DEGREE PROJECT IN MATHEMATICAL STATISTICS

SECOND CYCLE, 30 CREDITS

Predicting Customer Conversion using Supervised Machine Learning

STEPHANIE ABOUD

Stockholm, Sverige 2022

Abstract

The growth of e-commerce has been evident over the past years and for companies like Klarna that provides payment solutions, focusing on the purchase experience is more important than ever. With that goal in mind, more companies are using machine learning methods and tools to make predictions and forecast future outcomes, giving them a competitive advantage on the market. This thesis aims to apply supervised machine learning techniques to predict customer conversion, i.e. predict if a customer with a started shopping session will complete the purchase. The purpose of the project is to also determine which supervised learning algorithm performs the best when predicting customer conversion, with regards to a set of model evaluation metrics. The classical classification method Logistic Regression was tested, as well as the machine learning methods Support vector Machine, Random forest and XGBoost. The metrics used to evaluate the model performances were Precision, Recall, F1- and AUC-scores. Furthermore, the SHapley Additive exPlanations approach was implemented for feature importance and for interpreting tree-based models. The results showed that it is in fact possible to predict customer conversion using machine learning. All models yielded good performance and the difference in performance was relatively small. XGBoost performed slightly better than the rest of the models.

Sammanfattning

Tillväxten av e-handel har varit tydlig de senaste åren och för företag som Klarna, som erbjuder betalningslösningar, är det viktigare än någonsin förr att lägga stor fokus på kundernas köpupplevelse. Som hjälp använder allt fler företag maskininlärnings- metoder och verktyg för att prediktera och göra framtidsprognoser, något som gör dem konkurrenskraftiga på marknader. Syftet med detta examensarbete är att tillämpa övervakad maskininlärning för att prediktera kundkonvertering, med andra ord prediktera om en kund som påbörjat en shoppingsession kommer att slutföra beställningen. Syftet med projektet är även att avgöra vilken övervakad inlärningsalgoritm som presterar bäst vid predikteringen, med avseende på en uppsättning av valideringsmått. Den klassiska klassificeringsmetoden Logistisk Regression testades, så väl som maskininlärnings metoderna Stödvektormaskin, Random Forest och XGBoost. För att validera modellerna användes Precision, Recall, F1- och AUC-scores. Dessutom implementerades metoden SHapley Additive exPlanations för att företaget enklare ska förstå vikten av de olika variablerna och tolka de trädbaserade modellerna. Resultaten visade att det går att prediktera kundkonvertering med hjälp av maskininlärning. Alla modeller påvisade bra resultat och skillnaden i prestation var relativt liten. XGBoost presterade lite bättre än resterande modeller.

Acknowledgements

I would like to express my deepest gratitude to everyone involved at Klarna and in particular to Jamil Azrak for his support throughout this project and for acting as a mentor in life. I would also like to thank my KTH supervisor Tatjana Pavlenko for providing me with good feedback and constructive criticism.

List of Figures

List of Tables

Contents

1 Introduction

In this chapter I go through the background of the thesis, the purpose of this work and state the specific research questions that will be answered.

1.1 Background

Consumers are more often choosing to shop online, and the growth of e-commerce focused companies has been major over the past years. Klarna, founded in 2005, is a Swedish bank and company offering payment solutions to merchants. Today, Klarna is one of Europes biggest fintech companies and offers their payment solutions to merchants in 17 different countries. The company has over 90 million global active users and 2 million transactions a day [1].

An essential factor in the company's success is their focus on the purchase experience. In order for a company like Klarna to increase sales and partner with more merchants, they need to understand how their customers behave and what drives purchase conversion. Ultimately, the majority of all business decisions and strategies are made with the goal of boosting sales and becoming more competitive on the market. A change in conversion rates directly affects the revenue for both Klarna and their merchants.

Meanwhile, machine learning is becoming a bigger and more important part of e-commerce. The use of predictive models gives companies a competitive advantage as they can forecast future outcomes in advance. The use of machine learning makes it possible for e-commerce companies to personalize interactions with their customers, leading to an overall better user experience. The goal is often to increase purchase conversion and studies show significant increases in conversion rates when adding machine learning to the mix [2].

A use case for machine learning within e-commerce is to predict the customers conversion probability. Such predictions give valuable insight to the company as they gain better understanding of the key factors driving conversion and gives them a chance to make early interventions. Depending on the prediction response, the company can take different strategic actions in order to increase conversion rates. Separating the customers that will most likely convert, i.e place an order, from the ones that will most likely not, open doors for personalizing the shopping flow for each type of customer.

As Klarna is widely used over the world, the purchase experience can look different in each of their operating markets. Not only can the market play a part in the purchase experience, other factors like the choice of e-store could affect the purchase flow due to different integrations. The company has over the years developed many products, with Klarna Checkout being one of their biggest. One of the screens in the Klarna checkout flow is the payment selector. As indicated by the name, the payment selector screen lists the available pay-

ment methods the customer can choose from. After selecting, or keeping the pre-ticked payment method, the customer can choose to place the order or drop out of the flow.

1.2 Purpose

The purpose of this thesis is to determine if it is possible to predict customer conversion, i.e. predict if a customer with a started shopping session will convert/place the order. Furthermore, the purpose is to investigate which algorithm yields best performance results when predicting customer conversion. A classical approach, Logistic regression, will be compared to the three machine learning methods Support-vector Machines, Random Forest and XGBoost. The algorithms will be compared based on the evaluation metrics F1 score, Macro average, ROCAUC and PRAUC.

The research questions is as follows:

- Is it possible to predict if a customer starting a shopping session will convert/place the order?
- Which Supervised learning algorithm performs best when predicting conversion with regards to specific model evaluation metrics?

1.3 Limitations

As this project is done in collaboration and within a specific team at Klarna, their team-specific vision and goal needs to be considered. To fit their vision, limitations has to be made to the data and the project scope in general.

- The data is based on the Swedish market.
- The data is limited to shopping sessions made with the Klarna Checkout product.
- The shopping session starts at the Payment Selector screen and ends when a customer has either successfully placed an order or dropped out.
- The data contains events from 2020 only.

1.4 Thesis outline

The remaining parts of the thesis are structured as follows. Section 2 focuses on introducing and explaining the relevant mathematical theory used in this project. The mathematical concepts and algorithms are explained. The section also includes the theory behind the Model evaluation metrics considered when building the model and analyzing the results. Section 3 covers the methodology used to carry out the project. This includes a description of how the data was processed and how the hyperparameters were tuned. The results are presented

in Section 4. A summary of the different models is presented, along with relevant plots. In Section 5 the results are discussed and the thesis concluded, along with some suggestions for future work.

2 Theory

In this Theory section all the relevant mathematical concepts and algorithms will be introduced and explained.

2.1 Supervised machine learning and Binary classification

In statistical learning, most of the problems either fall under supervised or unsupervised learning. The former is the most common and will be used in this project.

To gain an understanding of the concept supervised learning, we begin by looking at the function

$$Y = f(\mathbf{X}) + \epsilon \tag{1}$$

where f is a fixed function and the random *irreducible error* $\epsilon > 0$ has mean and variance $E[\epsilon] = 0$ and $Var(\epsilon) = \sigma^2$ respectively [3].

The goal is, using an algorithm, to approximate the mapping function so that for any future input data $\mathbf{X}$ the trained model can predict the associated output target Y. As this thesis handles a qualitative response with two possible categories, the target variable is modelled as a binary random variable Y_i taking on the values $Y \in \{0, 1\}$. The observed values of the target variable are denoted by y_i and furthermore $y_i = 1$ if a customer converts after a started shopping session while $y_i = 0$ if the customer drops out before converting. The random feature variable is denoted $\mathbf{X}_i \in \mathbb{R}^p$ and the observed equivalent feature vector is represented by $\mathbf{x}_i = [x_{i1}, x_{i2}, \cdots, x_{ip}]^T$ where $i = 1, \cdots, N$ and p is the number of features. The model learns from historical data as every input has a known labeled output and applies this knowledge to predict future events [3]. Combining these notations, we will in the coming sections express the observed training dataset as $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, for a set of N observations.

As mentioned, other machine learning problems might fall under the framework of unsupervised learning. In those cases, a predictor measurement is observed, but no corresponding response. There is no training data available and the model is required to learn by itself. Instead, the goal is to learn about the underlying distribution of the data and unsupervised learning algorithms are often categorized as clustering models [4].

2.2 Logistic Regression

Logistic Regression is a classical modeling approach and widely used for binary classification. Its easy implementation and efficiency when training often makes it a go-to technique for binary classification and often used as a benchmark. The logistic function is designed to model the posterior probabilities of the K

classes via linear functions in x [5]. Again, as this is a binary classification problem, the model considers two classes. The posterior probability of a customer belonging to class 1, given an observed feature vector $\mathbf{x}_i$ where $i = 1, \cdots, N$, is given through a logistic function

$$P(Y_i = 1 \mid \mathbf{X}_i = \mathbf{x}_i) = \frac{e^{\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i}}{1 + e^{\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i}} \tag{2}$$

where β_0 and $\boldsymbol{\beta} = [\beta_1, \beta_2, \cdots, \beta_p]^\top$ are the model parameters, representing the intercept and the coefficient vector respectively [3]. The function curve is S-shaped, taking on values between 0 and 1 as shown in the figure below [6]

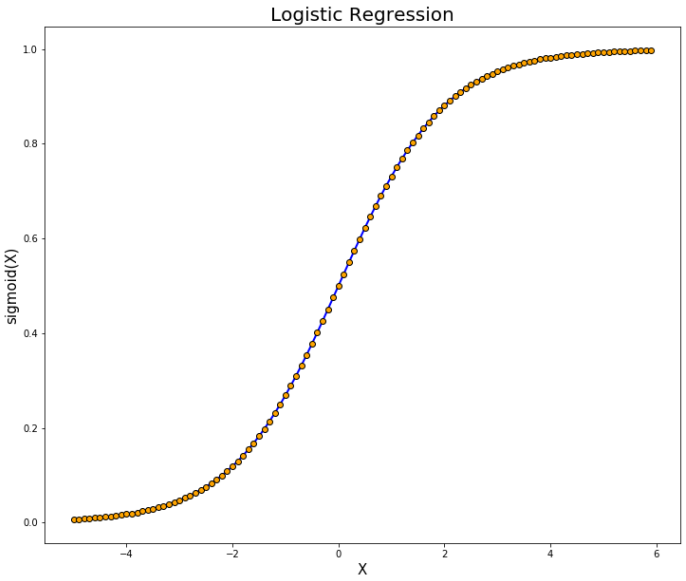

Figure 1: The shape of the logistic function. Source of Figure: [6]

