Boosting Gradient Boosting Interpretability

Predicting and Interpreting Cross-Selling Purchase Probabilities of a Large German Savings Bank

Seminar Paper

submitted to

Hon. Prof. Dr. Martin Schmidberger
Gabriela Alves Werb

Goethe University Frankfurt am Main
School of Business and Economics
Chair for E-Commerce

by

Jonathan Ratschat

in partial fulfillment of the requirements of the seminar

Data Mining in Marketing:
Data Driven Customer Analytics with Machine Learning
Summer Semester 2020

June 10, 2020

Contents

List of Figures

List of Tables

List of R Code Chunks

1 Introduction

The availability of new data sources and marketing analytics enables companies to create more value for their customers, leverage customer experiences, and to increase customer loyalty. New metrics and more powerful analytical methods are needed to be more efficient and effective in data-driven marketing (Wedel & Kannan 2016, p. 97). Besides descriptive metrics, machine learning methods have become popular due to their high predictive performance. A disadvantage is, however, that the interpretability of complex machine learning models is questionable. Therefore, eliminating the trade-off between a model's accuracy and a model's interpretability has gained attention from many researchers (Ribeiro et al. 2016, Lundberg & Lee 2017, Chen et al. 2018, Lipton 2018).

In this paper, I use a data set from a large German savings bank to predict cross-selling purchase probabilities and decisions in the customer base. The goals of this paper are (1) to accurately predict whether an already existing customer will open a checking account and (2) to explore which affect the features have on the prediction to enhance the interpretability of the model. The paper leverages one of the leading gradient boosting algorithms there is, namely XGBoost, to reach these goals. It has been used with great success in many machine learning and data mining challenges. XGBoost models tend to easily scale, avoid overfitting, and are used for a wide range of problems (Chen & Guestrin 2016, p. 785-786). To tackle the lack of easy interpretability of boosted trees (Friedman 2001, p. 1229–1230), this paper implements SHapley Additive exPlanations (SHAP) values to explain the output of the XGBoost model.

The paper is structured in the following way: In the first section, I give an introduction of cross-selling theory while I afterward describe the implemented statistical methods. Following the methodological explanation, the paper presents the results concerning the implemented model and its interpretability. Finally, managerial and research implications on predictive modeling and interpretability are highlighted.

2 Cross-Selling

Cross-selling is defined as the "practice of selling an additional product or service to an existing customer" (Li et al. 2011, p. 683). Kamakura et al. (1991) state that technological developments help to identify customers that are likely to participate in cross-selling activities (p. 348). Also, focusing on retaining customers is associated with lower costs compared to newly acquired customers (Keaveney 1995, p. 71). Therefore, cross-selling can help to increase market share and sales volume (Kamakura et al. 1991, p. 348). Cross-selling is especially important in the financial industry since bank customers face different financial objectives at different points in time. To address these objectives, banks offer a wide selection of borrowing and saving products to their customers (Kamakura et al. 1991, p. 331). The importance of cross-selling in the financial industry is also influenced by bank customers' relatively high switching costs. Switching costs arise due to the high number of services and the inconvenience of filling out the related paperwork when switching to another bank (Li et al. 2005, pp. 234–235). Although cross-selling ranks as a top strategic priority for many industries, related marketing campaigns are not always profitable (Li et al. 2011, p. 683). Amongst others, targeting every customer that is likely to buy an additional service may not be beneficial due to different customer profitability levels (Shah et al. 2012). Also, a problem could be that inaccurately targeted customers eventually churn because of the constant stream of irrelevant or annoying information (Keaveney 1995). Therefore, it is a high managerial priority to target the right customers to improve the response rates and the profitability of these marketing campaigns (Li et al. 2011, p. 683). This paper addresses this challenge by implementing an XGBoost model that seeks to predict the purchase probabilities and decisions of existing customers accurately.

3 Methodology

In this section, I describe the methodology that led to the results in section 4. First, the paper prepares and modifies the data set to make it accessible to the XGBoost algorithm and to optimize the predictive power of the model. In the second part of this section, I describe the XGBoost model and the process of improving the model through

cross-validation and hyperparameter-tuning. In the last part, the interpretability and trust of the model are addressed by introducing SHAP values.

3.1 Feature Engineering and Feature Reduction

The data set is provided from a large German savings bank containing information of 10,000 customers. The dependent variable is `xsell`, which measures whether a checking account was successfully opened within six months from an already existing customer.[1] The data set includes information about demographics (e.g., age, gender, or occupation), bank-related data (e.g., the number of logins, total savings, or total loans), and marketing data (email advertising, postal advertising). In total, 32 features are reported.

To prepare the dataset for the algorithm, the variable `pref_device` is dropped. The reasoning is that `pref_device` is automatically created when customers log in to their bank accounts. Thus, missing values of `pref_device` are indicating a low level of activity as shown by the significant differences in the number of logins for customers with missing values (mean: 1.2, SD: 5.7) and the number of logins for customers without missing values (mean: 14, SD: 46.3). Therefore, the effect of `pref_device` is already included in the model through the `logins` variable. Next, the variable `occupation` is simplified since more than half of the customers did not provide information about their profession. This is due to the voluntary response option of that question when becoming a customer of the bank. The information if a customer has answered the question or not is meaningful. Therefore, it is included in the model as a binary variable. Due to the high segmentation of professions and the high amount of missing data, the base `occupation` variable is overwritten with this binary variable.

Further preparation is done by transforming all character variables to factor type. Additionally, all numeric variables that are thought to be nominal are transformed into factor variables. Missing values are present in many other variables, but do not have to be imputed for decision trees and gradient boosting models (Friedman 2001, p. 1229). XGBoost takes care of these values by learning the direction that minimizes

[1] Variable names and descriptions are presented in table 4 in the appendix.

the objective (Nielsen 2016, p. 51). The data set is randomly split into training (6,400 customers), validation (1,600 customers), and test (2,000 customers) sets to train the model and evaluate the performance of the model in section 3.2. Since XGBoost cannot handle factor variables, the data sets are one-hot encoded using the `model.matrix()` and `xgboost.DMatrix()` command, which creates sparse matrices. XGBoost is a sparsity-aware algorithm that significantly improves runtimes compared to algorithms that do not consider sparsity(Chen & Guestrin 2016). After feature engineering, the data sets contain 58 features that are used to predict the probability of opening a checking account.

3.2 Gradient Boosting with XGBoost, Model Fitting, and Hyperparameter-Tuning

The R interface of the implementation of the gradient boosting framework **xgboost** by Chen & Guestrin (2016) includes efficient tree learning algorithms that are implemented for the prediction. To train the model, the `xgboost.train()` command is used. I specify the model as a gradient booster tree and set binary classification as the objective. The maximum number of iterations is set to 1,000. Then, the boosting algorithm could be implemented and would create 1,000 iterations of the model. This would train an overfitted model that perfectly fits the training data, but cannot be generalized. This behavior needs to be prevented while ensuring that the model learns the generalized underlying relationships of the features with the target variable to obtain a better model. For these reasons, the classification accuracy of the validation set must be increased (Athey 2019, p. 511). Using this cross-validation technique, the training and validation set are therefore used to create and ensure the stability of the model while the test set is used to evaluate the performance of the model. Thus, it is first described how model performance is increased by using cross-validation and then hyperparameter-tuning.

The classification accuracy is set as the evaluation metric to optimize the performance of the validation set. The model is optimized through this metric because this paper focuses on the perceived trade-off between accuracy and interpretability. Only using accuracy can be misleading (Burez & van den Poel 2009, Chawla 2010),

therefore, this paper also reports additional evaluation metrics. The hyperparameter `early_stopping_rounds` is configured to 30. The algorithm stops the when the validation accuracy has not improved for the specified number of rounds and chooses the iteration with the highest accuracy. Therefore, this parameter prevents overfitting and saves computation time.

In figure 1, accuracy and AUC of an XGBoost model are displayed by iteration of training and validation set. As explained, the `xgboost.train()` command would choose the 25th iteration because the accuracy of the validation set decreases for the following 30 rounds. Therefore, the accuracy of the validation set is highest at this iteration. If the evaluation metric were changed to AUC, the iteration would stop at the same iteration. As exhibited by the increasingly diverging accuracy and AUC of training and validation set, overfitting is indeed a problem that is mitigated by early stopping.

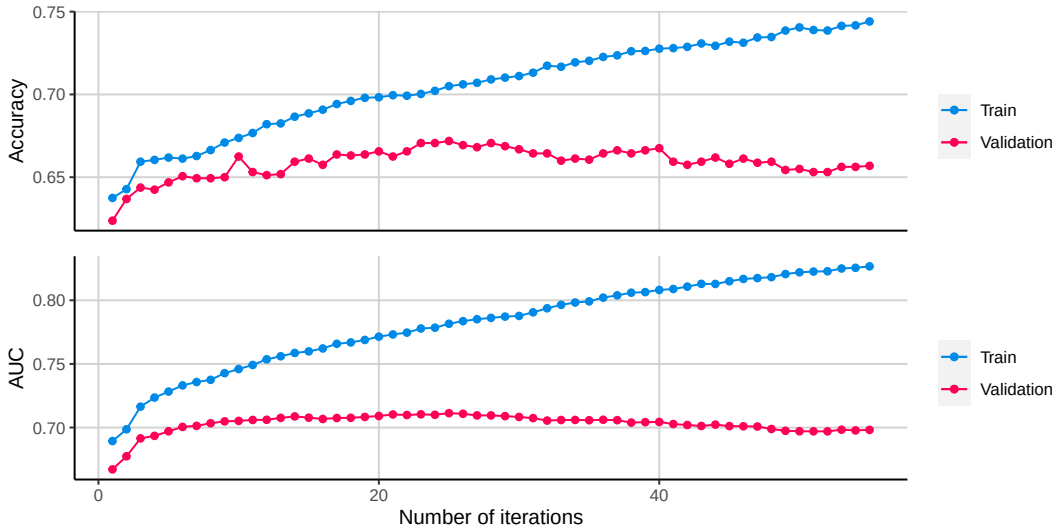

Figure 1: Accuracy and AUC by Iteration of a XGBoost Model[2]

