Un-Black-Boxing Artificial Neural Networks

Predicting and Explaining Bank Customer's Cross-Sell Likelihood

Seminar Thesis

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

Lukas Jürgensmeier

in partial fulfillment of the requirements
for the degree of

Master of Science in Business Administration

July 15, 2019

Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial Neural Network
API	Application Programming Interface
AUC	Area Under Curve
LIME	Local Interpretable Model-agnostic Explanation
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristics
SP	Submodular Pick

1 Introduction

The rise of machine learning techniques has come with a great increase of predictive accuracy in many settings (Jordan & Mitchell 2015). The hype around those techniques has led to a widespread use of them in many fields in science and practice—from medicine to business or self-driving cars (Chen & Asch 2017). However, some of those methods are increasingly computationally expensive and might not in every case be superior to traditional regression-based methods such as simple multivariate linear regression or logit models.

This paper uses a data set from a large German bank with numerous customer characteristics to predict whether an existing customer buys another product—also referred to as cross-selling. The goal of this research is first to predict weather a customer will cross-buy and open a checking account and second to identify certain characteristics of typical customers that increase the likelihood of opening that account. Therefore, this paper introduces a complex Artificial Neural Network (ANN) with optimized predictive power and compares the predictions to a simple logit model. One downside of ANN models in particular the lack of interpretability—they are widely regarded as black-boxes (Benítez et al. 1997, Dayhoff & DeLeo 2001). This paper addresses this shortcoming by implementing LIME—a method to explain individual predictions of neural networks (Ribeiro et al. 2016). The paper is structured as follows: The first chapter gives a short introduction to cross-selling theory and is followed by the explanation of the implemented statistical methods. The subsequent chapter presents the empirical results and especially compares the ANN to the simple logit model. The paper concludes with implications for practitioners and outlines further research avenues.

2 A Short Primer on Cross-Sell

Cross-selling is an important action field for established companies with an existing customer base (Li et al. 2011). Selling additional products to present customers taps into a readily accessible target group for additional sales potential. Felvey (1982) argues that a businesses' client base might already be more inclined to buy another product from the company that they already have a relationship with, compared to non-

customers. Also, the cost of selling additional products to already existing customers is cheaper than targeting new ones (Reichheld & Sasser Jr 1990). Benefits of cross-selling from the company point-of-view include increased switching costs for customers, while customers benefit from an easy integration of different products and a higher certainty about the quality of the new product (Kamakura et al. 1991, 2003). However, as in most business settings, there is no free lunch and hence cross-selling efforts are also associated with costs and risks. Apart from trivial costs for cross-selling efforts such as mailing or telephone campaigns, there is a significant churn risk involved. If not accurately targeted, customers can be annoyed by irrelevant and continuous advertisement and might eventually churn as a result (Keaveney 1995). Due to the significant sales potential on the one hand and costs and risks involved on the other hand, it is therefore paramount to accurately identify the target group that is most likely to cross-buy another product. This paper addresses this issue and presents an Artificial Neural Network to accomplish that task in the best possible way.

3 Methodology

This section provides an overview over the implemented methodology by first outlining the feature engineering process that transforms the original data set to suitable format for neural networks. The second part then describes the ANN and its initial architecture, while the third part describes the hyperparameter tuning process that leads to the *best* model[1]. Lastly, this section addresses a common criticism of machine learning technologies in general and neural networks in particular—the uninterpretable black box—by introducing the Local Interpretable Model-agnostic Explanation (LIME) as a method to "look under the hood" of a neural network and determine which features contributed by how much to a prediction.