The logistic function is a derivation from the *log-odds* transformation of the outcome variable which has a linear relationship with the predictor variables, according to

$$log \frac{P(Y_i = 1 \mid \mathbf{X}_i = \mathbf{x}_i)}{1 - P(Y_i = 1 \mid \mathbf{X}_i = \mathbf{x}_i)} = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i \tag{3}$$

As the regression coefficients are unknown and need to be estimated, the model is fit using the *maximum likelihood* method. After some algebraic manipulation of Eq.(2), the probability can be rewritten as

$$P(Y_i = 1 \mid \mathbf{X}_i = \mathbf{x}_i) = \frac{1}{1 + e^{-(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i)}} \tag{4}$$

With binary classification and N observations, the *log-likelihood* function can then be defined as

$$\begin{aligned} l(\beta_0, \boldsymbol{\beta}) &= \sum_{i=1}^{N} \{y_i \log p(\mathbf{x}_i; \beta_0, \boldsymbol{\beta}) + (1 - y_i) \log(1 - p(\mathbf{x}_i; \beta_0, \boldsymbol{\beta}))\} \\ &= \sum_{i=1}^{N} \left\{ y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i) - \log(1 + e^{\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i)} \right\} \end{aligned} \tag{5}$$

where l represents the negative log-loss function and the conditional probability $p(\mathbf{x}_i; \beta_0, \boldsymbol{\beta}) = P(Y_i = 1 \mid \mathbf{X}_i = \mathbf{x}_i; \beta_0, \boldsymbol{\beta})$

Let $\beta = \{\beta_0, \boldsymbol{\beta}\}$ and assume $\mathbf{x}_i$ includes the constans term 1 to accomodate the intercept β_0. To optimize the log-likelihood function above by maximization, the derivatives are set to zero

$$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{N} \mathbf{x}_i (y_i - p(\mathbf{x}_i; \beta)) = 0 \tag{6}$$

This yields $p+1$ equations which are *nonlinear* in β and are solved by using the *Newton-Raphson* algorithm [3]. The algorithm requires the second-derivative which is computed according to

$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^\top} = -\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^\top p(\mathbf{x}_i; \beta)(1 - p(\mathbf{x}_i; \beta)) \tag{7}$$

Then, starting at β_{old}, a single Newton-Raphson update is given by

$$\beta^{new} = \beta^{old} - \left(\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^\top}\right)^{-1} \frac{\partial l(\beta)}{\partial \beta} \tag{8}$$

The values of the β-vector are improved until a suitable stopping criteria is reached. This could be when the dual gap is smaller than the predefined tolerance. In the case of no conversion, the value of the *Maximum iterations* parameter decides when the iteration stops [7].

2.2.1 Regularization techniques

Regularization is a common way of controlling for and reducing overfitting in a flexible way. An overfitted model is an overly complex model that tends to perform well in reference to its initial dataset. This is problematic due to the model's inability to generalize and make good predictions on unseen data. Two commonly used methods are L_1- and L_2- regularization and are also known as LASSO and Ridge regression, respectively. The methods helps reduce the variance of the model, leading to a less overfitted model. A linear combination of both L_1- and L_2 can also be used and referred to as *elastic net* [8].

In L_1**- regularization**, or LASSO, the model is penalized by adding:

$$-\lambda_{L_1} \sum_{j=1}^{p} |\beta_j| \tag{9}$$

to the negative log-loss function given in Eq. (5). Here, $\lambda_{L_1} \geq 0$ is the shrinkage parameter and is also defined as $\lambda_{L_1} = \frac{1}{C}$ where $C \geq 0$. As the shrinkage parameter grows, the regression coefficients are shrunk towards zero. LASSO also has the ability to shrink some coefficients all the way down to zero and by doing so, less important features can be removed entirely. L_1- regularization can therefore serve as a feature selection method [3].

The penalized negative log-loss function can now be written as follows

$$l(\beta_0, \boldsymbol{\beta}) = \sum_{i=1}^{n} [y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i) - log(1 + exp(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i))] - \lambda_{L_1} \sum_{j=1}^{p} |\beta_j| \tag{10}$$

To estimate the coefficients, second order Taylor expansion is used to maximize the penalized function above. We denote the current coefficient estimates as $\hat{\beta} = (\hat{\beta}_0, \hat{\boldsymbol{\beta}})$ and let $\hat{p}(\mathbf{x}_i) = (\mathbf{x}_i; \hat{\beta}_0, \hat{\boldsymbol{\beta}})$ and $w_i = \hat{p}(\mathbf{x}_i)(1 - \hat{p}(\mathbf{x}_i))$.

Second order Taylor expansion around the estimates yields the following quadratic objective function

$$l_Q(\beta_0, \boldsymbol{\beta}) = -\frac{1}{2} \sum_{i=1}^{n} [w_i(z_i - \beta_0 - \boldsymbol{\beta}^\top \mathbf{x}_i)^2 + C(\hat{\beta_0}, \hat{\boldsymbol{\beta}})^2] - \lambda_{L_1} \sum_{j=1}^{p} |\beta_j| \tag{11}$$

where $z_i = \hat{\beta}_0 + \hat{\boldsymbol{\beta}}^\top \mathbf{x}_i + \frac{y_i - \hat{p}(\mathbf{x}_i)}{\hat{p}(\mathbf{x}_i)(1 - \hat{p}(\mathbf{x}_i))}$ is the current working response and C a constant independent of $(\beta_0, \boldsymbol{\beta})$.

By maximizing $l_Q(\beta_0, \boldsymbol{\beta})$, a Newton-Raphson update of the coefficients can be obtained. This is a simple weighted least-squares problem, solved by a generalized Newton algorithm. A sequence of nested loops are created where, for each decrement of λ_{L_1}, a quadratic approximation l_Q is computed using the current coefficients. To find the coefficients that maximizes Eq. (11), an inner loop is created with the generalized Newton algorithm running on the optimization problem [3].

L_2**- regularization**, or Ridge, works in a similar way. The main difference between the two methods can be found in the penalty term as Ridge penalizes using the sum of square given below

$$-\lambda_{L_2} \sum_{j=1}^{p} \beta_j^2 \tag{12}$$

Unlike LASSO, the value of the coefficients can never be shrunk to reach zero [3].

For this regularization technique, the penalized negative log-loss function can be written as follows

$$l(\beta_0, \boldsymbol{\beta}) = \sum_{i=1}^{n} [y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i) - log(1 + exp(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i))] - \lambda_{L_2} \sum_{j=1}^{p} \beta_j^2 \quad (13)$$

Estimation of the coefficients $\beta = (\beta_0, \boldsymbol{\beta})$ follows the same procedure as for L_1-regularization above, but now maximizing Eq. (13) instead.

Another regularization technique candidate is a linear combination of the two previous presented. An **Elastic net** combines both penalty terms into one, benefitting from the two methods. The following penalty term is used

$$-\lambda_e \sum_{j=1}^{p} \alpha \beta_j^2 + (1 - \alpha) \left| \beta_j \right| \quad (14)$$

where $\alpha \in [0, 1]$ is the ratio-variable giving more or less weight to either term. Thus, $\alpha = 0$ would be equivalent to using LASSO regularization and $\alpha = 1$ would be the same as only using Ridge regularization [8].

Using this combination, the negative log-loss function is written as follows

$$l(\beta_0, \boldsymbol{\beta}) = \sum_{i=1}^{n} [y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i) - log(1 + exp(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i))] - \lambda_e \sum_{j=1}^{p} \alpha \beta_j^2 + (1 - \alpha) \left| \beta_j \right| \quad (15)$$

and once again, the procedure of estimating the coefficients $\beta = (\beta_0, \boldsymbol{\beta})$ is the same as for both previous regularization techniques [3].

2.3 Support Vector Machine

Another, very common approach to binary classification is Support Vector Machine (SVM). The machine learning method has similarities to Logistic Regression and using both methods quite often yields the same results. SVM tend to perform a bit better when the two classes are well separated [4]. To best grasp the theory behind Support Vector Machine, the two other concepts *Maximal Margin Classifier* and *Support Vector Classifier* will be introduced first.

2.3.1 Maximal Margin Classifier

Consider again a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ with N observations. To classify these observations, a Maximal Margin Classifier uses a separating hyperplane. A hyperplane in a p-dimensional space is a flat affine subspace of dimension $p-1$. The hyperplane in $\mathbb{R}^p$ is defined by

$$\mathbf{x}_i^T \boldsymbol{\beta} + \beta_0 = 0 \tag{16}$$

where $\boldsymbol{\beta}$ is a unit parameter vector, $\|\boldsymbol{\beta}\| = 1$. In the case of binary classification, with $p = 2$, the hyperplane is simply a line.

Using Eq. (16) above and letting $f(\mathbf{x}) = \mathbf{x}_i^T \boldsymbol{\beta} + \beta_0$, then $f(\mathbf{x}) > 0$ for points on one side of the hyperplane and $f(\mathbf{x}) < 0$ for points on the other side. This gives hold to the following equation

$$y_i(\mathbf{x}_i^T \boldsymbol{\beta} + \beta_0) \geq 0 \tag{17}$$

for $i = 1, \cdots, N$ [3].

The figure below illustrates a separating hyperplane shown in black, distinguishing between two colour classes.

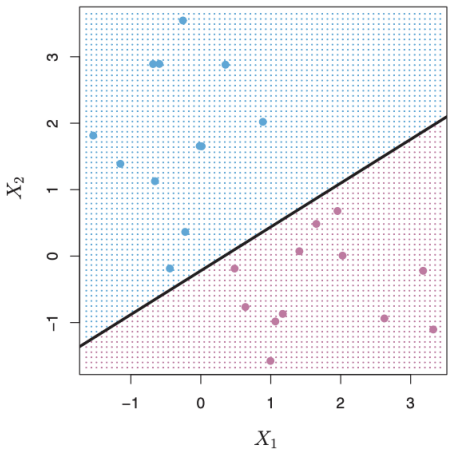

Figure 2: Example of a hyperplane separating two colour classes. Source of Figure: [4]

The idea behind Maximal Margin classifier is to find, among all separating hyperplanes, the one that yields the biggest gap or margin between the two classes. In the figure below the maximal margin hyperplane is manifested in a solid line and the margin represented by the distance between the solid line and either one of the dashed lines. The three points located on the dashed lines are referred to as *support vectors*. The maximal margin hyperplane is dependent solely on these observations. If the observations were to slightly move, the solid line would be adjusted according to this move [4].

