Customer Churn Prediction using Quotation Data

Master Thesis

Submitted on: January 4, 2022

at the University of Cologne

Name:	Abdurahman Maarouf
Adress:	Schulstrasse 31
Postcode, Area:	53332, Bornheim
Country:	Germany
Matriculation number:	736481
Supervisor:	Prof. Dr. Dominik Wied

Contents

1 Introduction

Predicting customer churn in order to retain customers has become one of the most important issues for companies. The goal is to estimate probabilities for a costumer churning in the next period of time, in order to be able to detect potential churners before they leave the company. To tackle this issue, more and more advanced Machine-Learning-Algorithms are used guaranteeing high accuracy in their out-of-sample predictions.
Fortunately for most of the companies, churn rates from one period to another are very small. However in classification models predicting a rare event can become challenging. In this so called "Imballanced Classes" issue certain arrangements to the underlying training data need be made. Without these arrangements and with highly imballanced classes, a poor algorithm will simply never predict the outcome of the minority class. In a dataset with 1000 customers containing 5 churners for example, this loss-minimizing algorithm would have an in-sample accuracy of 99.5%.
In order to avoid the high amount of "False-Negative" classifications there are many methods ranging from upsampling the minority class or downsampling the majority class to more advanced techniques. In this work we will present and compare the different methods while applying them to the underlying problem.
We also want to emphasize (or not) the importance of using quotation data for predicting customer churn. A company can track (potential) customer behavior on their distribution channels. Nowadays, in most cases the products or services are offered online on websites, which makes it easy to track website visitor data. In the context of dealing with customer churn this data can be matched to the customers already having a product or contract of this company. We believe (?) that the number of visits of a current customer in the last period (?) plays a big role in predicting the probability of that customer leaving in the next period. (Coming from high correlation between Nvisits and churn)
In order to evaluate the importance of not only the number of website visits but also the other explanatory variables there is typically a trade-off during model selection. The trade-off is between the model complexity or corresponding accuracy and the model interpretability. Deep neural networks or boosted trees belong to the complex models which are famous for their high accuracy in the fields of computer vision and natural language processing. Understanding and interpreting the model is of no big interest in these areas. However in the topic of this work and in many other areas understanding which variables lead to the resulting outcome of the model becomes desirable. The most transparent models in terms of interpretability are linear or logistic models. There the magnitude and sign of the corresponding coefficients (after being testet for significance) illustrate the changes of the outcome for a change in the specific explanatory variable. These models however lack in terms of accuracy when being compared to the comlex ones. In this work we will present the accuracy and interpretability of "Explainable Boosting Machines" developed by (?) for predicting customer churn. It aims to combine the high accuracy of complex models on the one hand and the interpretability of linear models on the other hand.