3.1 Feature Engineering and Data Set Preparation

In order to feed seamlessly into a neural network, extensive feature engineering and data preparation is required (Hastie et al. 2017). Before training the model, I replace

[1]This model is definitely not the best model *existing*, but the best model *found* through the trial-and-error tuning process described in detail in section 3.3.

missing values in `pref_device` and `occupation` by `None_or_missing`, transform all `char` variables to a `factor` type, and remove the misleading feature `ID`. However, there are still numerous features including missing values. Since they are not missing at random, they need to be imputed in order to not bias the prediction. Deleting all observations with missing values would do exactly that. Hence, I impute missing values with a Random Forest from the **randomForest**`::rfImpute()` function. This algorithm first replaces missing numeric (factor) variables with the median (mode) and then uses proximity measures from `randomForest` as weights to replace `NA's` with the weighted average of the non-missing values and repeats this for a pre-defined number of iterations[2] (Liaw & Wiener 2002). I furthermore bin the continuous features `age`, `entry_age` and `last_acc`[3] in four distinct groups. Since a neural network cannot handle multi-categorical features, the routine dummy-codes those. Finally—to not distort the input weights for the first hidden layer and thus not negatively affect the prediction quality (Hastie et al. 2017, pp. 398)—I standardize all input features to exhibit $\bar{x} = 0$ and $s = 1$. The feature engineered data set consists of $n_x = 74$ input variables that will be used to predict cross-sell.

3.2 Artificial Neural Network with Keras

While the previous section discussed the steps required to transform the data to a suitable format for an ANN, this part describes the basic architecture of this classification method. This study uses the R interface **keras** (Chollet & Allaire 2017) to the popular same name Python library. As a high-level API, it enables an easier way to access the features of Google's **TensorFlow** (Abadi et al. 2015).

The neural network in this study is a common feedforward network which uses the backpropagation algorithm to adjust its weights (Werbos 1990). Based on the feature

[2]After four iterations the out-of-bag error did not decrease notably anymore, hence four iterations were chosen.

[3]Those three features exhibit a highly non-linear relationship. According to the universal approximation theorem, a neural network with enough units in a hidden layer can approximate any continuous function (Hornik 1991). The best neural network will be found by trying out different hyperparameters, including some combinations which would not have *enough* hidden layer units for a solid approximation. Hence, I bin those three variables to make sure that this non-linear relationship can be taken into account by the model in any case. Also, the benchmark logit model could not capture this effect without further specification.

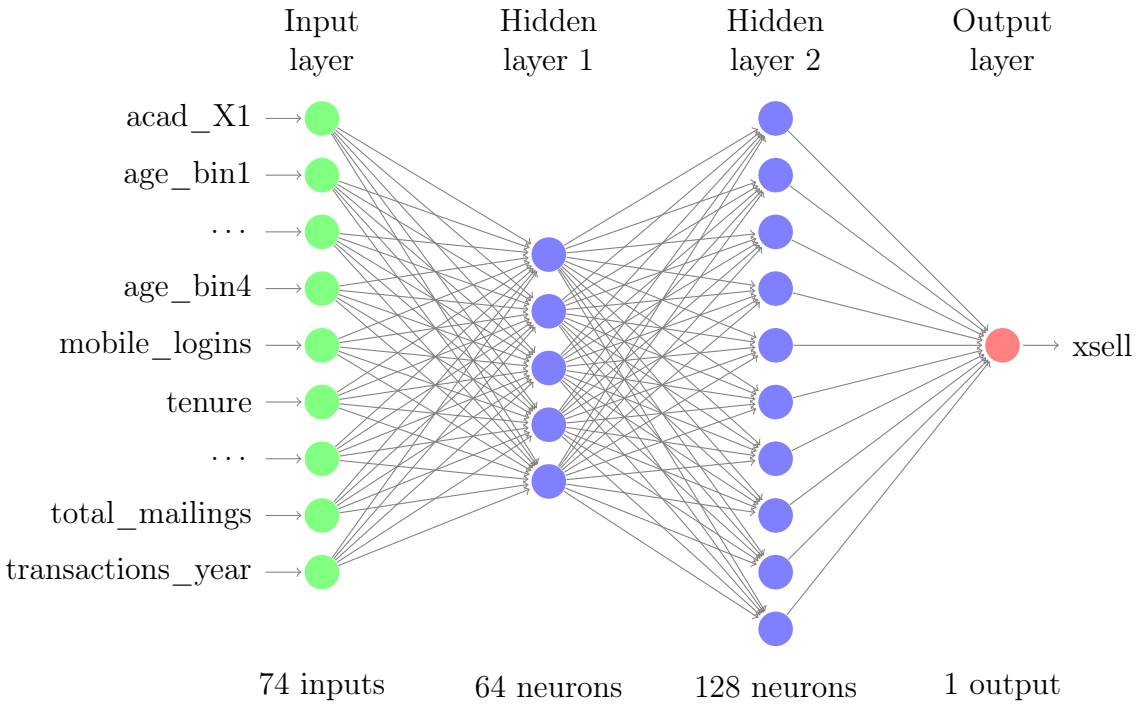

Figure 1: Architecture of the Artificial Neural Network implemented in this study

