | Austria | | **Invoice 1** | ... | 1 | 0 | 0 |
| **Invoice 2** | ... | Italy | $\Rightarrow$ | **Invoice 2** | ... | 0 | 1 | 0 |
| **Invoice 3** | ... | Italy | $\Rightarrow$ | **Invoice 3** | ... | 0 | 1 | 0 |
| **Invoice 4** | ... | Germany | | **Invoice 4** | ... | 0 | 0 | 1 |

Table 4.8: **Example of applying *one-hot encoding* on a categorical feature**. The original feature *country* gets replaced by three new binary features which values are set according to the original feature state of the removed categorical feature- this strategy might drastically increase the dimensions of the underlying feature matrix.

**Feature Scaling and Standardization** relates generally to the process of transforming the independent feature values into a common range. Usually, all the provided features

values tend to vary drastically in their original range of values which might strongly influence the outcome and performance of different ML classifiers. To overcome this problem, we distinguish between *scaling* which refers to the *min-max scaling* process and *standardization* which refers to the *z-score standardization* (see Section 3.3.1). Whether a scaling or standardization process needs to be performed is thereby again task dependent. Note that many of the proposed classifiers in Section 3.1 do not necessarily require such a preprocessing step. However, for convenience purposes applying the *z-score standardization* method is generally known as a good habit which may results in a more robost classifier behavior. Nevertheless, some classification methods needed some special attention. The proposed KNN classifier considers for example the use of the Euclidean or Manhatten distance to classify samples. Thereby it is very important to respectively scale all the features such that a higher or lower feature value does not distort the prediction outcome of the classifier per se. We provided an overview of the applied *scaling* and *standardization* processes for each considered classifier in Table 4.9.

**Oversampling** is a common method to overcome the problem of imbalanced datasets as described in Section 3.4. During the initial dataset review we discovered that both of our used datasets are rather imbalanced (see Figure 4.1), and further also rather small considering their sample sizes. Consequently, we concluded to finally use the oversampling strategies ROS, SMOTE and SMOTE + ENN. All of these three oversampling strategies have been finally in use during our model fitting procedure. Consequently, we provided an overview of the basic class distribution after applying these three oversampling methods in Figure 4.11 for *Dataset A* and Figure 4.12 for *Dataset B*. Thereby it is important to note, that these results serve only for illustrative purposes since the actual oversampling procedure has been repeated several times during the model fitting process. The multiple application of oversampling methods had thereby been followed to overcome lucky situations where the sampling process might have picked only easy samples considering the reproduction of synthetic samples or the later classification process. As one can see from our presented results, ROS and SMOTE produced exactly balanced class distribution. Unlike SMOTE + ENN which did not necessarily guarantee an accurate class balance of 50% due to the additional focus on an outlier removal process with the help of the ENN algorithm.

| Classifier | Scaling/Standardization Method | Required |
|---|---|---|
| **Naive Bayes** | z-score standardization | No |
| **Logistic Regression** | z-score standardization | No |
| **Decision Tree** | z-score standardization | No |
| **Random Forest** | z-score standardization | No |
| **K-Nearest Neighbors** | min-max scaling | Yes |
| **Support Vector Machine** | min-max scaling | Yes |

Table 4.9: **Feature scaling and standardization overview per classifier**. All classifiers which make predictions based on distances or similarity measurements require a mandatory scaling or standardizing process. However, the application of *z-score-standardization* is usually a good habit to ensure a more robust classifier behavior.

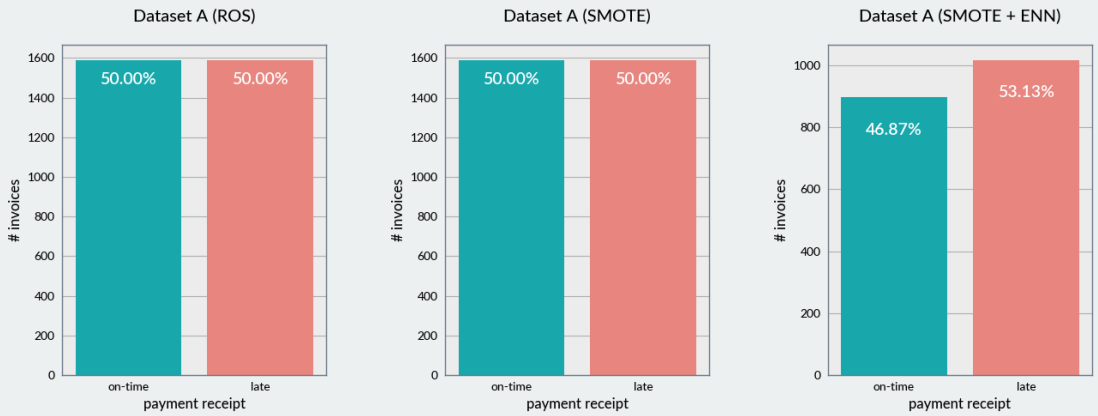


Figure 4.11: **Application of ROS, SMOTE and SMOTE + ENN on *Dataset A***. Note that ROS and SMOTE focused only on the oversampling process, which resulted in a class balanced dataset of the same size. SMOTE + ENN applied an additional outlier detection process on the results of SMOTE which resulted in fewer samples with a slightly skewed dataset classes - fewer on-time payment invoice samples.

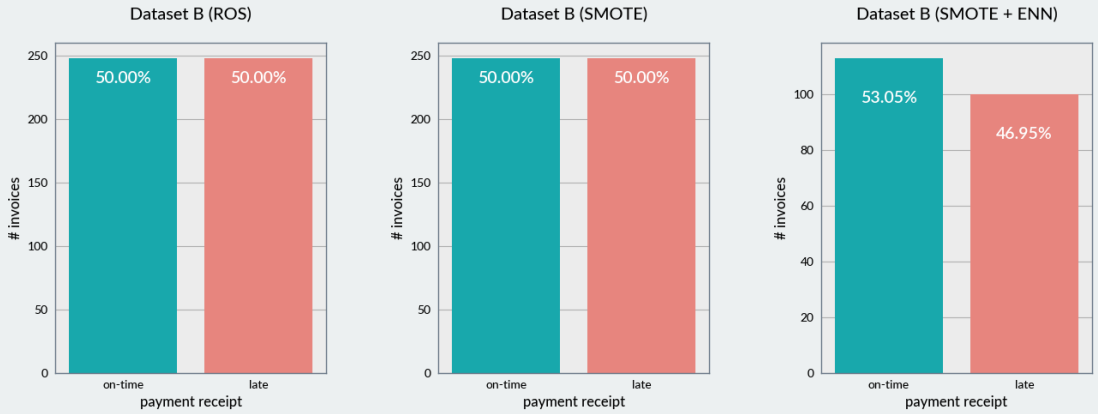


Figure 4.12: **Application of ROS, SMOTE and SMOTE + ENN on *Dataset B***. Note that ROS and SMOTE focused only on the oversampling process, which resulted in a class balanced dataset of the same size. SMOTE + ENN applied an additional outlier detection process on the results of SMOTE which resulted in fewer samples with a slightly skewed dataset classes - fewer late payment invoice samples.

The last step which we needed to perform before starting with the model fitting process was **Dataset Splitting**. This step refers to splitting the original dataset into a corresponding training and test set. Generally, we can see this step as a very essential requirement to properly evaluate a ML classifier. The basic idea bind this subset construction is that a ML classifier should never see a specified portion of data until the final evaluation process to prevent the model from possible overfitting. In other words, the subset which gets extracted as the test set should never be touched again until the final model evaluation. The training set is thereby the only source which serves for training and validation purposes. The splitting process itself is rather straightforward, but still, two main concepts need to be considered. First, the size of the test set should not be chosen too small but also not too big. An established ratio for the test set is thereby around 20% to 30% of the original

dataset size. The intention is thereby to pick a decent amount of samples which cover the most of the variance in the original dataset so that real-world circumstances can be simulated in the final model evaluations. The second important step refers to the random sampling process when splitting the data. This means that the data samples have to be picked randomly (without replacement) such that approximately the same class distribution within in the final training and test set can be expected. In our experiments we decided to stick with a ratio of 70% (training set) to 30% (test set) for both of our datasets. We presented the final results of this dataset splitting procedure in Figure 4.13 for *Dataset A* and in Figure 4.14 for *Dataset B*.It is further necessary to note that we previously sorted all our invoices within the datasets according to their issue date and selected the last 30% of the dataset as or test set, which allowed us to create some realistic training and test sets for the upcoming model training and evaluation. Note that the underlying class distribution within our final training and test sets held thereby approximately the same distribution of on-time and late payment invoices in both of our datasets.

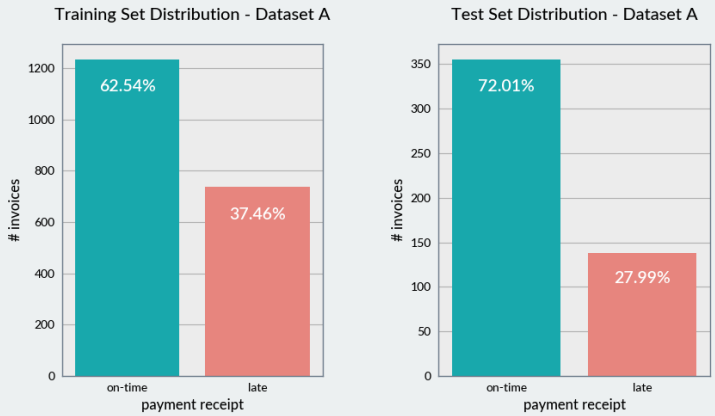


Figure 4.13: **Class distribution of the final training and test set in *Dataset A*,** whereby the test set represented the last 30% of issued invoices to ensure realistic circumstances during our classifier evaluations.

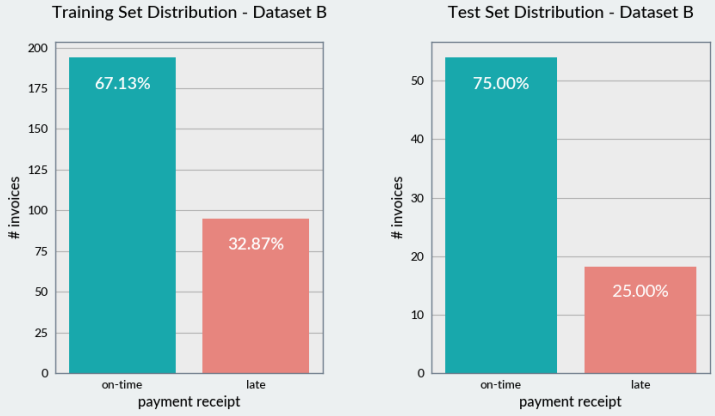


Figure 4.14: **Class distribution of the final training and test set in *Dataset B*,** whereby the test set represented the last 30% of issued invoices to ensure realistic circumstances during our classifier evaluations.

## 4.3 Model Fitting, Selection and Optimization

In this section, we will finally cover the conducted steps to develop a supervised classification model which should be able to classify invoices such that on-time payments can be distinguished from late payment already upon the creation of an invoice. For clarity reasons, we split this section into two subsection where each covers different steps in our development procedures. *Model Parameter Optimization* reviews the process to determine the best-suited hyperparameters for each model, and *Model Fitting, Selection and Testing* covers the process of evaluating the overall performance of each classifier based on their reported accuracy scores. Furthermore, we finally present in this section also the model selection process and the construction of additional ensemble strategies on which the final classifier performances got evaluated.

Our datasets were at this point already prepared and preprocessed, which is why we could further use the defined training set for training and validation purposes while the test set got only used for the model evaluation process only. This means that the test set was only used to check how well our created models performed on unseen data (simulate real-world circumstances). Throughout our hyperparameter determination process, and the model fitting procedure, we performed multiple oversampling iterations to overcome lucky situations where the oversampling process might have lead to evaluation problems. Moreover, to support the upcoming model fitting, selection, and optimization steps, we provided an illustrative workflow procedure in Figure 4.15 which serves as a baseline to back the understanding while guiding through our model development process. To determine the most suitable classification models, we evaluated each of the discussed classifiers in Section 3.1, namely: Naive Bayes, Logistic Regress, Decision Tree, Random Forest, K-Nearest Neighbors, and Support Vector Machine. Moreover, we made use of the ensemble strategies *AdaBoost* and a *Soft-Voting* classifier which served as additional models to possibly enhance out evaluations performances even more.

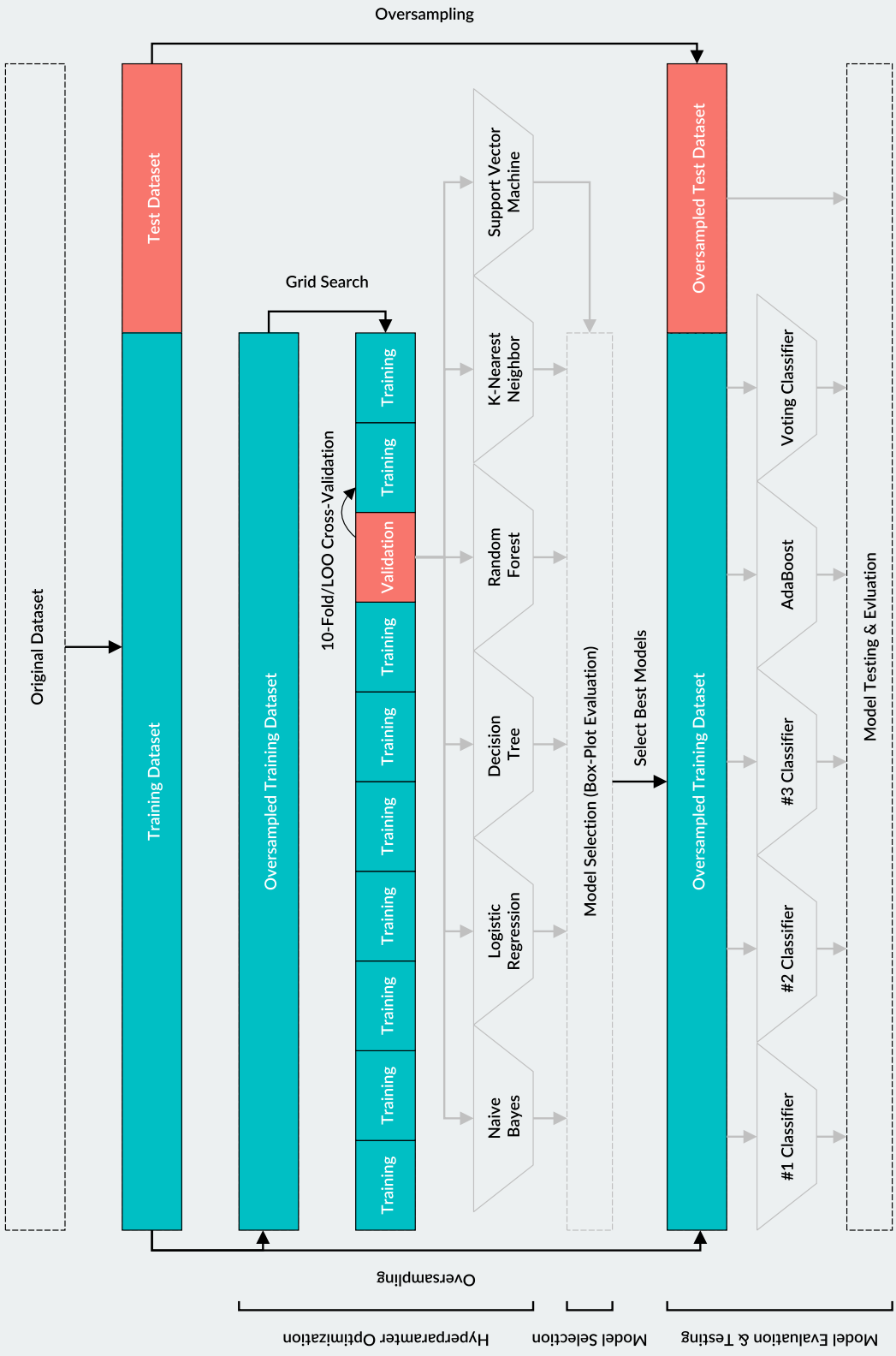


Figure 4.15: **Illustrative model fitting, selection, and optimization workflow** which was used in this thesis to develop a suitable supervised classification model for invoice payment classification into on-time and late payments - supportive workflow to enhance the understanding of our development strategy.

### 4.3.1 Model Parameter Optimization

Before we started with the actual model training process, we needed to find some suitable hyperparameter setups for the individual classifiers. To do this, we followed the upcoming procedure. First, we needed to determine a set of available and useful hyperparameters which were supported by the *scikit-learn* library for each of the used classifiers. Thereby it is important to note, that each model holds its own hyperparameters which may differ in their quantity and value ranges. Once we had chosen the most suitable hyperparameters and their corresponding value ranges (see Table 4.10) we continued to take the prepared training set and applied an oversampling technique to achieve a balanced class setup. Next, we started to evaluate the best combination of hyperparameters for each model by applying $k$-fold CV and the Grid-Search technique. The parameters for $k$-fold CV have thereby been adapted to the size of the corresponding datasets. This means that with the bigger *Dataset A* we used $k = 10$ and for *Dataset B* we used the *leave-one-out* (LOO) strategy due to the lack of available data. Regarding the scoring method which was used by Gird-Search to compare the individual model results, we used the *accuracy* score. The procedure of oversampling and application of $k$-fold CV has thereby been repeated 100 times to overcome lucky situations during the oversampling process (especially important for SMOTE and SMOTE + ENN). In each iteration, the best hyperparameter setup and the corresponding accuracy score got reported for each classifier. In the end, we had a list of hyperparameters with the corresponding average accuracy score which was reported from the $k$-fold CV process. This list of hyperparameters got consequently sorted for each classifier by their accuracy scores, and finally, the hyperparameter setup with the overall median accuracy had been picked as the final setup. We presented the results of these hyperparameter setups for each model and dataset in Table 4.11.

| Classifier | Hyperparemters | Values |
|---|---|---|
| **Naive Bayes** | – | – |
| **Logistic Regression** | $C$ | [0.001, 0.01, 0.1, 1, 10, 100] |
| **Decision Tree** | $max\_depth$ | [1, 5, 10, 15, 20] |
| | $max\_features$** | $[2-15]$ |
| | $min\_samples\_leaf$ | [1, 5, 10, 15, 20] |
| | $criterion$ | [$entropy$] |
| **Random Forest** | $max\_depth$ | [1, 5, 10, 15, 20] |
| | $max\_features$** | $[2-15]$ |
| | $min\_samples\_leaf$ | [1, 5, 10, 15, 20] |
| | $n\_estimators$ | [10, 15, 20, 25] |
| | $criterion$ | [$entropy$] |
| **K-Nearest Neighbors** | $n\_neighbors$* | [15, 20, 25, 30, . . . , 100] |
| **Support Vector Machine** | $kernel$ | [$rbf$] |
| | $gamma$ | [0.01, 0.1, 0.2, 0.5, 1, 10] |
| | $C$ | [0.001, 0.01, 0.1, 1, 10, 100] |

Table 4.10: **Hyperparameters and value ranges for the used classifier models** which have been considered in this thesis. Note, that this table does not cover all possible parameter settings which were provided by the *scikit-learn* library. All parameters which are not present in this table, but supported by the *scikit-learn* library, were set to the suggested default values.

*We ran into problems while evaluating the rather small *Dataset B* and the sampling method SMOTE + ENN. The dataset did not provide enough samples to calculate the corresponding neighbors for KNN up to 100. To compensate this problem, we needed to reduce this range to a maximum of 50 neighbors. Moreover, it is also important to state that the corresponding range of neighbors for KNN did not start at exactly 1 due to possible overfitting problems which could be caused by the oversampling methods.