Equally important, hyperparameter-tuning is a crucial step in machine learning practice. It allows changing the model fit options to adjust the model to the data (Lantz 2013, pp. 361–362). Therefore, hyperparameter-tuning can be applied to find a balanced relationship between underfitting and overfitting in the model to the data (Choudhury et al. 2018, p. 16). Different approaches are used to select hyperparameters. Often, users choose the values of hyperparameters based on reputation, instinctive appeal, or

[2]Model specifications: max_depth (4), eta (0.287), subsample (0.844), min_child_weight (6), colsample_bytree (0.624), others (default)

adhere to the default parameter values due to this high degree of choice. This behavior may result in a model whose potential has not been fully utilized (Thornton et al. 2013, p. 847). Following the argumentation of Bergstra & Bengio (2012), a random search algorithm is implemented to find optimal model hyperparameters. Their research shows that random search has higher performance compared to grid trials and manual search when allowing the same computation time. Also, opposite to the manual search of hyperparameters, the results obtained through random search are reproducible. For that purpose, hyperparameter values are randomly created within a given search space[3]. The model performance is then evaluated by the accuracy of the validation set. The results are reproducible since a seed is specified before the random creation of hyperparameters and the search algorithm. It took less than two hours with a standard computer to compare 10,000 XGBoost-models.[4] Although the ransom search algorithm compares a high number of models, it is certain that the reported approach cannot find the very best model performance. One could adjust the search space or even add more hyperparameters to achieve better results. Still, finding the ideal model could not be declared after additional random searches with changed settings.

3.3 SHapley Additive exPlanations values

Complex models seek to optimize prediction evaluation metrics, such as accuracy. This behavior can negatively affect the interpretability of the models (Friedman 2001, Lundberg & Lee 2017, Chen et al. 2018). This trade-off reduces confidence in the model, gives fewer insights into how the model could be improved, and reduces the knowledge that could be gained from the model. Also, better interpretability leads to better adoption. Therefore, the interpretability of such models is very important (Lundberg & Lee 2017, p. 4768).

[3]For this analysis, hyperparameter values for `eta`, `max_depth`, `subsample`, `colsample_bytree`, and `min_child_weight` are randomly created.

[4]The runtime of this procedure to find optimal hyperparameters is low. It was compared with a random search algorithm from the hyperparameter-tuning package **mlr** and running chunks of computations in parallel with the **parallel** package. Of course, the runtime of this procedure is high compared to the runtime of a simple logit regression.

xgboost provides feature importance measures such as gain to increase interpretability (appendix figure 5). Lundberg et al. (2019) argue that a model evaluated by such measures can change to rely more on a given feature while the assigned importance decreases. This leads to inconsistent results (p. 2).

To tackle the trade-off of accuracy and interpretability and to get consistent results, this paper focuses on SHapley Additive exPlanations (SHAP) values as introduced by Lundberg & Lee (2017). SHAP values stem from game theory and measure the marginal effect that the observed level of a variable for an individual has on the final predicted probability for that individual (Lundberg et al. 2019, p. 1). In other words, SHAP values measure the feature importance of all features for every prediction (Lundberg & Lee 2017, p. 4768). This is done by measuring the "sequential impact on the model's output of observing each input feature's value, averaged over all possible feature orderings" (Lundberg et al. 2020, p. 61). Through this approach, it becomes feasible to explain why a case receives its prediction and how the features contribute to this prediction. This local interpretability increases transparency. Also, it becomes feasible to combine local explanations to derive global interpretability. Generally, using SHAP values has many advantages. Opposed to importance charts provided in the **xgboost** package, it is shown how feature values relate to the feature impact and which range and distribution feature impacts have. SHAP values can be used for any tree-based model and are much easier interpreted by humans than other current methods like LIME, DeepLIFT, or Saabas values (Lundberg & Lee 2017, Lundberg et al. 2020). However, it is essential to note that SHAP values cannot identify causal effects or produce generalizable theoretical insights.

With regards to this paper, the **SHAPforxgboost**[5] package is used to generate SHAP values and SHAP interaction values. After these steps, summary plots, dependence plots, and interaction plots can easily be visualized. Therefore, this method enables the model to be not only accurate but also interpretable. While this chapter discussed the deployed methods theoretically, the following chapter presents and evaluates the results of the machine learning process on the cross-selling dataset.

[5]The original implementation of SHAP is found in the Github repository slundberg/shap. The **SHAPforxgboost** package was created to make this module accessible to R users.

4 Results

This section first introduces the final XGBoost architecture derived by cross-validation and hyperparameter-tuning. Then the associated results of the final XGBoost model are compared to a simple benchmark logit model. This chapter concludes with the interpretability of the tuned model through SHAP values.

4.1 Selecting the Optimized XGBoost Architecture

As described in section 3.2, the process of finding a model with optimal performance is a challenging process. To obtain optimal performance, the model must neither underfit nor overfit. Through feature engineering and feature reduction, choosing evaluation metrics, and hyperparameter-tuning, there are many steps involved that influence the prediction quality of the model. Since this process does not necessarily follow a certain order, it is impossible to state that one has found the "best" model. Decisions with regards to feature engineering/reduction, cross-validation and hyperparameter-tuning are described in section 3.1 and 3.2. Following the described approach, the random search procedure created 10,000 models with differing hyperparameters. Table 1 shows the three best and the three worst models as ranked by the accuracy of the validation set with the corresponding hyperparameters.

rank	val_acc	max_depth	eta	subsample	colsample_bytree	child[6]
1	0.672	4	0.287	0.844	0.624	6
2	0.669	4	0.104	0.846	0.743	7
3	0.669	5	0.17	0.749	0.653	3
...	...	...	...	...	...	...
9,998	0.614	10	0.234	0.993	0.636	0
9,999	0.613	9	0.282	0.889	0.88	0
10,000	0.609	10	0.258	0.934	0.894	0

Table 1: Results of 10,000 Random Search Runs Ordered by val_acc in Descending Order

Indeed, hyperparameter-tuning has a strong effect on the performance of the model. The validation accuracy ranges between 60.9 percent and 67.2 percent, with a median of 64.7 percent and a mean of 64.64 percent. When dividing these models into deciles,

[6] min_child_weight

significant differences between the hyperparameter values are found between the best and worst 10 percent of the models. It is worth noting that the mean of maximum depth of a tree is 115 percent higher for the top models. The differences in the mean for the other hyperparameters are much lower.

Although this procedure outperformed the previous testing by manual search by far, the performance is heavily reliant on the previously determined hyperparameters and search space. Including more hyperparameters could improve or harm the model performance. Also, specifying the search space dependent on the practitioner's recommendations and own experience could have the same effect. However, the focus of this paper is not to identify the best random search procedure existing.

4.2 Comparing Hyperparameter-Tuned XGBoost Model with Benchmark Logit Model

The performance of a model must be compared to alternative models to assess the quality of the prediction. The XGBoost model with the highest validation accuracy that was identified in section 4.1 is chosen as the challenger model. This tuned model is benchmarked against a simple logit regression model. To compare these models, `xsell` of the test set is predicted, and the resulting model statistics and confusion matrices are calculated.[7] This comparison is based on the same input features.[8]

Benchmark Logit		*Hyperparameter-Tuned XGBoost*	
Accuracy	0.626	Accuracy	0.657
Sensitivity	0.633	Sensitivity	0.680
Specificity	0.618	Specificity	0.635
Pos Pred Value	0.616	Pos Pred Value	0.643
Neg Pred Value	0.635	Neg Pred Value	0.672
AUC	0.678	AUC	0.713
Kappa	0.251	Kappa	0.314
Prevalence	0.492	Prevalence	0.492

Table 2: Model Performance Comparison - Performance Metrics

[7]For a look at the confusion matrices, please consult table 3 in the appendix.

[8]Features are not one-hot encoded with the discussed procedure for the simple logit regression model. Also, missing values are imputed using the **randomForrest**::`rfImpute()` command since logit models would otherwise omit rows with missing values.

In table 2, the results with regards to the benchmark logit model and the hyperparameter-tuned XGBoost model are shown. Unsurprisingly, the XGBoost model performs better in all analyzed metrics.[9] It increases accuracy by 3.1 percentage points. Bearing in mind that 50 percent of the customers did acquire a checking account, the prediction is 15.7 percentage points better than random guessing. The models differ by 4.7 percentage points in sensitivity. Therefore, the hyperparameter-tuned XGBoost model is better at predicting customers that buy a checking account. When inspecting specificity, the difference of 1.7 percentage points is not as severe as with regards to sensitivity. Still, the XGBoost model is also better in predicting customers that do not buy a checking account.

4.3 Increasing Interpretability through SHapley Additive exPlanations values

In the previous section, it was shown that the hyperparameter-tuned XGBoost model could better predict whether existing customers will open a checking account than the simple benchmark logit model. To increase the transparency and interpretability of the model with the help of the in section 3.3 introduced SHAP values, I report relations between feature importance, range, distribution, dependence, and interaction.

In figure 2, the features are ordered by their global impact (SHAP value) on the tuned XGBoost model. The dots represent the feature impact on the model output of individual predictions (e.g., customer 1) of the training set. Depending on the feature value, the dots are colored from blue (low) to red (high). Therefore, it is feasible to explore the extent, strength, and direction of a feature's effect. On the one hand, the highest global feature importance in the model is taken by `logins` followed by `age`, `giro_mailing`, and `last_acc`. It becomes clear that the range of the feature effects is widely distributed. E.g., for some individuals, receiving giro account advertising has a strong influence on their prediction score while for others, it only has a small effect. Such behavior of the model can be found, especially with the more important features. On the other hand, several features have a feature importance score of 0 and

[9] Most likely, the performance of the logit model could be improved via feature engineering so that the model can follow non-linear relations more closely. However, I refrain from doing so since the logit model is used as a benchmark.