2 Data and Methodology

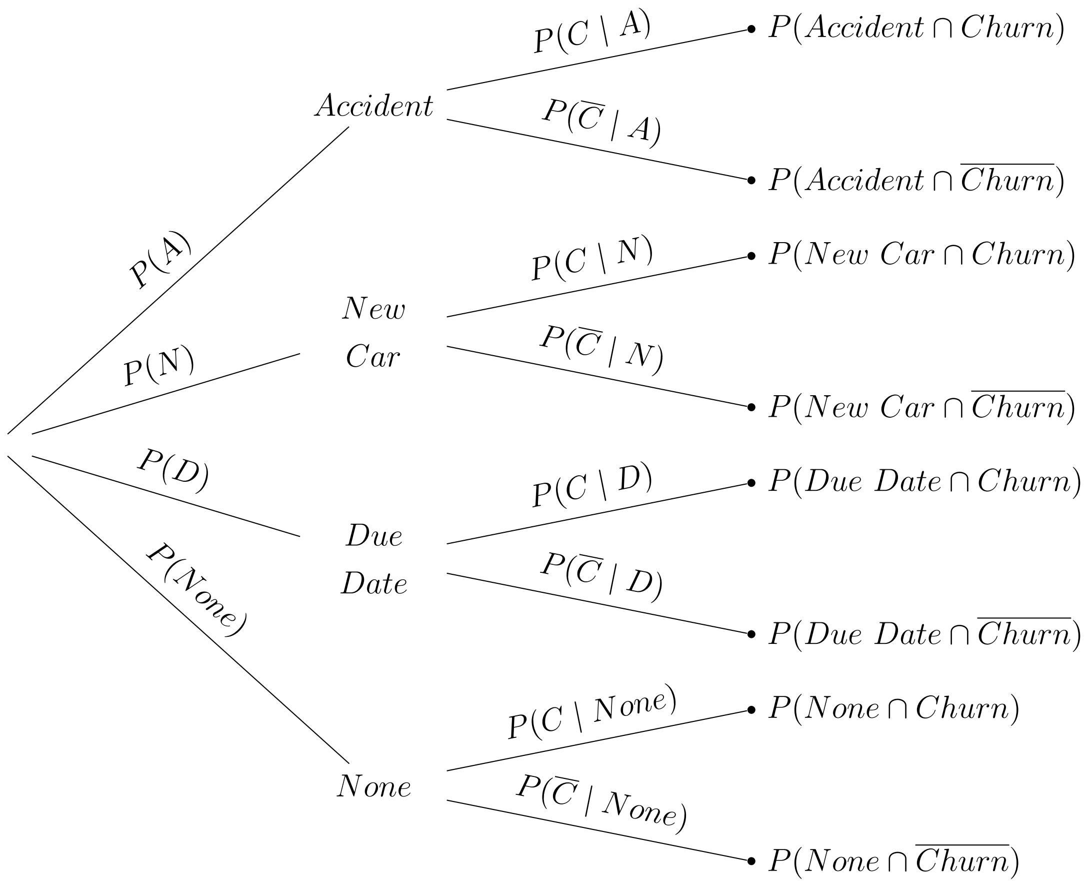
2.1 Understanding the Problem

For this work we use customer data from a big insurance company in Germany. Due to data pretection the data is anonymized which does not affect the model accuracy and interpretability in any form. We focus on the product of automobile liability insurance, which is by law a mandatory service every car owner must hold in Germany.
Typically car owners close a deal with an insurance company which can be terminated by the end of each year. In rare cases both sides agree on a contract with a due date during the year. If the contract does not get terminated it is automatically extended for another year. Besides the option to terminate the contract at the due date there is also an option to terminate it earlier in a few special cases. These cases mainly involve car accidents and vehicle changes of the contractor. To sum up, here are the three cases in which a churn can (but not must) occur during a year:

Event A:	Contractor is involved in an Accident.
Event N:	Contractor buys a new Car.
Event D:	Due date during the year.

The problem of modelling customer churn needs to be seperated into the probability of a costumer leaving during the year and at the end of a year (why noch ausführen). In this work we will focus on predicting churns occuring during the year. The purpose is to build a model which can be used at any time t of the year besides on January the 1st to predict the probability of a costumer leaving the company in the next period of time $(t, t+s]$.
It can be argued that in order to provide a model with maximized utility for production one would want to keep s small. For example a company would highly benifit from a model, which can predict the churn-probability of tomorrow or the next week. However we will see that having a small s will decrease the accuracy of our models (drastically?), creating a trade-off situation between model accuracy and the benifits of a small s. With a smaller period the classes of the data become more imballanced, creating a higher challange of preprocessing the training data and feeding the algorithm enough information on potential churners. Furthermore, a small s decreases the scope of action for a company to retain potential customers leaving.

Figure 1 (richtiger Verweis) illustrates how the probability of a churn during the year can be decomposed using the Events A (Accident), N (New Car), D (Due date during the year) and C (Churn).



By assumption we set $P(C \mid None) = 0$ as the amount of terminated contracts during the year which are not being caused by a new car, an accident or a due date is very small and can be omitted. Therefore we leave these cases out of our data (?). Also, the probability $P(D)$ can only take values 0 and 1, as either the due date of a customer lies in the next period of time $(t, t+s]$ or not. What we are interested in predicting is the overall probability of a churn, which can be rewritten as: (Change this to a formula with a number next to it?)

$$\begin{aligned} P(C) &= P(A \cap C) + P(N \cap C) + P(D \cap C) \\ &= P(A)P(C \mid A) + P(N)P(C \mid N) + P(D)P(C \mid D) \end{aligned}$$

One idea would be to model the three branches of A, N and D seperately. The logic behind this is that different models and sets of explanatory variables have the best fit for the probabilities of the three branches. Not only the the three branches but also the unconditional and conditional probabilities of a single branch may vary in their best modelling approach. For prediciting an accident a model of type (XY) is more suitable, whereas (YZ) would go better with modelling the probability of churn given an accident occured. In the course of this work we will begin with a general modelling approach trying to predict the probability $P(C)$. At a later stage we will compare the accuracy outcomes of seperately predicting the branch probabilities with the baseline approach.