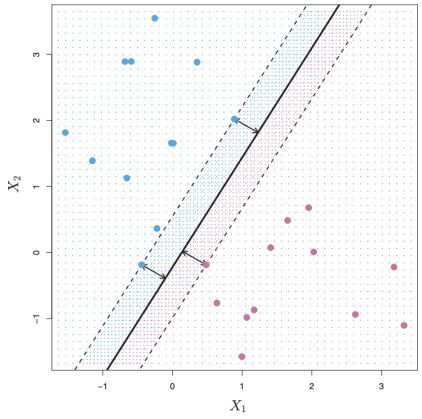

Figure 3: Example of a maximal margin hyperplane, shown in a solid line. Source of Figure: [4]

The construction of a Maximal Margin Classifier gives the following constrained

optimization problem

$$\begin{cases} \underset{\boldsymbol{\beta},\beta_0}{\text{maximize}}\ M \\ \text{subject to } \|\boldsymbol{\beta}\| = 1 \\ y_i(\mathbf{x}_i^T\boldsymbol{\beta} + \beta_0) \geq M, \forall i = 1, .., N \end{cases} \tag{18}$$

where the width of the margin is given by $2M = \frac{2}{\|\boldsymbol{\beta}\|}$.

The optimization problem can be rewritten in the following way

$$\begin{cases} \underset{\boldsymbol{\beta},\beta_0}{\text{minimize}}\ \|\boldsymbol{\beta}\| \\ \text{subject to } y_i(\mathbf{x}_i^T\boldsymbol{\beta} + \beta_0) \geq 1, \forall i = 1, .., N \end{cases} \tag{19}$$

after rescaling with $M = \frac{1}{\|\boldsymbol{\beta}\|}$ [3].

2.3.2 Support Vector Classifier

In many cases, the existing data is not exactly separable and there is no perfect hyperplane that can be fit through the data. Another problem can occur when the data is separable but noisy, causing one extra point to have dramatic effect on the maximal margin classifier. This sensitivity to single observations indicates that there is a risk of overfitted data [9].

To deal with these problems, a *Support Vector Classifier* is introduced. Also referred to as *soft margin*, this is a classifier that does not perfectly separate the two classes but allows for some misclassified observations. This is done for the greater good leading to a more robust model and better overall classification results. The figure below shows two examples of how a support vector classifier is fit to a small dataset, allowing for some observations to be misclassified.

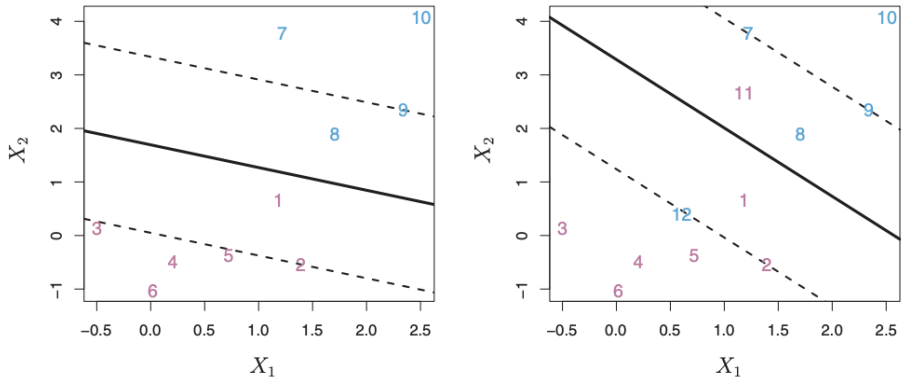

Figure 4: Two examples of a support vector classifier. The left shows observations on the wrong side of the margin and the right shows observations on the wrong side of both the margin and the solid line hyperplane. Source of Figure: [4]

Constructing a support vector classifier, the formulation of the problem given in Eq. (18) needs to be modified to accommodate it. The optimization problem can now be formulated as

$$\begin{cases} \underset{\boldsymbol{\beta},\beta_0,\boldsymbol{\xi}}{\text{maximize}}\ M \\ \text{subject to } \|\boldsymbol{\beta}\| = 1 \\ y_i(\mathbf{x}_i^T\boldsymbol{\beta} + \beta_0) \geq M(1-\xi_i) \\ \xi_i \geq 0, \sum_{i=1}^{N} \xi_i \leq C \end{cases} \tag{20}$$

where C is a tuning parameter and the bigger the value of C, the more tolerant the classifier is to classification. It can be seen as if the equation is discounted by a discount vector $(1-\xi_i)$, indicating that the support vector classifier allow some "slacks". The *slack variables* $\xi_1, .., \xi_N$ indicates how much each observation is allowed to be on the wrong side of the margin.

Rescaling again yields the following rewritten optimization problem

$$\begin{cases} \underset{\boldsymbol{\beta},\beta_0,\boldsymbol{\xi}}{\text{minimize}}\ \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to } y_i(\mathbf{x}_i^T\boldsymbol{\beta} + \beta_0) \geq 1 - \xi_i \\ \xi_i \geq 0, \sum_{i=1}^{N} \xi_i \leq C \end{cases} \tag{21}$$

This is solved by minimizing the Lagrange (primal) function below

$$L_P = \frac{1}{2}\left\|\boldsymbol{\beta}\right\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i(y_i(\mathbf{x}_i^T\boldsymbol{\beta}+\beta_0) - (1-\xi_i)) - \sum_{i=1}^{N}\mu_i\xi_i \tag{22}$$

with the positivity constraints $\alpha_i, \xi_i, \mu_i \geq 0$ [3].

The minimization is done by setting the derivatives w.r.t $\boldsymbol{\beta}, \beta_0, \xi_i$ to zero as follows

$$\sum_{i=1}^{N}\alpha_i y_i = 0 \tag{23}$$

$$\boldsymbol{\beta} = \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i \tag{24}$$

$$\alpha_i = C - \mu_i \tag{25}$$

Insertion of the equations above into Eq. (22) yields the Lagrangian dual objective function

$$L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \tag{26}$$

This function is furthermore maximized subject to $\sum_{i=1}^{N}\alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$. Combined with the derivatives above, the Karush-Kuhn-Tucker conditions

$$\alpha_i(y_i(\mathbf{x}_i^T\boldsymbol{\beta}+\beta_0) - (1-\xi_i)) = 0 \tag{27}$$

$$\mu_i\xi_i = 0 \tag{28}$$

$$y_i(\mathbf{x}_i^T\boldsymbol{\beta}+\beta_0) - (1-\xi_i) \geq 0 \tag{29}$$

for $i = 1, ..N$ are used to solve both Lagrangian functions [3] [9].

As can be seen from Eq. (24), the solution for $\boldsymbol{\beta}$ has the form

$$\hat{\boldsymbol{\beta}} = \sum_{i=1}^{N}\hat{\alpha}_i y_i \mathbf{x}_i \tag{30}$$

The coefficients $\hat{\alpha}_i$ have the constraint $\hat{\alpha}_i > 0$ for the observations where Eq. (29) is exactly met. The observations are referred to as *support vectors*. Additional constraint falls on these observations as some of them lie on the margin where $\xi_i = 0$ and thus, by Eq. (25) and Eq. (28) it can be seen that $0 < \hat{\alpha}_i < C$. The rest lie inside the margin where $\xi_i > 0$ and thus, given by the same equations, $\hat{\alpha}_i = C$.

2.3.3 Support Vector Machine

When dealing with a non-linear decision boundary, the use of a any linear classifier will most likely yield poor performance. One way to solve this problem is by enlarging the feature space. This can be done by including transformations, e.g. quadratic or high-order polynomial functions of the predictors. Fitting a support vector classifier in the enlarged space results in a non-linear decision boundary in the original space. The concept of *Support Vector Machine* is built on exactly this. With the use of *kernels*, it is a controlled and computationally efficient way to introduce non-linearities. The combination of a non-linear kernel with a support vector classifier leads to what is known as a support vector machine [9].

To fully grasp the concept, we start by expressing the transformation functions as
$h(\mathbf{x}_i) = (h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \cdots, h_m(\mathbf{x}_i))$, $i = 1, ...N$.

Thus, the modified hyperplane is given by

$$f(\mathbf{x}) = h(\mathbf{x}_i)^T \boldsymbol{\beta} + \beta_0 = 0 \tag{31}$$

where $\boldsymbol{\beta}$ in the enlarged feature space is given by

$$\boldsymbol{\beta} = \sum_{i=1}^{N} \alpha_i y_i h(\mathbf{x}_i) \tag{32}$$

Following the same procedure as in Section 2.3.2, the Lagrange dual objective function is written as

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle h(\mathbf{x}_i), h(\mathbf{x}_j) \rangle \tag{33}$$

with $\langle \cdot, \cdot \rangle$ defined as the inner product.

To solve the new optimization problem, a kernel function

$$K(\mathbf{x}, \mathbf{x}') = \langle h(\mathbf{x}), h(\mathbf{x}') \rangle \tag{34}$$

is used to compute the inner product in the transformed space and is a positive (semi-) definite function. Using Eq. (30) again, the solution function can be expressed as

$$f(\mathbf{x}) = \sum_{i=1}^{N} a_i y_i K(\mathbf{x}, \mathbf{x}_i) + \beta_0 \tag{35}$$

The choice of a kernel function can vary and some commonly used ones are

-**Radial basis**: $K(\mathbf{x}, \mathbf{x}') = exp(-\gamma \left\| \mathbf{x} - \mathbf{x}' \right\|^2)$
-***d*th degree polynomial**: $K(\mathbf{x}, \mathbf{x}') = (r + \gamma \langle \mathbf{x}, \mathbf{x}' \rangle)^d$

where γ is a positive scale parameter, d is an integer and r is a constant [3].