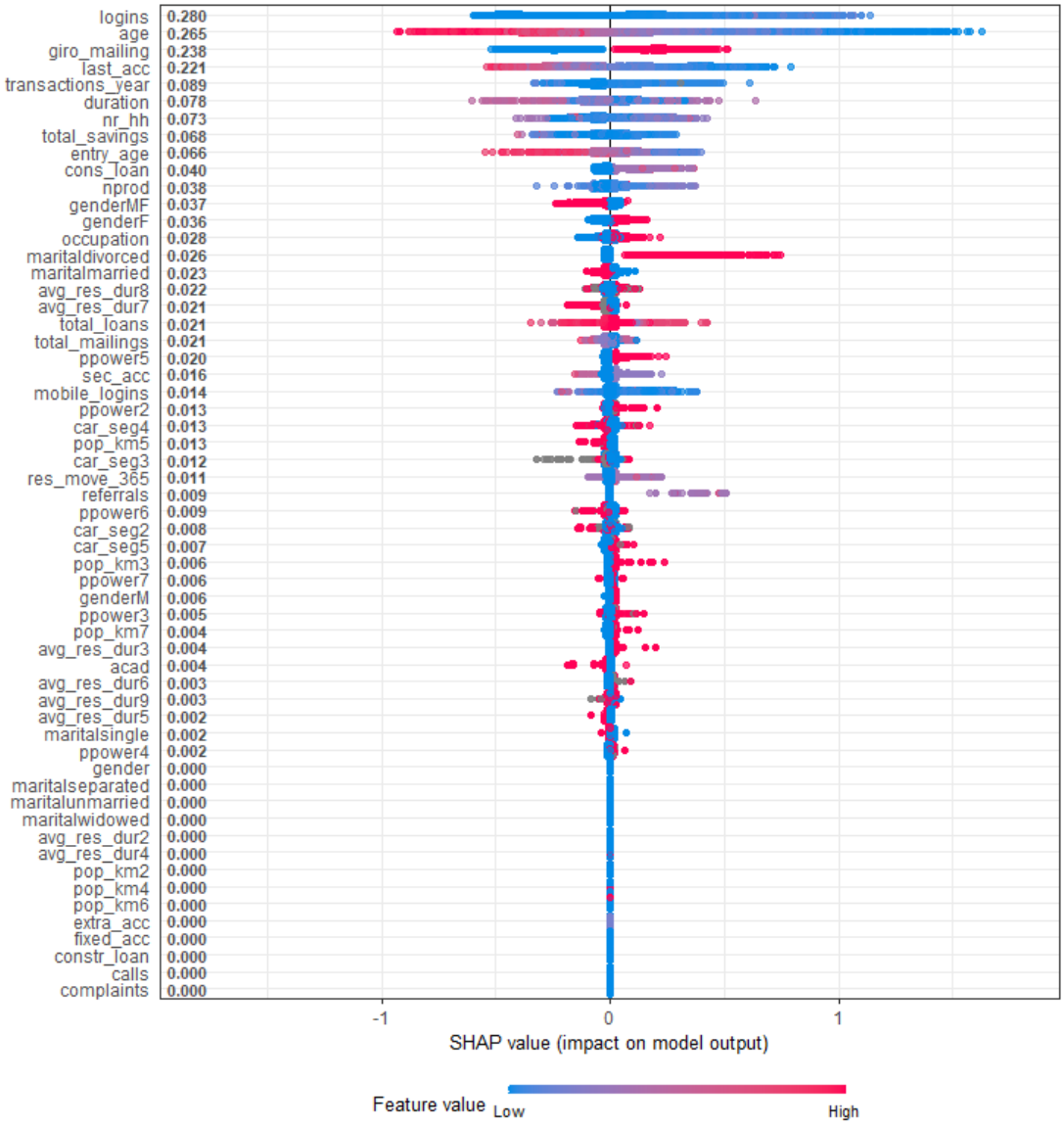

Figure 2: SHAP Summary Plot of Hyperparameter-Tuned XGBoost Model

do not vary in the extent of the feature effect on the prediction. Exploring this figure is helpful to understand how the model works and can help to check the plausibility of the model. At first sight, none of the more important features shows a counterintuitive relation to `xsell`.[10]

This paper focuses only on a few variables with high global importance with a special focus on `giro_mailing` to derive meaningful results and show the potential of this

[10]The feature values of some variables are not easily interpretable because there are outliers present in these features. This leads to the visual misconception that displays nearly all values of a feature as being low. Therefore, this paper provides additional information (figures 6 and 7) concerning these features in the appendix.

method. It should be noted that receiving checking account ads always has a positive influence on the prediction. In figure 3, a SHAP dependence and a SHAP interaction plot for `age` and `giro_mailing` are displayed. As with the summary plot, each dot is a customer. On the left panel, the x-axis is their age, and the y-axis is SHAP value attributed to customer age. Higher SHAP values represent a higher prediction of acquiring a checking account due to age. The dependence plot reveals that younger customers have a higher cross-sell prediction value than older customers. Coloring each dot by giro advertising, it becomes clear that `giro_mailing` lowers the feature's impact on younger customers but increases the feature's impact on older customers. This interaction is confirmed by the interaction plot on the right panel, which more closely displays this effect.

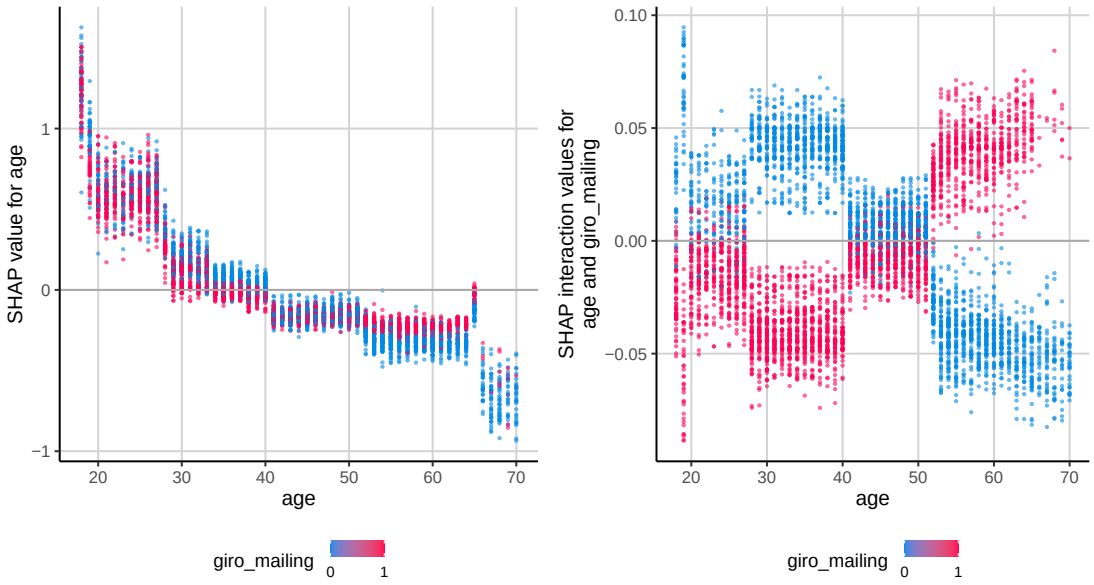

Figure 3: SHAP Dependence Plot and Interaction Plot for age and giro_mailing

This paper visualizes similar results in figure 4. Here, customers who have recently acquired another product have a higher prediction of cross-buying than customers that have not recently acquired another product. Likewise, `giro_mailing` again lowers this effect for recent acquirers and increases the effect for past acquirers.

By displaying all individual predictions, one can create global trends so that generalizations on the entire dataset can be derived. What was found out so far is that the global feature importance does not mean that the feature is equally important for every

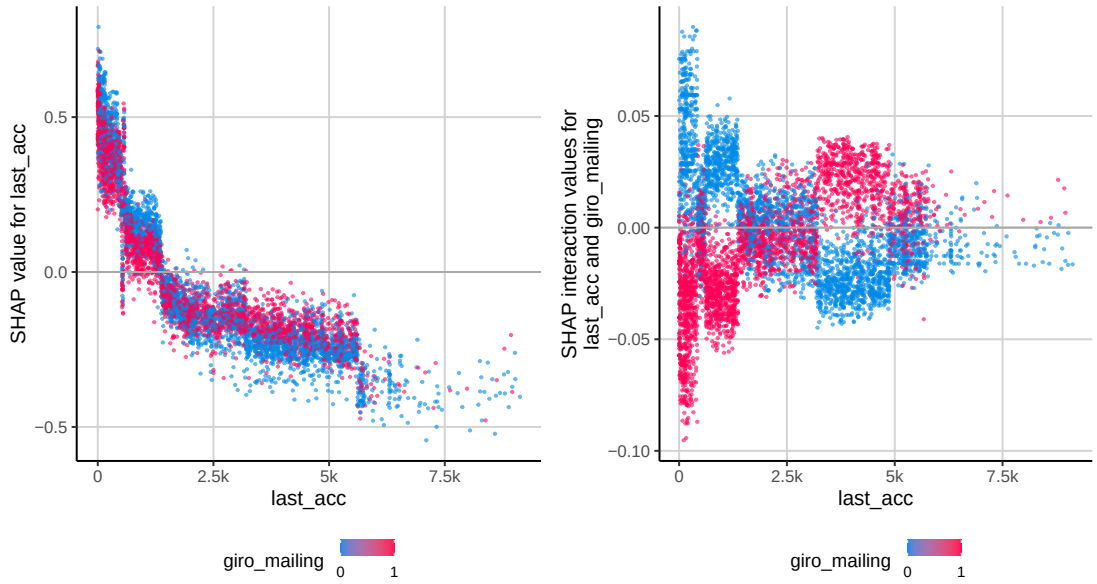

Figure 4: SHAP Dependence Plot and Interaction Plot for last_acc and giro_mailing

single prediction. There are predictions in which a feature is essential, while in other predictions, this feature could be negligible. Also, features do interact with each other. The following section further elaborates on the findings of this section and discusses managerial and research implications.