**For evaluations on *Dataset B*, we used the parameter range 2-10 due to the reduced set of features.

| | | Dataset A | | | Dataset B | | |
|---|---|---|---|---|---|---|---|
| **Classifier** | **Hyperparameter** | **ROS** | **SMOTE** | **SMOTE + ENN** | **ROS** | **SMOTE** | **SMOTE + ENN** |
| **Naive Bayes** | – | – | – | – | – | – | – |
| **Logistic Regression** | *C* | 0.1 | 0.01 | 1 | 0.01 | 0.01 | 0.1 |
| **Decision Tree** | *max_depth* | 16 | 6 | 11 | 16 | 16 | 11 |
| | *max_features* | 6 | 15 | 12 | 15 | 10 | 12 |
| | *min_samples_leaf* | 1 | 11 | 1 | 1 | 1 | 1 |
| | *criterion* | *entropy* | *entropy* | *entropy* | *entropy* | *entropy* | *entropy* |
| **Random Forest** | *max_depth* | 16 | 16 | 16 | 16 | 16 | 11 |
| | *max_features* | 12 | 8 | 4 | 4 | 8 | 4 |
| | *min_samples_leaf* | 1 | 1 | 1 | 1 | 1 | 1 |
| | *n_estimators* | 25 | 25 | 25 | 25 | 25 | 25 |
| | *criterion* | *entropy* | *entropy* | *entropy* | *entropy* | *entropy* | *entropy* |
| **K-Nearest Neighbors** | *n_neighbors* | 85 | 75 | 15 | 40 | 25 | 15 |
| **Support Vector Machine** | *kernel* | *rbf* | *rbf* | *rbf* | *rbf* | *rbf* | *rbf* |
| | *gamma* | 10 | 10 | 10 | 10 | 10 | 10 |
| | *C* | 10 | 10 | 100 | 100 | 100 | 10 |

Table 4.11: **Best hyperparameter setups for each model in *Datset A* and *Dataset B***. We selected the best combination of hyperparameters based on the most suited accuracy score evaluations which were calculated by repetitive application of *k*-fold CV.

### 4.3.2 Model Fitting, Selection and Testing

Once the most suitable hyperparameters for our models were found, we started to evaluate the performance of the individual classification models. Our goal was thereby to determine the classifiers which were most suited for our invoice payment classification task. To do this, we used again the training set, oversampled it and trained each of the six models with the optimized hyperparameters, followed by a $k$-fold CV which reported the average accuracy score. Again, for the bigger *Dataset A* we used $k = 10$ and for *Dataset B* we used *LOO*. As in the search for the best hyperparameters, we repeated this process 100 times to overcome lucky situations during the oversampling process. At this point, we want to note again that we used three different oversampling strategies which is why the whole procedure was reproduced three times with ROS, SMOTE, and SMOTE + ENN. The results of these different classifier evaluations were finally summarized in several boxplots to neatly present the average performances of the 100 training and validation iterations for each classifier. We reported the results for *Dataset A* in Figure 5.1 (ROS), Figure 5.3 (SMOTE), and Figure 5.5 (SMOTE + ENN). Similarly, we reported the results for *Dataset B* in Figure 5.11 (ROS), Figure 5.13 (SMOTE), and Figure 5.15 (SMOTE + ENN).

These boxplot evaluations further revealed that the classifier performances seemed to be strongly influenced by the used oversampling methods. At this point, we already assumed that the results from the applied sampling methods with ROS and SMOTE + ENN had a rather strong tendency to overfit our models on the training data. To approve this assumption, we continued to test our models on the held-back test set, which finally allowed us to report the actual performance of our models on unseen data (constructed real-world circumstances). We reported the respective confusion matrices and evaluation metrics on the test set for each of our models in the following result Chapter 5. Furthermore, we picked the top three classification models which showed the highest overall accuracy score in the related boxplot evaluations, and constructed four additional ensemble classifiers. This was done with the help of AdaBoost and a Soft-Voting classifier to further possibly increase the performance of our classification models. From the resulting evaluations on the test set, we received the approving feedback that the models which were trained on the oversampled datasets with ROS and SMOTE + ENN were actually overfitting. Moreover, we revealed that SMOTE + ENN removed too many samples during the related outlier detection process which most likely removed only samples which were hard to classify (more details in Chapter 6). This is why we decided to performe the final evaluations of the ensemble classifiers only on the oversampled dataset with SMOTE. We reported also these evaluation metrics in the following results Chapter 5. To further clarify this final model training and testing process, we want to state again that the defined training set and the held-back test set were both oversampled with the respectively used methods of ROS, SMOTE, and SMOTE + ENN. This means that the models were trained on the fully oversampled training set, whereby the final evaluations were performed on the oversampled test set. Moreover, it is important to note that we did not repeat the oversampling process 100 times for our final evaluations because we already learned from previous experiments that the use of SMOTE did not influence the actual classification results by much. In the end, we finally selected the most suited classifiers for our task of on-time and late payment classifications with the help of our evaluation

metrics on ROC and the F1-Score. We present all accompaning result, evaluation metrics, and confusion matrices in the next Chapter 5. Chapter 6 continues thereby to interpret these results and further tries to answer the stated research question of this thesis.

# Chapter 5

# Results

In this chapter, we present the results of our invoice payment classifications on two datasets with the used ML classifiers of Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), Naive Bayes (NB), K-Nearest Neighbors (KNN), and Support Vector Machine (SVM). Additionally, we present in this chapter the results of the ensemble classifiers (AdaBoost and Voting classifier) which we built upon the three classification models which reported on average the highest accuracy score on the training and test set. Note that we evaluated all ML classifiers (except for the ensemble classifiers), on three different oversampling strategies, namely: Random Over-Sampling (ROS), Synthetic Minority Over-Sampling Technique (SMOTE), and Synthetic Minority Over-Sampling Technique with Edited Nearest Neighbors (SMOTE + ENN). At this point, we want to anticipate and point out that the results of ROS and SMOTE + ENN tended to raise some problems which were mainly caused by the invoice duplication process with ROS or by the outlier removing process with ENN (more details in Chapter 6). Consequently, we needed to interprete the corresponding results with caution, which is why we further reported the evaluation results of the top three ML classifiers, including their ensemble strategies, only for the oversampled dataset with SMOTE.

To identify the features which contributed the most to our classifications, we further present the constructed decision trees of the DT classifiers, and the feature coefficients of the LR classifiers. Due to limiting time constraints in this thesis, we did not evaluate any detailed feature analysis on NB, KNN, nor on the black-box models with SVM and RF.

## 5.1 Company Specific Results - Dataset A

To evaluate the results for our classifiers on ***Dataset A***, we analyzed the constructed accuracy score boxplots from the training procedure and compared them to the related performances on the oversampled test set. It is important to note again that we created these boxplot results during the classifier training procedure which hold the 100 average accuracy scores of the 100 training iteration with 10-fold CV. To identify the most-suited strategy for overcoming the problem of class imbalance, we further evaluated all classifier performances for each used oversampling method individually. Besides that, this procedure allowed us to determine also possible issues throughout the use of different oversampling methods.

For proper evaluations, we trained each classifier with optimized hyperparameters on the oversampled training set and evaluated them on the oversampled test set. Furthermore, we used a constant classifier as a baseline model which classified each invoice as a "late payment". To evaluate the performances of the individual classifiers on the test set, we respectively reported the confusion matrix, accuracy, precision, recall, and F1-Score. If the respective classifiers supported the use of a reliable probabilistic output, we further stated the AUC measurement form the ROC.

For the oversampling strategy with **ROS**, we reported the boxplot evaluations from the training in Figure 5.1, and an illustration of the oversampled data subsets in Figure 5.2. We listed all related classifier evaluations on the test set in Table 5.1 - Table 5.7.

For the oversampling strategy with **SMOTE**, we reported the boxplot evaluations from the training in Figure 5.3, and an illustration of the oversampled data subsets in Figure 5.4. We listed all related classifier evaluations on the test set in Table 5.8 - Table 5.14.

For the oversampling strategy with **SMOTE + ENN**, we reported the boxplot evaluations from the training in Figure 5.5, and an illustration of the oversampled data subsets in Figure 5.6. We listed all related classifier evaluations on the test set in Table 5.15 - Table 5.21.

Last but not least, we reported the ROC evaluations of our top three classifiers and the corresponding ensemble strategies with AdaBoost and a Soft-Voting classifier in Figure 5.7 and Figure 5.8. Moreover, we listed the accompanying evaluation metrics in Table 5.22 - Table 5.25. To support the feature analysis for the most important characteristics during the classification procedure, we plotted the related feature coefficients of the LR classifier in Figure 5.9, and the constructed decision tree of the DT classifier in Figure 5.10. As a side note, we want to mention that we used *n_estimators: 100* and *learning_rate: 0.1* as the respective hyperparameters for AdaBoost, and the default *scikit-learn* parameters for the Soft-Voting classifier - corresponding to a uniformly weighting among the individual classifiers.

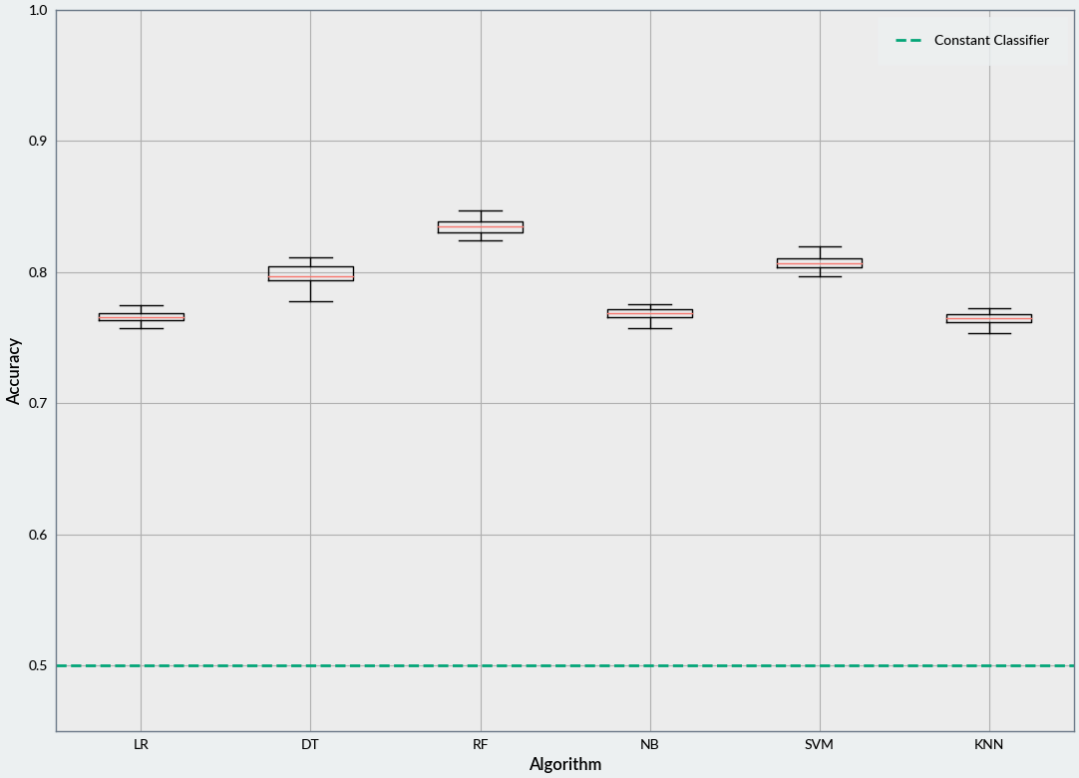


Figure 5.1: **Boxplots of all accuracy score evaluations in *Dataset A* (ROS)** for each used classifier on the training set. Each boxplot contains the 100 average accuracy scores during the repeated training evaluations by 10-fold CV. On the x-axis, we reported the used classifiers whereby we listed the respective accuracy scores on the y-axis. Moreover, we used a constant classifier (green dashed line) as the baseline model.

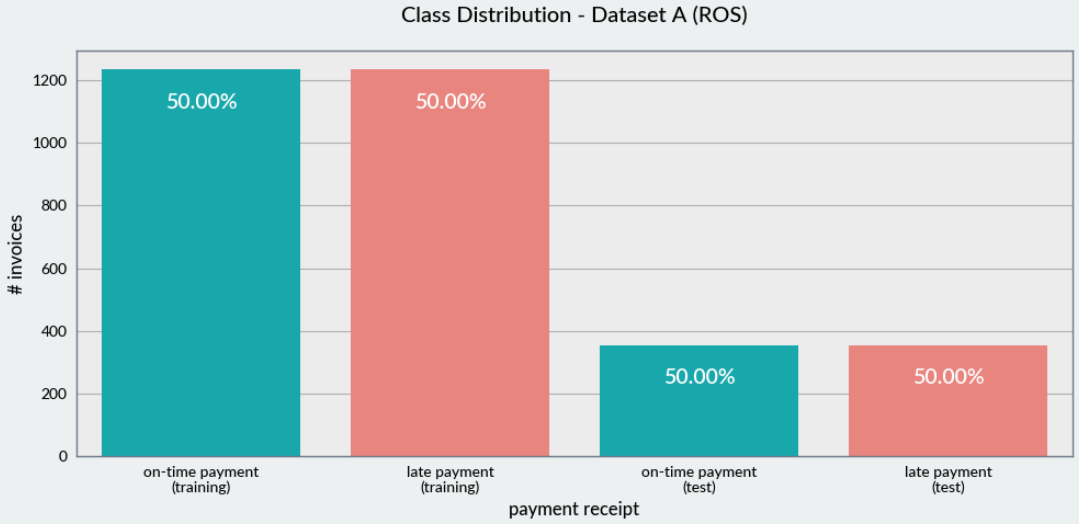


Figure 5.2: **Oversampled training and test set in *Dataset A* (ROS)** that we used for the final model evaluations; illustrating the achieved class balance of 50% in both subsets. The data split on the original dataset was performed on a ratio of 80% (training set) to 20% (test set).

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 355 | 0 | *late* | 0.50 | 1.00 | 0.67 | – |
| *on-time* | 355 | 0 | *on-time* | 0.00 | 0.00 | 0.00 | – |
| **Accuracy: 0.50** | | | **avg/total** | **0.25** | **0.50** | **0.33** | **0.50** |

Table 5.1: **Evaluation results of the Constant classifier - *Dataset A* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 283 | 72 | *late* | 0.81 | 0.80 | 0.80 | – |
| *on-time* | 68 | 287 | *on-time* | 0.80 | 0.81 | 0.80 | – |
| **Accuracy: 0.80** | | | **avg/total** | **0.80** | **0.80** | **0.80** | **0.86** |

Table 5.2: **Evaluation results of the LR classifier - *Dataset A* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 197 | 158 | *late* | 0.78 | 0.55 | 0.65 | – |
| *on-time* | 57 | 298 | *on-time* | 0.65 | 0.84 | 0.73 | – |
| **Accuracy: 0.70** | | | **avg/total** | **0.71** | **0.70** | **0.69** | **0.69** |

Table 5.3: **Evaluation results of the DT classifier - *Dataset A* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 216 | 139 | *late* | 0.82 | 0.61 | 0.70 | – |
| *on-time* | 47 | 308 | *on-time* | 0.69 | 0.87 | 0.77 | – |
| **Accuracy: 0.76** | | | **avg/total** | **0.76** | **0.74** | **0.73** | **0.82** |

Table 5.4: **Evaluation results of the RF classifier - *Dataset A* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 300 | 55 | *late* | 0.76 | 0.85 | 0.80 | – |
| *on-time* | 97 | 258 | *on-time* | 0.82 | 0.73 | 0.77 | – |
| **Accuracy: 0.79** | | | **avg/total** | **0.79** | **0.79** | **0.79** | **–** |

Table 5.5: **Evaluation results of the NB classifier- *Dataset A* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 298 | 57 | *late* | 0.79 | 0.84 | 0.81 | – |
| *on-time* | 80 | 275 | *on-time* | 0.83 | 0.77 | 0.80 | – |
| **Accuracy: 0.81** | | | **avg/total** | **0.81** | **0.81** | **0.81** | **0.87** |

Table 5.6: **Evaluation results of the KNN classifier - *Dataset A* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 180 | 175 | *late* | 0.74 | 0.51 | 0.60 | – |
| *on-time* | 64 | 291 | *on-time* | 0.62 | 0.82 | 0.71 | – |
| **Accuracy: 0.66** | | | **avg/total** | **0.68** | **0.66** | **0.65** | **0.74** |

Table 5.7: **Evaluation results of the SVM classifier - *Dataset A* (ROS)**.

We will present the accompanying discussions on these results in Chapter 6. Next, we reported the results of *Dataset A* using the SMOTE-based sampling method where we again used the oversampled test set to evaluate our classifiers on previously unseen data.

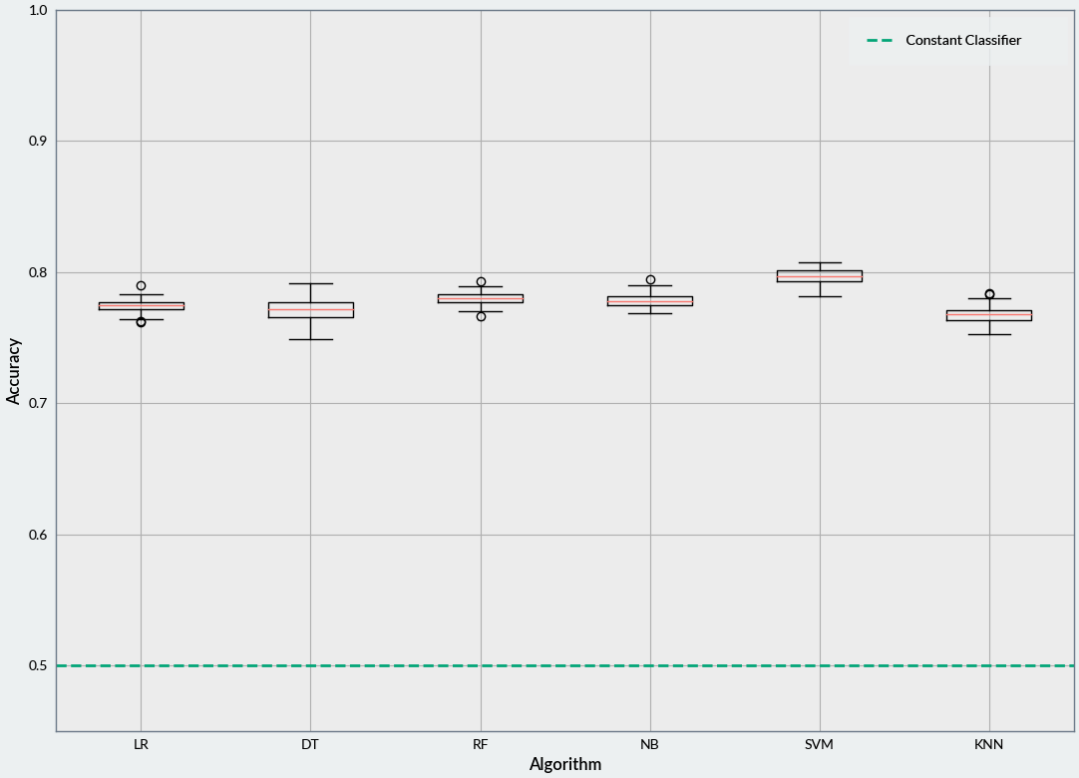


Figure 5.3: **Boxplots of all accuracy score evaluations in *Dataset A* (SMOTE)** for each used classifier on the training set. Each boxplot contains the 100 average accuracy scores during the repeated training evaluations by 10-fold CV. On the x-axis, we reported the used classifiers whereby we listed the respective accuracy scores on the y-axis. Moreover, we used a constant classifier (green dashed line) as the baseline model.