2.4 Tree-based concepts

2.4.1 Decision trees

Decision trees are supervised learning methods, used to split a dataset based on different conditions. The idea is to split a feature space into J distinct and non-overlapping regions: $R_1, R_2, ..., R_J$. Then, for all input observations that falls into the same region, the same prediction response is made. This response can be quantitative or qualitative, the former if a regression tree is used and the latter if a classification tree is used. The goal of this project is to predict whether or not a customer will convert and thus presenting a classification tree with a qualitative response will be the focus forward. Therefore, for each test observation that falls into a region R_j, we take a majority vote and assign it to the most commonly occurring class in that region, according to

$$k(j) = argmax_k \hat{p}_{jk} \tag{36}$$

where

$$\hat{p}_{jk} = \frac{1}{N_j} \sum_{x_i \in R_j} I\{y_i = k\} \tag{37}$$

is the proportion of class k observations that are in region R_J [4].

An illustration of a simple classification tree is given in Fig. (5) below. The tree has two internal nodes and three terminal nodes, which are the predictive Yes/No outcomes and also referred to as *leaves*.

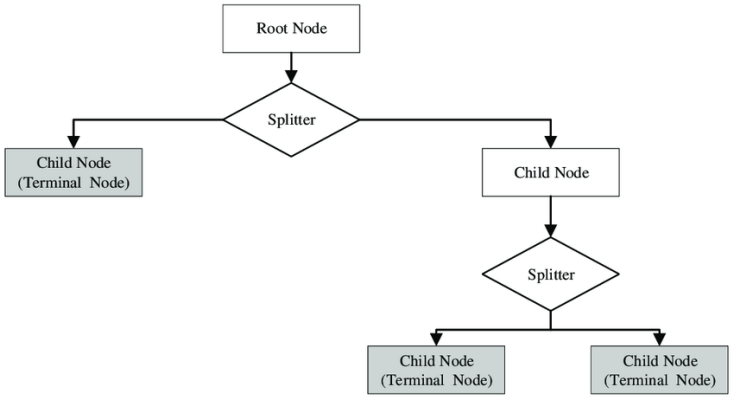

Figure 5: Example of a classification tree. Source of Figure: [10]

For binary splitting of the tree, a criterion needs to be chosen. Both *Cross-entropy* and the *Gini index* are differentiable and thus suitable for numerical optimization. The Gini index will be used as a criteria and is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk}) \tag{38}$$

The index is a measure of node purity across the K classes. A node mainly containing observations from one class will result in a small value of the Gini index as all values of $\hat{p}_{jk}$ will be close to zero or one [3].

Although classification trees are easy to understand there are also some downsides to using them. Two potential problems are the trees non-robustness and their risk of overfitting if too many splits are made [4]. In order to improve tree performance, classification trees can be aggregated many times using different methods presented in the following sections.

2.4.2 Bootstrap and Bagging

The *bootstrap method* is a resampling technique and can be used to estimate statistical uncertainty or accuracy. In order to generate new samples from the original population, the bootstrap method uses resampling with replacement. Considering again the training dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$, the bootstrapped method will repeatedly draw random samples from the dataset, with replacement. Each bootstrapped sample will obtain N observations and thus be the same size as the original training dataset. Repeating this B times, we can measure the sample mean for each of the B bootstrapped samples and estimate the overall standard error of the means [3].

Bootstrap aggregation or *bagging* is a procedure built on the concept of bootstrap, aiming to reduce the generally high variance of decision trees. The first step of bagging is to generate B different bootstrapped training datasets, as described above. The idea is to then grow a decision tree on each bootstrapped set and for each tree the model is trained, we assess the prediction $f^{*b}(\mathbf{x}_i)$ for an observation $\mathbf{x}_i$, where $b = 1, \cdots, B$. All the predictions can then be averaged in order to obtain the bagged prediction estimate

$$\hat{f}_{bag}(\mathbf{x}_i) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(\mathbf{x}_i) \tag{39}$$

To use this in a classification setting, as in this thesis, we grow a tree on each bootstrapped training dataset and record the qualitative class response made for each tree. We can then, as described in previous sections, take a majority vote and obtain the overall prediction which is the most commonly occurred class among the B different predictions [4].

2.4.3 Random Forest

Although bagging aims to reduce variance of trees, a problem could arise if the datasets contains one strong predictor along with some moderately strong predictors. Most of the bagged trees will make the same top split using the strong predictor and thus end up looking similar. As the predictions will be highly correlated, the goal of a large reduction in variance will not be fulfilled.

Random Forest is a remedy to this problem and is an improvement over bagged trees. The method decorrelates the trees by only considering a sample of m random predictors as split candidates at each split. Instead of choosing from a full set of p predictors, each split can now only choose one of m random predictors. To avoid ending up with the same options of m predictors, the method samples m new random candidates at each split. For a classification tree the relation is typically $m = \sqrt{p}$ and the algorithm is described below [3].

Algorithm 1: Random Forest Algorithm for classification

---

1. For $b = 1$ to B;
   (a) Draw a bootstrap sample $\mathcal{D}^*$ of size N from the training data by sampling with replacement
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
   i. Select m variables at random from the p variables
   ii. Pick the best variable/split-point among the m
   iii. Split the node into two daughter nodes

2. Output the ensemble of trees $\{T_b\}_1^B$.
To make a prediction at a new point $\mathbf{x}_i$:
Let $\hat{k}_b(\mathbf{x}_i)$ be the class prediction of the bth random-forest tree.
Then $\hat{k}_{rf}^B(\mathbf{x}_i) = majority\ vote \left\{ \hat{k}_b(\mathbf{x}_i) \right\}_1^B$

2.4.4 Boosting

Boosting is another technique used to improve the predictions made. Similar to bagging, a boosting model is an ensemble of many trees. The difference is that in boosting the trees can not be built in parallel as they are instead grown sequentially. Thus, the current tree learns from the previous fitted trees [4].

2.4.5 XGBoost

XGBoost, an abbreviation of eXtreme Gradient Boosting, is a powerful machine learning algorithm. XGBoost includes regularization as part of the learning objective, helping to prevent overfitting and penalizes complexity. A factor behind

the methods success is its scalability in all scenarios. The mathematics behind XGBoost follows the general idea from existing theory on gradient boosting [11].

We consider a dataset with N samples and p features, $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$. A tree ensamble model uses K additive functions to predict the output

$$\phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), f_k \in \mathcal{S} \tag{40}$$

where $\mathcal{S} = \left\{f(\mathbf{x}) = w_{q(x)}\right\} (q : \mathbb{R}^p \to T, w \in \mathbb{R}^m)$ is the regression trees' space. Here, each $f_k(\mathbf{x}_i)$ corresponds to an independent tree structure q and leaf weights w. T denotes the number of leaves in each tree and w_i represents the score for the ith leaf.

As mentioned, the models learning objective is regularized and consists of two parts

$$\mathcal{L}(\phi) = \sum_i l(y_i, \phi(\mathbf{x}_i)) + \sum_k \Omega(f_k) \tag{41}$$

where

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2 \tag{42}$$

The first term is the convex loss function, l, measuring the difference between the prediction $\phi(\mathbf{x}_i)$ and the target y_i. The other part, Ω, contains an additional term and penalizes the complexity of the model and helps avoid overfitting. Here, γ and λ are parameters that penalizes the number of leaves and the leaf weights respectively.

Considering $\phi(\mathbf{x}_i)^{(t)}$ as the prediction of the ith tree at the tth iteration, a term f_k needs to be added in order to minimize the following objective

$$\mathcal{L}^{(t)} = \sum_{i=1}^{N} l(y_i, \phi(\mathbf{x}_i)^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t) \tag{43}$$

The f_t that most improves the model is greedily added.

By using second-order Taylor approximation, we can optimize the objective quickly in the general setting:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{N} [l(y_i, \phi(\mathbf{x}_i)^{(t-1)} + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t) \tag{44}$$

with

$$g_i = \frac{\partial l(y_i, \phi(\mathbf{x}_i)^{(t-1)})}{\partial \phi(\mathbf{x}_i)^{(t-1)}} \tag{45}$$

and

$$h_i = \frac{\partial^2 l(y_i, \phi(\mathbf{x}_i)^{(t-1)})}{\partial(\phi(\mathbf{x}_i)^{(t-1)})^2} \tag{46}$$

as first and second order derivatives.

Removing the constant terms and expanding $\Omega(f_t)$ we obtain the following objective, at step t

$$\begin{aligned} \tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{N} g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \end{aligned} \tag{47}$$

with $I_j = \{i \mid q(\mathbf{x}_i) = j\}$ defined as the set of instances belonging to leaf j where $j = 1, \cdots, T$ and $i = 1, \cdots, N$.

From this equation and a fixed structure $q(\mathbf{x})$, the optimal weight w_j^* of leaf j can be computed by

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{48}$$

Finally, this equation can be used to obtain the corresponding optimal value

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \tag{49}$$

The last equation measures how good a tree structure $q(\mathbf{x})$ is. Ideally, all possible trees would be enumerated and the best one picked but this is impossible in practice. Instead, the levels of a tree are optimized one at a time. A greedy algorithm is used, which starts from a single leaf and iteratively adds branches to the tree [12].

2.5 Model evaluation

Evaluating the built models is a core part of the machine learning model process. To ensure good accuracy, feedback from continuously evaluating the model is crucial. It is also of importance when comparing different models to each other. Multiple metrics can be used to evaluate the accuracy of a model and the choice depends on the model type.

2.5.1 Confusion Matrix

A confusion matrix is a NxN matrix, with N in this case denoting the number of possible response classes. The matrix can be visualized as a table containing different combinations of the predicted class and actual class, similar to the figure below [13].

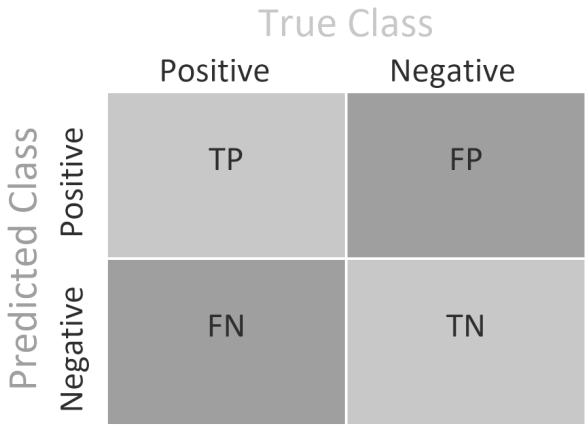

Figure 6: Confusion matrix. Source of Figure: [14]

As this thesis consists of a binary classification with classes "Conversion" and "No Conversion", the confusion matrix consists of a 2x2 matrix with outcomes:

- **TP (True positive):** The model predicts "Conversion", and the true class is "Conversion".