5 Managerial and Research Implications for Predictive Modeling and Interpretability

The implications resulting from this analysis can be divided into two parts: The first one discusses implications related to predictive modeling, while the second part outlines implications derived from the model's interpretation.

The result is unambiguous when comparing the hyperparameter-tuned XGBoost model's accuracy with the benchmark logit model's accuracy. When accuracy is the primary goal, then one should use more complex models. Although tuning an XGBoost model is computationally expensive, once it is tuned, it becomes computationally cheap. It becomes apparent that such prediction systems have their place in marketing analytics departments. Instead of only predicting the likelihood of buying a checking account, marketing departments could extend this approach to any product offering to compare purchasing probabilities and target customers with the products for which they have

the highest probability to buy. Through this strategy, it could become feasible to target the right customers and to increase profitability ultimately. For this concept to be successful, several additional topics need to be addressed. One has to assess which customers have the potential to be profitable while excluding customers that lead to losses (Shah et al. 2012). Also, one must answer the question of when and where a customer should be targeted.

Generally, SHAP values enable its users to examine complex models critically and to understand how dependent variables were predicted. Through this method, users gain further knowledge about the importance, extent, and direction of feature variables on the target variable. Although causal statements cannot be made through this approach, it still helps users to gain trust in the model, to find ways of improving the model, and to get a new understanding of the data. Therefore, this enhanced interpretability should increase user adoption. When only using tools from the **xgboost** package, this would not be feasible to such an extent. The trust of the analysis through SHAP values is increased because suggested underlying trends of the data have been amongst others discovered by RFM-models[11] (Bauer 1988, Miglautsch 2000). Customers that have recently acquired another product have a higher predictive value than customers that have not bought another product for a more extended period. Also, the more active customers are (as measured by `logins`), the higher is the prediction that these customers cross-buy a checking account. Other vital trends found in the data are that younger customers exhibit a higher prediction probability than older customers and that checking account ads always lead to a positive effect on the prediction, although with varying extent. This paper shows that `giro_mailing` does lead to a negative interaction effect on the prediction when appearing with younger or recent customers. This could indicate that the model finds autoresponse within the group of younger or recent customers and punishes this effect with a negative interaction value.

XGBoost models should be used in practice for predicting cross-selling purchase probabilities and decisions. One of the biggest disadvantages - lack of interpretability - can be mitigated through the use of SHAP values. These values greatly expand the

[11] Recency, Frequency, and Monetary Value

transparency, explainability, interpretability of complex tree-based models.

6 Conclusion

The first research question of this paper was to accurately predict whether an existing customer will open a checking account based on customer characteristics. This paper applied XGBoost, a leading machine learning gradient boosting algorithm, and compared it to the performance of a simple benchmark logit model to get an accurate prediction as possible. It proved to perform better in terms of accuracy, sensitivity, specificity, and AUC. Although the tuning of the XGBoost model is computationally expensive, this paper argues that this gradient boosting algorithm should be used in practice for predicting cross-selling purchase probabilities and decisions.

The second research question was related to understanding the underlying magnitude, extent, and distribution of the features. This paper presented a relatively new tree-based explainer model - namely SHAP values derived through the research of Lundberg & Lee (2017) to make the model interpretable. Without this method, the interpretability of the model would have been very limited. However, the paper was able to show how feature values relate to feature impacts and which range and distribution feature impacts have. Also, using dependencies and interactions of features, it was feasible to explore trends with regards to the prediction. It was found that specific customer characteristics have a strong influence on the prediction (activity, age, giro ads, recent purchases) while others have a low to non-existing influence (construction loans, calls, complaints).

Although the two methods discussed led to useful results concerning the research questions, further work could be dedicated to these findings. Regarding model performance, a model with comparably high performance was found. SHAP values increased the model's interpretability, even though causality cannot be established with this kind of model.

A Appendix

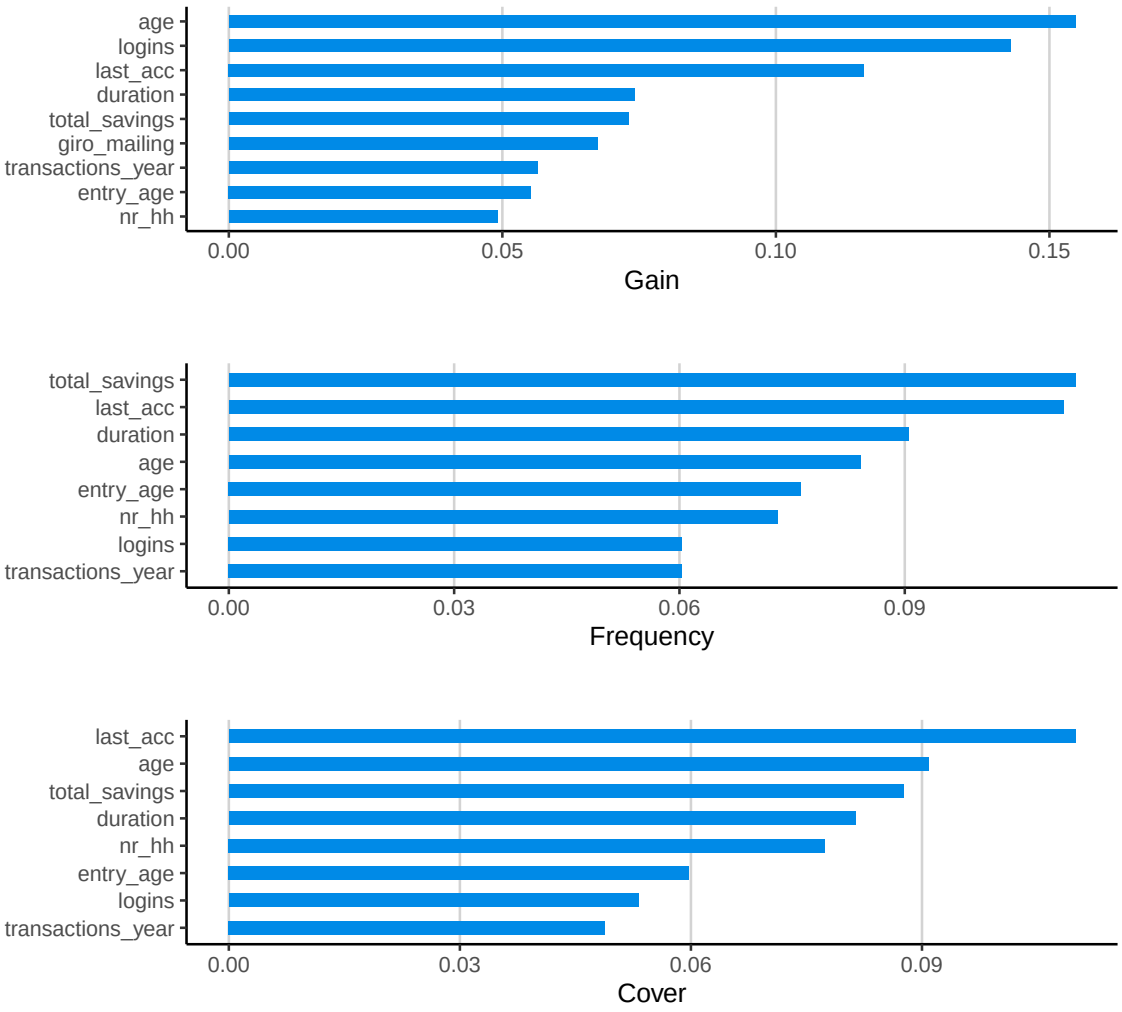

Figure 5: Conventional Feature Importance Bar Plots Included in the xgboost Package

Benchmark Logit			*Hyperparameter-Tuned XGBoost*		
	Actual			*Actual*	
Prediction	0	1	*Prediction*	0	1
0	628	361	0	645	315
1	388	623	1	669	726

Table 3: Model Performance Comparison - Confusion Matrices

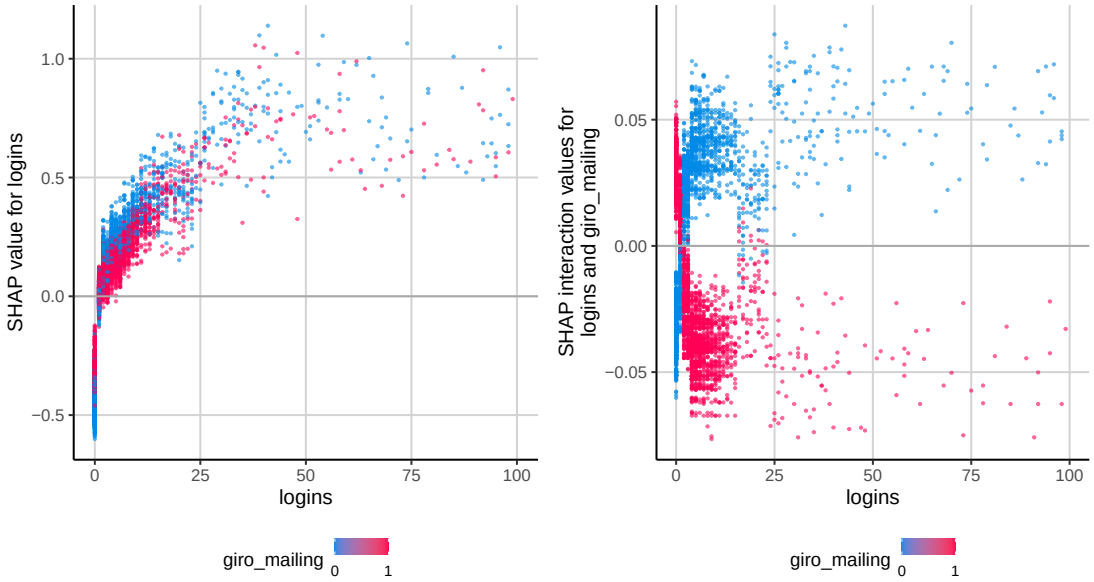

Figure 6: SHAP Dependence Plot and Interaction Plot for logins and giro_mailing (for better representation of the data without extreme values)