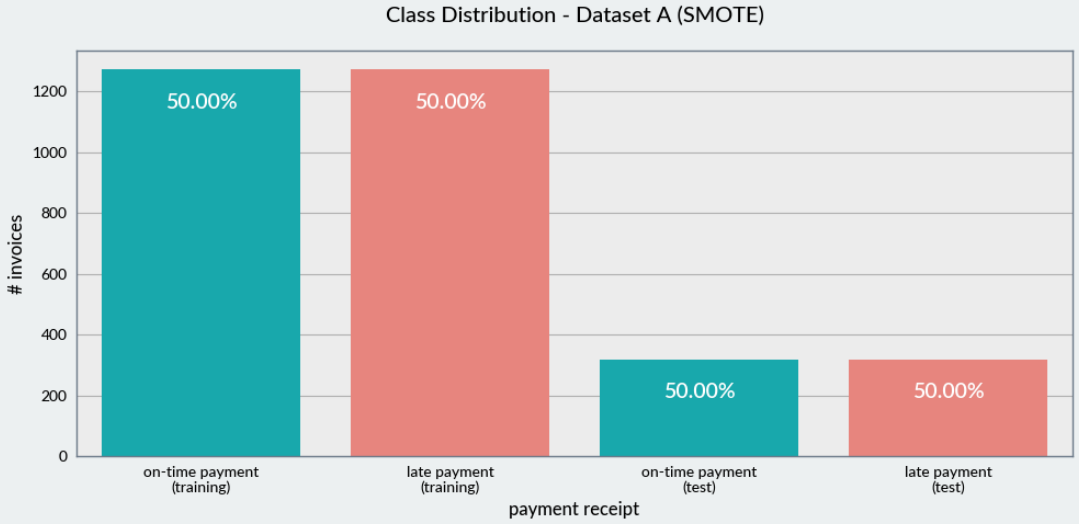


Figure 5.4: **Oversampled training and test set in *Dataset A* (SMOTE)** that we used for the final model evaluations; illustrating the achieved class balance of 50% in both subsets. The data split on the original dataset was performed on a ratio of 80% (training set) to 20% (test set).

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 355 | 0 | *late* | 0.50 | 1.00 | 0.67 | – |
| *on-time* | 355 | 0 | *on-time* | 0.00 | 0.00 | 0.00 | – |
| **Accuracy: 0.50** | | | **avg/total** | **0.25** | **0.50** | **0.33** | **0.50** |

Table 5.8: **Evaluation results of the Constant classifier - *Dataset A* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 294 | 61 | *late* | 0.79 | 0.83 | 0.81 | – |
| *on-time* | 76 | 279 | *on-time* | 0.82 | 0.79 | 0.80 | – |
| **Accuracy: 0.81** | | | **avg/total** | **0.81** | **0.81** | **0.81** | **0.86** |

Table 5.9: **Evaluation results of the LR classifier - *Dataset A* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 246 | 109 | *late* | 0.78 | 0.69 | 0.73 | – |
| *on-time* | 69 | 286 | *on-time* | 0.72 | 0.81 | 0.76 | – |
| **Accuracy: 0.75** | | | **avg/total** | **0.75** | **0.75** | **0.75** | **0.84** |

Table 5.10: **Evaluation results of the DT classifier - *Dataset A* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 231 | 124 | *late* | 0.81 | 0.65 | 0.72 | – |
| *on-time* | 54 | 301 | *on-time* | 0.71 | 0.85 | 0.77 | – |
| **Accuracy: 0.75** | | | **avg/total** | **0.76** | **0.75** | **0.75** | **0.86** |

Table 5.11: **Evaluation results of the RF classifier - *Dataset A* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 304 | 51 | *late* | 0.76 | 0.86 | 0.81 | – |
| *on-time* | 95 | 260 | *on-time* | 0.84 | 0.73 | 0.78 | – |
| **Accuracy: 0.79** | | | **avg/total** | **0.80** | **0.79** | **0.79** | – |

Table 5.12: **Evaluation results of the NB classifier - *Dataset A* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 287 | 68 | *late* | 0.79 | 0.81 | 0.80 | – |
| *on-time* | 77 | 278 | *on-time* | 0.80 | 0.78 | 0.79 | – |
| **Accuracy: 0.80** | | | **avg/total** | **0.80** | **0.80** | **0.80** | **0.85** |

Table 5.13: **Evaluation results of the KNN classifier - *Dataset A* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| ***late*** | 175 | 180 | ***late*** | 0.75 | 0.49 | 0.59 | – |
| ***on-time*** | 59 | 296 | ***on-time*** | 0.62 | 0.83 | 0.71 | – |
| **Accuracy: 0.66** | | | **avg/total** | **0.68** | **0.66** | **0.65** | **0.72** |

Table 5.14: **Evaluation results of the SVM classifier -** ***Dataset A*** **(SMOTE)**.

We will present the accompanying discussions on these results in Chapter 6. Next, we reported the results of *Dataset A* using the SMOTE + ENN-based sampling method where we again used the oversampled test set to evaluate our classifiers on previously unseen data.

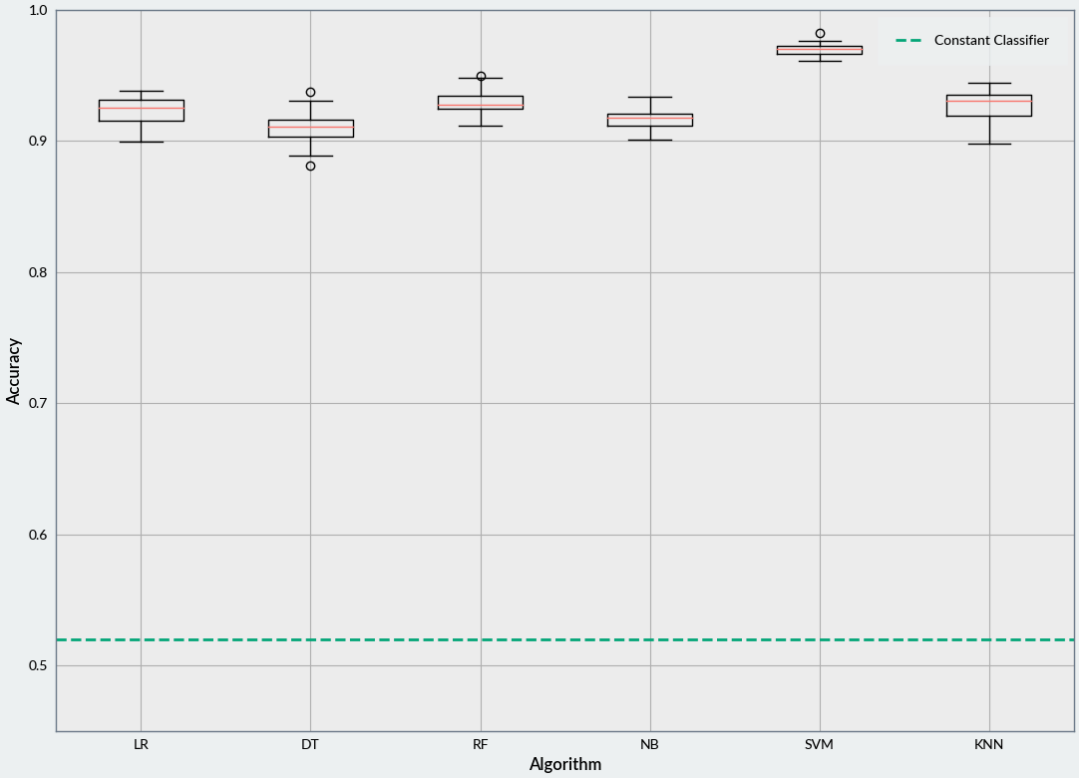


Figure 5.5: **Boxplots of all accuracy score evaluations in *Dataset A* (SMOTE + ENN)** for each used classifier on the training set. Each boxplot contains the 100 average accuracy scores during the repeated training evaluations by 10-fold CV. On the x-axis, we reported the used classifiers whereby we listed the respective accuracy scores on the y-axis. Moreover, we used a constant classifier (green dashed line) as the baseline model.

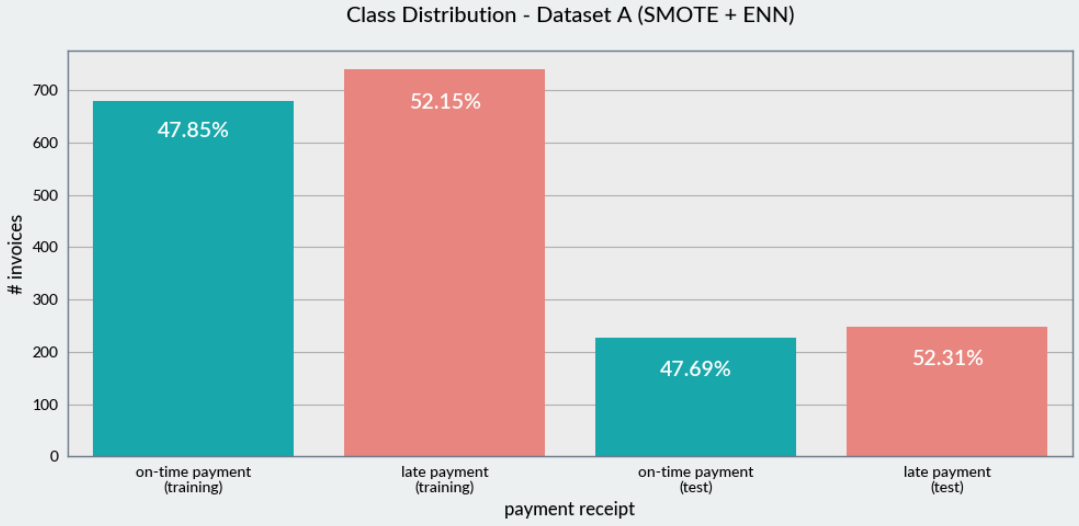


Figure 5.6: **Oversampled training and test set in *Dataset A* (SMOTE + ENN)** that we used for the final model evaluations; illustrating the skewed class balance which was caused by ENN. The data split on the original dataset was performed on a ratio of 80% (training set) to 20% (test set).

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 260 | 0 | *late* | 0.52 | 1.00 | 0.69 | – |
| *on-time* | 239 | 0 | *on-time* | 0.00 | 0.00 | 0.00 | – |
| **Accuracy: 0.52** | | | **avg/total** | **0.27** | **0.52** | **0.36** | **0.50** |

Table 5.15: **Evaluation results of the Constant classifier - *Dataset A* (SMOTE + ENN).**

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 237 | 23 | *late* | 0.92 | 0.91 | 0.92 | – |
| *on-time* | 20 | 216 | *on-time* | 0.90 | 0.92 | 0.91 | – |
| **Accuracy: 0.91** | | | **avg/total** | **0.91** | **0.91** | **0.91** | **0.96** |

Table 5.16: **Evaluation results of the LR classifier - *Dataset A* (SMOTE + ENN).**

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 230 | 30 | *late* | 0.93 | 0.88 | 0.91 | – |
| *on-time* | 16 | 220 | *on-time* | 0.88 | 0.93 | 0.91 | – |
| **Accuracy: 0.91** | | | **avg/total** | **0.91** | **0.91** | **0.91** | **0.91** |

Table 5.17: **Evaluation results of the DT classifier - *Dataset A* (SMOTE + ENN).**

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 230 | 30 | *late* | 0.94 | 0.88 | 0.91 | – |
| *on-time* | 14 | 222 | *on-time* | 0.88 | 0.94 | 0.91 | – |
| **Accuracy: 0.91** | | | **avg/total** | **0.91** | **0.91** | **0.91** | **0.95** |

Table 5.18: **Evaluation results of the RF classifier - *Dataset A* (SMOTE + ENN).**

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 240 | 20 | *late* | 0.85 | 0.92 | 0.89 | – |
| *on-time* | 42 | 194 | *on-time* | 0.91 | 0.82 | 0.86 | – |
| **Accuracy: 0.88** | | | **avg/total** | **0.88** | **0.88** | **0.87** | – |

Table 5.19: **Evaluation results of the NB classifier - *Dataset A* (SMOTE + ENN).**

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 231 | 24 | *late* | 0.94 | 0.91 | 0.92 | – |
| *on-time* | 16 | 217 | *on-time* | 0.90 | 0.93 | 0.92 | – |
| **Accuracy: 0.92** | | | **avg/total** | **0.92** | **0.92** | **0.92** | **0.97** |

Table 5.20: **Evaluation results of the KNN classifier - *Dataset A* (SMOTE + ENN).**

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| ***late*** | 219 | 36 | ***late*** | 0.88 | 0.86 | 0.87 | – |
| ***on-time*** | 30 | 203 | ***on-time*** | 0.85 | 0.87 | 0.86 | – |
| **Accuracy: 0.86** | | | **avg/total** | **0.87** | **0.86** | **0.86** | **0.90** |

Table 5.21: **Evaluation results of the SVM classifier - *Dataset A* (SMOTE + ENN)**.

We will present the accompanying discussions on these results in Chapter 6. Next, we reported the results of the identified top three classifiers, and the four applied ensemble strategies on *Dataset A*, using the SMOTE-based oversampling method.

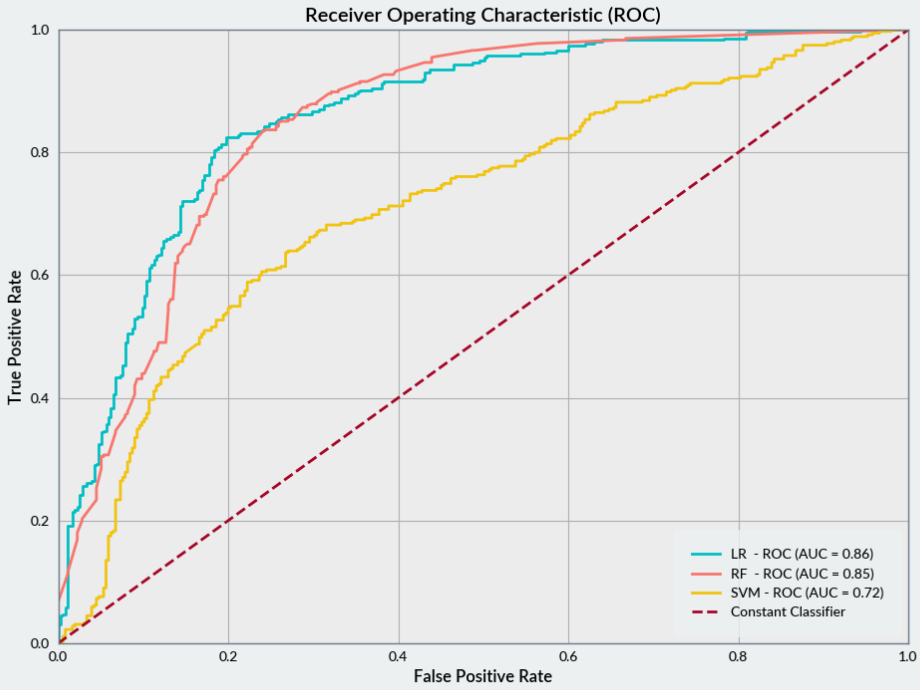


Figure 5.7: **ROC evaluations of the top three classifiers - *Dataset A* (SMOTE)**, whereby we reported the ROC-Curve and accompanying AUC measurements for LR, RF, and SVM to enable a supportive comparison.

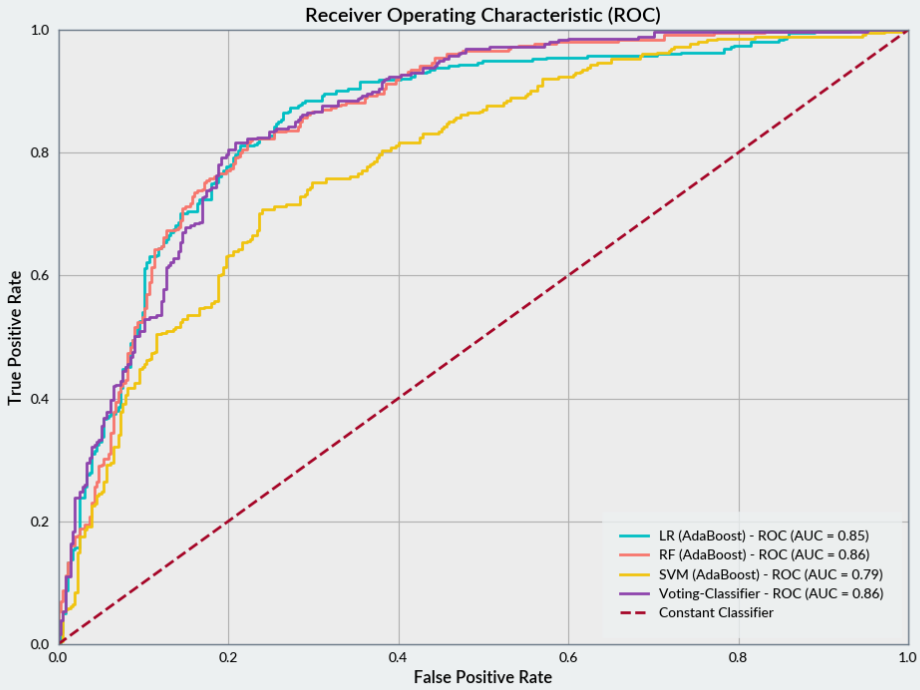


Figure 5.8: **ROC evaluations of the top three ensemble classifiers - *Dataset A* (SMOTE)**, whereby we reported the ROC-Curve and accompanying AUC measurements of AdaBoost LR, AdaBoost RF, AdaBoost SVM, and the Soft-Voting classifier (LR, RF, SVM) to enable a supportive comparison among the ensemble strategies.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 326 | 29 | *late* | 0.70 | 0.92 | 0.79 | – |
| *on-time* | 141 | 214 | *on-time* | 0.88 | 0.60 | 0.72 | – |
| **Accuracy: 0.76** | | | **avg/total** | **0.79** | **0.76** | **0.75** | **0.85** |

Table 5.22: **Evaluation results of the AdaBoost LR classifier - *Dataset A* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 268 | 87 | *late* | 0.81 | 0.75 | 0.78 | – |
| *on-time* | 63 | 292 | *on-time* | 0.77 | 0.82 | 0.80 | – |
| **Accuracy: 0.79** | | | **avg/total** | **0.79** | **0.79** | **0.79** | **0.86** |

Table 5.23: **Evaluation results of the AdaBoost RF classifier - *Dataset A* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 207 | 148 | *late* | 0.76 | 0.58 | 0.66 | – |
| *on-time* | 67 | 288 | *on-time* | 0.66 | 0.81 | 0.73 | – |
| **Accuracy: 0.70** | | | **avg/total** | **0.71** | **0.70** | **0.69** | **0.79** |

Table 5.24: **Evaluation results of the AdaBoost SVM classifier - *Dataset A* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 253 | 102 | *late* | 0.81 | 0.71 | 0.76 | – |
| *on-time* | 67 | 295 | *on-time* | 0.74 | 0.83 | 0.78 | – |
| **Accuracy: 0.77** | | | **avg/total** | **0.78** | **0.77** | **0.77** | **0.86** |

Table 5.25: **Evaluation results of the Soft-Voting classifier- *Dataset A* (SMOTE)**.