engineering process described in the previous section, the ANN has $n_x = 74$ inputs X_i, two hidden layers with neurons H_i^l referring to the i^{th} neuron in the l^{th} hidden layer. The number of neurons in the hidden layers is subject to hyperparameter tuning in the following section. As a final layer, the ANN predicts the single binary output $\hat{Y}_i$, in our case `xsell`. Figure 3.2 graphically visualizes the architecture. This network uses a random uniform distribution $U(-0.05, 0.05)$ to initialize the weights for each neuron H_i^l. Also, a bias initialized at zero feeds into the two hidden layers, for which each H_i^l is activated by a rectifier function (`ReLU`). Glorot et al. (2011) show that rectifying units in an ANN yield superior outcomes compared to others such as the hyperbolic tangent activation function. Since the output $\hat{Y}_i$ is a binary feature, I use a sigmoid activation function for the output layer. To avoid overfitting, the model implements dropout for the hidden layers. This process randomly drops neurons and their connections from the ANN and thus leads to a more generalizable model (Srivastava et al. 2014). Mitigating the risk of overfitting to the training data is furthermore a key objective of the next section.

3.3 Cross-validation and Hyperparameter Tuning

After setting up the basic architecture of the neural network, this section describes the process that leads to the *best* model[4]. An important feature of the *best* model is that it does not memorize the exact relationships in the training data set—overfitting—but rather learns the generalizable underlying mechanisms an thereby performs well on the holdout validation and subsequently on the test or even an out-of-time data set. For this purpose I first describe the optimization of the ANN by cross-validating predictions of several training epochs with a holdout sample and then introduce hyperparameter tuning—a process that leads to the most desirable[5] configuration of the ANN by running many combinations of different model specifications.

To compile the ANN, I use the adaptive moment estimation algorithm `Adam` (Kingma & Ba 2014). Ruder (2016) recommend this gradient descent optimizer as computationally inexpensive and performing best in most applications. This optimizer minimizes the cross-entropy loss (Zhang & Sabuncu 2018) for each training epoch and accordingly updates the weights for the next model. Figure 2 shows the training history of the chosen ANN over ten epochs. The cross-entropy loss is plotted in the top pane, and the accuracy displayed in the bottom part, each for the training and validation data set.

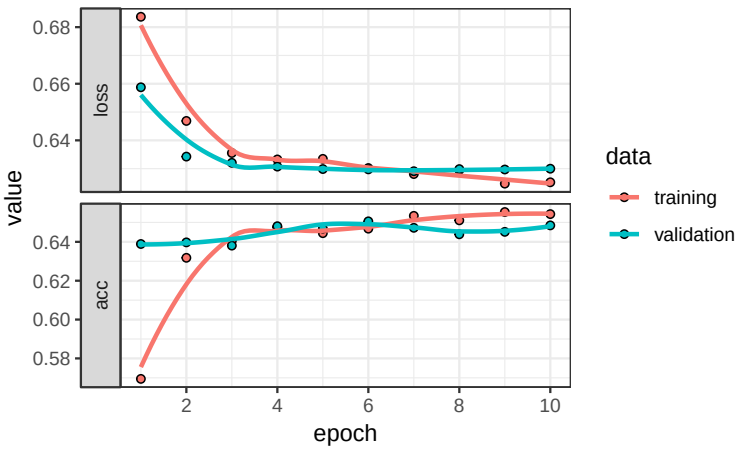

Figure 2: Loss and accuracy for training and test data over training history of the tuned model

[4] as mentioned earlier, this is not the best model *existing*, but *found*.