- **FP (False positive):** The model predicts "Conversion", but the true class is "No Conversion".

- **TN (True negative):** The model predicts "No Conversion", and the true class is "No Conversion".

- **FN (False negative):** The model predicts "No Conversion", but the true

class is "Conversion".

The matrix can be used to calculate more complex metrics, presented in the coming sections, but also basic performance metrics as:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{50}$$

and

$$Error\ rate = \frac{FP + FN}{TP + FP + FN + TN} \tag{51}$$

Accuracy is best fit to be used for balanced data. In case of a highly imbalanced dataset, all predictions could be made based on the majority class which would yield a very high, but misleading, accuracy [15].

2.5.2 Precision-Recall Curve

Two additional metrics often used to measure performance are *Precision* and *Recall* defined by

$$Precision = \frac{TP}{TP + FP} \tag{52}$$

and

$$Recall = \frac{TP}{TP + FN} \tag{53}$$

respectively.

Precision represents the proportion of predicted positive classes which were actually correct. In our case, this would answer the question: *"Among all observations predicted as "Conversion", how many were actually correct?*. Recall on the other hand measures the proportion of actual positive classes that was predicted as positive. In our case, this would answer the question: *"Among all actual "Conversion" cases, how many did the model predict correctly as "Conversion"?*.

Ideally one would like to maximize both Precision and Recall, but usually one measure is prioritized over the other depending on the models use case. To fully understand how the model is performing, Precision and Recall are examined together in a *Precision-Recall Curve*. In contrast to the accuracy presented in the above section, PR-curves works good with imbalanced data [16] [17]. An example of a PR-curve is shown in the figure below.

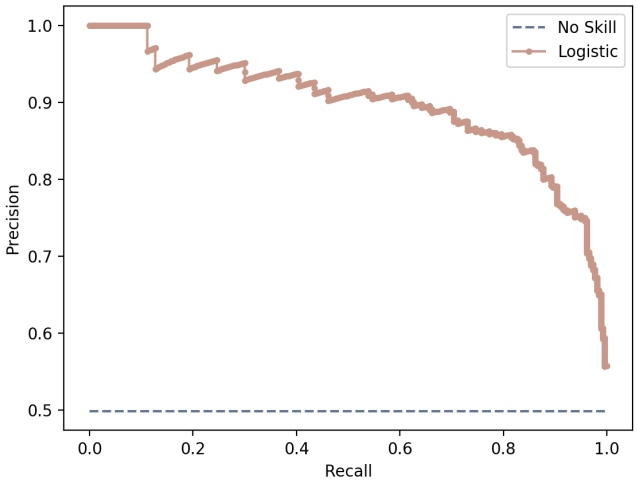

Figure 7: Example of a Precision-Recall curve of a Logistic regression model. Source of Figure: [18]

The No skill line in a PR-curve changes as the class ratio of the dataset changes. In the figure the no-skill is a horizontal line at y=0.5, i.e. the data is perfectly balanced with a 50/50 class distribution. A model that performs along the no-skill class performs as good as a random guess [19].

2.5.3 F1 Score

Built on Precision and Recall, *F1 Score* is another performance measure defined by

$$F1\ Score = 2 \times \frac{Precision * Recall}{Precision + Recall} \tag{54}$$

This score works in a way that it maintains a balance between both Precision and Recall and therefore if one of the measures are low the F1 Score will also be low. It differs from Accuracy in the sense that it works well for balancing the two metrics when also dealing with an imbalanced dataset [16].

2.5.4 ROC: Receiver Operating Characteristics

Another alternative when evaluating models is examining the *ROC*-curve. The curve is constructed by plotting the *true positive rate* against the *false positive rate* for each possible threshold, where the rates are defined by

$$True\ positive\ rate\ (TPR) = \frac{TP}{TP + FN} \tag{55}$$

and

$$False\ positive\ rate\ (FPR) = \frac{FP}{FP + TN} \tag{56}$$

From Eq. (55) above we see that the true positive rate is also known as the Recall [16].

An example of a ROC-curve is shown in the figure below. Generally, the closer a ROC curve is to the top left corner, the better the classifier.

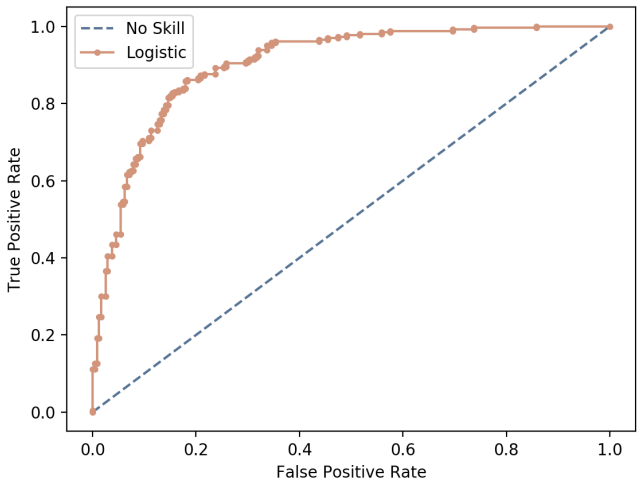

Figure 8: Example of a ROC-curve of a Logistic regression model. Source of Figure: [18]

The No skill line above is a diagonal from the bottom left corner (predict all observations as negative) to the upper right corner (predict all observations as positive). This is a classifier with no discriminative power between the two different classes. A model represented by points below this diagonal have worse than no skill [19].

2.5.5 AUC: Area Under the Curve

One challenge that can arise when using ROC curves is to plot the curve of multiple classifiers and compare them solely based on the form of the curves.

Instead, the area under the ROC curve can be calculated and used for better comparison. In the same way, the area under the PR curve can be calculated and used. This yields one single score for a classifier model across all possible thresholds. The AUC ranges in values between 0.0 to 1.0 where 1.0 is considered a perfect classifier. The figure below shows four examples of ROC curves with different AUC scores.

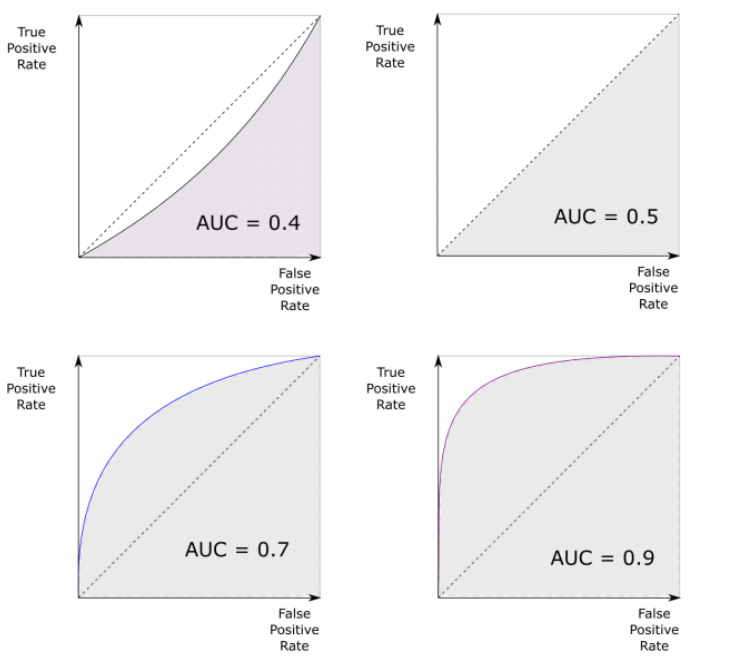

Figure 9: Four different ROC curves with their respective AUC scores. Source of Figure: [20]

A drawback of only considering the AUC value is that the metric treats TPR and FPR as equally important across all thresholds. The score alone provides no information about how the curve looks like. A ROC curve which is skewed towards the left could have the same AUC score as a curve skewed towards the right, though their characteristics suggest they perform differently. It is therefore of importance to keep the misclassification cost in mind when using this metric [21].

2.6 Cross validation

In order to avoid skewed results in the evaluation process, the machine learning models need to be tested on unseen data. Testing on unseen data provides information about how well-generalized a classifier is. One way of doing this is by *Cross validation*. In Cross validation, the data is divided into a training set, a

test set and a validation set. The idea is to use the training set to fit the model, the validation set for model selection by estimating the prediction error and finding the best model and lastly the test set solely to evaluate the performance of the best model. The test set is therefore held out until the final model is chosen.

One commonly used Cross validation method is *K-Folds Cross validation*. The idea is to randomly divide the data into k *folds*, k equally small sets of the data. One of the sets is treated as the validation set while the model is fit on the remaining $k-1$ sets of the data. Each time the validation set is predicted, the prediction error of the fitted model is calculated and the process is repeated K times. For each repetition, a different set is treated as the validation set.

An observation i is randomly assigned a partition to which it belongs and the index of that partition is given by the mapping function $\kappa : \{1, \cdots, N\} \mapsto \{1, \cdots, K\}$. Furthermore, let $\hat{f}^{-k}(\mathbf{x})$ denote the function that was fit with the kth set of the data excluded. Using this, the CV error is given by

$$CV = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-k(i)}(\mathbf{x}_i)) \tag{57}$$

where $L(\cdot)$ is the loss function.

The validation set is used as model selection with the aim of finding the best model from a set of models $f(\mathbf{x}, \alpha)$, indexed by the tuning parameter α. We denote the αth fitted model with the kth set of the data excluded as $\hat{f}^{-k}(\mathbf{x}_i, \alpha)$. The CV error is instead given by

$$CV = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-k(i)}(\mathbf{x}_i, \alpha)) \tag{58}$$

The goal is to find the optimal $\hat{\alpha}$ that minimizes the CV error above. This is more commonly known as *Hyperparameter tuning* which is done in this project and described in this thesis as part of the Methodology. After tuning and obtaining the best set of hyperparameters, the final model $f(\mathbf{x}, \hat{\alpha})$ is chosen and its performance is evaluated using the test set, as explained above [3].

To better illustrate this method, the figure below shows an example of how K-fold CV works with K = 5.