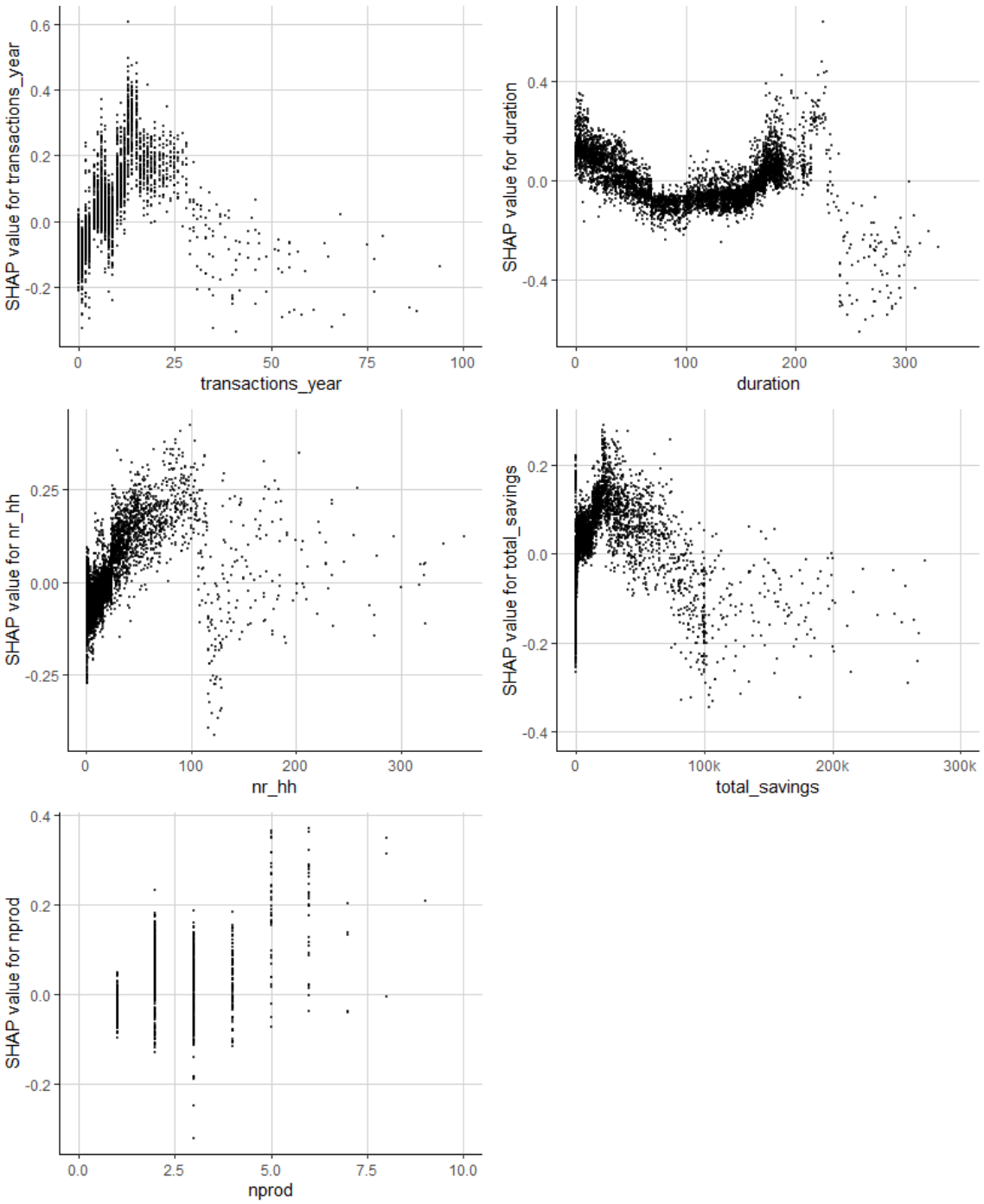

Figure 7: SHAP Dependence Plots for transactions_year, duration, nr_hh, total_savings and nprod (for better representation of the data without extreme values)

Variable name	Variable description
xsell	Customer opened a checking account: 1 (yes), 0 (no)
age	Age
entry_age	Age when became a customer
gender	Gender: female, male, joint account, missing
acad	Customer has an academic title: 1 (yes), 0 (otherwise)
marital	Marital status
nprod	No. of products/accounts the customers owns
duration	Customer duration in months
last_acc	Days since the last time the customer opened an account
occupation	Customer provided occupation information: 1 (yes), 0 (no)
nr_hh	No. of customers (households) in the neighborhood
ppower	Average purchasing power in the residential block: 1 (very low) to 7 (very high)
avg_res_dur	Average duration of residence in the customer's building: 1 (0-1 year), 2 (1-2 years), 3 (2-3 years), 4 (3-4 years), 5 (4-5 years), 6 (5-6 years), 7 (6-8 years), 8 (8-10 years), 9 (more than 10 years)
pop_km	Population density per sq. km in the neighborhood: 1(very low) to 7(very high)
car_seg	Predominant vehicle category in the neighborhood: 1 (subcompact), 2 (compact), 3 (mid-size), 4 (full size), 5 (mixed)
res_move_365	No. of times the customer moved his residence (past year)
giro_mailing	Received an email about opening a checking account: yes (1), no (0)
total_mailings	No. of mailings sent to the customer in the last 2 years
extra_acc	No. of additional accounts the customers owns
fixed_acc	No. of fixed deposit accounts the customer owns
constr_loan	No. of construction financing loans the customer owns
cons_loan	No. of consumer loan products the customer owns
sec_acc	No. of securities accounts the customer owns
total_savings	Volume of customer's savings except securities accounts in euros
total_loans	Volume of customer's outstanding loans in euros
transactions_year	No. of transactions in the current account (past year)
logins	No. of logins in the online banking (past 6 months)
mobile_logins	No. of mobile logins in the online banking (past 6 months)
calls	No. of calls to customer service (past 6 months)
referrals	No. of customers the customer referred to the bank (past year)
complaints	No. of customer complaints (past year)
pref_device	Customer's preferred type of device for logins

Table 4: Variable Name and Variable Description Used or Described for Analysis Before One–Hot Encoding

B R Code

Listing 1: Main Analysis

```
# Packages
library(Hmisc)
library(GGally)
library(ggplot2)
library(dplyr)
library(data.table)
library(tidyr)
library(e1071)
library(xgboost)
library(caret)
library(pROC)
library(mlr)
library(missForest)
library(randomForest)
library(sjlabelled)
library(readr)
library(SHAPforxgboost)
library(cowplot)
library(Ckmeans.1d.dp)
library(here)
library(stargazer)
library(StatMeasures)
library(sjlabelled)

# Load data
load(file= "data/xsell.RData")

# Create table of variable names and descriptions
(variable_names <- as.data.table(names(as.data.table(xsell))))
(variable_description <- as.data.table(attr(xsell, "variable.labels")))
(variable_table <- cbind(variable_names, variable_description))
stargazer(variable_table, summary = FALSE)

# Create table with number of missing values per feature (pref_device does have blank
    cells, not included here)
(nas <- rbindlist(lapply(names(as.data.table(xsell)), function(colname) {
  nMissingValues <- sum(is.na(as.data.table(xsell)[, get(colname)]))
  return(data.table(feature = colname, number_of_nas = nMissingValues))
})))

# Transform occupation variable
xsell$occupation <- ifelse(is.na(xsell$occupation), 0, 1)

# Delete pref_device
xsell <- xsell %>%
  dplyr::select(-pref_device)

# Transform all character variables into factor variables
xsell <- xsell %>%
  mutate(gender = as.factor(gender),
         marital = as.factor(marital))

# Transform numeric variabbles that should better be factor variables
xsell <- xsell %>%
  mutate(ppower = as.factor(ppower),
         avg_res_dur = as.factor(avg_res_dur),
         pop_km = as.factor(pop_km),
         car_seg = as.factor(car_seg))

# Write csv for benchmark model
write_csv(xsell, "data/xsell.csv")

# Randomly select 80% of the observations without replacement
set.seed(20)
train_id <- sample(1:nrow(xsell), size = floor(0.8 * nrow(xsell)), replace=FALSE)