2.2 Data

Most customer churn predicition models have been using only one timestamp to train a model. We want to emphasize the usage of multiple timestamps and show that it significantly improves prediction accuracy. The improvement is a result of an increased training set size and the ability of the model to especially learn more about the rare class. Furthermore, the model becomes more generalizable in time which is crucial if this model is applied in production. (see: How training on multiple time slices improves performance in churn prediction). Explain more. Use Graphics like in the paper.
Part of this work will also be to evaluate if the churn probability of customers is time invariant. Therefore we statistically test the equality of monthly and yearly (and weekly?) mean churn rates using the ANOVA (or other? seasonality tests). We will see that it is time variant (or not?) which underlines the importance of including (or not) different timestamps containing different years/months(/weeks) in the data for the model to learn the time differences (or not). (These will simply be appended to the data as additional rows creating a panel dataset.)
Therefore to build the models we use historical data of the insurance company. More specifically we pick N (one or many?) timestamp(s) t_i with $i = 1, ..., N$ in the past and collect all the active contracts to that(these) timestamp(s). One row corresponds to one active contract at t_i. So if one hypothetical contract is active in all N periods it will appear as N seperate rows in our data. To each row we merge the status of that contract in $t_i + s$. Furthermore, we join the number of requests corresponding to that contract in the period $[t_i - m, t_i]$. The period length m will be another parameter to tune.
Illustrate graphically like in Gattermann?
Describe the ETL Process?

-Statt "STORNO" im data-load sql query zu definieren, definieren wir es einfach in dem Feature-Engineering Teil?
-Test if customer churn probability is time invariant or time variant!!! Is it enough to argue with the Aggregate Churn rate? Show statistically (Voll gute Idee)
-How do I exclude 1.1. churners in my data? Exclude the entire contract or simply say y=0 as he didnt churn during the year?
-Safe noch als Variable einbauen, was sein Preis bei einer jetzigen Berechnung wäre (und dann Differenz zwischen Preis, den er zahlt und Preis den er angezeigt bekommt!)

3 Modelling Approach

3.1 Literature Review

see Gattermann et al. 2. Literature Review to get an idea on how to do this Literature Review.
Put an introduction in to the next subsections. Say sth like "there is a wide range of literature on classification techniques and specifically on customer churn predicition. Baseline: Logistic Regression. They propose following advanced techniques. In chapter 3.4 we are going to focus on X, Y and Z"
Also include introductory words for the other subsections.

3.2 Preprocessing

Put the ETL process here? Including multiple time stamps explanation here instead of Chapter 2?

3.3 Handling Class Imbalance

Studying the rarity of an event in the context of machine learning has become an important challange in the recent two decades. Rare events, such as a customer churning in the next period, are much harder to identify and learn for most of the models. HaiYing Wang et al (Logistic Regression for Massive Data with Rare Events) study the convergence rate and distributaional properties of a Maximum-Likelihood estimator for the parameters of a logistic regression while prediciting rare events. Their finding is that the convergence rate of the MLE is equal to the inverse of the number of examples in the minority class rather then the overall size of the training data set. So the accuracy of the estimates for the parameters is limited to the available information on the minority class, even if the size of the dataset is massive.
Therefore some methods have been developed to decrease the problematic of imballanced classes. In this chapter we will present (three?) different methods which can be applied to the training set, before feeding it to the model. To handle and evaluate the outcomes of prediciting rare events also the appropriate models and model evaluation metrics must be chosen. This will be discussed in the next two chapters.

(i) Downsampling

The first basic sampling method is named downsampling. It randomly eliminates examples from the majority class in order to artifficially decrease the imballance between the two classes. The downside of this approach is that it possibly eliminates useful examples for the model to maintain a high accuracy in predicting the majority class (Mining with Rarity: A Unifying Framework). HaiYing Wang et al (Logistic Regression for Massive Data with Rare Events) also study the convergence rate and distributaional properties when applying downsampling. According to their findings the asymptotic distribution of the resulting parameters may be identical to the MLE's using the full data set. Under this condition there is no loss in terms of efficiency (minimum possible variance of an unbiased estimator devided by its actual variance).