Finally, we presented the constructed decision tree of the DT classifier and the feature coefficients of the LR classifier; further used to analyze the respective feature importance.

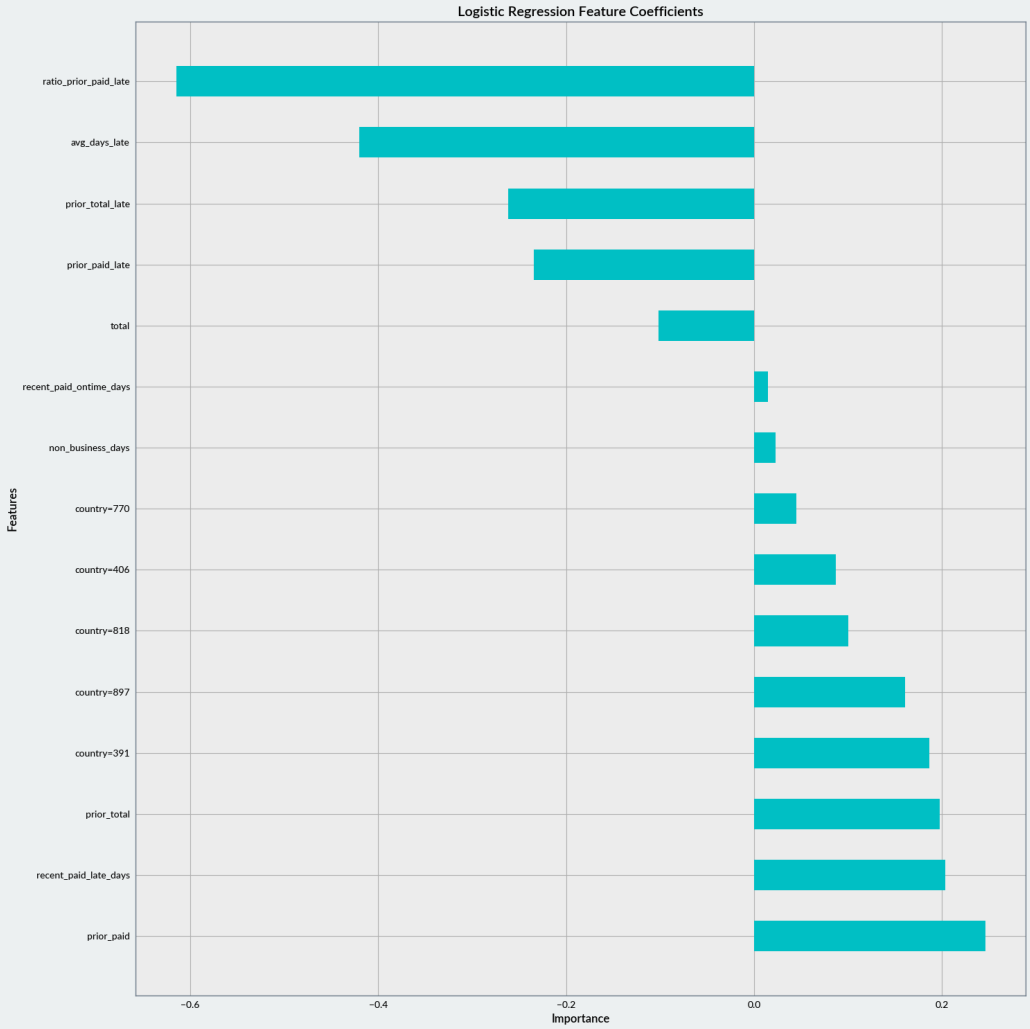


Figure 5.9: **Feature coefficients used by the LR classifier - *Dataset A* (SMOTE)** to classify the invoices into on-time and late payments. On the y-axis, we reported the categorical and numerical features, and on the x-axis, we respectively reported their feature importance. The feature importance represents thereby the calculated coefficient values in the LR classifier which lie in the range between -1 and +1. The closer a feature's coefficient value is to -1, the more important is the feature for classifying an invoice as a late payment. Contrary, the closer a feature's coefficient value is to +1, the more important is the feature for an on-time payment classification. This implies for the trained LR classifier in *Dataset A*, that the features *ratio_prior_paid_late* and *avg_days_late* held the essential characteristics to predict a late payment classification, while the features *prior_paid*, and *recent_paid_late_days* were considered as the most important ones for an on-time payment classification.

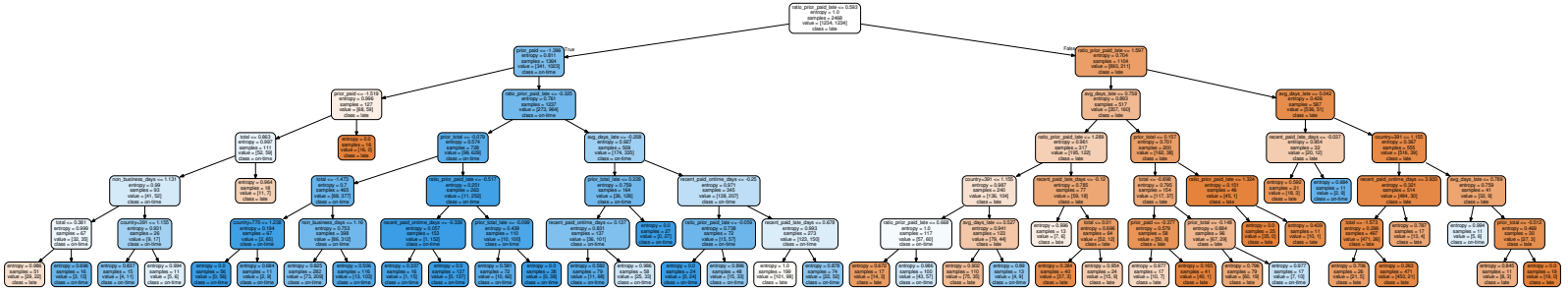

Figure 5.10: **Constructed decision tree by the DT classifier - *Dataset A* (SMOTE)** to classify the invoices into on-time and late payments. The most characteristic features are reported at the top of the tree and decrease by their importance while descending the tree structure. The invoice features *ratio_prior_paid_late*, *avg_days_late*, and *prior_paid* were consequently considered as the most important features for the DT to classify an invoice into a late or on-time payment. We trained the constructed DT classifier for *Dataset A* with the optimized hyperparameters on the oversampled training set and evaluated it on the oversampled test set. The best AUC corresponded thereby to 86%. The blue nodes in the presented decision tree represent an on-time payment classification while the orange nodes represent a late payment classification. The color saturation of the individual nodes represents the respective class distribution, whereby white nodes denote a uniform distribution.

## 5.2 Company Specific Results - Dataset B

To evaluate the results for our classifiers on ***Dataset B***, we analyzed the constructed accuracy score boxplots from the training procedure and compared them to the related performances on the oversampled test set. It is important to note again that we created these boxplot results during the classifier training procedure which hold the 100 average accuracy scores of the 100 training iteration with LOO CV. To identify the most-suited strategy for overcoming the problem of class imbalance, we further evaluated all classifier performances for each used oversampling method individually. Besides that, this procedure allowed us to determine also possible issues throughout the use of different oversampling methods.

For proper evaluations, we trained each classifier with optimized hyperparameters on the oversampled training set and evaluated them on the oversampled test set. Furthermore, we used a constant classifier as a baseline model which classified each invoice as a "late payment". To evaluate the performances of the individual classifiers on the test set, we respectively reported the confusion matrix, accuracy, precision, recall, and F1-Score. If the respective classifiers supported the use of a reliable probabilistic output, we further stated the AUC measurement form the ROC.

For the oversampling strategy with **ROS**, we reported the boxplot evaluations from the training in Figure 5.11, and an illustration of the oversampled data subsets in Figure 5.12. We listed all related classifier evaluations on the test set in Table 5.26 - Table 5.32.

For the oversampling strategy with **SMOTE**, we reported the boxplot evaluations from the training in Figure 5.13, and an illustration of the oversampled data subsets in Figure 5.14. We listed all related classifier evaluations on the test set in Table 5.33 - Table 5.39.

For the oversampling strategy with **SMOTE + ENN**, we reported the boxplot evaluations from the training in Figure 5.15, and an illustration of the oversampled data subsets in Figure 5.16. We listed all related classifier evaluations on the test set in Table 5.40 - Table 5.46.

Last but not least, we reported the ROC evaluations of our top three classifiers and the corresponding ensemble strategies with AdaBoost and a Soft-Voting classifier in Figure 5.17 and Figure 5.18. Moreover, we listed the accompanying evaluation metrics in Table 5.47 - Table 5.50. To support the feature analysis for the most important characteristics during the classification procedure, we plotted the related feature coefficients of the LR classifier in Figure 5.19 , and the constructed decision tree of the DT classifier in Figure 5.20. As a side note, we want to mention that we used *n_estimators: 15* and *learning_rate: 1* as the respective hyperparameters for AdaBoost, and the default *scikit-learn* parameters for the Soft-Voting classifier - corresponding to a uniformly weighting among the individual classifiers.

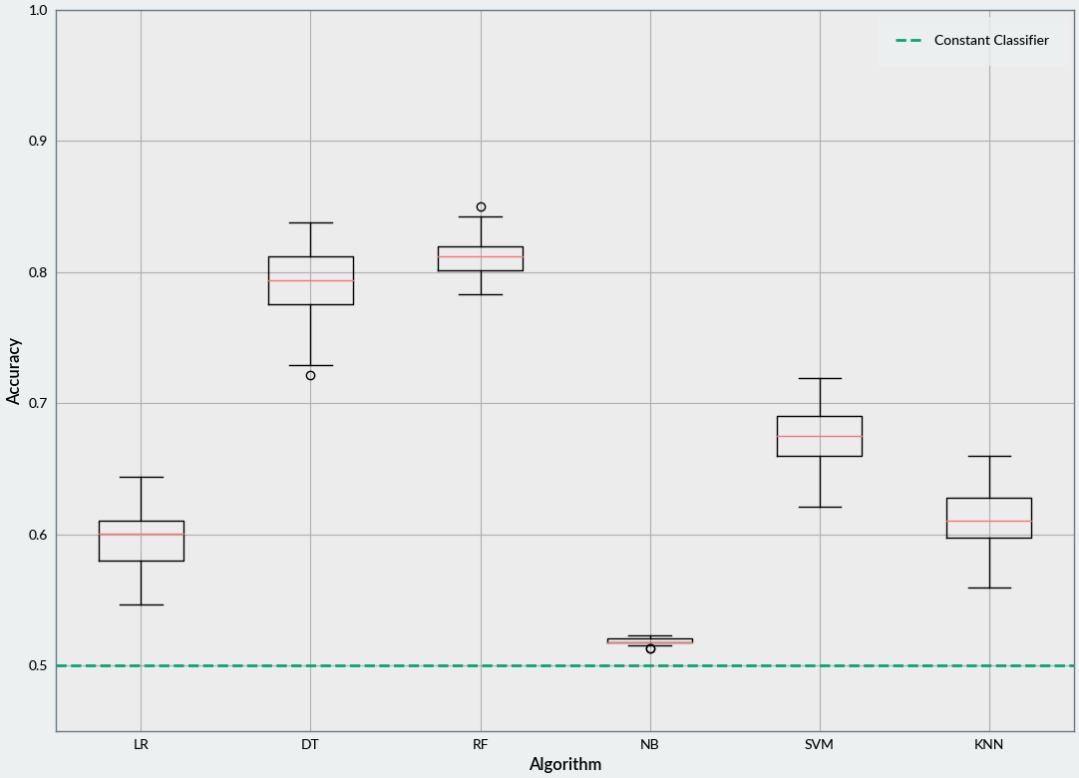


Figure 5.11: **Boxplots of all accuracy score evaluations in *Dataset B* (ROS)** for each used classifier on the training set. Each boxplot contains the 100 average accuracy scores during the repeated training evaluations by LOO CV. On the x-axis, we reported the used classifiers whereby we listed the respective accuracy scores on the y-axis. Moreover, we used a constant classifier (green dashed line) as the baseline model.

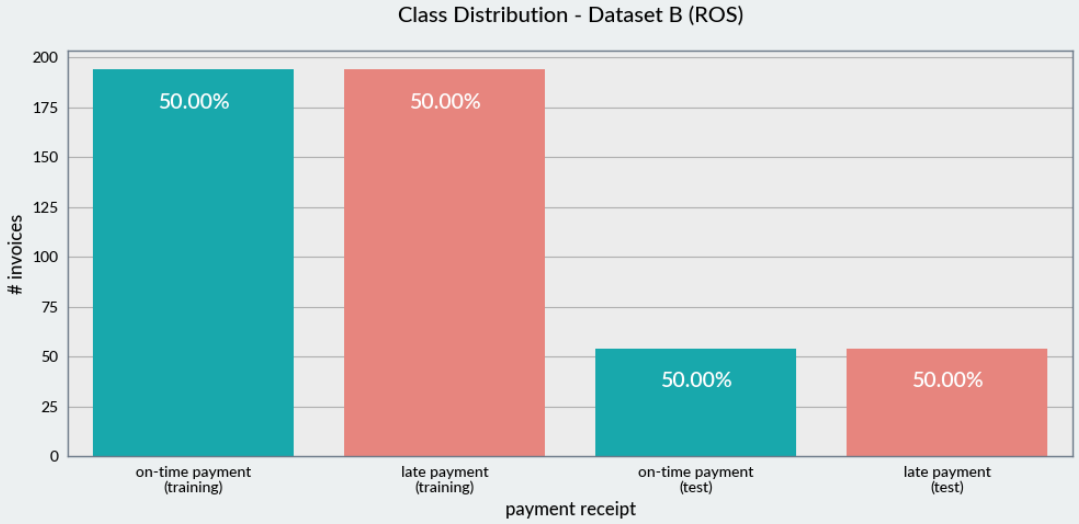


Figure 5.12: **Oversampled training and test set in *Dataset B* (ROS)** that we used for the final model evaluations; illustrating the achieved class balance of 50% in both subsets. The data split on the original dataset was performed on a ratio of 80% (training set) to 20% (test set).

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 54 | 0 | *late* | 0.50 | 1.00 | 0.67 | – |
| *on-time* | 54 | 0 | *on-time* | 0.00 | 0.00 | 0.00 | – |
| **Accuracy: 0.50** | | | **avg/total** | **0.25** | **0.50** | **0.33** | **0.50** |

Table 5.26: **Evaluation results of the Constant classifier - *Dataset B* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 40 | 14 | *late* | 0.56 | 0.74 | 0.64 | – |
| *on-time* | 31 | 23 | *on-time* | 0.62 | 0.43 | 0.51 | – |
| **Accuracy: 0.58** | | | **avg/total** | **0.59** | **0.58** | **0.57** | **0.70** |

Table 5.27: **Evaluation results of the LR classifier - *Dataset B* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 18 | 36 | *late* | 0.50 | 0.33 | 0.40 | – |
| *on-time* | 18 | 36 | *on-time* | 0.50 | 0.67 | 0.57 | – |
| **Accuracy: 0.50** | | | **avg/total** | **0.50** | **0.50** | **0.49** | **0.51** |

Table 5.28: **Evaluation results of the DT classifier - *Dataset B* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 25 | 29 | *late* | 0.58 | 0.46 | 0.52 | – |
| *on-time* | 18 | 36 | *on-time* | 0.55 | 0.67 | 0.61 | – |
| **Accuracy: 0.56** | | | **avg/total** | **0.57** | **0.56** | **0.56** | **0.55** |

Table 5.29: **Evaluation results of the RF classifier - *Dataset B* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 54 | 0 | *late* | 0.52 | 1.00 | 0.68 | – |
| *on-time* | 50 | 4 | *on-time* | 1.00 | 0.07 | 0.14 | – |
| **Accuracy: 0.54** | | | **avg/total** | **0.76** | **0.54** | **0.41** | – |

Table 5.30: **Evaluation results of the NB classifier - *Dataset B* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 36 | 18 | *late* | 0.61 | 0.67 | 0.64 | – |
| *on-time* | 23 | 31 | *on-time* | 0.63 | 0.57 | 0.60 | – |
| **Accuracy: 0.62** | | | **avg/total** | **0.62** | **0.62** | **0.62** | **0.47** |

Table 5.31: **Evaluation results of the KNN classifier - *Dataset B* (ROS)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| ***late*** | 29 | 25 | ***late*** | 0.59 | 0.54 | 0.56 | – |
| ***on-time*** | 20 | 34 | ***on-time*** | 0.58 | 0.63 | 0.60 | – |
| **Accuracy: 0.58** | | | **avg/total** | **0.58** | **0.58** | **0.58** | **0.47** |

Table 5.32: **Evaluation results of the SVM classifier - *Dataset B* (ROS)**.

We will present the accompanying discussions on these results in Chapter 6. Next, we reported the results of *Dataset B* using the SMOTE-based sampling method where we again used the oversampled test set to evaluate our classifiers on previously unseen data.

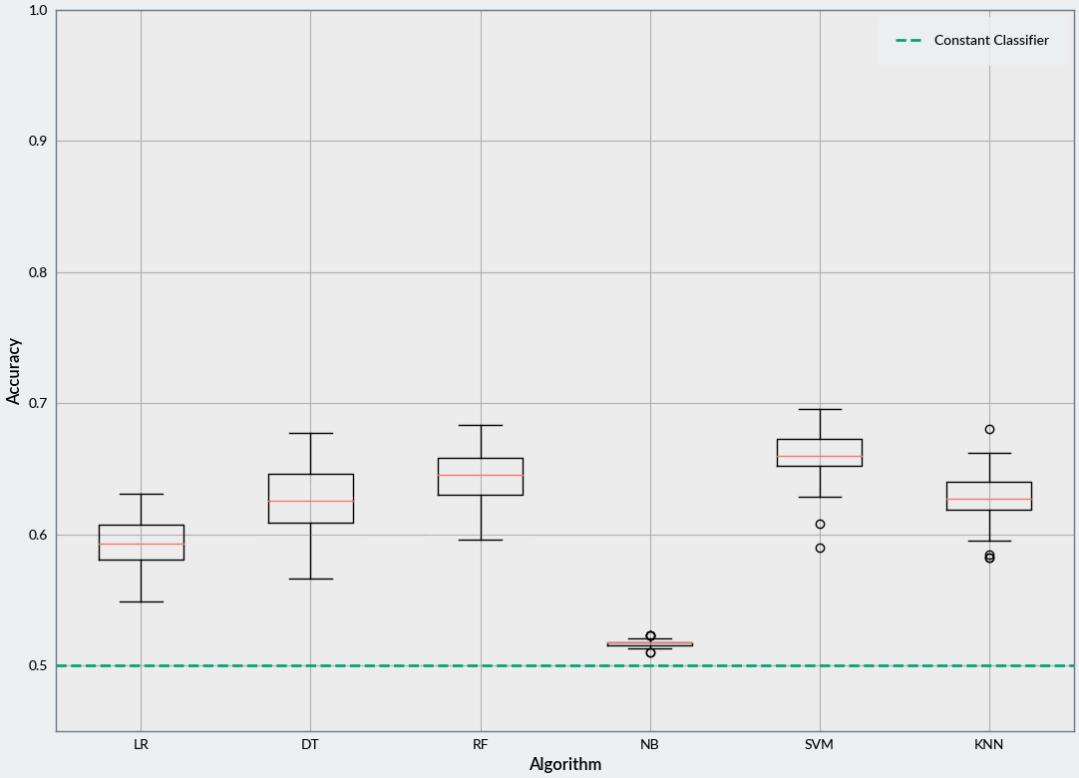


Figure 5.13: **Boxplots of all accuracy score evaluations in *Dataset B* (SMOTE)** for each used classifier on the training set. Each boxplot contains the 100 average accuracy scores during the repeated training evaluations by LOO CV. On the x-axis, we reported the used classifiers whereby we listed the respective accuracy scores on the y-axis. Moreover, we used a constant classifier (green dashed line) as the baseline model.