[5] "Desirable" is defined as most accurate in this study. However, other metrics such as loss are also valid optimization criteria.

The validation accuracy in comparison with the training accuracy in this case indicates first how accurate the ANN predicts true positives and true negatives in both data sets, but more importantly, gives an indication whether overfitting is an issue. If the training accuracy kept steadily increasing, while the validation accuracy remained relatively constant, the model would overfit—which means that the model is not generalizable beyond the training data set (Hansen & Salamon 1990). As exhibited in figure 2, both accuracies are relatively constant and rather similar during the last training epochs. This is a strong indication that the model does not overfit and will perform similarly well on the test or out-of-time data.

However, one question remains unanswered with cross-validation: Which model architecture leads to the most accurate predictions? Hyperparameter tuning provides a remedy to this issue (Bergstra & Bengio 2012). In short, this process runs numerous models with different combinations of hyperparameters[6]. The model with the highest validation accuracy `val_acc` is then chosen to be the *best* model for predicting `xsell`. This study uses the "naive" grid method, which runs all possible combinations of pre-defined values for hyperparameters[7]. However, this method is computationally expensive. It requires $N = a^b$ different models, where a is the number of discrete values for each parameter and b is the number of hyperparameters to be tuned. Running two separate sets of hyperparameter combinations for 324 different models in total took roughly five hours on a standard consumer laptop. It can be assumed that this limited set of combinations did by far not yield the best model. However, this naive grid method runs an exponentially increasing number of different models and thus gets difficult to handle quite quickly. A possible solution to this problem is random search, which is even more efficient than grid search (Bergstra & Bengio 2012).

By cross-validating the predictions with a loss function and by tuning the ANN's hyperparameter, this section described the approach to finding the most suitable model for the prediction task at hand. The next part turns to prediction explanation—a key

[6]Hyperparameters refer to the specifications of an ANN, such as number of units per hidden layer, dropout rate and weights, but also different optimization algorithms and activation functions (Bengio 2000)

[7]I use each three different values of the hyperparameters `val_acc`, `acc`, `units1`, `dropout1`, `units2`, `dropout2`, `epochs` and `batch`

discussion point and critique of neural networks.

3.4 Local Interpretable Model-agnostic Explanation (LIME)

In contrast to traditional regression-based methods that yield beta coefficients and enable the construction of a functional form of the underlying mechanisms in the data, a neural network does not provide an easily interpretable function that links Y_i to X_i. A common criticism of ANN hence is the "black-box" argument (Benítez et al. 1997, Dayhoff & DeLeo 2001). Though the model might yield more accurate predictions, there is no clear answer to the question why the model predicted those particular values or which features contributed by how much.

With the development of new methods that address this problem, an ANN is less a black box. Consistent with the research question to uncover factors that drive cross-sell, this section focuses on the Local Interpretable Model-agnostic Explanation (LIME) introduced by Ribeiro et al. (2016) as a methodology to assess feature importance on a single customer level[8]. LIME uses a trick to come up with an interpretable explanation: While the highly non-linear function of the ANN is unknown over the total space, it can be approximated locally by an easier, readily interpretable model such as a linear regression or a decision tree.

While those models globally do not match the functional form of the neural network at all, they can yield accurate *local* approximations (Ribeiro et al. 2016, see figure 3 for a visual example). Turning to the task at hand, this paper uses a multiple linear regression with the ten most important features for every customer. LIME permutes every observation and calculates the `gower`-distance between the observation and the permutation (Pedersen & Benesty 2018). It then fits a forward-selection procedure on the basis of a ridge regression (Pedersen 2018) to identify the ten most important features, weighted by the distance calculated earlier. Following Ribeiro et al. (2016), the beta coefficients ("weights") are then reported as a locally accurate approximation of feature importance. This method hence enables to go beyond the typical "black-

[8]Other tools such as neural interpretation diagrams, Garson's algorithm, sensitivity analysis or randomized approaches (Olden & Jackson 2002) offer similar explanations on an aggregate level. However, covering those is beyond the scope of this paper.

box" description of a neural network. While this chapter described the implemented methods theoretically, the remainder of this paper reports and interprets the results of those machine learning techniques applied on the cross-sell data set.