(ii) Upsampling

The second basic sampling method is the upsampling approach. This method simply duplicates examples from the minority class until the classes are more balanced. While duplicating examples though, the chances of overfitting to these duplicates becomes a more probable threat. Also, no new data is being generated in order to let the model learn more valuable characteristics about the minority class (Mining with Rarity: A Unifying Framework). Additionally, the computational performance of this approach can get rather poor, espacially with large datasets and highly imballanced classes. While evaluating the asymptotics of the MLE's with upsampling, HaiYing Wang et al find out that it also decreases the efficiency. A probable higher asymptotic variance of the estimators is the reason for that.

(iii) SMOTE

The more advanced SMOTE-approach (Synthetic Minority Oversampling Technique) (SMOTE: Synthetic Minority Over-sampling Technique) also creates more artifficial examples of the minority class. Instead of simply duplicating some rows SMOTE creates new nearest neighbors in terms of feature values for the minority class examples. While constructing the feature values of the new example $(n + 1)$ (where n is the size of the unsampled dataset) as a new nearest neighbor for example i of the minority class one has to differentiate between continuous and nominal features. The k-nearest neighbors for the minority class are typically constructed with the Euclidean Distance for continuous features and the Value Distance Metric for nominal features (maybe include other distance measures here? Mahalanobis Distance?).
A. Continuous features:
A.1) Construct difference between corresponding feature value of example i and one of its k nearest neighbors.
A.2) Multiply this difference with a random value drawn from a uniform distribution between 0 and 1.
A.3) Construct the feature value of the new example by adding the multiplied difference to the feature value of example i.

B. Nominal features:
B.1) Choose the feature value which is the majority vote between the feature value i and its k nearest neighbors.
B.2) Assign this value to the corresponding feature of the new example.
With this approach it is ensured that the model learns more about the neighborhood regions of the minority class. It decreases the probability, that the model overfits to the duplicates created in upsampling.

(iv) Cost-Sensitive Classifiers

One drawback of the presented sampling methods is that the sample distribution is being changed. A different starting point is to change the objective cost function which will be minimized during the model estimation. Thereby the sample size and the distribution stay the same. In the modified cost function we want to penalize false-negative classifications with a higher weight. Reason behind this is to ensure that the event of interest, represented by the minority class, is predicted correctly.
(How does our cost-function look like then? Gattermann use balanced class weights which are inversely proportional to the class frequency and thereby assign more weight to samples from the minority class.) (or cite: Cost-sensitive learning methods for imbalanced data)
The challange of this approach is to find the appropriate penalty parameters. It is hard to measure the cost of misclassifying an insurance-customer regarding churn-probabilities.The fact that these costs can come from multiple sources which are not easily definable is one part of the reasoning. Therefore we will focus on two types of modified loss functions.
One conventional method is to assign simple parameter weights $w0$ and $w1$ to the loss function. By default one might use the inverse of the class frequency as the associated weights in the cost function. These parameters however can also be optimized during grid search. We will call this method the "Simple Weighted Loss Function". The derivation will be part of the next subchapter.
The second approach is called "Focal Loss Function" and was originally designed by FAIR (Facebook Artifficial Intelligence Research) for an object detection purpose (Focal Loss for Dense Object Detection). As it also aims to penalize false-negative classifications it is widely used for predicting rare events. Again, the exact structure will be shown in the next subchapter and the parameters can be tuned with grid search.

3.4 Machine Learning Models and Interpretability

(i) Logisitic Regression

The logistic regression is used as a benchmark model for all classification problems. It has high advantages in terms of computational complexity and interpretability, but fails in capturing complex relationships and interactions. As in most cases it has the lowest accuracy of all classifiers we use it as a baseline model in order to evaluate the advantages of additional complexity in the models, as in (Random Forests,) Boosted Trees and Explainable Boosting Machines. Thereby we focus on model performance on unseen data and model interpretability.
Logisitic regression applies a sigmoid function to the input characteristics in order to get values (probabilities) between 0 and 1 as a Bernoulli distribution. We will use the following conventional notation throughout this work:

$$\widehat{C_{t+s}} = P(C_{t+s} = 1 \mid X_t) = f\Big(g(X_t)\Big) = \frac{e^{g(X_t)}}{1 + e^{g(X_t)}} \tag{1}$$