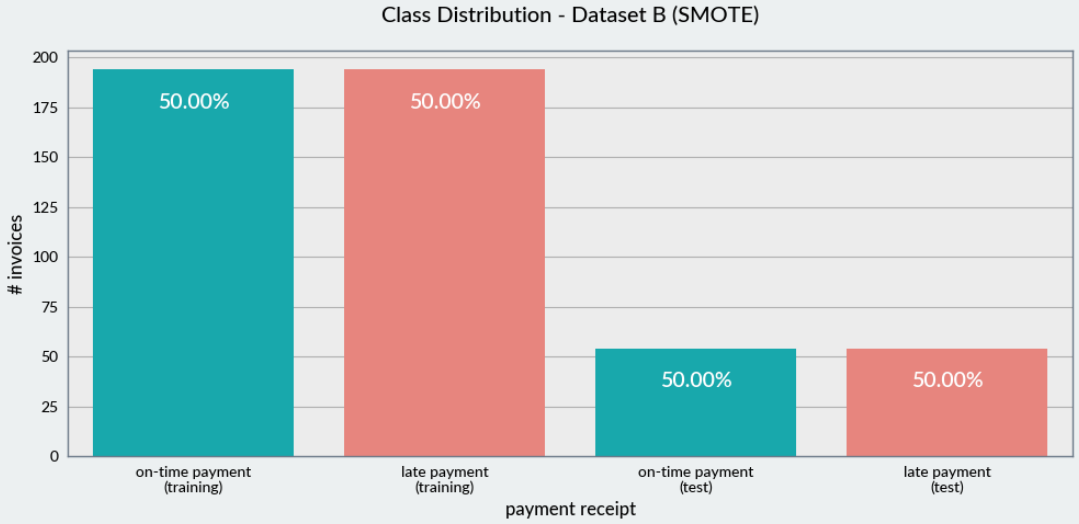


Figure 5.14: **Oversampled training and test set in *Dataset B* (SMOTE)** that we used for the final model evaluations; illustrating the achieved class balance of 50% in both subsets. The data split on the original dataset was performed on a ratio of 80% (training set) to 20% (test set).

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 54 | 0 | *late* | 0.50 | 1.00 | 0.67 | – |
| *on-time* | 54 | 0 | *on-time* | 0.00 | 0.00 | 0.00 | – |
| **Accuracy: 0.50** | | | **avg/total** | **0.25** | **0.50** | **0.33** | **0.50** |

Table 5.33: **Evaluation results of the Constant classifier - *Dataset B* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 43 | 11 | *late* | 0.66 | 0.80 | 0.72 | – |
| *on-time* | 22 | 32 | *on-time* | 0.74 | 0.59 | 0.66 | – |
| **Accuracy: 0.69** | | | **avg/total** | **0.70** | **0.69** | **0.69** | **0.75** |

Table 5.34: **Evaluation results of the LR classifier - *Dataset B* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 26 | 28 | *late* | 0.60 | 0.48 | 0.54 | – |
| *on-time* | 17 | 37 | *on-time* | 0.57 | 0.69 | 0.62 | – |
| **Accuracy: 0.58** | | | **avg/total** | **0.59** | **0.58** | **0.58** | **0.54** |

Table 5.35: **Evaluation results of the DT classifier - *Dataset B* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 29 | 25 | *late* | 0.62 | 0.54 | 0.57 | – |
| *on-time* | 18 | 36 | *on-time* | 0.59 | 0.67 | 0.63 | – |
| **Accuracy: 0.60** | | | **avg/total** | **0.60** | **0.60** | **0.60** | **0.59** |

Table 5.36: **Evaluation results of the RF classifier - *Dataset B* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 54 | 0 | *late* | 0.52 | 1.00 | 0.68 | – |
| *on-time* | 50 | 4 | *on-time* | 1.00 | 0.07 | 0.14 | – |
| **Accuracy: 0.54** | | | **avg/total** | **0.76** | **0.54** | **0.41** | **–** |

Table 5.37: **Evaluation results of the NB classifier - *Dataset B* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 40 | 14 | *late* | 0.61 | 0.74 | 0.67 | – |
| *on-time* | 26 | 28 | *on-time* | 0.67 | 0.52 | 0.58 | – |
| **Accuracy: 0.63** | | | **avg/total** | **0.64** | **0.63** | **0.62** | **0.56** |

Table 5.38: **Evaluation results of the KNN classifier - *Dataset B* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| ***late*** | 36 | 18 | ***late*** | 0.65 | 0.67 | 0.66 | – |
| ***on-time*** | 19 | 35 | ***on-time*** | 0.66 | 0.65 | 0.65 | – |
| **Accuracy: 0.66** | | | **avg/total** | **0.66** | **0.66** | **0.66** | **0.58** |

Table 5.39: **Evaluation results of the SVM classifier - *Dataset B* (SMOTE)**.

We will present the accompanying discussions on these results in Chapter 6. Next, we reported the results of *Dataset B* using the SMOTE + ENN-based sampling method where we again used the oversampled test set to evaluate our classifiers on previously unseen data.

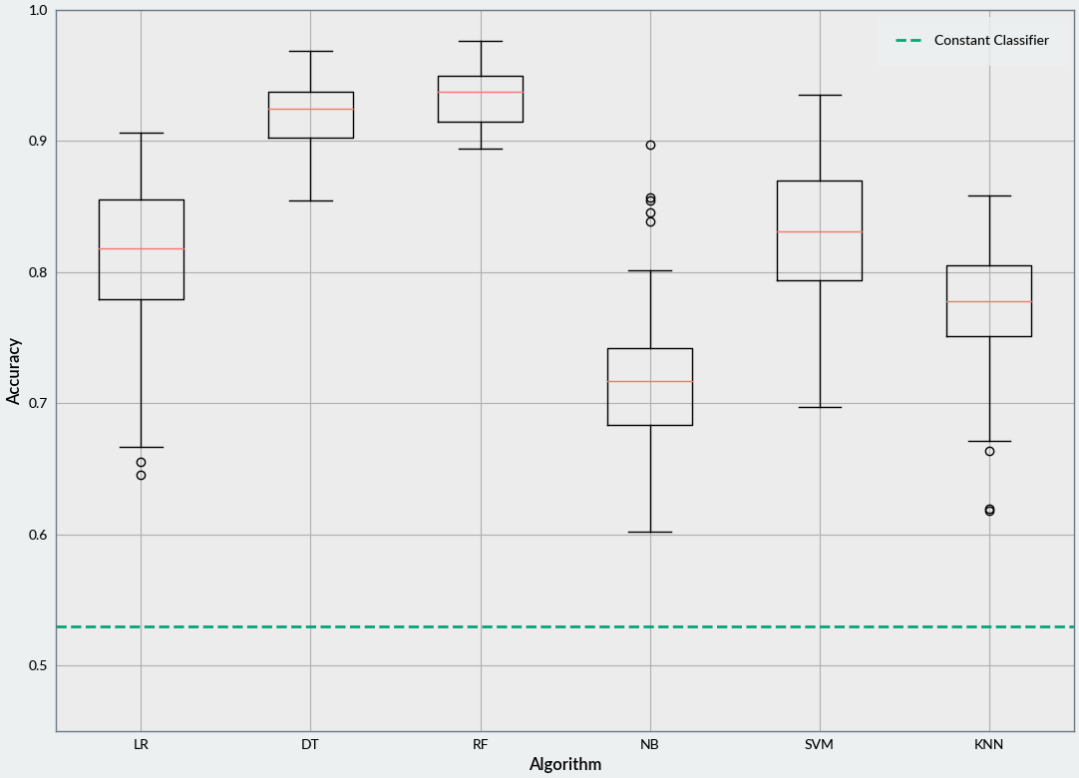


Figure 5.15: **Boxplots of all accuracy score evaluations in *Dataset B* (SMOTE + ENN)** for each used classifier on the training set. Each boxplot contains the 100 average accuracy scores during the repeated training evaluations by LOO CV. On the x-axis, we reported the used classifiers whereby we listed the respective accuracy scores on the y-axis. Moreover, we used a constant classifier (green dashed line) as the baseline model.

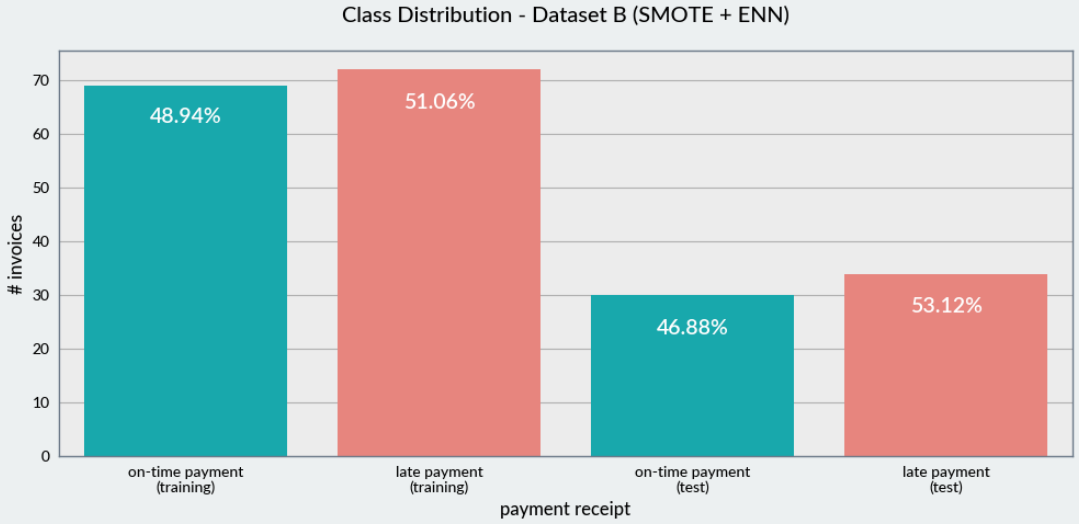


Figure 5.16: **Oversampled training and test set in *Dataset B* (SMOTE + ENN)** that we used for the final model evaluations; illustrating the skewed class balance which was caused by ENN. The data split on the original dataset was performed on a ratio of 80% (training set) to 20% (test set).

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 34 | 0 | *late* | 0.53 | 1.00 | 0.69 | – |
| *on-time* | 30 | 0 | *on-time* | 0.00 | 0.00 | 0.00 | – |
| **Accuracy: 0.53** | | | **avg/total** | **0.28** | **0.53** | **0.37** | **0.50** |

Table 5.40: **Evaluation results of the Constant classifier - *Dataset B* (SMOTE + ENN)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 27 | 7 | *late* | 0.75 | 0.79 | 0.77 | – |
| *on-time* | 9 | 21 | *on-time* | 0.75 | 0.70 | 0.72 | – |
| **Accuracy: 0.75** | | | **avg/total** | **0.75** | **0.75** | **0.75** | **0.90** |

Table 5.41: **Evaluation results of the LR classifier - *Dataset B* (SMOTE + ENN)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 21 | 13 | *late* | 0.68 | 0.62 | 0.65 | – |
| *on-time* | 10 | 20 | *on-time* | 0.61 | 0.67 | 0.63 | – |
| **Accuracy: 0.64** | | | **avg/total** | **0.64** | **0.64** | **0.64** | **0.64** |

Table 5.42: **Evaluation results of the DT classifier - *Dataset B* (SMOTE + ENN)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 28 | 6 | *late* | 0.68 | 0.82 | 0.75 | – |
| *on-time* | 13 | 17 | *on-time* | 0.74 | 0.57 | 0.64 | – |
| **Accuracy: 0.70** | | | **avg/total** | **0.71** | **0.70** | **0.70** | **0.77** |

Table 5.43: **Evaluation results of the RF classifier - *Dataset B* (SMOTE + ENN)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 29 | 5 | *late* | 0.81 | 0.85 | 0.83 | – |
| *on-time* | 7 | 23 | *on-time* | 0.82 | 0.77 | 0.79 | – |
| **Accuracy: 0.81** | | | **avg/total** | **0.81** | **0.81** | **0.81** | – |

Table 5.44: **Evaluation results of the NB classifier - *Dataset B* (SMOTE + ENN)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 27 | 5 | *late* | 0.73 | 0.84 | 0.78 | – |
| *on-time* | 10 | 20 | *on-time* | 0.80 | 0.67 | 0.73 | – |
| **Accuracy: 0.76** | | | **avg/total** | **0.76** | **0.76** | **0.76** | **0.82** |

Table 5.45: **Evaluation results of the KNN classifier - *Dataset B* (SMOTE + ENN)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| ***late*** | 30 | 2 | ***late*** | 0.73 | 0.94 | 0.82 | – |
| ***on-time*** | 11 | 19 | ***on-time*** | 0.90 | 0.63 | 0.75 | – |
| **Accuracy: 0.79** | | | **avg/total** | **0.82** | **0.79** | **0.78** | **0.72** |

Table 5.46: **Evaluation results of the SVM classifier - *Dataset B* (SMOTE + ENN)**.

We will present the accompanying discussions on these results in Chapter 6. Next, we reported the results of the identified top three classifiers, and the four applied ensemble strategies on *Dataset B*, using the SMOTE-based oversampling method.

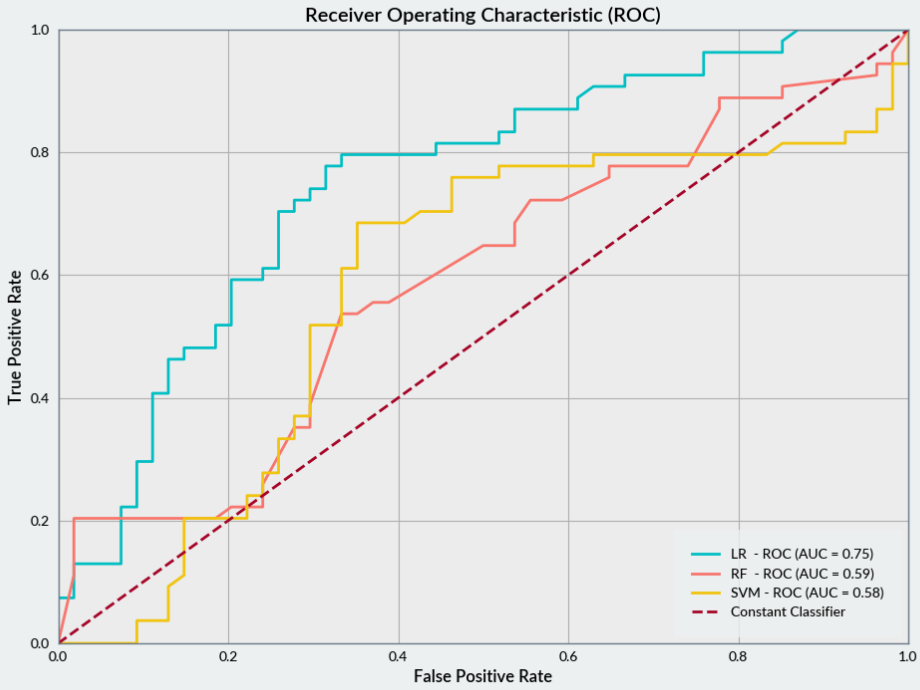


Figure 5.17: **ROC evaluations of the top three classifiers - *Dataset B* (SMOTE)**, whereby we reported the ROC-Curve and accompanying AUC measurements for LR, RF, and SVM to enable a supportive comparison.

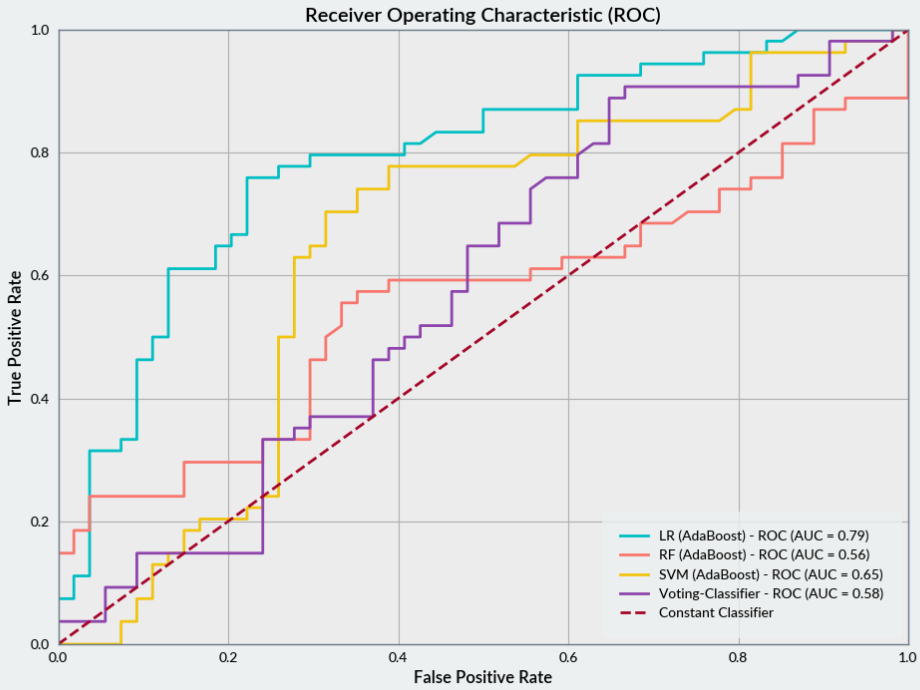


Figure 5.18: **ROC evaluations of the top three ensemble classifiers - *Dataset B* (SMOTE)**, whereby we reported the ROC-Curve and accompanying AUC measurements of AdaBoost LR, AdaBoost RF, AdaBoost SVM, and the Soft-Voting classifier (LR, RF, SVM) to enable a supportive comparison among the ensemble strategies.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 51 | 3 | *late* | 0.55 | 0.94 | 0.70 | – |
| *on-time* | 41 | 13 | *on-time* | 0.81 | 0.24 | 0.37 | – |
| **Accuracy: 0.59** | | | **avg/total** | **0.68** | **0.59** | **0.54** | **0.79** |

Table 5.47: **Evaluation results of the AdaBoost LR classifier - *Dataset B* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 29 | 25 | *late* | 0.62 | 0.54 | 0.57 | – |
| *on-time* | 18 | 36 | *on-time* | 0.59 | 0.67 | 0.63 | – |
| **Accuracy: 0.60** | | | **avg/total** | **0.60** | **0.60** | **0.60** | **0.56** |

Table 5.48: **Evaluation results of the AdaBoost RF classifier - *Dataset B* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 42 | 12 | *late* | 0.64 | 0.78 | 0.70 | – |
| *on-time* | 24 | 30 | *on-time* | 0.71 | 0.56 | 0.63 | – |
| **Accuracy: 0.67** | | | **avg/total** | **0.68** | **0.67** | **0.66** | **0.65** |

Table 5.49: **Evaluation results of the AdaBoost SVM classifier - *Dataset B* (SMOTE)**.

| Confusion Matrix | | | Evaluation Metrics | | | | ROC |
|---|---|---|---|---|---|---|---|
| Ground-Truth/Predicted | *late* | *on-time* | | Precision | Recall | F1-Score | AUC |
| *late* | 23 | 31 | *late* | 0.53 | 0.43 | 0.47 | – |
| *on-time* | 20 | 34 | *on-time* | 0.52 | 0.63 | 0.57 | – |
| **Accuracy: 0.53** | | | **avg/total** | **0.53** | **0.53** | **0.52** | **0.58** |

Table 5.50: **Evaluation results of the Soft-Voting classifier - *Dataset B* (SMOTE)**.