4 Results and Model Comparison

This section proceeds as follows: First, it reports the results of the hyperparameter tuning and cross-validation as well as the resulting optimal ANN architecture. The second part then compares the predictions of the neural network to a simple benchmark logit model. It concludes by reporting the feature importance for individual customers derived by LIME.

4.1 Choosing the Neural Network Architecture

The task of choosing the *best* ANN architecture is not trivial. Using the approach described in section 3.3, I ran a total of 324 different models, each with different combinations of hyperparameters. Ranked by validation accuracy, table 1 shows the top five and bottom two models with its respective model specification. It is noteworthy

Table 1: Resulting statistics of 324 hyperparameter tuning model runs

rank	val_acc	acc	units1	dropout1	units2	dropout2	epochs	batch
1	0.656	0.654	64	0.600	128	0.600	10	100
2	0.654	0.692	128	0.600	64	0.600	30	150
3	0.653	0.674	64	0.600	64	0.600	30	100
4	0.651	0.658	64	0.600	64	0.600	10	100
5	0.651	0.673	64	0.600	128	0.600	30	50
...	...	...	...	...	...	...	...	...
323	0.604	0.731	64	0.200	32	0.400	30	128
324	0.602	0.735	64	0.200	32	0.200	30	128

that this evaluation metric only varies by five percentage points from the best (65.6%) to the worst model (60.2%). Also, the dropout rate for all five best performing models is 0.6 for both hidden layers, while the number of units in those varies between 64 and 128. In contrast to that the least preferable models exhibit only 32 neurons in the second hidden layer. In this case, a higher number of neurons in those layers appears to

better capture the effects of the input features on `xsell`. The winning model used ten training epochs, that means the model adjusted weights by backpropagation ten times before stopping. This value is rather low[9] and suggests an advantage of stopping early to avoid overfitting. Finally, the size of the batches—that is the number of samples sent back and forth through the network at once (Chollet & Allaire 2017)—apppears to not have an obvious impact on model performance.

Even though this routine identified the best out of 324 models, it only tested three different values for eight hyperparameters in two seperate runs. Due to the computational complexity, important parameters such as the *learning rate* were not included. Furthermore, I only tested three more or less arbitrarily chosen discrete values for the hyperparameters—it is therefore obvious that better models do exist. It is however difficult to identify them with the grid method and limited computational resources.

4.2 Comparing the Predicions of the ANN with the Benchmark Logit Model

The ANN from the previous section with the highest validation accuracy is furthermore used to predict `xsell` in the remaining unseen test data set[10]. To assess the quality of the model, this paper now compares the resulting model statistics and confusion matrices of the discussed ANN with those of a simple logit model based on the same 74 input features used for the neural network[11]. Table 2 displays all relevant model performance metrics from accuracy to kappa and each confusion matrix. To a surprise, there is no apparent difference in model accuracy between the hyperparameter-tuned ANN and the simple benchmark logit model. The reference model even predicts `xsell` 0.4 percentage points more accurate[12]. Since 50% of all customers in the data set made a purchase, this is the benchmark accuracy for any predictive model. Compared to a

[9] I tested epoch values of 5, 10, and 30 for the tuning routine.

[10] 20% of the 10,000 observations in the data set were used for testing, while the remaining 8,000 observations form the training set. Out of those, I reserved 2,400 rows (30%) as a holdout validation sample for cross-validation during the fitting process.

[11] In fact, this logit model has a relatively simple functional form, since it throws in every feature at hand without any selection mechanism or concern for multicolllinearity. However, those features have undergone the same extensive feature engineering process as described for the ANN. Hence, a lot of concerns are dealt with, e.g. though not functionally mapped, the logit model incorporates non-linear effects of `age`, `entry_age`, and `last_acc` through binning.