```

```r
# Split in Training and Test (80/20)
xsell_first_round_test <- xsell[train_id,]
xsell_test <- xsell[-train_id,]

# Randomly select 80% of the observations without replacement
set.seed(20)
train_id <- sample(1:nrow(xsell_first_round_test), size = floor(0.8 * nrow(xsell_first_round_test)), replace=FALSE)

# Split in Training and Validation (80/20)
xsell_train <- xsell_first_round_test[train_id,]
xsell_valid <- xsell_first_round_test[-train_id,]

# Returns the NA object unchanged, if not changed, NA would be dropped
options(na.action='na.pass')

# Prepare matrix for XGBoost algorithm
xsell_train_matrix <-model.matrix(xsell ~ .-1, data = xsell_train)
xsell_valid_matrix <-model.matrix(xsell ~ .-1, data = xsell_valid)
xsell_test_matrix <-model.matrix(xsell ~ .-1, data = xsell_test)
dtrain <- xgb.DMatrix(data = xsell_train_matrix, label = xsell_train$xsell)
dvalid <- xgb.DMatrix(data = xsell_valid_matrix, label = xsell_valid$xsell)
dtest <- xgb.DMatrix(data = xsell_test_matrix, label = xsell_test$xsell)

# Base XGBoost model
set.seed(20)
params <- list(booster = "gbtree", objective = "binary:logistic")
xgb_base <- xgb.train (params = params,
                       data = dtrain,
                       nrounds =1000,
                       print_every_n = 10,
                       eval_metric = "auc",
                       eval_metric = "error",
                       early_stopping_rounds = 50,
                       watchlist = list(train= dtrain, val= dvalid))

# Make prediction directly on dtest
xsell_test$pred_xgb_base <- predict(xgb_base, dtest)
xsell_test$pred_xgb_factor_base <- factor(ifelse(xsell_test$pred_xgb > 0.5, 1, 0), labels=c("No XSell","XSell"))

# Check accuracy with the confusion matrix
confusionMatrix(xsell_test$pred_xgb_factor_base, factor(xsell_test$xsell ,labels=c("No XSell", "XSell")),
                positive = "XSell", dnn = c("Prediction", "Actual Data"))

##################### Tuned XGBoost model ############

# Tuned XGBoost model - hyperparameter values derived from random search
set.seed(20)
params <- list(booster = "gbtree", objective = "binary:logistic", eta = 0.28655333, max_depth = 4, min_child_weight = 6, subsample = 0.8440675, colsample_bytree = 0.6240213)
xgb_tuned <- xgb.train (params = params,
                        data = dtrain,
                        nrounds =300,
                        print_every_n = 1,
                        eval_metric = "auc",
                        eval_metric = "error",
                        early_stopping_rounds = 30,
                        watchlist = list(train= dtrain, val= dvalid))

##################### Validation
# Make prediction
xsell_valid$pred_xgb <- predict(xgb_tuned, dvalid, type = "response")

# Create prediction with 1 or 0
xsell_valid$pred_xgb_factor <- factor(ifelse(xsell_valid$pred_xgb > 0.5, 1, 0), labels=c("No XSell","XSell"))

```

```r
# Check accuracy with the confusion matrix
confusionMatrix(xsell_valid$pred_xgb_factor, factor(xsell_valid$xsell ,labels=c("No XSell","XSell")),
                positive="XSell",dnn = c("Prediction", "Actual Data"))

##################### Test

# Make prediction
xsell_test$pred_xgb <- predict(xgb_tuned, dtest, type = "response")

# Find optimal cutoff value
roc_test_xgb <- roc(xsell_test$xsell, xsell_test$pred_xgb, percent=TRUE, plot=TRUE, print.auc=TRUE,grid=TRUE)
roc_test_xgb

# Create prediction with 1 or 0
xsell_test$pred_xgb_factor <- factor(ifelse(xsell_test$pred_xgb > 0.5, 1, 0), labels=c("No XSell","XSell"))

# Check accuracy with the confusion matrix
(cvtuned <- confusionMatrix(xsell_test$pred_xgb_factor, factor(xsell_test$xsell , labels=c("No XSell","XSell")),
                            positive="XSell",dnn = c("Prediction", "Actual Data")))

# Prepare data for table
crosstable <- as.data.frame.matrix(cvtuned$table)
stargazer(crosstable, summary = FALSE)

##################### Plotting ####################

# Set theme for plots
mytheme <- theme(
  ## plotregion
  panel.background = element_rect(fill = "white",
                                  colour = "white"),
  panel.border = element_blank(),
  panel.grid.major = element_line(size = 0.5,
                                  linetype = 'solid',
                                  colour = "lightgrey"),
  panel.grid.minor = element_blank(),
  panel.spacing = unit(0.25, "lines"),
  ## axis line
  axis.line = element_line(colour = "black",
                           size = 0.5,
                           linetype = "solid"),
  ##background of fill
  strip.background = element_rect(fill="#f2f2f2") +
    scale_fill_discrete())

## AUC and Accuracy Plot

g1 <- ggplot(xgb_tuned$evaluation_log, aes(x= iter)) +
  geom_line(aes(y= 1-train_error, col = "Train")) +
  geom_point(aes(y= 1-train_error, col = "Train")) +
  geom_line(aes(y= 1-val_error, col = "Validation")) +
  geom_point(aes(y= 1-val_error, col = "Validation")) +
  xlab("") +
  ylab("Accuracy") +
  scale_y_continuous(labels = scales::number_format(accuracy = 0.01),breaks = c(0.60, 0.65, 0.70, 0.75)) +
  mytheme +
  theme(legend.title = element_blank(),
        axis.title.x=element_blank(),
        axis.text.x=element_blank(),
        axis.ticks.x=element_blank()) +
  scale_color_manual(values = c("#008AE7", "#FF0056"))

g2 <- ggplot(xgb_tuned$evaluation_log, aes(x= iter)) +
  geom_line(aes(y= train_auc, col = "Train")) +
  geom_point(aes(y= train_auc, col = "Train")) +
```

```
  geom_line(aes(y= val_auc, col = "Validation")) +
  geom_point(aes(y= val_auc, col = "Validation")) +
  xlab("Number of iterations") +
  ylab("AUC") +
  scale_y_continuous(labels = scales::number_format(accuracy = 0.01)) +
  mytheme +
  theme(legend.title = element_blank()) +
  scale_color_manual(values = c("#008AE7", "#FF0056"))

plot_grid(g1, g2, align = "v", nrow = 2, rel_heights = c(0.455, 0.545))

## Benchmark logit

# Run script that sets up a simple benchmark logit model
source("Benchmark_Logit_Model.R", echo = TRUE)

## ROC

ggroc(list("Hyperparameter-tuned XGBoost" = roc_test_xgb, "Benchmark Logit" = roc_test_logit)) +
  geom_point(size = 0.8) +
  mytheme +
  annotate(geom= "text", x=99, y=92, label="AUC: FALSE",
           color= "#008AE7", hjust = 0) +
  annotate(geom= "text", x=99, y=85, label="AUC: FALSE",
           color= "#FF0056", hjust = 0) +
  annotate(geom= "text", x=49, y=17, label="Hyperparameter-tuned XGBoost",
           color= "#008AE7", hjust = 0) +
  annotate(geom= "text", x=49, y=10, label="Benchmark Logit",
           color= "#FF0056", hjust = 0) +
  geom_abline(intercept = 100, slope = 1, color = "lightgrey") +
  xlab("Specificity (%)") +
  ylab("Sensitivity (%)") +
  theme(legend.title = element_blank()) +
  scale_color_manual(values = c("#008AE7", "#FF0056")) +
  theme(legend.position = "none")

## Distribution prediction

ggplot(xsell_valid, aes(x= pred_xgb, fill = pred_xgb_factor, color = pred_xgb_factor)) +
  geom_histogram(binwidth = 0.01) +
  mytheme +
  xlab("Predicted Xsell") +
  labs(fill = "Predicted Xsell Factor") +
  scale_fill_manual(values = c("#008AE7", "#FF0056")) +
  guides(color = FALSE)

## Feature Importance Plot

# Feature importance Gain
importance_matrix <- xgb.importance(model = xgb_tuned)
importance_matrix <- importance_matrix %>%
  dplyr::filter(Gain > 0.04)
g1 <- xgb.ggplot.importance(as.data.table(importance_matrix), measure = "Gain", n_clusters = 1) +
  labs(title = "") +
  xlab("") +
  ylab("Gain") +
  mytheme +
  theme(legend.position = "none") +
  scale_fill_manual(values = c("#008AE7", "#FF0056"))

# Feature performance Frequency
importance_matrix <- xgb.importance(model = xgb_tuned)
importance_matrix <- importance_matrix %>%
  dplyr::filter(Frequency > 0.05)
g2 <- xgb.ggplot.importance(as.data.table(importance_matrix), measure = "Frequency", n_clusters = 1) +
  labs(title = "") +
```

```
  xlab("") +
  ylab("Frequency") +
  mytheme +
  theme(legend.position = "none") +
  scale_fill_manual(values = c("#008AE7", "#FF0056"))

# Feature performance Cover
importance_matrix <- xgb.importance(model = xgb_tuned)
importance_matrix <- importance_matrix %>%
  dplyr::filter(Frequency > 0.05)
g3 <- xgb.ggplot.importance(as.data.table(importance_matrix), measure = "Cover", n_
    clusters = 1) +
  labs(title = "") +
  xlab("") +
  ylab("Cover") +
  mytheme +
  theme(legend.position = "none") +
  scale_fill_manual(values = c("#008AE7", "#FF0056"))

plot_grid(g1, g2, g3, align = "v", nrow = 3, rel_heights = c(1/3, 1/3, 1/3))

#################### SHAP ####################

# SHAP preparation
set.seed(20)
shap_values <- shap.values(xgb_tuned, X_train = xsell_train_matrix)
shap_values$mean_shap_score

# To prepare the long-format data:
set.seed(20)
shap_long <- shap.prep(xgb_model = xgb_tuned, X_train = xsell_train_matrix)
# is the same as: using given shap_contrib
shap_long <- shap.prep(shap_contrib = shap_values$shap_score, X_train = xsell_train_
    matrix)

# SHAP for single predictions - not mentioned in paper
library(arm)
set.seed(20)
shap_data <- shap_values$shap_score
shap_data[, BIAS := shap_values$BIAS0]
set.seed(20)
pred_mod <- predict(xgb_tuned, newdata = dtrain)
shap_data[, `:=`(rowSum = round(rowSums(shap_data),6), pred_mod = round(pred_mod,6))]
shap_data$rowSum <- invlogit(shap_data$rowSum)
shap_data[1:20,]

## SHAP summary plot
shap.plot.summary(shap_long) +
  scale_color_gradient(low = "#008AE7", high = "#FF0056",
                       breaks=c(0,1), labels=c(" Low","High "),
                       guide = guide_colorbar(barwidth = 12, barheight = 0.3))

## SHAP dependence plots for five variables
g1 <- shap.plot.dependence(data_long = shap_long, x = "transactions_year", y = '
    transactions_year',  smooth = FALSE) +
  mytheme +
  scale_x_continuous(limits=c(0,100))
g2 <- shap.plot.dependence(data_long = shap_long, x = "duration", y = 'duration',
    smooth = FALSE) +
  mytheme +
  scale_x_continuous(limits=c(0,350))
g3 <- shap.plot.dependence(data_long = shap_long, x = "nr_hh", y = 'nr_hh',  smooth =
    FALSE) +
  mytheme
g4 <- shap.plot.dependence(data_long = shap_long, x = "total_savings", y = 'total_
    savings',  smooth = FALSE) +
  mytheme +
  scale_x_continuous(limits=c(0,300000),
                     breaks = c(0, 100000, 200000, 300000),
                     labels = c("0", "100k", "200k", "300k"))
```

```r
g5 <- shap.plot.dependence(data_long = shap_long, x = "nprod", y = 'nprod',  smooth = FALSE) +
  mytheme +
  scale_x_continuous(limits=c(0,10))
gridExtra::grid.arrange(g1, g2, g3, g4, g5, ncol = 2)

# Prepare SHAP Interaction plots
shap_int <- shap.prep.interaction(xgb_mod = xgb_tuned, X_train = xsell_train_matrix)

## Age
g1 <- shap.plot.dependence(data_long = shap_long, x = "age", y = 'age',
                           smooth = FALSE, color_feature = "giro_mailing") +
  mytheme +
  scale_color_gradient(low = "#008AE7", high = "#FF0056",
                       breaks=c(0,1), labels=c("0","1"),
                       guide = guide_colorbar(barwidth = 2, barheight = 0.6)) +
  labs(color = "giro_mailing  ") +
  geom_hline(yintercept=0, colour = "darkgrey")

g2 <- shap.plot.dependence(data_long = shap_long, data_int = shap_int, x= "age", y = "giro_mailing",
                           smooth = FALSE, color_feature = "giro_mailing") +
  mytheme +
  scale_color_gradient(low = "#008AE7", high = "#FF0056",
                       breaks=c(0,1), labels=c("0","1"),
                       guide = guide_colorbar(barwidth = 2, barheight = 0.6)) +
  labs(color = "giro_mailing  ") +
  geom_hline(yintercept=0, colour = "darkgrey")
gridExtra::grid.arrange(g1, g2, ncol = 2)

## last_acc
g1 <- shap.plot.dependence(data_long = shap_long, x = "last_acc", y = 'last_acc',
                           smooth = FALSE, color_feature = "giro_mailing") +
  mytheme +
  scale_color_gradient(low = "#008AE7", high = "#FF0056",
                       breaks=c(0,1), labels=c("0","1"),
                       guide = guide_colorbar(barwidth = 2, barheight = 0.6)) +
  scale_x_continuous(breaks = c(0, 2500, 5000, 7500, 10000),
                     labels = c("0", "2.5k", "5k", "7.5k", "10k")) +
  labs(color = "giro_mailing ") +
  geom_hline(yintercept=0, colour = "darkgrey")
g2 <- shap.plot.dependence(data_long = shap_long, data_int = shap_int, x= "last_acc", y = "giro_mailing",
                           smooth = FALSE, color_feature = "giro_mailing") +
  mytheme +
  scale_color_gradient(low = "#008AE7", high = "#FF0056",
                       breaks=c(0,1), labels=c("0","1"),
                       guide = guide_colorbar(barwidth = 2, barheight = 0.6)) +
  scale_x_continuous(breaks = c(0, 2500, 5000, 7500, 10000),
                     labels = c("0", "2.5k", "5k", "7.5k", "10k")) +
  labs(color = "giro_mailing ") +
  geom_hline(yintercept=0, colour = "darkgrey")
gridExtra::grid.arrange(g1, g2, ncol = 2)

## logins
g1 <- shap.plot.dependence(data_long = shap_long, x = "logins", y = 'logins',
                           smooth = FALSE, color_feature = "giro_mailing") +
  mytheme +
  scale_color_gradient(low = "#008AE7", high = "#FF0056",
                       breaks=c(0,1), labels=c("0","1"),
                       guide = guide_colorbar(barwidth = 2, barheight = 0.6)) +
  scale_x_continuous(limits=c(0,100)) +
  labs(color = "giro_mailing ") +
  geom_hline(yintercept=0, colour = "darkgrey")
g2 <- shap.plot.dependence(data_long = shap_long, data_int = shap_int, x= "logins", y = "giro_mailing",
                           smooth = FALSE, color_feature = "giro_mailing") +
  mytheme +
  scale_color_gradient(low = "#008AE7", high = "#FF0056",
                       breaks=c(0,1), labels=c("0","1"),
```

```
                        guide = guide_colorbar(barwidth = 2, barheight = 0.6)) +
  scale_x_continuous(limits=c(0,100)) +
  labs(color = "giro_mailing ") +
  geom_hline(yintercept=0, colour = "darkgrey")
gridExtra::grid.arrange(g1, g2, ncol = 2)
```