Finally, we presented the constructed decision tree of the DT classifier and the feature coefficients of the LR classifier; further used to analyze the respective feature importance.

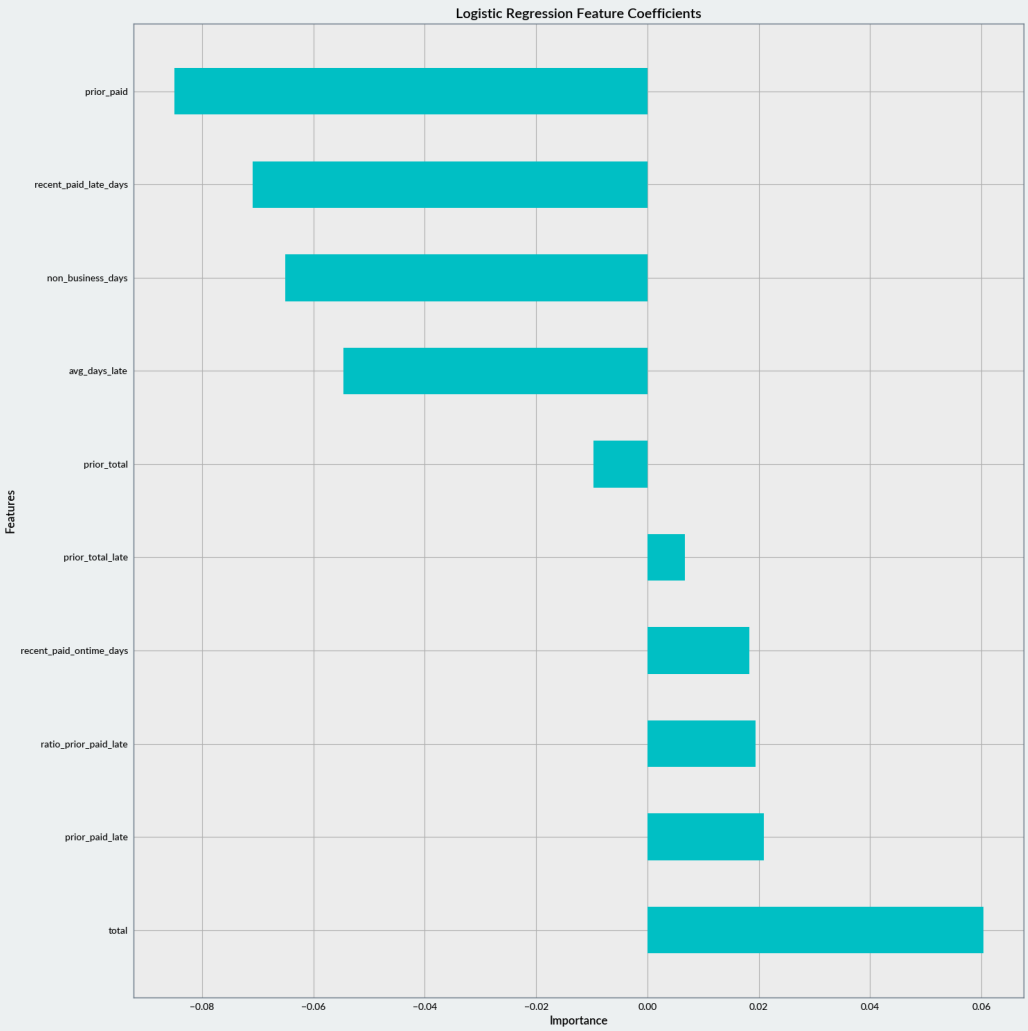


Figure 5.19: **Feature coefficients used by the LR classifier -** ***Dataset B*** **(SMOTE)** to classify the invoices into on-time and late payments. On the y-axis, we reported the categorical and numerical features, and on the x-axis, we respectively reported their feature importance. The feature importance represents thereby the calculated coefficient values in the LR classifier which lie in the range between -1 and +1. The closer a feature's coefficient value is to -1, the more important is the feature for classifying an invoice as a late payment. Contrary, the closer a feature's coefficient value is to +1, the more important is the feature for an on-time payment classification. This implies for the trained LR classifier in *Dataset B*, that the features *prior_paid* and *recent_paid_late_days* held the essential characteristics to predict a late payment classification, while the features *total*, and *prior_paid_late* were considered as the most important ones for an on-time payment classification.

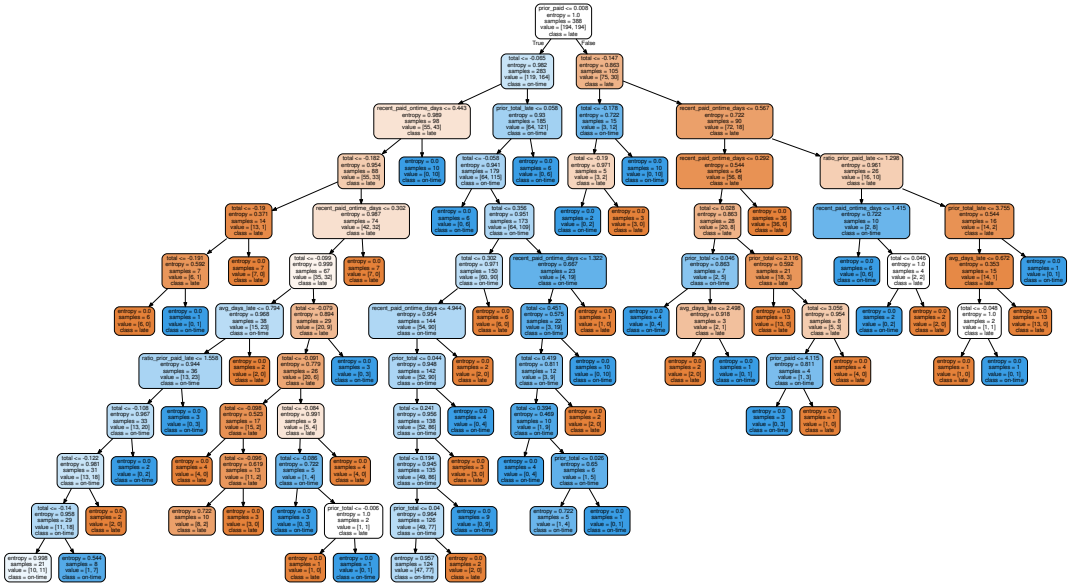

Figure 5.20: **Constructed decision tree by the DT classifier - *Dataset B* (SMOTE)** to classify the invoices into on-time and late payments. The most characteristic features are reported at the top of the tree and decrease by their importance while descending the tree structure. The invoice features *prior_paid*, *total*, and *recent_paid_ontime_days* were consequently considered as the most important features for the DT to classify an invoice into a late or on-time payment. We trained the constructed DT classifier for *Dataset B* with the optimized hyperparameters on the oversampled training set and evaluated it on the oversampled test set. The best AUC corresponded thereby to 54%. The blue nodes in the presented decision tree represent an on-time payment classification while the orange nodes represent a late payment classification. The color saturation of the individual nodes represents the respective class distribution, whereby white nodes denote a uniform distribution.

# Chapter 6

# Discussion

In this chapter, we will discuss the presented classification results from Chapter 5 and further try to answer the stated research questions in this thesis. To do this, we first review and discuss the different sampling methods by comparing the results from the training evaluations with the ones from the test evaluations. Next, we will discuss the results of our individual classifiers to identify the most suited ML model for a proper invoice payment classification. Followed by the discussion on our constructed ensemble strategies which we build upon the most promising classifiers. Subsequent, we will review the feature analysis to determine the essential features for a successful classification. Finally, we will summarize our main findings on the conducted experiments and thereby try to answer the stated research questions in this thesis.

## 6.1 Sampling Methods

Throughout our experiments, we evaluated three sampling methods to overcome the problem of class imbalance in our datasets, namely: ROS, SMOTE, and SMOTE + ENN. Comparing the classifier evaluations from the training set with those from the test set, allowed us to identify the most appropriate oversampling method for our classification problem. Thereby, we discovered that some sampling methods tended to raise some difficulties concerning an overfitting on the training data, or unrealistic performance evaluations on a reduced subset of invoices. To judge which sampling method was most suited, we compared each model's average accuracy scores from the classifier training with the respective accuracy score of the test evaluations. Note that the boxplot evaluations from the previously presented training procedure hold thereby the 100 average accuracy scores which were evaluated by 10-fold CV or respectively LOO CV on the oversampled training set. To graphically support this comparison, we collected in Figure 6.1 all evaluated boxplots of the different sampling methods and manually marked the accuracy evaluations of the test set to identify possible issues more intuitively.

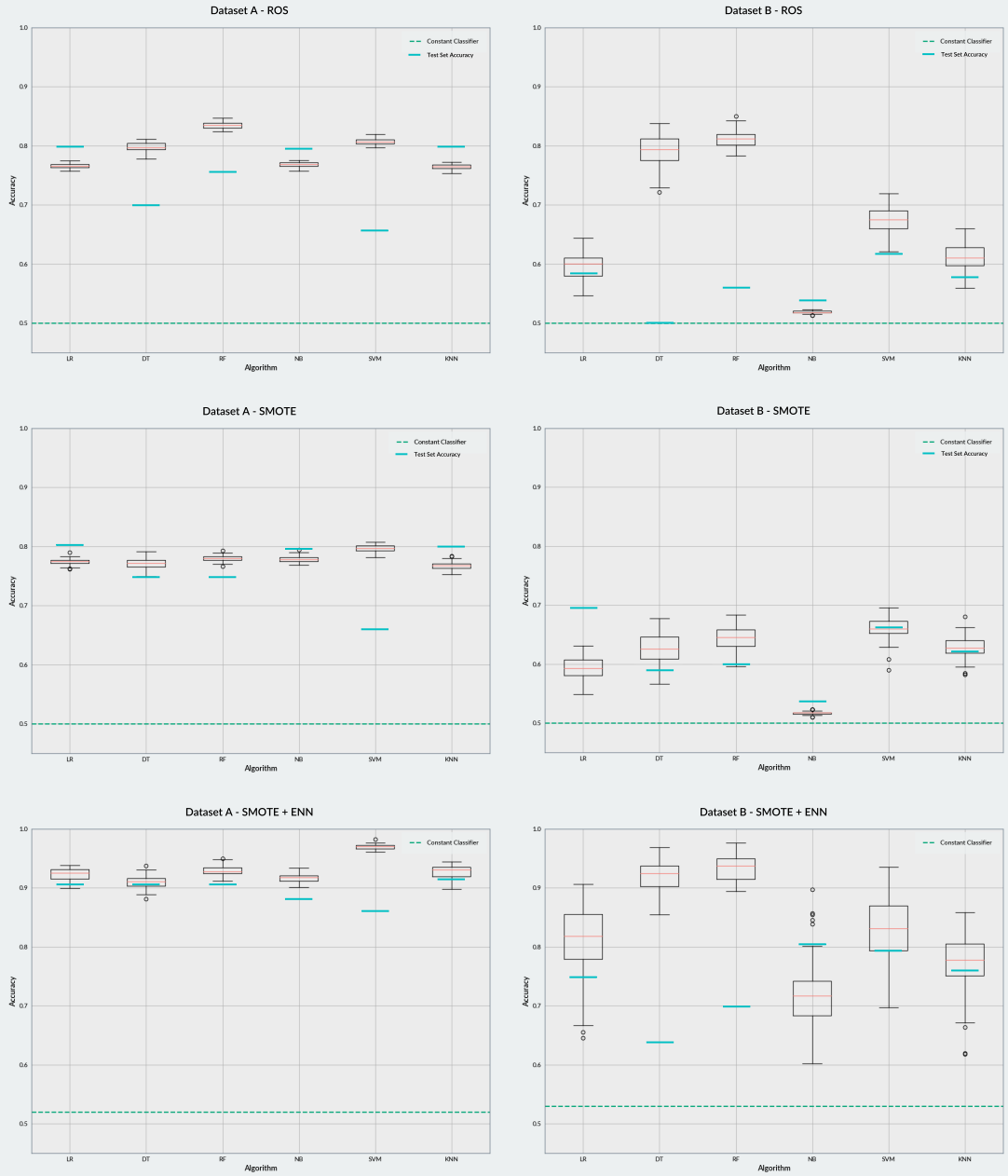


Figure 6.1: **Comparison of training and test accuracy scores for *Dataset A* and *Dataset B***. We reported the individual classifier boxplot evaluations containing 100 accuracy score from the *k*-fold CV training process for each of the used sampling methods and datasets. From top to bottom we listed the sampling methods for ROS, SMOTE, and SMOTE + ENN. On the left-hand side, we always reported the boxplots for *Dataset A* while on the right-hand side, we respectively reported the boxplots for *Dataset B*. Additionally, we manually marked for each of the individual models the respective accuracy evaluation of the test set (blue lines).

Applying **ROS** on ***Dataset A***, allowed us to observe that the DT, RF, and SVM classifiers were strongly overfitting on the training data. We could perceive this behavior while comparing the accuracy boxplots in Figure 5.1 with the respective evaluation results in

Table 5.3 (DT), Table 5.4 (RF), and Table 5.7 (SVM); or top left plot in Figure 6.1. Thereby, we saw that the reported training accuracy scores of the RF classifier (on average around 0.84) were much higher than the reported accuracy on the final test set (0.76). Consequently, we were able to argue that the corresponding classifier was overfitting on the training data. Similarly, for the DT and SVM classifiers, which reported an accuracy score of 0.70 and 0.66 on the test set and a much higher one during the training evaluations (on average around 0.80). Contrary to the LR, NB and KNN classifiers which showed a much more stable behavior considering the respectively reported accuracy score evaluations, F1-Score, and AUC evaluations on the test set.

The results of ***Dataset B*** showed similar outcomes while applying **ROS**. Again, the DT and RF classifier were overfitting on the training data while lower accuracy scores got reported on the final test set than on the training set (see Table 5.28 and 5.29; or top right plot in Figure 6.1). The LR, NB, SVM and KNN classifiers were performing quite stable while their accuracy showed similar results on the training and test set. However, all used classifiers reported a rather poor performance compared to *Dataset A*. Especially the NB model was reporting a very low performance with an F1-Score of 0.41 and an accuracy of 0.54 (see Table 5.30).

These observations of performance loss were however already expected since *Dataset B* contained only very few invoice samples with a very low number of individual invoice records for each customer (as revealed during the initial dataset review in Section 4.2). Consequently, the NB classifier was not able to calculate the respective late payment probabilities as precisely as in *Dataset A* (which guaranteed a minimum number of 15 invoices for each customer). The overfitting observations of the DT and RF classifiers in *Dataset A* and *Dataset B* were thereby also somehow expected due to the invoice duplication process within ROS. This particular invoice duplication (without modifying any feature values), resulted likely in a classifier overfitting due to possible k-fold CV on same samples as used in the training process. Nevertheless, we initially expected the RF classifier to compensate this behavior since it trains the classifier with multiple invoice subsets which get randomly drawn from the original training set. Further investigations revealed however that the used *scikit-learn* implementation constructs these subsets of the same size as the original training set. Consequently, identical samples are still very likely to be contained within these training subsets (mainly due to the small size of *Dataset B*), which further explained our observed behavior. Note that the performance of LR, SVM and KNN was not influenced in *Dataset B* since the classifiers were able to compensate the sample duplication in ROS with the help of their hyperparameters (e.g. higher number of considered nearest neighbors in KNN - see Table 4.11). The only unexpected results which we observed during these evaluations with ROS was the poor test performance of the SVM classifier on *Dataset A*. Thereby we assumed that the underlying training data must have contained some structures which were not represented enough during the training process. Indeed, as we compared the invoices within the test and training set, we noticed a considerable lack of invoices with a high number of past payments in the test set. This means that we observed in the test set mostly invoices which held on average only two previously issued invoices or invoices from new customers, whereby the training set

consisted mostly of invoices from already established customers which held on average around six previously issued invoices. Consequently, also other related invoices features as *prior_total_late* or *recent_paid_late_days* got influenced, which is thereby likely the reason for the overfitting of the SVM classifier in *Dataset A*.

As a conclusion, we argued that even though ROS solved the problem of class imbalance, we could not consider it as the most appropriate method for our classification task due to the related overfitting problems with invoice duplicates.

By using the **SMOTE**-based sampling method, we were able to overcome this specific problem of overfitting due to its use of synthetic sample generation as described in Section 3.4. On ***Dataset A***, we observed that it almost resulted in the same performance as with ROS while we compared the training accuracy scores in Figure 5.1 (ROS) and 5.3 (SMOTE). However, this time the individual classifiers did not overfit anymore (except for the SVM classifier). As we compared the average accuracy scores of the DT and RF classifiers on the training set with the ones on the test set, we only observed a difference of about 0.05 which we considered as proper behavior. We were further able to observe similar results for the LR, NB, and KNN classifiers, which was already an indicator for the preferable use of SMOTE as the final sampling method. Besides that, we can see in Figure 5.4 that SMOTE was also able to solve the class imbalance problems within our used datasets. The only problem which we still encountered was the poor performance result of the SVM classifier on the test set. Even though we optimized its hyperparameters, we again received the unexpected result of an overfitting behavior on the training data. As we can see in Table 5.14, the reported F1-Score of 0.65 and the accuracy of 0.66 were much lower as the reported results in the training evaluations (average accuracy of 0.79). Overall we would have expected that the SVM classifier performs on approximately the same level as the other proposed ML models. However, as we observed the same situation already within ROS, we can assume that we mainly note this loss of performance due to the different data structure within the training and test set.

Applying SMOTE on ***Dataset B*** brought similar results. Same as in *Dataset A*, the boxplot evaluations of the accuracy score were comparable to the one from ROS whereby LR, NB, SVM, and KNN showed a stable performance and were no more overfitting on the test set. Especially the LR classifier showed thereby a performance boost with an accuracy of 0.69, which increased compared to the training performance with ROS by 0.11. Furthermore, the DT and RF classifiers were able to perform quite stable and were not overfitting anymore. However, we noticed that the variance of the individual accuracy score evaluations on the training set was much higher than in *Dataset A*, which is likely caused by the minimal number of invoice records per customer. We could observe this behavior for almost all classifiers by looking at the individual boxplot evaluations of the training set. Most of these illustrated boxplots hold thereby larger quartiles and whiskers which consequently indicate a higher standard deviation considering the 100 accuracy score evaluations with k-fold CV (see Figure 5.13). Another interesting observation was that the SVM model did not have any overfitting problems concerning the training data as in *Dataset A*. The reported average accuracy score on the training data was around

0.67 and the respective test accuracy was 0.66. Additional investigations revealed that the individual invoices within the training and test set were thereby very similar which explains why the SVM classifier was not overfitting while using the invoices from *Dataset B*.

We concluded that SMOTE enabled us to overcome the class imbalance problem while at the same time the creation of synthetic samples also enhanced the model training process and raised no further overfitting problems on the training data.