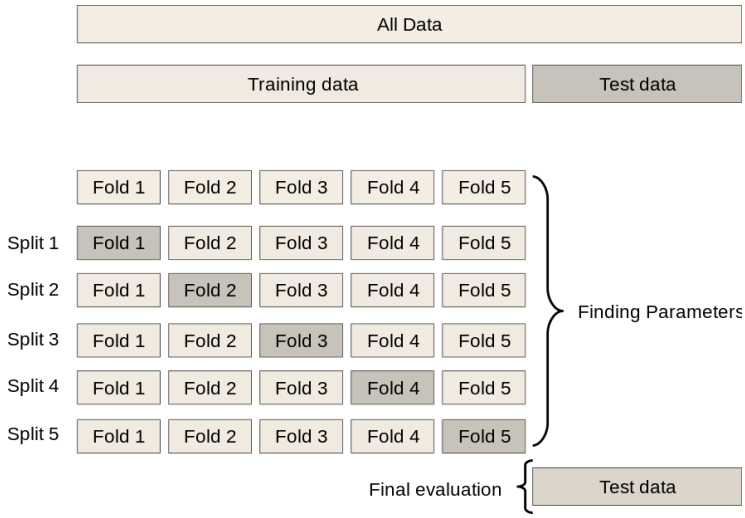

Figure 10: An illustration of the K-fold Cross validation approach with K=5. Source of Figure: [22]

2.7 SHAP: SHapley Additive exPlanations for Feature Importance

An important and valuable part of predictive modeling is being able to interpret the built model. It helps when trying to understand how each feature impacts the model and how the model can be improved. Some machine learning models, like logistic regression, are easier to interpret than others but comes with the cost of being less accurate. There is often a trade-off between model complexity and interpretability. As more complex, tree-based models are used in this thesis oen can argue that there is a need for a good model interpretation method.

SHapley Additive exPlanations, SHAP, is a relatively new framework for interpreting the output of machine learning models. It was introduced by Lundberg and Lee as a unified approach to interpreting model predictions and the method is built on game theory [23]. SHAP provides multiple explainers depending on the model built and the TreeExplainer is used in this thesis for interpreting tree-based models. One of the strong sides of SHAP is its global interpretability. The method has the ability to show the positive and negative correlation for each feature with the target variable. Another benefit of using SHAP is that it also has a good local interpretability. In contrast to traditional variable importance methods, SHAP gives each observation in the dataset its own set of SHAP values. Thus, the method can show why a particular observation led to a certain prediction.

More specifically, the SHAP method is built on coalition game theory. The

feature values are seen as players with importance values based on their contribution to the total score. To make the mathematical theory behind SHAP comprehensible, the method is simplified as follows:

We denote S as the subset of all features F, such that $S \subseteq F$. Let $v(S)$ be the total importance value of the subset S. The idea is to train a model that assigns an importance value to each feature i in the subset S. The importance value can be seen as the resulting effect on the model prediction from including the feature i. The model is trained on all subsets and if a model is trained including feature i, the value is denoted $v(S \cup \{i\})$. If the feature i on the other hand is dismissed from the model, the value will be denoted $v(S)$ instead. The marginal contribution of feature i can then be written as $v(S \cup \{i\}) - v(S)$. The resulting effect of excluding a feature i from a model is dependent on the other existing features. Therefore, the difference has to be computed for all possible subsets $S \subseteq F \setminus \{i\}$ and the resulting SHAP values are the weighted average of these differences, given by

$$\phi_i(N, v) = \frac{1}{N!} \sum_{S \subseteq N \setminus \{i\}} |S|!(|N| - |S| - 1)! \left[v(S \cup \{i\}) - v(S)\right] \tag{59}$$

Furthermore, SHAP has a fast implementation for tree-based models which makes it convenient to use in this thesis [24].

3 Methodology

In this chapter, the process of how the results are achieved is described. The chapter also aims to make the work reproducible for others.

3.1 Data description

The data used in this project is provided by Klarna Bank AB. Due to the sensitive nature of customer data I will not go into details about the variables used, but rather give an overview of the data while clearly explaining how it was processed.

The input data contains information about customers shopping sessions. As mentioned in Section 2.3, the data was limited to the Swedish market and the Klarna Checkout product. The shopping session has been defined to start when the customer reaches the Payment Selector screen and ends when the customer either completes the purchase or drops out. The full original dataset consisted of 1M rows and 23, unprocessed, columns. Three types of variables were used: categorical, numerical and booleans. The chosen variables include information about both customer characteristics and about the specific attempted order. After spending time learning about the data, the raw data was used to build some relevant point-in-time variables using Redshift SQL.

The target variables indicate whether or not a customer that starts a shopping session in the Klarna Checkout ends up converting/placing the order. A positive class corresponds to "Conversion" while a negative class corresponds to "No conversion".

3.1.1 Pre-processing

When putting together an initial dataset, the set can contain a bunch of unstructured data. Prior to training a model, the input data needs to be cleaned and processed. Different machine learning classifiers can handle different types of input data, making it important to keep in mind that pre-processing can look different for each method. As all variables in the dataset was thought through and selected based on their relevance for the thesis, the focus was put on cleaning the data rows rather than removing columns and feature selection.

Before starting the pre-processing, the original dataset is split into train and test data. This is done as the first step to prevent that information from the test set "leaks" into the training data. In such case, the model performance can risk becoming biased. When splitting the data, the ratio is chosen and set to 80/20. After the split, the cleaning of the data is done on each separate set.

Pre-processing of **numerical variables**:

- One part of data processing is deciding on how to handle missing data. Some of the numerical variables contain null values. While some machine learning methods can handle missing values themselves, others require the null values to be deleted or imputed with some other value. One common imputation that can be made is to replace the null values by the training mean for that column. Some values can be missing by random, but other null values can be dependent on other variables and have an impact on the thesis. I chose to impute the missing values with the value -9999 and will thus be able to distinguish these rows from the others. By doing so, the potential impact or meaning of a missing value can be preserved.
- Based on statistic summary for all numerical variables, outliers are identified and removed. This is done in order to avoid misleading the training process as machine learning algorithms are sensitive to the range and distribution of variables [25].
- As there is a big difference in the range of values between the different variables, all numerical variables need to be standardized. This is done using a built in Standard Scaler function which sets the mean to zero and scales to unit variance.

Pre-processing of **categorical variables**:

- The missing values in categorical columns can be handled differently. One approach is to replace them with the most frequent value in the that column, an approach usually taken when an assumption is made that the data is missing at random. Sometimes, just as for numerical data, the fact that the value is missing can itself be valuable information. In this thesis, all null values are imputed with a new category called 'Missing' in order to again preserve the meaning of a missing value.
- As the chosen machine learning models can not handle categorical input data, all categorical columns have to be *one-hot-encoded*. One-hot-encoding leads to an increase in the number of columns, as the distinct categorical values are transformed into their own columns.

3.1.2 Balancing data

The initial dataset suffers from a class imbalance. The training target variable shows that approximately 90% of the data belongs to the positive class, which corresponds to an 1:10 imbalance ratio. In the case of class imbalance, classifiers tend to make a biased learning model that gives high accuracy for the majority class but undertrains for the minority class [26].

One way to get a more balanced classification dataset is to apply undersampling on the majority class. A random undersampling was applied as the initial

dataset was considered large enough to reduce from. The more balanced ratio was set to 2:3. Training all machine learning models with the new class ratio resulted in four additional models, which could all be compared to the models with imbalanced class ratio.

3.2 Hyperparameter tuning

In the process of training a model, different machine learning algorithms contain different hyperparameters to be tuned. As its challenging to know which hyperparameters values to use in each algorithm, various grid search strategies were implemented to obtain the most optimal hyperparameters.

For Logistic Regression and SVM, the sklearn method *GridSearchCV* was used. The method performs an extensive search over parameters specified by the user. The models are evaluated for each combination of parameters, using Cross validation described in Section 2.6. Some sets of parameters are given by the user while some are set to their default values. The method evaluates the models and finds a set of optimal hyperparameters based on minimizing a loss function [22]. The loss function is predefined and was based on the PRAUC metrics described in the previous section. In other words, the optimal set of hyperparameters is the one that maximizes the PRAUC score and used in the final results to compare with other models.

For the tree based machine learning models, the sklearn method *RandomizedSearchCV* is used. In contrast to the grid search, a randomized search does not test all parameter combinations sequentially as it would be computationally too expensive. The user defines the number of different combinations to be tested, while the method itself picks out the random combinations. The models are again evaluated using Cross validation and based on the same metrics as in *GridSearchCV* [22].

Below, the sets of hyperparameters for each learning method are presented with a short description. Furthermore, the optimal hyperparameter values are represented in bold.

Logistic Regression

- **Solver:** [newton-cg', 'lbfgs', 'liblinear', 'sag', '**saga**']
  The solver states which algorithm to use in the optimization problem. The best choice can depend on the penalty chosen as well as the size of the dataset. Two examples are 'sag' and 'saga' that works faster for large datasets while 'liblinear' works well for small datasets.

- **Penalty:** ['**l1**', 'l2', 'elasticnet']
  This penalty specifies which type of regularization used. The different choices were closer described in Section 2.2.1. Not all solvers are com-

patible with all penalties. An example is the solver 'liblinear' which only supports 'l2' as a penalty, or the choice of having no penalty at all.

- **C:** [0.001, 0.01, **0.1**, 1, 10]
  This parameter determines the strength of the regularization, described in detail in Section 2.2.1. The smaller the C-value, the stronger the regularization. The parameter is thus an inverse of the regularization parameter λ.

Support Vector Machine

- **Kernel:** ['linear', 'poly', '**rbf**', 'sigmoid']
  This parameter specifies which Kernel to use in the algorithm. An explanation of a kernel was presented in Section 2.3.3 where the mathematical formulas for 'rbf' and 'poly' were given.

- **Gamma:** [0.001, 0.01, 0.1, 1, **10**, 100, 1000]
  The Gamma parameter is a positive scale parameter for 'rbf', 'poly' and 'sigmoid'.

- **C:** [100, **1000**, 10000]
  This is a regularization parameter. Again, the smaller the C-value, the stronger the regularization.

Random Forest

- **N-estimators:** [50, 100, 150, **200**]
  Determines the number of trees in the forest. In section 2.4.3 this is denoted as B. A higher number of trees yields better performance but comes at the cost of slowing down the training process considerably.

- **Max depth:** [5, 10, 20, **40**, 80, 160]
  The maximum depth of each tree in the forest. A deeper tree can lead to overfitting as they allow for more splitting and thus captures more information about the data.