Listing 2: Benchmark Logit Model

```
# Packages
library(readr)
library(missForest)
library(randomForest)
library(sjlabelled)
library(caret)
library(MASS)
library(pROC)

# Load xsell
xsell_logit <- as.data.frame(read_csv("data/xsell.csv"))
xsell_logit <- remove_all_labels(xsell_logit)

# Transform all character variables into factor variables
xsell_logit <- xsell_logit %>%
   mutate(xsell = as.factor(xsell),
          gender = as.factor(gender),
          marital = as.factor(marital))

# Transform numeric variabbles that should better be factor variables
xsell_logit <- xsell_logit %>%
   mutate(ppower = as.factor(ppower),
          avg_res_dur = as.factor(avg_res_dur),
          pop_km = as.factor(pop_km),
          car_seg = as.factor(car_seg))

# Impute missing values
set.seed(20)
xsell_logit <- rfImpute(xsell ~ . ,xsell_logit, iter = 4, ntree =100)

# Randomly select 80% of the observations without replacement
set.seed(20)
train_id <- sample(1:nrow(xsell_logit), size = floor(0.8 * nrow(xsell_logit)), replace
    =FALSE)

# Split in Training and Test (80/20)
xsell_first_round_test <- xsell_logit[train_id,]
xsell_logit_test <- xsell_logit[-train_id,]

# Randomly select 80% of the observations without replacement
set.seed(20)
train_id <- sample(1:nrow(xsell_first_round_test), size = floor(0.8 * nrow(xsell_first
    _round_test)), replace=FALSE)

# Split in Training and Validation (80/20)
xsell_logit_train <- xsell_first_round_test[train_id,]
xsell_logit_valid <- xsell_first_round_test[-train_id,]

# Logit model
model_logit <- glm(xsell ~., data = xsell_logit_train, family = "binomial")

# Make prediction
xsell_logit_test$pred_logit <- predict(model_logit, xsell_logit_test, type = "response
    ")
xsell_logit_test$pred_logit_factor <- factor(ifelse(xsell_logit_test$pred_logit> 0.5,
    1, 0), labels=c("No XSell","XSell"))

#Check accuracy with the confusion matrix
confusionMatrix(xsell_logit_test$pred_logit_factor, factor(xsell_logit_test$xsell ,
    labels=c("No XSell","XSell")),
                positive="XSell",dnn = c("Prediction", "Actual Data"))

```

```r
# ROC
roc_test_logit <- roc(xsell_logit_test$xsell, xsell_logit_test$pred_logit, percent=
    TRUE, plot=TRUE, print.auc=TRUE,grid=TRUE)
roc_test_logit
```

Listing 3: Random Search

```r
# Take start time to measure time of random search algorithm
start.time <- Sys.time()

# Create empty lists
lowest_error_list = list()
parameters_list = list()


# Create 10,000 rows with random hyperparameters
set.seed(20)
for (iter in 1:10000){
  param <- list(booster = "gbtree",
                objective = "binary:logistic",
                max_depth = sample(3:10, 1),
                eta = runif(1, .01, .3),
                subsample = runif(1, .7, 1),
                colsample_bytree = runif(1, .6, 1),
                min_child_weight = sample(0:10, 1)
  )
  parameters <- as.data.frame(param)
  parameters_list[[iter]] <- parameters
}

# Create object that contains all randomly created hyperparameters
parameters_df = do.call(rbind, parameters_list)

# Use randomly created parameters to create 10,000 XGBoost-models
for (row in 1:nrow(parameters_df)){
  set.seed(20)
  mdcv <- xgb.train(data=dtrain,
                    booster = "gbtree",
                    objective = "binary:logistic",
                    max_depth = parameters_df$max_depth[row],
                    eta = parameters_df$eta[row],
                    subsample = parameters_df$subsample[row],
                    colsample_bytree = parameters_df$colsample_bytree[row],
                    min_child_weight = parameters_df$min_child_weight[row],
                    nrounds= 300,
                    eval_metric = "error",
                    early_stopping_rounds= 30,
                    print_every_n = 100,
                    watchlist = list(train= dtrain, val= dtest),
                    maximize = TRUE
  )
  lowest_error <- as.data.frame(1 - min(mdcv$evaluation_log$val_error))
  lowest_error_list[[row]] <- lowest_error
}

# Create object that contains all accuracy measures
lowest_error_df = do.call(rbind, lowest_error_list)

# Bind columns of accuracy values and random hyperparameter values
randomsearch = cbind(lowest_error_df, parameters_df)

# Quickly display highest accuracy
max(randomsearch$`1 - min(mdcv$evaluation_log$val_error)`)

# Stop time and calculate difference
end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken

write_csv(randomsearch, "randomsearch.csv")
```

```
64
65 # Load random search output
66 randomsearch <- read_csv("randomsearch.csv")
67
68 # Prepare table
69 randomsearch <- randomsearch %>%
70   rename(val_acc = `1 - min(mdcv$evaluation_log$val_error)`) %>%
71   arrange(-val_acc) %>%
72   dplyr::select(-booster, -objective) %>%
73   mutate(val_acc = round(val_acc, 3),
74          eta = round(eta,3),
75          subsample = round(subsample,3),
76          colsample_bytree = round(colsample_bytree,3))
77
78 # Create tables for latex
79 stargazer(randomsearch[1:3,], summary = FALSE)
80 stargazer(randomsearch[9998:10000,], summary = FALSE)
81
82 # Compare best 10% with worst 10%
83 randomsearch$decile <- decile(vector=randomsearch$val_acc)
84 (randomsearch_group_by <- randomsearch %>%
85     filter(decile == 1 | decile == 10) %>%
86     group_by(decile) %>%
87     summarise(
88       max_depth = mean(max_depth),
89       eta = mean(eta),
90       subsample = mean(subsample),
91       colsample_bytree = mean(colsample_bytree),
92       min_child_weight = mean(min_child_weight)))
93
94 # Conduct t-tests between best and worst models
95 randomsearch_top <- randomsearch %>%
96   filter(decile == 1)
97 randomsearch_flop <- randomsearch %>%
98   filter(decile == 10)
99 t.test(randomsearch_top$max_depth, randomsearch_flop$max_depth)
100 t.test(randomsearch_top$eta, randomsearch_flop$eta)
101 t.test(randomsearch_top$subsample, randomsearch_flop$subsample)
102 t.test(randomsearch_top$colsample_bytree, randomsearch_flop$colsample_bytree)
103 t.test(randomsearch_top$min_child_weight, randomsearch_flop$min_child_weight)
```