Last but not least we reviewed the results of SMOTE + ENN on *Dataset A* and *Dataset B*. It is important to mention that the use of *ENN* did not result in an exactly balanced dataset since it removed various samples after the oversampling process with SMOTE (see Figure 5.6 and 5.16). At first glance, it seemed like this method outperformed the prior sampling methods concerning the high evaluation results on the training and test sets. However, even though SMOTE + ENN did not cause any overfitting problems on the training set (except for DT and RF in *Dataset B* due to the discussed lack of samples), a major problem during the "outlier" detection could be recognized in both datasets. We observed that a rather large amount of invoices got detected as outliers and consequently removed by the ENN algorithm. This process removed all invoices which were somehow difficult to classify regarding their payment outcome and lead thereby to the observed boost in performance for the remaining invoices. As discussed in Section 3.4, SMOTE + ENN tends to remove many samples by undersampling the dataset with the help of a *k*-nearest neighbor strategy. The alternative use of *"Tomek links"* would thereby likely have been a better choice to detect outliers as it is known to perform more stable while not removing as many samples as ENN. Such experiments were however left for further work.

In the end, we can conclude for SMOTE + ENN, that it was not suited for our classification problems even though the outlier removing process seemed useful during the experimental setup. It turned out that ENN was removing way too many samples and thereby artificially increasing the performance of the individual classifiers. However, we want to mention at this point that our manual outlier detection process on *Dataset B* (see Section 4.2), likely influenced the ENN algorithm and further also the evaluation results of SMOTE + ENN. Because we already removed most of the outliers manually, ENN was likely not the best approach to choose while it could have lead to more promising evaluations when we would have worked with the original dataset. Nevertheless, for the Datset A such an approach would not have been applicable since it was already provided as filtered and cleaned dataset by EmcienScan.

As an overall conclusion on the choice of the most appropriate oversampling strategy, we can say that even though ROS solved the problem of data imbalance, it was not a suitable approach due to overfitting problems on the training data. Neither was the use of SMOTE + ENN due to the vast removal process of invoice "outliers" which caused an artificially increased performance during training and testing of the individual classifiers. Consequently, we decided to stick with the SMOTE-based oversampling method. It provided proper evaluation performances and was not overfitting on the training set. Furthermore, SMOTE guaranteed a balanced dataset setup which further allowed us do

provide comparable evaluations results regarding future work and improvements.

## 6.2 Classifier Comparison

Once we revealed that the most appropriate oversampling method for our classification task was SMOTE, we evaluated and compared the top three classifiers of our two datasets against each other to determine the best classifier for our research objectives. To discover these top three classifiers, we compared the training evaluations (Figure 5.3 for *Dataset A*, and Figure 5.3 for *Dataset B*), with the corresponding performances on the test set (Table 5.8 - Table 5.14 for *Dataset A*, and Table 5.33 - Table 5.39 for *Dataset B*). Consequently, we tried to identify the best combination of classifiers by evaluating a proper tradeoff between training and test set performances.

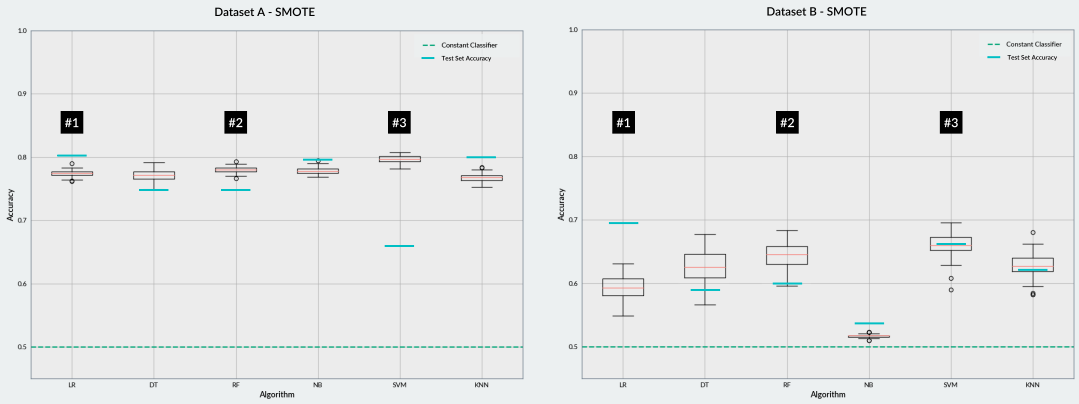


Figure 6.2: **Selection of the top three classifiers for *Dataset A* and *Dataset B***. On the left-hand side, we reported the accuracy boxplots of the 100 training iterations on *Dataset A*, and on the right-hand side, we respective reported the boxplot evaluations on *Dataset B*. To visually back the performance differences on the training and test set, we additionally marked each classifiers test accuracy in the related boxplot (blue lines). Finally, by comparing the evaluations on the training and test set, we were able to identify our top three classifiers which we used for the final evaluations (marked by black boxes #1, #2, #3).

To support the understanding of how we picked these three models, we present in Figure 6.2 an overview of the training and test accuracy scores on *Dataset A* and *Dataset B*. First of all, we picked the LR classifier due to its outstanding performance on both Datasets. Concerning the DT and RF classifiers, we observed that both models performed somewhat similarly while the DT was performing slightly worst than RF, which is why we decided to pick RF as the second model. Regarding the NB approach, we observed in prior experiments that it was not performing very well in *Dataset B* due to the lack of available data. Consequently, this model was not considered to be part of the selection of the overall best classifiers. Last but not least, we looked at SVM and KNN. Even though KNN performed good on both test evaluations, SVM showed slightly better results in the training and test evaluations. However, we knew that SVM was overfitting on the training data in *Dataset A*. Regardless of this overfitting problem, we decided to stick with the SVM classifier as the third model since we hoped

for improvements when applying ensemble strategies like AdaBoost or the Voting classifier.

To summarize, we selected LR, RF, and SVM as our top three classifiers for the respective task of invoice payment classifications. Since we selected the same models for both datasets, we were further able to properly compare the different results of our two rather diverse datasets. Note that we reported the evaluated ROC and its AUC for each of the selected models in Figure 5.7 for *Dataset A*, and in Figure 5.17 for *Dataset B*.

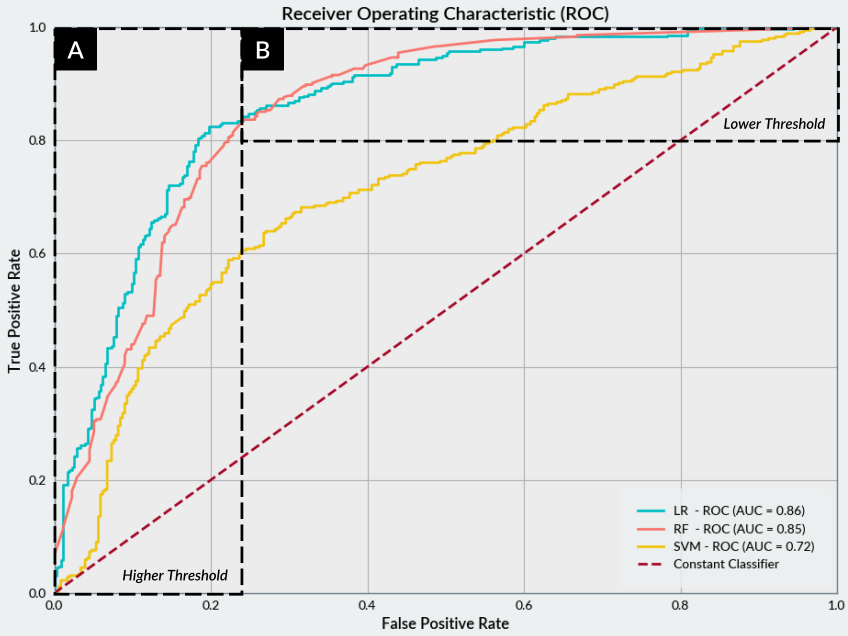


Figure 6.3: **Comparison of LR and RF threshold calibrations**. Note that a threshold of 1.00 means that every classification which is not reaching a prediction probability of 1.00 is marked as on-time payment, while a threshold of 0.00 implies that every sample is considered as a late payment. **A:** We can observe that the LR classifier (blue line) outperforms the RF classifier (red line) in the higher section of thresholds. Setting a high threshold means that the classifier needs high confidence in classifying an invoice as a late payments. **B:** We can observe that the RF classifier (red line) slightly outperforms the LR classifier (blue line) in the lower end of the threshold values. A lower threshold does thereby mean that the classifier does not need such high confidence in classifying an invoice as late payments.

For evaluations on *Dataset A*, we observed again that the LR and the RF classifiers outperformed the SVM classifier whereby both models hold an AUC of 0.86 or respectively 0.85. The LR model is thereby slightly better with higher threshold values, whereby RF performs better with lower ones. For a final classifier application, we could thus base the choice of the right classifier on the pursued target strategy. If we want to have a classifier which performs well on late payment predictions with a high level of confidence, we would have to set a high threshold on the LR classifier. On the other hand, when we want a classifier which performs quite good with lower classification confidences, we would

have to pick a lower threshold on the RF classifier. We provided a graphical illustration of this observation in Figure 6.3. For the SVM classifier, we could observe that it particularly lacked in performance compared to the LR and RF classifiers, which is also reflected by the lower AUC of 0.72 and F1-Score of 0.65 (see Table 5.14). As already explained above, we argued this behavior due to problems within the test and training data in *Dataset A*.

On *Dataset B* we observed that LR outperformed the RF and SVM classifiers which is also indicated by the respective AUC measurements. While LR held an AUC of 0.75, RF and SVM could only provide an AUC of 0.59 or respectively 0.58. We were already expecting such low results for SVM and RF due to the lack of performance in prior test evaluations. However, this evaluation allowed us to compare the individual effects of our models on both datasets. Consequently, we could reveal that in both datasets, independent of their size and underlying data structure, LR performed as the overall best classifier while SVM as the worst one. For RF we identified a huge loss in performance in *Dataset B* while comparing it to *Dataset A*, where LR and RF performed almost equivalent. This lack of performance is therby most likely attributed to the size of *Dataset B* where the number of recorded invoices per customers is much lower than in *Dataset A*.

For a conclusion on our top three classifiers, we can say that LR was overall the most-suited classifier for classifying invoices into on-time and late payments. Even if we reported a higher performance on *Dataset A*, its stability was mostly untouched by the amount of available data and also worked surprisingly good in classifying invoices where we had only a small amount of recorded past invoices. The LR classifier reported thereby an F1-Score of 0.81 and an AUC of 0.86 on *Dataset A*. For *Dataset B* it reported an F1-Score of 0.57 with an AUC of 0.70 whereby the low number of available invoices influenced the corresponding performance results. The performance of the RF and SVM classifiers was thereby however strongly dependent on the underlying dataset. While the RF classifier reported similar performances to LR on the larger *Dataset A*, it notably lacked on the smaller *Dataset B*. Similarly with the SVM classifier which performed quite stable in *Dataset B* but struggled with the diverse training and test data on *Dataset A*.

## 6.3 Ensemble Strategies

As we knew how our single classifiers performed on the individual datasets, we wanted to review also whether we could further boost their performances by the construction of additional ensemble classifiers with AdaBoost and Soft-Voting. The respective ROC evaluation are thereby reported in Figure 5.8 for *Dataset A*, and Figure 5.18 for *Dataset B*. To support the understanding on how boosting influenced our classification results, we further provided an additional illustration in Figure 6.4, where the ROC-Curve for the RF and SVM classifiers got compared with the respectively applied AdaBoost strategy.

The evaluations of the applied Boosting strategy with AdaBoost for *Dataset A* showed that AdaBoost LR and AdaBoost RF were now performing identically. As observable in Figure 6.4, the application of AdaBoost increased the performance for the RF classifier

in the higher range of its threshold values. The Soft-Voting classifier which combined the LR, RF, and SVM classifiers, also performed almost identically with an AUC of 0.86. Contrary, the AdaBoost SVM classifier was still not able to compete with the other boosted classifiers except for the rather high and low end of its threshold values. All evaluations of the accuracy, precision, recall and F1-Score can thereby be found in Table 5.22 - 5.25. These evaluation measurements showed very similar outcomes and further indicated that there is not much of a difference between the use of AdaBoost LR, AdaBoost RF, or the combining Soft-Voting classifier in *Dataset A*. As mentioned, the performance lack with the SVM classifier lies likely in the used test set structure, whereby the use of the AdaBoost ensemble strategy was not able to compensate this behavior.

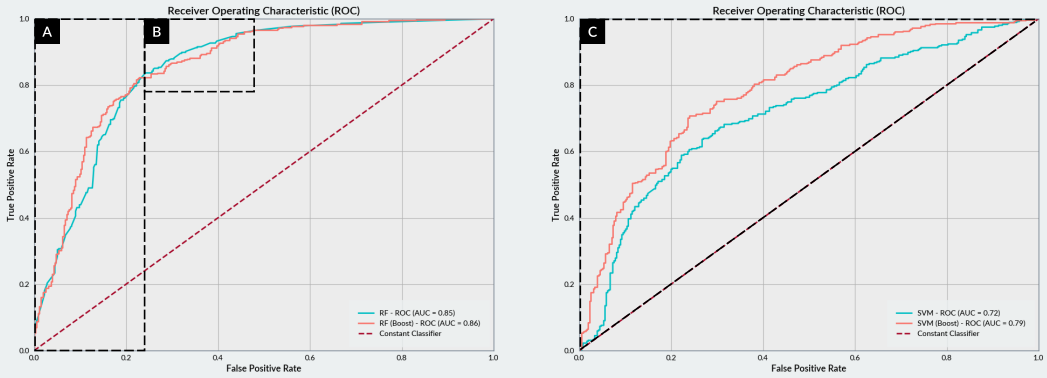


Figure 6.4: **Influence of AdaBoost on the RF and SVM classifiers** on *Dataset A*. On the left-hand side, we reported the ROC for the RF and AdaBoost RF classifier, and on the right-hand side, we reported the Boosting approach with SVM. **A:** We can see that by applying AdaBoost on the RF classifier, we were able to push the performance in the higher threshold ranges which means that the classifier improved especially for late payment predictions with higher confidence. **B:** Even though the application of AdaBoost generally pushes the overall performance of a classifier, it is also possible that it partially decreases which must not necessarily imply a negative effect for the classifier. **C:** For the SVM classifier, we can see that the application of AdaBoost increased the overall performance of the classifier within all threshold values, which is also observable by the higher AUC of 0.79.

For *Dataset B*, the ensemble strategy with AdaBoost pushed the LR classifiers AUC from 0.75 to 0.79, which we already considered as a proper result concerning the rather low amount of available invoices. However, the evaluation metrics reported only an F1-Score of 0.54 (see Table 5.47). This revealed that the Boosting strategy mainly focused on the correct classification of late payments, as further visible from the same table while observing its confusion matrix. The use of this classifier is thereby only valuable for late payment predictions while it performs rather poorly for on-time predictions. Besides that, the evaluation results of AdaBoost RF and AdaBoost SVM did no further impact the classification performance by much. Consequently, we could argue that the few invoices within *Dataset B*, and the particular low number of recorded invoices per customer could not provide enough information for the Boosting strategy to improve the overall performance of the used classifiers.

To conclude our findings on the ensemble strategies, we can say that the application of

AdaBoost is overall very well suited to increase the performance of individual classifiers. Nevertheless, on *Dataset A*, it lead to only minor improvements for the RF and SVM classifiers while the LR classifier was almost not affected at all. Moreover, AdaBoost was also no suitable approach for *Dataset B*, since the dataset did not provide enough invoices to train the classifiers properly (problems with LR). For the Soft-Voting classifier we made similar observations, whereby not enough data was provided by *Dataset B* to efficiently use the RF and SVM classifier in combination with LR. On *Dataset A*, the Soft-Voting classifier was able to combine the strengths of LR, RF, and SVM, nevertheless, it did not outperform the used LR classifier by much.

## 6.4 Feature Importance

Since we identified the LR classifier as the most appropriate model for both of our datasets, we were further able to investigate whether we considered the same feature during the classification of different datasets. To identify the most characteristic features for the LR classifier, we consequently reported the respective feature coefficients on both of our datasets in Figure 5.9 and 5.19. Moreover, we sorted these features according to their reported coefficient value (independent of their sign) so that we could guarantee a proper ranking among the individual features. Consequently representing a ranking among the features which is sorted by their importance for classifying an invoice as on-time or late payment. Even though we did not identify the DT classifier as part of the most valuable classifiers, we further investigated also the list of used classification features of the constructed decision tree, which allowed us to determine possible reasons for the performance loss with the DT classifier. We presented the constructed decision trees in Figure 5.10 and 5.20. To rank the features of the DT in the same order as we did for LR, we reported the features from top to bottom (and left to right), which consequently represented the same importance among the features as in LR.

Next, we compared the list of identified DT features with the extracted list of LR features. We presented the complete list of features (ranked by their importance for each classifier - from top to bottom) for both datasets in Table 6.1.

An important insight which we revealed by the help of this table, is that features differ drastically while considering their importance among different datasets. In the case of *Dataset A*, the LR classifier considered *ratior_prior_paid_late*, *avg_days_late*, and *prior_total_late* as the top three essential features to distinguish between on-time and late payments. Moreover, we revealed that the least significant features were *non_business_days*, and *recent_paid_ontime_days*. Contrary to *Dataset B*, where the LR classifier considered *prior_paid*, *recent_paid_late_days*, and *non_business_days* as the most important features for the respective classification, whereby the features *prior_total* and *prior_total_late* were rather insignificant. To determine whether a particular feature was considered for on-time or late payment classification, we need to review the corresponding sing of the feature coefficients within the respective LR plots. Thereby, a negative value indicates that it was used for late payment classification while a positive value indicates that it was used for on-time classifications.

| *Rank* | *Dataset A (LR)* | *Dataset B (LR)* | *Dataset A (DT)* | *Dataset B (DT)* |
|---|---|---|---|---|
| 1 | *ratio_prior_paid_late* | *prior_paid* | *ratio_prior_paid_late* | *prior_paid* |
| 2 | *avg_days_late* | *recent_paid_late_days* | *prior_paid* | *total* |
| 3 | *prior_total_late* | *non_business_days* | *avg_days_late* | *recent_paid_ontime_days* |
| 4 | *prior_paid_late* | *total* | *total* | *prior_total_late* |
| 5 | *prior_paid* | *avg_days_late* | *prior_total* | *ratio_prior_paid_late* |
| 6 | *recent_paid_late_days* | *prior_paid_late* | *recent_paid_late_days* | *prior_total* |
| 7 | *prior_total* | *ratio_prior_paid_late* | *country=391* | *avg_days_late* |
| 8 | *country=391* | *recent_paid_ontime_days* | *recent_paid_ontime_days* | *prior_paid* |
| 9 | *country=897* | *prior_total* | *prior_total_late* | ~~*non_business_days*~~ |
| 10 | *total* | *prior_total_late* | *non_business_days* | ~~*prior_paid_late*~~ |
| 11 | *country=818* | | *country=770* | |
| 12 | *country=406* | | ~~*prior_paid_late*~~ | |
| 13 | *country=770* | | ~~*country=406*~~ | |
| 14 | *non_business_days* | | ~~*country=818*~~ | |
| 15 | *recent_paid_ontime_days* | | ~~*country=897*~~ | |

Table 6.1: **Most characteristic features for the LR and DT classifiers** to classify invoice into on-time and late payments. We ranked the features from top to bottom which means that the higher a feature is listed, the more important it was for the respective classifier to determine the payment outcome for an invoice. The strikethrough features were not considered by the DT classifiers due to set constraints by the hyperparameters.