Note that from now on we use a vectorized notation. So $\hat{C}$ represents a vector of probabilities, where the i'th element ($i = 1, ..., N$) corresponds to the probability of a churn in time $t + s$ of the same at t active contract. This probability vector is constructed by a model conditional on a characteristic matrix X at time t. Again, each row of X represents a different contract and column j is the characteristic j ($j = 1, ..., M$) of that contract.
Therefore $f(\cdot)$ is the sigmoid function used on function $g(\cdot)$, where in logistic regression $g(\cdot)$ is a linear model. For Gradient Boosting and Explainable Boosting Machines this function is allowed to be more complex.

$$g(X_t) = \beta X_t + \epsilon_t \tag{2}$$

Maximum Likelihood is then used to estimate the vector of parameters β of the model. It maximizes the joint probability that the status of the training data contracts are drawn from the Bernoulli distribution stated in (1). For the final model in production, when estimating probabilities for a churn in the next period, t is of course equal across all contracts in X. But as already stated, for training the model we use multiple timestamps of the historized data. Therefore the index t is left out for the input matrix X and output vector C while estimating model parameters, avoiding a misleading notation. Still, the status of contract i is determined after s units of time for all contracts. The likelihood function to be maximized is described as follows:

$$L(C;X,\beta) = \prod_{i=1}^{N} P\Big(C^{(i)} = 1 \mid X^{(i)}\Big)^{I(C^{(i)}=1)} \Big(1 - P\Big(C^{(i)} = 1 \mid X^{(i)}\Big)^{1-I(C^{(i)}=1)}\Big) \tag{3}$$

To describe this term as loss function it is characterized by the negative log of (3), which can be minimized with respect to β using Gradient Descent, Newton Raphson (Zitieren!) or other optimization proceedings.

$$\begin{aligned} l(C;X,\beta) &= -\sum_{i=1}^{N} C^{(i)} log\Big(P(C^{(i)} = 1 \mid X^{(i)})\Big) + (1 - C^{(i)}) log\Big(1 - P(C^{(i)} = 1 \mid X^{(i)})\Big) \\ &= -\sum_{i=1}^{N} C^{(i)} log(\widehat{C^{(i)}}) + (1 - C^{(i)}) log(1 - \widehat{C^{(i)}}) \end{aligned} \tag{4}$$

In our case we have a lot (how many?) of characteristics in the raw matrix X. Consequently it makes sense to add a penalty term to the loss function to avoid overfitting to the training data. While minimizing this loss function during training, this additional penalty term either shrinks some parameters of β towards zero (L2 regularization), or sets some to exact 0 (L1 regularization). However, we can also reduce the dimension of characteristics prior to fitting the model. It will be discussed at a later stage of this work. Additionally, as already mentioned in (3.3), one technique to handle imbalanced classes is to use cost-sensitive classifiers. We can implement this by setting additional penalty weights to our loss function. The resulting objective "Simple Weighted Loss Function" is described as:

$$L(C,\widehat{C}) = -\sum_{i=1}^{N} w_1 C^{(i)} log(\widehat{C^{(i)}}) + w_0(1 - C^{(i)}) log(1 - \widehat{C^{(i)}}) + \lambda h(\beta) \tag{5}$$

Here, the penalty term $h(\cdot)$ can either be an L1 or L2 regularization on β. The additional parameters of λ, w_0 and w_1 are hyperparameters which can be tuned during grid search. Following (whose?) approach, w_1 and w_0 can also be set by default to the inverse of their corresponding class frequencies in the training data.
The objective "Focal Loss Function" can be set up like this:

$$L(C,\widehat{C}) = -\sum_{i=1}^{N} \alpha(1-\widehat{C^{(i)}})^{\gamma} C^{(i)} log(\widehat{C^{(i)}}) + (1-\alpha)\widehat{C^{(i)}}^{\gamma}(1-C^{(i)}) log(1-\widehat{C^{(i)}}) + \lambda h(\beta) \tag{6}$$

After having found the optimal set of parameters it is time to define the classification cutoff τ for predicting a churn. The logistic regression model outputs a probability which is between 0 and 1. The rule of thumb is to set $\tau = 0.5$, such that probabilities larger

than 0.5 are predicted as churns and the rest as non-churns.
Interpretability.