- **Min samples split:** [**2**, 5, 10, 15]
  This parameter specifies the minimum number of samples required to split an internal node. If the number of samples is less than the minimum set, no split will be done and the internal node will turn into a leaf. Increasing the value of this parameter can lead to underfitting as each tree has to consider more samples at each node and thus become constrained.

- **Min samples leaf:** [**1**,2,4,6]
  This determines the minimum number of samples required to be at a leaf node. A split will only be made if it results in a child with fewer samples, otherwise the split will be avoided and the node will turn into a leaf. The same conclusion as to previous parameter, increasing the value of this parameter can lead to underfitting.

- **Bootstrap:** ['**True**', 'False']
  A boolean parameter indicating whether or not to use Bootstrap samples when building the trees. The concept behind Bootstrap was explained in Section 2.4.2. If the value is set to False, the full dataset is used to build each tree.

XGBoost

- **N-estimators:** [50, 100, **150**, 200]
  A Boosting model is an ensemble of many trees, as described in Section 2.4.4. *N-estimators* specifies the number of trees in the ensemble. Model accuracy can grow with a larger number of trees, as predictions based on a majority vote can become more reliable. However, this is not always preferred as a large number of trees can be computationally too expensive. Works in the same way as for Random Forest.

- **Max depth:** [3,5,**7**,9]
  The maximum depth of each tree in the ensemble. Works in the same way as for Random Forest.

- **Learning rate:** [0.01, **0.05**, 0.1, 0.03]
  As explained earlier, XGBoost is built on the concept of Gradient Boosting where trees are added sequentially to the model. Specifying a low learning rate makes it possible to slow down the learning in order to prevent overfitting. If the learning rate however is too low, the model may take significantly longer to train as the number of necessary iterations will increase.

- **Colsample-bytree:** [0.5, **1**]
  The subsample ratio of features when constructing each tree. The default value is set to 1, i.e. each tree uses all features available. It follows the same idea as in Random Forest where we in Section 2.4.3 presented this process as a way to remedy to correlated trees. This way of decorrelating the trees can result in a large reduction of variance.

4 Results

In the following section, the results of this project will be presented.

All models will be evaluated and compared to one another using the metrics *Precision*, *Recall*, *F1-Score*, *ROCAUC* and *PRAUC*, all defined in Section 2.5. Comparing one single metric alone comes with the risk of drawing incorrect conclusions and thus all metrics will be considered.

As described in Section 3.2 about Hyperparameter Tuning, an extensive grid search was implemented in this project to tune some hyperparameters for each model. K-folds Cross validation was used as part of the validation process in order to find the optimal set of parameters, with optimality being based on maximizing the PRAUC score. The best combination of parameter values were presented in that same section.

The performance results for each supervised learning model are summarized in the table below.

Imbalanced data					
Model	Precision	Recall	F1	ROCAUC	PRAUC
Logistic Regression	0.92	0.99	0.96	0.558	0.970
SVM	0.91	1.00	0.96	0.510	0.971
Random Forest	0.92	1.00	0.96	0.562	0.978
XGBoost	0.93	0.99	0.96	0.606	0.979
Balanced data					
Model	Precision	Recall	F1	ROCAUC	PRAUC
Logistic Regression	0.94	0.94	0.94	0.665	0.970
SVM	0.94	0.95	0.94	0.644	0.969
Random Forest	0.95	0.95	0.95	0.685	0.978
XGBoost	0.95	0.94	0.94	0.706	0.979

Table 1: Model performance metrics

To visualize the results, the ROC-curves and Precision-Recall curves for all models are plotted below. The metrics are plotted across a set of thresholds between 0 and 1.

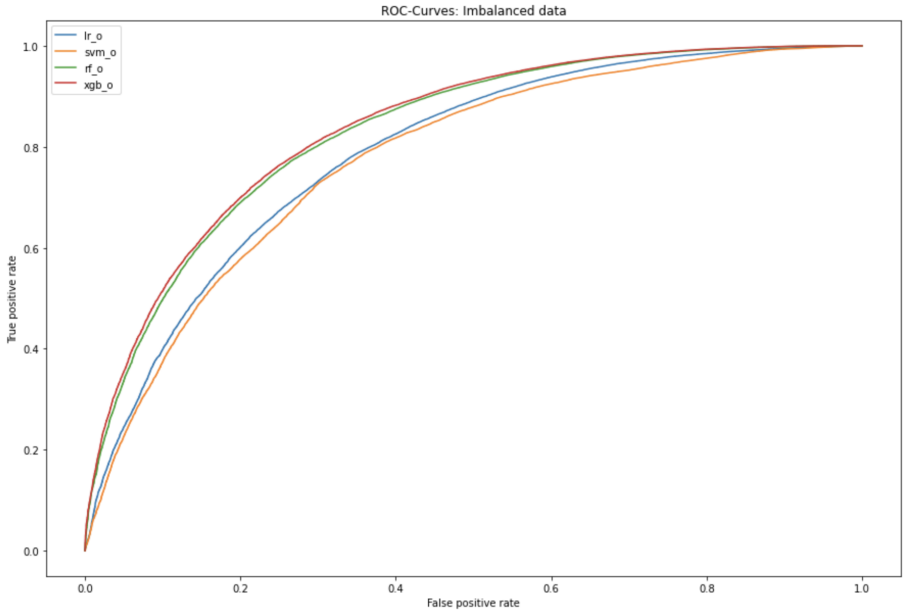

Figure 11: ROC-curves for the Test set: data not balanced

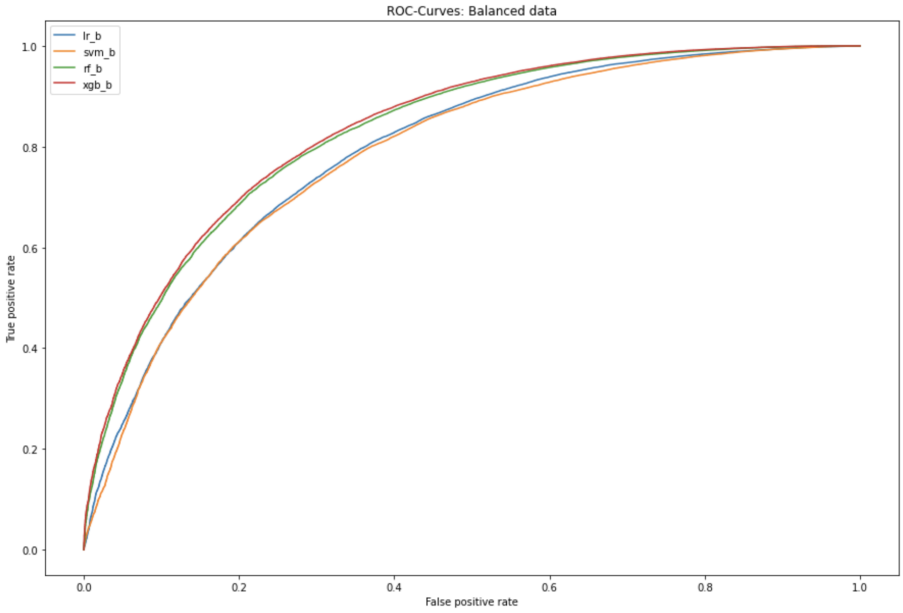

Figure 12: ROC-curves for the Test set: data balanced

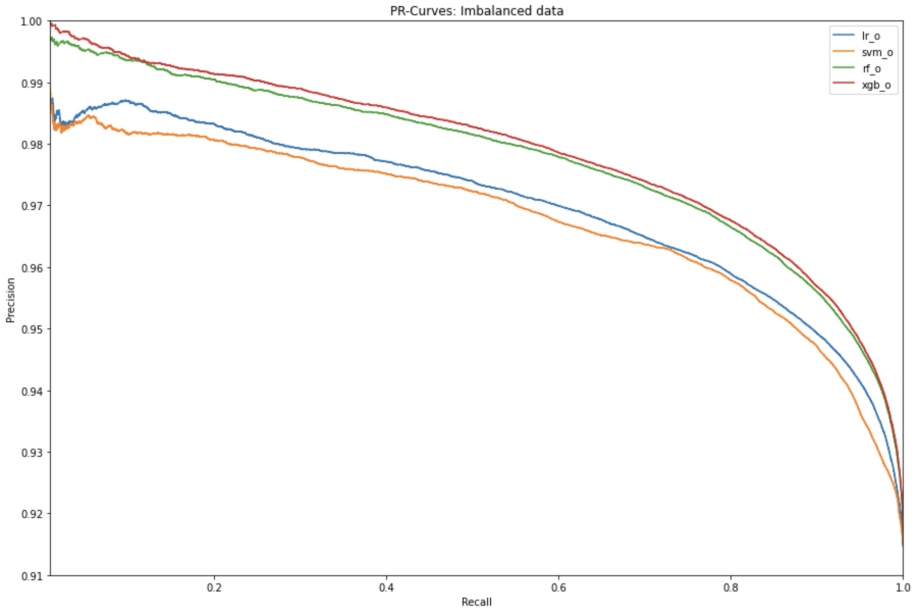

Figure 13: PR-curves for the Test set, data not balanced

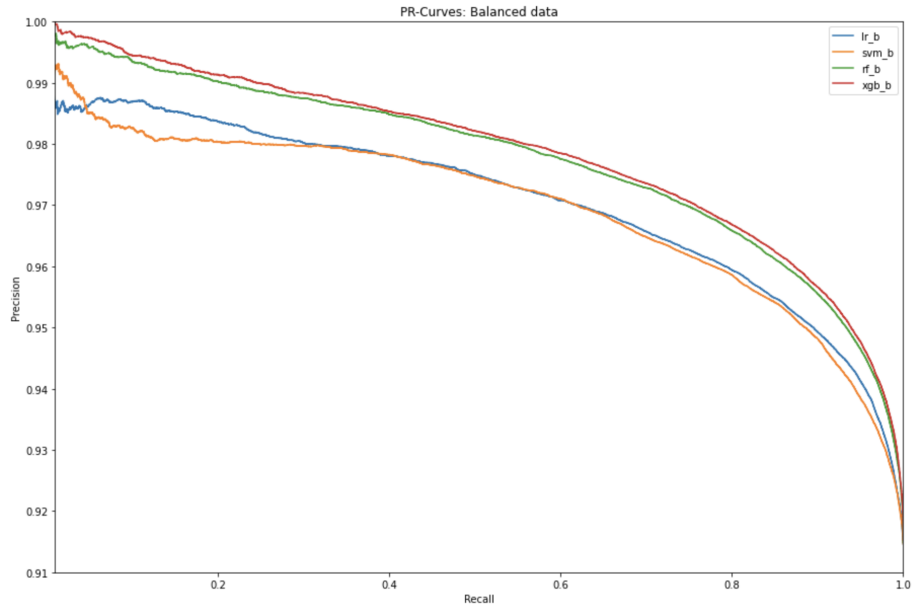

Figure 14: PR-curves for the Test set, data balanced

To get the full picture of the model performances, the macro average metrics are also presented. In addition to the True class, the macro average considers the performance if the problem was reversed and the goal was to predict the False, minority class instead. For this project, the aim would thus change into predicting if a customer with a started shopping session drops out. The macro average is found computing the metrics for both classes and taking their unweighted mean. The table below summarized the macro average Precision, Recall and F1-score for all models.