Next, we determined a similar ranking for the features in the decision tree for *Dataset A* and *Dataset B*. However, not all features were used to by the respective DT classifiers since the optimized hyperparameters limited its depth or the number of considered feature. Consequently, we could reveal that the DT classifier in Datasets A did not make use of the feature *prior_paid_late*, whereby it further ranked the features *prior_paid_late* and *total* very differently than the used LR classifier. For *Dataset B* found similar results, where the DT classifier did not include the characteristic feature *non_business_days*. Thereby we could argue that the exclusion of these features were maybe one of the main causes which explain the lack of performance compared to the LR models.

As a conclusion for the feature importance, we want to point out again that we could not determine any overall best set of features for the classification process of on-time and late payment considering both of our datasets. Each dataset held its own set of characteristic features whereby the ranking differed very drastically among the used datasets, e.g. *non_business_days* held a rank of 14/15 in *Dataset A* whereby *Dataset B* ranked the same feature at 3/10. Thereby we think that the diversity of our two datasets is likely the main cause for the different feature ranking throughout our two datasets. While we know that both datasets contain invoices with very different characteristics, from different countries and business sectors, the payment behavior of the respective customers is also very likely to differ. Consequently, we were not able to identify an overall set of best features, but we rather identified a set of features for each dataset individually. Nevertheless, we think that it is very likely to identify similar feature sets when exploring invoices from companies of similar business sectors or regions.

## 6.5 Conclusion

To summarize our discussion about the individual results of sampling methods, classifier comparisons, ensemble strategies, and identification of feature importance, we concluded this chapter by concisely presenting our main findings while trying to answer the stated research questions in this thesis.

**RQ 1: Which ML classifiers are most suited for the problem of invoice payment classification into ”on-time payments” and ”late payments”?**

In this thesis, we conducted various experiments for invoice payment classification with the ML classifiers of Naive Bayes (NB), Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), and K-Nearest Neighbor (KNN). Additionally, we evaluated ensemble strategies with a Soft-Voting classifier (LR, RF, SVM - uniformly weighted), and Adaptive Boosting (AdaBoost) on LR, RF, and SVM. All evaluations where thereby based on two rather diverse datasets from different business sectors. After conducting various experiments and investigations, we could conclude that the LR classifier was the best suited ML model which provided the best performance results regarding F1-Score and AUC measurements. However, it is particularly important to note that the use of different datasets influenced these results. Using the dataset which contained more records on customer's historical payment behaviors (*Dataset A*), revealed that most of the considered ML models performed more or less equally. Conducting the same experiments on a smaller dataset with fewer invoice records (*Dataset B*), revealed however that LR was outperforming all other presented classifiers very drastically. The biggest difference was thereby observable in the use of the NB classifier, whereby we can not recommend using it when working with only few invoice samples since our experiments on *Dataset B* demonstrated that it performed only slightly better than the constant classifier. We made similar observations while investigating the results of the ensemble classifiers. Even though the application of AdaBoost and the Soft-Voting classifier provided useful results in use on the bigger *Dataset A*, they were still not able to outperform the single LR model by much (performed rather identically). On *Dataset B* we could observe that the application of Boosting somewhat worsens the overall results while they forced the classifiers to concentrate on specific classes only (e.g. LR was only usable for late payment classifications).

To summarize, we can conclude that our main findings for the best ML classifiers revealed that LR outperformed all other evaluated classifiers, but only in the case of using a dataset with rather few data samples. Considering a decent amount of invoice records for returning customers, also RF or NB would be appropriate models to pick when trying to classify invoices into on-time and late payments. The use of ensemble strategies was thereby considered to be of little use, due to the rather low performance boost and the high computational efforts during training and classification.

**RQ 2: Which invoice and customer features are the most significant ones to predict whether an invoice will be paid on-time or late?**

Throughout our experiments on feature importances, we mainly focused on the LR and DT classifiers, whereby the investigations on the feature importance for the DT classifier were only considered to determine possible problems while comparing it to the LR classifier. The LR classifier, which generally outperformed all other considered classifiers in this thesis, was consequently the one on which we focused to find the most valuable feature characteristics. The individual features coefficients of the trained LR classifiers were thereby used to distinguish between valuable features for on-time and late payment classifications. Similarly to the process of determining the most suited ML classifier itself, also the identification of feature characteristics differed rather drastically for our two datasets. Consequently, we were not to able to identify an overall unique set of features for both datasets in general, but rather provided a set of features for each dataset independently.

For *Dataset A*, the two most valuable features for a late payment classification were *ratio_prior_paid_late* (representing the ratio of previous late paid invoices compared to the total number of issued invoices), and *avg_days_late* (reflecting the average payment delay in days considering all previous issued invoices). On the other side, the two most valuable features for an on-time payment classification were *prior_paid* (representing the total amount of all previous paid invoices), and *recent_paid_late_days* (representing the number of elapsed days from the last late payment).

Contrary, for *Dataset B*, we determined that the two most valuable features for a late payment classification were *prior_paid* (number of previous paid invoices), and *recent_paid_late_days* (representing the number of elapsed days from the last late payment). Furthermore, the two most characteristic features for an on-time payment classification were *total* (total invoice amount of previous on-time and late paid invoices), and *prior_paid_late* (number of previous late paid invoices).

It is important to note that throughout our two datasets, the different features had different interpretations. While LR in *Dataset A* used *recent_paid_late_days* to identify on-time payments, LR in *Dataset B* used the same feature to identify late payments. This difference is thereby mainly caused due to the different customer payment behaviors in our dataset. Further investigations revealed for example that the feature value for late payments in *Dataset B* were always respectively low, whereby our LR classifier used them for late payment classifications. On the other side, in *Dataset A*, we observed mostly very high values for the respective feature, which is why the LR classifier used them mainly to determine on-time classifications.

To summarize these observations, we concluded that the feature importance for our classifications is strongly dependent on the underlying datasets on which we trained the respective LR classifier. However, we think that especially by using invoice datasets from similar business sectors or regions (holding same customers), an overall feature set detection would be definitely possible.

**RQ 3: Would such invoice and customer features differ drastically while constructing ML classifiers for different companies?**

As we observed from prior evaluations, it depends on the interpretation of ”different companies”. When we consider the scenario which we had in our experiments (two rather diverse datasets from companies with different business sectors and customers), than the feature importance differs very drastically. However, assuming that we have similar companies or at least companies which serve the same customers, we strongly assume to receive also similar customer and invoice features for the respective payment classification. In this thesis, we were however not able to conduct such experiments due to missing access to additional invoice datasets. Consequently, we can only make assumptions based on our conducted experiments whereby we left a detailed answer for future work.

As a conclusion for this discussion, we can say that the use of the LR classifier was overall the most suited strategy to classify invoice into on-time and late payments while it also provided a respectively good overview of used classification features which argued its decisions. To finally provide a proper tool for businesses which supports the choice of selecting customers for preventive actions like email reminders, phone calls, etc., all invoices would need to be sorted by their reported late payment probability. Once all invoices have been sorted, one can continue to filter out the underlying customers, which then finally leads to a list of sorted customers who are ranked by their probability of late payments. Consequently, the respective invoice collection department can use this extracted list to smartly decide on the choice of customers who need to get contacted to take primarily actions for preventing possible payment outages.

# Chapter 7

# Conclusion and Future Work

This chapter provides a brief outline of future work which may be considered when advancing the work in this field of predictive analytics on invoice payment classifications. Additionally, we conclude by reviewing and summarizing the main objectives of this thesis.

## 7.1 Future Work

The ability to classify invoices into two groups which distinguish between on-time and late payments can be seen as a valuable approach towards the use of predictive analysis and ML techniques in invoice collection and deductions/dispute management. To further improve the performance and practicality of the classifiers which we presented in this thesis, one could consider several steps as future work:

- **Gathering more invoice data** from companies which have a higher invoice issuing frequency. Since our assembled datasets in this thesis are both rather small, this step would further imply that more invoices from returning customers are likely to get collected which enables a more detailed customer-specific payment behavior analysis.

- **Collection and use of customers background information** like natural or legal person status, age, credit limit, business sector etc., which might enable an additional invoice classification of one-time or first-time customers.

- **Unification of classifiers** regarding companies individual models which work in the same business sector or handle the same customer base (similar regions). Customer-specific payment behaviors could thereby be learned trough inheriting and combining features from different company models. This additional information might especially be valuable when working with small companies which provide only a rather small amount of invoices.

- **Forecasting the magnitude of payment delays** could also be a possible goal in future work to more carefully support invoice collection processes of late paying customers. Thereby, not only their respective late payment probabilities could be forecasted, but also the individual amount of invoice delay (days) to support the

customer selection process regarding related preventive action considerations.

- **Customer clustering** could also be a valuable strategy to reveal whether payment behaviors can be assigned to individual customer groups. That might especially be interesting when considering companies who work with various business or private clients from different sectors or regions.

- **Neural Network** based ML classifiers and strategies might be tested and evaluated to improve the performance of the classification tasks itself, while further revealments of hidden structures in the payment behavior of the individual customer groups are also imaginable.

- **A/B testing** could be considered to evaluate the actual benefit which is gained through these respective invoice classifications. That refers to an evaluations where the predicted late payments are split into two groups, and one group gets contacted through preventive actions like phone calls, e-mail reminders, etc., while the second group does not get influenced through any preventive actions. The results can thus be used to assess if the gained knowledge could be used in a meaningful manner.

## 7.2 Conclusion

The overall goal of this master thesis was to develop and review a strategy to overcome the lack of information which many companies encounter while handling their invoice collection process. This problem referred mainly to the ability to identify and distinguish between possible late and on-time payments during the invoice creation process. Furthermore, the thesis suggested a ranking of the outstanding invoices according to their forecasted payment probability, such that preventive actions for late payments can be considered in a smart order. In other words, outstanding invoices which get classified to stay very likely unpaid within the defined invoice lead time, are listed on the top while invoices which are very likely to get paid within this timeframe are listed at the bottom. The main idea behind this concept was thereby to provide a suitable strategy such that companies are able to incorporate preventive actions during this invoice lead time, which could enable to minimize the time of outstanding payments.

This thesis focused thereby on using six different state-of-the-art classification models, namely: Naive Bayes, Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors, and Support Vector Machine. Moreover, additional ensemble strategies with AdaBoost and a Soft-Voting classifier were used to combine the strengths of the individual models. Besides the training, testing and evaluation process of these classifiers, a fundamental focus was also set on the data interpretation and preprocessing steps which were necessary to understand and prepare the underlying data on which this thesis was built upon. Consequently, the thesis guided through the different steps of data cleaning and review, feature engineering and selection, data analysis, and oversampling as well as optimization strategies.

The final evaluation results of this thesis can be summarized into two scenarios. On the one hand, when we consider working with a decent amount of invoices which hold multiple invoice records for returning customers, the use of the LR or RF classifier turned out to provide the most promising classification performances. Thereby, it is important to note, that LR can easily be used to identify additional feature characteristics which are especially valuable for the classification process itself. The RF classifier is thereby known as a black-box model which classification results cannot be interpreted as easily. Moreover, the use of the NB classifier was also very promising since the rather large amount invoices enabled us to depict the respective probabilities of customer payment behaviors. On the other hand, when we consider working with only few invoice information (on average only two or three invoice records for each customer), we concluded that the only suitable classifier was the LR model. However, the respective classification performances were thereby much lower compared to the larger dataset. Consequently, we were able to conclude that a proper classification of invoices into on-time and late payments also requires an adequate amount of invoice records for returning customers. Moreover, the most characteristic features which were used by the LR model to classify invoices into their respective class were also differing rather strongly along both datasets. These results further indicated that a suitable classification tool which can identify on-time and late payments during the invoice creation process needs to be trained for each company (dataset) individually.

# Bibliography


[AA16] Maher Ala'raj and Maysam F Abbod. "Classifiers consensus system approach for credit scoring." In: *Knowledge-Based Systems* 104 (2016), pp. 89–105 (cit. on p. 11).

[Bar+04] Ricardo Barandela et al. "The imbalanced training sample problem: Under or over sampling?" In: *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer. 2004, pp. 806–814 (cit. on p. 37).

[BB12] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305 (cit. on pp. 13, 15, 37).

[BPM04] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. "A study of the behavior of several methods for balancing machine learning training data." In: *ACM SIGKDD explorations newsletter* 6.1 (2004), pp. 20–29 (cit. on pp. 40, 41).

[Bre96] Leo Breiman. "Bagging predictors." In: *Machine learning* 24.2 (1996), pp. 123–140 (cit. on p. 25).

[BS07] Rajat Bhagwat and Milind Kumar Sharma. "Performance measurement of supply chain management: A balanced scorecard approach." In: *Computers & Industrial Engineering* 53.1 (2007), pp. 43–62 (cit. on p. 7).

[Cha+02] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique." In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357 (cit. on pp. 39, 40).

[Cha09] Bongsug Chae. "Developing key performance indicators for supply chain: an industry perspective." In: *Supply Chain Management: An International Journal* 14.6 (2009), pp. 422–428 (cit. on p. 7).

[Che+13] Chun-Hao Chen et al. "A combined mining-based framework for predicting telecommunications customer payment behaviors." In: *Expert Systems with Applications* 40.16 (2013), pp. 6561–6569 (cit. on p. 10).

[Das+03] Sophia Daskalaki et al. "Data mining for decision support on customer insolvency in telecommunications business." In: *European Journal of Operational Research* 145.2 (2003), pp. 239–255 (cit. on p. 10).

[F+96] Yoav Freund, Robert E Schapire, et al. "Experiments with a new boosting algorithm." In: *Icml*. Vol. 96. Bari, Italy. 1996, pp. 148–156 (cit. on p. 26).

[GK07] Angappa Gunasekaran and Bulent Kobu. "Performance measures and metrics in logistics and supply chain management: a review of recent literature (1995–2004) for research and applications." In: *International journal of production research* 45.12 (2007), pp. 2819–2840 (cit. on p. 7).

[Har15] Terry Harris. "Credit scoring using the clustered support vector machine." In: *Expert Systems with Applications* 42.2 (2015), pp. 741–750 (cit. on p. 11).

[HCW07] Cheng-Lung Huang, Mu-Chen Chen, and Chieh-Jen Wang. "Credit scoring with a data mining approach based on support vector machines." In: *Expert systems with applications* 33.4 (2007), pp. 847–856 (cit. on p. 11).

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning, Data Mining, Interference, and Prediction*. 2nd ed. Springer, 2009 (cit. on pp. 18, 24, 33, 35).

[Inc16] FreshBooks Inc. *The Best Invoice Payment Terms to Get Paid Fast*. 2016. URL: `https://www.freshbooks.com/blog/invoice-payment-terms` (visited on 03/20/2018) (cit. on pp. 2, 46).

[K+06] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. "Handling imbalanced datasets: A review." In: *GESTS International Transactions on Computer Science and Engineering* 30.1 (2006), pp. 25–36 (cit. on p. 37).

[KK16] Jongmyoung Kim and Pilsung Kang. "Late payment prediction models for fair allocation of customer contact lists to call center agents." In: *Decision Support Systems* 85 (2016), pp. 84–101 (cit. on p. 9).

[KKL10] Amir E Khandani, Adlar J Kim, and Andrew W Lo. "Consumer credit-risk models via machine-learning algorithms." In: *Journal of Banking & Finance* 34.11 (2010), pp. 2767–2787 (cit. on p. 11).

[L+02] Andy Liaw, Matthew Wiener, et al. "Classification and regression by randomForest." In: *R news* 2.3 (2002), pp. 18–22 (cit. on p. 20).

[Lem16] Guillaume Lemaitre. *Illustration of the sample generation in the over-sampling algorithm*. 2016. URL: `http://contrib.scikit-learn.org/imbalanced-learn/stable/auto_examples/over-sampling/plot_illustration_generation_sample.html` (visited on 03/20/2018) (cit. on p. 40).

[Les+15a] Stefan Lessmann et al. "Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research." In: *European Journal of Operational Research* 247.1 (2015), pp. 124–136 (cit. on p. 11).

[Les+15b] Stefan Lessmann et al. "Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research." In: *European Journal of Operational Research* 247.1 (2015), pp. 124–136 (cit. on p. 20).

[Lim15] Xero Limited. *Invoice payment terms: Top seven tips*. 2015. URL: `https://www.xero.com/resources/small-business-guides/invoicing/invoice-payment-terms` (visited on 03/20/2018) (cit. on pp. 2, 46).

[LL11] Philip Leitch and Dawne Lamminmaki. "Refining measures to improve performance measurement of the accounts receivable collection function." In: *Journal of Applied Management Accounting Research* 9.2 (2011), p. 1 (cit. on p. 8).

[Par15] David Parmenter. *Key performance indicators: developing, implementing, and using winning KPIs*. John Wiley & Sons, 2015 (cit. on p. 8).

[Ram+12] Enislay Ramentol et al. "SMOTE-RSB*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory." In: *Knowledge and information systems* 33.2 (2012), pp. 245–265 (cit. on pp. 37, 40).

[Ras15] Sebastian Raschka. *Python machine learning*. Packt Publishing Ltd, 2015 (cit. on pp. 18, 20, 22, 23, 27).

[Sha+17] Torgyn Shaikhina et al. "Decision tree and random forest models for outcome prediction in antibody incompatible kidney transplantation." In: *Biomedical Signal Processing and Control* (2017) (cit. on p. 20).

[SKV09] Chris Seiffert, Taghi M Khoshgoftaar, and Jason Van Hulse. "Hybrid sampling for imbalanced data." In: *Integrated Computer-Aided Engineering* 16.3 (2009), pp. 193–210 (cit. on p. 37).

[SL09] Marina Sokolova and Guy Lapalme. "A systematic analysis of performance measures for classification tasks." In: *Information Processing & Management* 45.4 (2009), pp. 427–437 (cit. on p. 27).

[Ste+10] Ewout W Steyerberg et al. "Assessing the performance of prediction models: a framework for some traditional and novel measures." In: *Epidemiology (Cambridge, Mass.)* 21.1 (2010), p. 128 (cit. on p. 27).

[Tom76] Ivan Tomek. "Two modifications of CNN." In: *IEEE Trans. Systems, Man and Cybernetics* 6 (1976), pp. 769–772 (cit. on pp. 40, 41).

[Trk+10] Peter Trkman et al. "The impact of business analytics on supply chain performance." In: *Decision Support Systems* 49.3 (2010), pp. 318–327. ISSN: 0167-9236. DOI: https://doi.org/10.1016/j.dss.2010.03.007. URL: http://www.sciencedirect.com/science/article/pii/S0167923610000680 (cit. on p. 7).

[Wil72] Dennis L Wilson. "Asymptotic properties of nearest neighbor rules using edited data." In: *IEEE Transactions on Systems, Man, and Cybernetics* 3 (1972), pp. 408–421 (cit. on p. 40).

[Xia+17] Yufei Xia et al. "A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring." In: *Expert Systems with Applications* 78 (2017), pp. 225–241 (cit. on pp. 11, 12).

[Yap+14] Bee Wah Yap et al. "An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets." In: *Proceedings of the first international conference on advanced data and information engineering (DaEng-2013)*. Springer. 2014, pp. 13–22 (cit. on p. 37).

[You+14] Bashar Younes et al. "Overdue invoice management: Markov chain approach." In: *Journal of Construction Engineering and Management* 141.1 (2014), p. 04014062 (cit. on pp. 3, 9).

[Zen+08] Sai Zeng et al. "Using predictive analysis to improve invoice-to-cash collection." In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2008, pp. 1043–1050 (cit. on pp. 3, 4, 9, 10).

[ZS12] Tomasz S Zabkowski and Wieslaw Szczesny. "Insolvency modeling in the cellular telecommunication industry." In: *Expert Systems with Applications* 39.8 (2012), pp. 6879–6886 (cit. on pp. 3, 10).