(ii) Tree-based Classifier

Single Classification Tree

Classification trees have a different approach on builing a prediction model for $\widehat{C_{t+s}} = P(C_{t+s} = 1 \mid X_t)$. Unlike logistic regression, they allow for non-linearities and interactions. Classification trees search for the optimal sequential binary sample splits in order to minimize an objective loss function. So at each node of the tree the optimal characteristic j and its optimal split point r need to be found. The search at each node can be summarized as follows:

$$\begin{gathered} S_1(j,r) = \{X \mid X^j \geq r\}, S_2(j,r) = \{X \mid X^j < r\} \\ \{j,r\} \in \min_{j,s} \sum_{i:X^{(i)} \in S_1(j,r)} l(C^{(i)}; X^{(i)}, \widehat{p^k}) + \sum_{i:X^{(i)} \in S_2(j,r)} l(C^{(i)}; X^{(i)}, \widehat{p^k}) \end{gathered} \tag{7}$$

The loss function for region S_k is calculated using the resulting region prediction $\widehat{p^k}$ (applied to all obeservations which are assigned to this region after the splits $i : X^{(i)} \in S_k$), which are simply the shares of churns in S_k. The typical loss functions are based on evaluating the purity of the resulting regions. In the ideal case, one would like to find the splits in X which always correctly assign contracts of the two classes into two different regions. In this case, $\widehat{p^k}$ would always be either 1 or 0. In order to approach this case one either uses the Gini-index or the Cross-entropy for region S_k:

$$\begin{array}{ll} \text{Gini-index:} & 2\widehat{p^k}(1-\widehat{p^k}) \\ \text{Cross-entropy:} & -\widehat{p^k}log(\widehat{p^k}) - (1-\widehat{p^k})log(1-\widehat{p^k}) \end{array}$$

The splits at each node are performed until certain criteria for the loss functions are met (which can be tuned by the hyperparameters and are summerized in a table inshaallah). To get the prediction $\widehat{C^{(i)}}$, one assigns $\widehat{p^k}$ as the region prediction of the corresponding end node of observation i to that value.

Boosted Trees for Classification

In most applications the predictions of a single classification tree have high variance due to overfitting to the training data. The accuracy in the prediction on unseen data will therefore be rather poor compared to the accuracy during training. Random forests (Breiman 2001) are designed to reduce the variance of the predictions, by averaging over predictions of multiple independent trees. This phenomenon is called "Wisdom of the Crowds" and is underlined by the fact that the average of multiple estimators for the same parameter will have the same bias but a smaller variance (how much depends on N and the correlation (which is 0 if we assume that the base learners are independent) of

the estimators) than a single estimator.
The current literature (cite!!!) however agrees on the fact that Boosted Tree Algorithms outperform Random Forests, and therefore also a single tree. Therefore we will focus on the core methodology and hyperparameters of boosted trees in the following.
The idea of boosted trees for classification is to have a sequence of dependent base learners (trees), which improve in terms of accuracy where it is most needed. Thereby, the base learners $\hat{g}_b(X)$ $(b = 1, .., B)$ are constructed in a shallow shape, avoiding an overfitted single learner. Additionally the $\hat{g}_b(X)$ are now regression trees, which output values in $(-\infty, \infty)$. So for the optimal split at the nodes of each tree the squared error reduction (of the log-odds?) are now being compared. To get an interpretable result we again have to apply the sigmoid function $f(\cdot)$ to get a probability in $[0, 1]$. The resulting loss of each tree is then fitted by the preceeding trees. So the procedure can be summarized by these steps:
1. Set first fit to $\hat{G}(X) := 0$ and loss to $L := \sum_{i=1}^{N} l(C^{(i)}, f(\hat{G}(X^{(i)})))$
2. For the sequence of trees $b = 1, .., B$ repeat:
2.1 Fit shallow tree $\hat{g}_b(X)$ which minimizes the Loss $L = \sum_{i=1}^{N} l\Big(C^{(i)}, f\Big(\hat{G}(X^{(i)}) + \hat{g}_b(X^{(i)})\Big)\Big)$
2.2 Update fit to: $\hat{G}(X) := \hat{G}(X) + \lambda \hat{g}_b(X)$
3. Final prediction is then: $\hat{C} = \hat{f}(X) = f(\sum_{b=1}^{B} \lambda \hat{g}_b(X))$

Gradient boosted bagged trees noch?