Listing 4: Examining pref_device Variable

```
1 # Packages
2 library(readr)
3 library(dplyr)
4
5 # Import xsell
6 load(file= "data/xsell.RData")
7
8 # Replace NaN's or blank cells of occupation and pref_device with None_or_missing
9 xsell$pref_device <- ifelse(xsell$pref_device == "", "None_or_missing", xsell$pref_
      device)
10
11 # Data missing vs. not missing pref_device
12 missing_pref_device <- xsell %>%
13   dplyr::select(logins, mobile_logins, pref_device) %>%
14   dplyr::filter(pref_device == "None_or_missing")
15
16 not_missing_pref_device <- xsell %>%
17   dplyr::select(logins, mobile_logins, pref_device) %>%
18   dplyr::filter(pref_device != "None_or_missing")
19
20 # Standard deviation
21 sd(missing_pref_device$logins)
22 sd(not_missing_pref_device$logins)
23
24 # T-Test
25 t.test(missing_pref_device$logins, not_missing_pref_device$logins)
26 t.test(missing_pref_device$mobile_logins, not_missing_pref_device$mobile_logins)
```

Listing 5: Apply Hyperparameter-Tuned XGBoost Model to Predict xsell of Out-Of-Time Data Set

```
1
2  # Packages
3  library(Hmisc)
4  library(dplyr)
5  library(data.table)
6  library(tidyr)
7  library(e1071)
8  library(xgboost)
9  library(caret)
10 library(readr)
11
12 # Load data
13 load(file= "data/xsell_oot.RData")
14
15 # Transform occupation variable
16 xsell_oot$occupation <- ifelse(is.na(xsell_oot$occupation), 0, 1)
17
18 # Save obs
19 obs <- xsell_oot$obs
20
21 # Delete pref_device and obs
22 xsell_oot <- xsell_oot %>%
23   dplyr::select(-pref_device, -obs)
24
25 # Transform all character variables into factor variables
26 xsell_oot <- xsell_oot %>%
27   mutate(gender = as.factor(gender),
28          marital = as.factor(marital))
29
30 # Transform numeric variabbles that should better be factor variables
31 xsell_oot <- xsell_oot %>%
32   mutate(ppower = as.factor(ppower),
33          avg_res_dur = as.factor(avg_res_dur),
34          pop_km = as.factor(pop_km),
35          car_seg = as.factor(car_seg))
36
37
38 # Returns the NA object unchanged, if not changed, NA would be dropped
39 options(na.action='na.pass')
40
41 ## Create new variabe to build matrix like the trained dataset (problem: no missing
       values in gender in xsell_oot)
42 # Fast wasy of creating a column with only zeros
43 gendermissing <- ifelse(xsell_oot$age > 0, 0, 0)
44
45 # Put gendermissing befor gender
46 library(tibble)
47 xsell_oot <- add_column(xsell_oot, gendermissing, .after = 2)
48
49 # Create matrix
50 xsell_oot_train_matrix <-model.matrix(~ .-1, data = xsell_oot)
51
52 # Change column name to gender (like in training set) and create matrix again
53 xsell_oot_train_matrix <- as.data.frame(xsell_oot_train_matrix) %>%
54   rename(gender = gendermissing)
55 xsell_oot_train_matrix <-model.matrix(~ .-1, data = xsell_oot_train_matrix)
56
57 # Create DMatrix
58 dtest_oot <- xgb.DMatrix(data = xsell_oot_train_matrix)
59
60 # Load xgboost model
61 xgb_tuned <- xgb.load('data/xgb_tuned.model')
62
63 # Make prediction
64 xsell_oot$pred_xgb <- predict(xgb_tuned, dtest_oot, type = "response")
```

```r

# Create prediction with 1 or 0
xsell_oot$pred_xgb_factor <- factor(ifelse(xsell_oot$pred_xgb > 0.5, 1, 0), labels=c("
    No XSell","XSell"))

# Save as csv
xsell_oot_predicted <- cbind(xsell_oot, obs)
write_csv(xsell_oot_predicted, "data/xsell_oot_predicted.csv")
```

References

Athey, S. (2019), The impact of machine learning on economics, *in* A. Agrawal, J. Gans & A. Goldfarb, eds, 'The Economics of Artificial Intelligence: An Agenda: Conference held September 13-14, 2017', University of Chicago Press, pp. 507–547.

Bauer, C. L. (1988), 'A direct mail customer purchase model', *Journal of Direct Marketing* **2**(3), 16–24.

Bergstra, J. & Bengio, Y. (2012), 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research* **12**, 281–305.

Burez, J. & van den Poel, D. (2009), 'Handling class imbalance in customer churn prediction', *Expert Systems with Applications* **36**(3), 4626–4636.

Chawla, N. V. (2010), Data mining for imbalanced datasets: An overview, *in* O. Maimon & L. Rokach, eds, 'Data Mining and Knowledge Discovery Handbook', Springer, Boston, MA, pp. 875–886.

Chen, J., Le Song, Wainwright, M. J. & Jordan, M. I. (2018), Learning to explain: An information-theoretic perspective on model interpretation, *in* J. G. Dy & A. Krause, eds, 'Proceedings of the 35th International Conference on Machine Learning', pp. 882–891.

Chen, T. & Guestrin, C. (2016), Xgboost: A scalable tree boosting system, *in* B. Krishnapuram & M. Shah, eds, 'KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', pp. 785–794.

Choudhury, P. R., Allen, R. & Endres, M. G. (2018), 'Developing theory using machine learning methods'.

Friedman, J. H. (2001), 'Greedy function approximation: A gradient boosting machine', *The Annals of Statistics* **29**(5), 1189–1232.

Kamakura, W. A., Ramaswami, S. N. & Srivastava, R. K. (1991), 'Applying latent trait analysis in the evaluation of prospects for cross-selling of financial services', *International Journal of Research in Marketing* **8**(4), 329–349.

Keaveney, S. M. (1995), 'Customer switching behavior in service industries: An expioratory study', *Journal of Marketing* **59**, 71–82.

Lantz, B. (2013), *Machine Learning with R*, Packt Publishing.

Li, S., Sun, B. & Montgomery, A. L. (2011), 'Cross-selling the right product to the right customer at the right time', *Journal of Marketing Research* **48**(4), 683–700.

Li, S., Sun, B. & Wilcox, R. T. (2005), 'Cross-selling sequentially ordered products: An application to consumer banking services', *Journal of Marketing Research* **42**(2), 233–239.

Lipton, Z. C. (2018), 'The mythos of model interpretability', *Communications of the ACM* **61**(10), 36–43.

Lundberg, S. & Lee, S.-I. (2017), A unified approach to interpreting model predictions, *in* U. von Luxburg, I. M. Guyon, S. Bengio, H. M. Wallach & R. Fergus, eds, 'NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems', pp. 4768–4777.

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N. & Lee, S.-I. (2020), 'Explainable ai for trees: From local explanations to global understanding', *Nature Machine Intelligence* **2**, 56–67.

Lundberg, S. M., Erion, G. G. & Lee, S.-I. (2019), 'Consistent individualized feature attribution for tree ensembles', pp. 1–9.

Miglautsch, J. R. (2000), 'Thoughts on rfm scoring', *Journal of Database Marketing* **8**(1), 67–72.

Nielsen, D. (2016), 'Tree boosting with xgboost: Why does xgboost win "every" machine learning competition?', pp. 1–98.

Ribeiro, M. T., Singh, S. & Guestrin, C. (2016), "why should i trust you?" explaining the predictions of any classifier, *in* B. Krishnapuram & M. Shah, eds, 'KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', pp. 1135–1144.

Shah, D., Kumar, V., Qu, Yingge & Chen, S. (2012), 'Unprofitable cross-buying: Evidence from consumer and business markets', *Journal of Marketing* **76**, 78–95.

Thornton, C., Hutter, F., Hoos, H. H. & Leyton-Brown, K. (2013), Auto-weka: Combined selection and hyperparameter optimization of classification algorithms, *in* R. Ghani, T. E. Senator, P. Bradley, R. Parekh & J. He, eds, 'KDD '13: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining', pp. 847–855.

Wedel, M. & Kannan, P. K. (2016), 'Marketing analytics for data-rich environments', *Journal of Marketing* **80**(6), 97–121.