Macro average: Imbalanced data			
Model	Precision	Recall	F1
Logistic Regression	0.77	0.56	0.58
SVM	0.82	0.51	0.50
Random Forest	0.87	0.56	0.59
XGBoost	0.82	0.61	0.65
Macro average: Balanced data			
Model	Precision	Recall	F1
Logistic Regression	0.67	0.67	0.67
SVM	0.66	0.64	0.65
Random Forest	0.70	0.69	0.69
XGBoost	0.68	0.71	0.69

Table 2: Macro average metrics

Furthermore, a SHAP Feature Importance summary was plotted for the XGBoost trained model and is presented in the figure below. The y-axis contains the names of the different features used when training the model. As mentioned in Section 3.1, the nature of the customer data is sensitive and therefore the y-axis label is cropped out from the figure below. Features with large absolute Shapley values are considered valuable and important for predicting and in the figure below the features are ordered by decreasing importance.

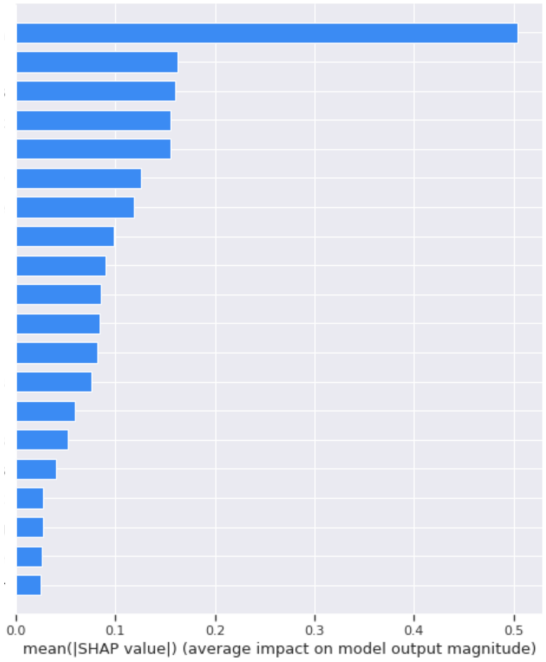

Figure 15: SHAP Feature importance summary plot for XGBoost trained model

5 Discussion

In the following chapter the results will be analyzed and discussed. Furthermore, suggestions for future work will be presented.

In order to best analyze and draw conclusions from the results in the previous chapter, it is of importance to keep the initial research question in mind. The purpose of this project was to answer the two following research questions:

- Is it possible to predict if a customer starting a shopping session will convert/place the order?
- Which Supervised learning algorithm performs best when predicting conversion with regards to specific model evaluation metrics?

The yielded results showed that it was, in the setting of this project, in fact possible to predict customer conversion with the use of Supervised machine learning. All four models had very high scores in regards to both Precision and Recall. As the F1-score is based on these two measures, the good results were reflected in that performance metric as well. The high numbers could, to the viewer, look too good to be true. It is thus important to keep some things in mind when analyzing the results. The mean of the target class is in the training data already very high. This means that, for the imbalanced data, the worst case scenario would be to predict all observations to the positive class. The Precision in that case would be equal to the mean of the target training set and the Recall would be equal to 1. The models are built with an optimization purpose and can thus only yield better results. The mean value can therefore be seen as a benchmark.

Table (1) summarizes the five performance metrics for all models. The table shows that all models are performing comparatively good and the differences in performance are minor. Separating the results into two groups, imbalanced and balanced data, indicates that the XGBoost model outperforms the rest by a small margin. This becomes more apparent in Fig (11)-(14)., as the ROC- and PR-curves show that XGBoost is strictly performing better across all thresholds. As the differences are relatively small it is also of interest to compare the macro average metric, to get another indication of which model can be favored over the others. Table (2) also confirms that XGBoost outperforms the other models with regards to the macro average metrics. This means that if the problem was reversed and we were to predict the False class, the XGBoost model would again yield the best results.

Even though XGBoost seems to perform slightly better than the remaining models, one question arises: is it worth implementing? As all models are showing good results, one could for example argue that the classical Logistic Regression is to be preferred. It is computationally cheap and very intuitive. Tree based models are more complex, computationally expensive and seen as "black boxes".

Difficulties can also be found when trying to understand the underlying theory of these complex models. This in itself could be enough to argue that it is not worth implementing XGBoost over Logistic Regression if the improvement in performance is minor. If a model would to be implemented by the company forward, the decisioning regarding what model to chose is outside of this thesis scope.

Furthermore, it is important to consider the misclassification cost when comparing. Reflecting on how the model will be used is essential when deciding on what metric should be the main performance indicator. Should one prefer a high Precision over a high Recall, or vice versa? If both measures are of equal importance, the F1-score could be enough to compare the models to each other. Depending on the end goal and the business actions that will be taken based on the prediction, one metric could be prioritized over the other. If the cost of misclassifying a session is considered devastating then the models ability to separate "Conversion" from "No conversion" will be of highest importance. That ability is best summarized in the ROC-curve together with the ROCAUC score. Many times, businesses has an already known trade-off balance between consumer preference and revenue stream. Hence, using this trade-off, there is a possibility to create a custom evaluation metric to compare different models. This is further explained in one of the future suggestions below.

Lastly, the SHAP Feauture Importance was added to the result chapter and showed in Fig (15). These results can be seen as internal results and beneficial solely to the company. It allows them to better understand customer behaviour and the weight of each variable on the predictions.

5.1 Future suggestions

If this project were to be further developed, one focus area could be expanding the training dataset. First of all, more point in time variables can be built and included in the data. The few point in time variables built in this thesis turned out to be of high importance when predicting customer conversion. Building more, similar variables could result in better predictions.

Secondly, the input variables were all handpicked based on available data and experience. Another alternative to putting together a dataset is to simply use all possible variables and let a feature selection tool instead decide on which ones are relevant to include. An example of such a tool could be *Recursive Feature Elimination*.

Furthermore, a deeper analysis on the misclassification cost could be made. Quantifying the cost of making wrong predictions would help when deciding on what evaluation metrics to prioritize over the other. The F1-score could then for example be weighted according to that misclassifciation cost. This is a lot of times referred to as "Adjusted F1-score", and defines a hybrid combination

of Recall and Precision with different weights to either the former or the latter.

References

[1] Klarna. *Klarna interim report January - June 2021*. 2021. URL: https://www.klarna.com/international/regulatory-news/klarna-interim-report-january-june-2021/.

[2] Susan Moore. *Gartner Says 25 Percent of Customer Service Operations Will Use Virtual Customer Assistants by 2020*. 2018. URL: https://www.gartner.com/en/newsroom/press-releases/2018-02-19-gartner-says-25-percent-of-customer-service-operations-will-use-virtual-customer-assistants-by-2020.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2001.

[4] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer Series in Statistics. Springer New York Inc., 2013.

[5] Thomas P Ryan. *Modern regression methods*. Vol. 655. John Wiley & Sons, 2008.

[6] Towards Data Science. *Logistic Regression Explained*. 2020. URL: https://towardsdatascience.com/logistic-regression-explained-9ee73cede081.

[7] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[8] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. Chapman and Hall/CRC, 2019.

[9] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Computation and Machine Learning Series, 2018.

[10] Bei Zhou et al. "Analysis of Factors Affecting Hit-and-Run and Non-Hit-and-Run in Vehicle-Bicycle Crashes: A Non-Parametric Approach Incorporating Data Imbalance Treatment". In: *Sustainability* 11 (Mar. 2019), p. 1327.

[11] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[12] Tianqi Chen and Carlos Guestrin. *Introduction to Boosted Trees*. 2016. URL: https://xgboost.readthedocs.io/en/stable/tutorials/model.html.

[13] Sarang Narkhede. *Understanding Confusion Matrix*. 2018. URL: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[14] Joydwip Mohajon. *Confusion Matrix for Your Multi-Class Machine Learning Model*. 2020. URL: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826.

[15] Rahul Agarwal. *The 5 Classification Evaluation metrics every Data Scientist must know*. 2019. URL: https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226.

[16] Takaya Saito and Marc Rehmsmeier. "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets". In: *PloS one* 10.3 (2015), e0118432.

[17] Paula Branco, Luis Torgo, and Rita Ribeiro. "A survey of predictive modelling under imbalanced distributions". In: *arXiv preprint arXiv:1505.01658* (2015).

[18] Jason Brownlee. *ROC Curves and Precision-Recall Curves for Imbalanced Classification*. 2020. URL: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/.

[19] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 233–240.

[20] Deparkes. *The ROC curve*. 2018. URL: https://deparkes.co.uk/2018/02/16/the-roc-curve/.

[21] Steve Halligan, Douglas G Altman, and Susan Mallett. "Disadvantages of using the area under the receiver operating characteristic curve to assess imaging tests: a discussion and proposal for an alternative approach". In: *European radiology* 25.4 (2015), pp. 932–939.

[22] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[23] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[24] Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions". In: *Proceedings of the 31st international conference on neural information processing systems*. 2017, pp. 4768–4777.

[25] Mayank Tripathi. *Knowing all about Outliers in Machine Learning*. 2020. URL: https://datascience.foundation/sciencewhitepaper/knowing-all-about-outliers-in-machine-learning.

[26] Wanwan Zheng and Mingzhe Jin. "The effects of class imbalance and training data size on classifier learning: an empirical study". In: *SN Computer Science* 1.2 (2020), pp. 1–13.