The hyperparameter λ is called learning rate and set to a small number to learn slowly and to avoid overfitting. Also notice, that the loss function defined here is the objective loss function used for gradient boosting and is defined differently than the loss function, when constructing a single tree. We will compare the two objective loss functions for Gradient Boosting derived in (5) and (6) with different hyperparameters. This allows us to handle the class imbalance. We will leave out the regularization term $h(\cdot)$ in this case. To apply the log-loss function in (5) or (6) we need to deliver the gradient as well as the hessian of this function. Herewith the lightgbm-package again aims to improve time-cost as is it takes the second order Taylor Expansion to compute the optimal loss minimizing $\hat{f}_b(X)$ in each iteration $b = 1, ..., B$. These are calculated in the appendix.
The main cost with boosted trees is learning the single trees, and the most time-consuming part in that is to find the best split-points. Therefore we will use the package lightgbm in our prediction model, developed by (XY), which aims to tackle this issue. The so called histogram-based approach is to bin the features which creates high efficiency gains during searching for the optimal splits, especially for continuous variables.
Another important aspect to note about lightgbm's implementation of boosted trees is the fact that the base trees grow leaf-wise instead of depth-wise. As proposed by Friedman et al (Friedman et al, 2000), the idea is to extend nodes in first-best order. Thereby, the best node is the one which maximally reduces the sum of the tree loss function in the

Table 1: Hyperparameters for lgbm

	Hyperparameter	Description	Grid Search Values
Boosting	learning rate	Boosting learning rate λ	Shrinking learning rate
	n_estimators	Number of boosted Trees B	
	objective	Objective or objective loss function	Custom loss function
	subsample	Subsample ratio of training data used	
	subsample_freq	Frequency of usage of subsample	
	reg_alpha	L1 regularization term on weights	
	reg_lambda	L2 regularization term on weights	
Single Tree	num_leaves	Maximum number of tree leaves	
	subsample_for_bin	Number of samples for constructing bin	
	min_split_gain	Minimum loss reduction for split	
	min_child_samples	Minimum number of data in new region	
	colsample_bytree	Subsample ratio of columns for tree	

resulting regions. This enables the possibility that a first-best split is found at a node in level $r+1$ eventhough in level r only one of two nodes are split. (Show graphically?)
If we let the trees grow without tree structure restrictions to full trees, both the leaf-wise and depth-wise approach will result in the same trees. As all gradient boosters rely on shallow trees, the combination of hyperparameters such as the maximum number of leaves, the minimum loss reduction required for a further split and first-best split approach result in a better (cite!!!) shallow tree structure.
What is regalpha and reglambda in boosted trees?
There are a lot of hyperparameters which allow to tune the learning structure. They can be split into the hyperparameters for single tree structure and hyperparameters for the learning process. In Table (3.4) is an overview of the hyperparameters we will focus on and the corresponding values we will use for grid search.

(iii) Explainable Boosting Classifiers

Explainable boosting machines (EBM) were developed by researchers at Microsoft (InterpretML: A Unified Framework for Machine Learning Interpretability) and aim to have high accuracy as most complex models, while maintaning an easy interpretable approach. It is maintained by the functional form which is similar to the standard GAM-Model, developed originally in 1987 by Hastie and Tibshirani (Generalized Additive Models):

$$g(X_t) = \beta_0 \sum_{j=1}^{J} k_j(X_t^j) + \epsilon_t \tag{8}$$

Thereby J is the number of characteristics and $k_j(\cdot)$ is the characteristic function for characteristic j. In standard GAM-models $k_j(\cdot)$ is approximated by scatterplot smoothers (Wenn noch Zeit und Platz ist, Running mean oder running lines smoother erklären, aber eigentlich unnötig wallah.). In EBM the approximation for $k_j(\cdot)$ is allowed to be done by a complex modern learner such as a decision tree or a random forest.
As the name of the method already reveals, gradient boosting is applied to improve the preceeding trees in direction of the gradient of the loss function. The main difference to the standard gradient boosting classifier is that no longer one tree fits all characteristics $\hat{g}_b(X) = \hat{g}_b(X^1, ..., X^J)$ in each iteration $b = 1, .., B$. Instead in each iteration, the algorithms cycles through all characteristics to improve the single characteristic function $k_j(\cdot)$, where it is most needed. By cycling through the features one at a time and maintaining a low learning rate the feature order does not matter and the potential problem of multicollinearity is reduced.

Unfortunetaly there is still a huge gap between standard GAM-models and full complexity models in terms of predictive power, even if we allow for non-linearities in $k_j(\cdot)$. Full complexity models, such as gradient boosted trees and deep neural nets, have the advantage of allowing for an arbitrary number of interactions between features. To allow for pairwise interactions, EBM's make use of the idea of GA2M-models developed by (Lou and Caruana: Accurate Intelligible Models with Pairwise Interactions). While the resulting characteristic functions $k_{m,n}(X^m, X^n)$, (where $m \neq n$ and $m, n \in \{1, ..., J\}$)

Parameter and hyperparameter search for the models

All inner parameters of the models will be constructed using a 3-fold cross-validation to ensure generalizable training scores and the selection of the best generalizable model. Additionally, the best hyperparameters of each model captured with a randomized implementation of grid-search.
Results of grid-search hier?

3.5 Model Evaluation Metrics

To get an unbiased estimate of the generalized performance of our model, it has to be tested on unseen data, which was not part of the training process. In order to espacially ensure time generalizability of our models, we use unseen data of an additional different timestamp to generate evaluations.
For the evaluation of classification models there is a huge variety of metrics. As already stated, selecting the correct metric is crucial. Let us first define the possible outcomes of our model with the confusion matrix:

Table 2: Confusion Matrix

	$\hat{C} = 0$	$\hat{C} = 1$
$C = 0$	# True Negatives (TN)	# False Positives (FP)
$C = 1$	# False Negatives (FN)	# True Positives (TP)

Using accuracy ($TN + TP/N$) will lead to very unreliable estimates of the performance with rare events. The accuracy will be high indicating a good fit, even if the churners are not frequently detected. Therefore it makes sense to either use the precision ($TP/TP + FP$) or the recall ($TP/TP + FN$) metrics. A combination of both measures is provided by the F1-score which is the harmonic mean of both metrics an therefore still defined between 0 and 1 (ganzer Paragraph: Mining with rarity):

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{9}$$

Another widely used metric is the area under the receiver-operating-characteristic-curve (AUROC). The ROC-curve represents the tradeoff between the false-positive-rate ($FPR = FP/FP + TN$) on the X-axis and recall on the y-axis of a classifier using different cutoff values τ.

The ROC always starts at the lower left-hand corner (FPR = 0, Recall = 0) for $\tau = 1$, and ends at the upper right-hand corner (FPR = 1, Recall = 1) for $\tau = 0$. The other values of the curve are defined by alternating τ in values between 0 and 1. An area under the curve of 1 is the best score and 0.5 is the worst score, representing the classifier of a simple coin-flipping algorithm.
This measure also has its downsides with very small positive classes, as the FPR is highly dependend on the number of negatives. The FPR will only improve by a small amount if a there is a substantial decrease in FP's as the number of TN's will be very high for most τ's.
For a small number of positive examples it makes more sense to look at the area under the precision-recall-curve (AUPRC) (cite: Area Under the Precision-Recall Curve: Point Estimates and Confidence Intervals). There the curve also plots values for the recall on the X-axis and precision on the y-axis for different τ's. It now starts at the upper left corner (Recall = 0, Precision = 1) for $\tau = 1$ (als * hinzufügen, dass für TP und FP gegen 0, Precision gegen 1) and ends in the lower right corner (Recall = 1, Precision = 0) for $\tau = 0$ (als * hinzufügen, dass precision gegen 0 geht, da wir sehr viele FP's haben. Die

Kurve wird niemals genau unten rechts aufhören.).
It is plotted as follows as an example:
Now when it comes to interpreting the metric, it is not as easy as the AUROC. The baseline now is not a simple coin-flip, but instead defined by the ratio of churners (#C/N) in the data. Every AUPRC score above that baseline is an improvement from predicting customers as Churners randomly using a Bernoulli-distribution for each ($C^{(i)} \sim B(\frac{\#P}{N})$).
For calculating the AUROC or the AUPRC there are multiple methods. All of them rely on estimates, as the true function of the curve can only be approximated using a finite number of cutoff values τ. (Mehr darauf eingehen, und Area Under the Precision-Recall Curve: Point Estimates and Confidence Intervals zitieren, falls du noch Zeit hast).

Bibliography