[12] Due to the small test set sample size of 2,000 this effect might not be statistically significant.

coin toss, both the ANN and logit model yield a roughly 29% increase in accuracy. Nevertheless, both models produce structurally different predictions. This becomes

Table 2: Model performance comparison—statistics and confusion matrix

Benchmark Logit			*Hyperparameter-tuned ANN*		
Accuracy	0.6463		Accuracy	0.6423	
Sensitivity	0.6653		Sensitivity	0.7331	
Specificity	0.6271		Specificity	0.5508	
Pos Pred Value	0.6429		Pos Pred Value	0.6221	
Neg Pred Value	0.6500		Neg Pred Value	0.6716	
Kappa	0.2925		Kappa	0.2841	
Prevalence	0.5023		Prevalence	0.5023	
	Actual			*Actual*	
Prediction	0	1	*Prediction*	0	1
0	624	336	0	548	268
1	371	668	1	447	736

clear when comparing sensitivity (true positive rate) and specificity (true negative rate). The ANN is relatively better at predicting customers that actually buy another product (`xsell = 1`), while the logit model more accurately predicts negative outcomes. This has far-reaching implications for business applications, since certain wrong predictions might be more costly than others[13].

Another widely used visual tool and metric for model comparison in machine learning is the receiver operating characteristics (ROC) curve and the associated area under curve (AUC) (Fawcett 2006). Figure 3 plots both ROC curves and the AUC in a single plot—the curves are visually inspected almost identical. Quantified by the AUC, the ANN performs 0.9 percentage points better than the logit model at 69.3%.

4.3 Beyond the Black Box: Feature Importance with LIME

The previous section introduced the implemented ANN model specification and reported the results in terms of predictive performance. However, prediction is only one part of the research question. The other equally important question to address is explainability. Out-of-the-box, machine learning algorithms such as ANN are indeed a

[13]The scope of this paper does unfortunately not allow a thorough discussion of different predictive error costs. See Domingos (1999) for a detailed discussion on this topic and a proposed solution that incorporates costs into any classifier.

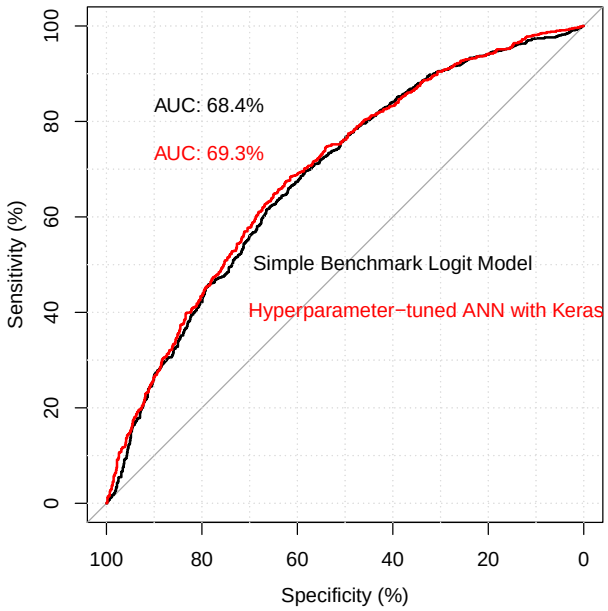

Figure 3: ROC curve comparison between benchmark logit (black) and hyperparameter-tuned neural network predictions (red)

"black-box" and thus it is impossible to understand why the model predicted which outcome. LIME, as introduced theoretically in section 3.4, provides insights into the reasoning of why the model came to a certain conclusion by locally fitting an explainable model and reporting the resulting feature weights as a measure for its contribution to the prediction (Ribeiro et al. 2016). In this paper, I focus on explanations for individual observations, i.e. individual customers. Explaining feature importance on a global level would be the next step. However, the limited scope of this paper does not permit a thorough discussion of both. In a business setting, it is crucial to understand why the model predicted a certain outcome for cross-selling of an individual customer, especially for building human trust in the machine learning algorithm's prediction (Ribeiro et al. 2016). The remainder of this section therefore focuses on this explanation only.

Figure 4 visualizes the results of **lime**`::explain()` for the first four customers in the test data set. For each of those customers, the ten most important features are reported in descending order of their influence on the model's prediction. *Label* is the model's prediction whether a customer buys another product, *probability* gives a degree of the

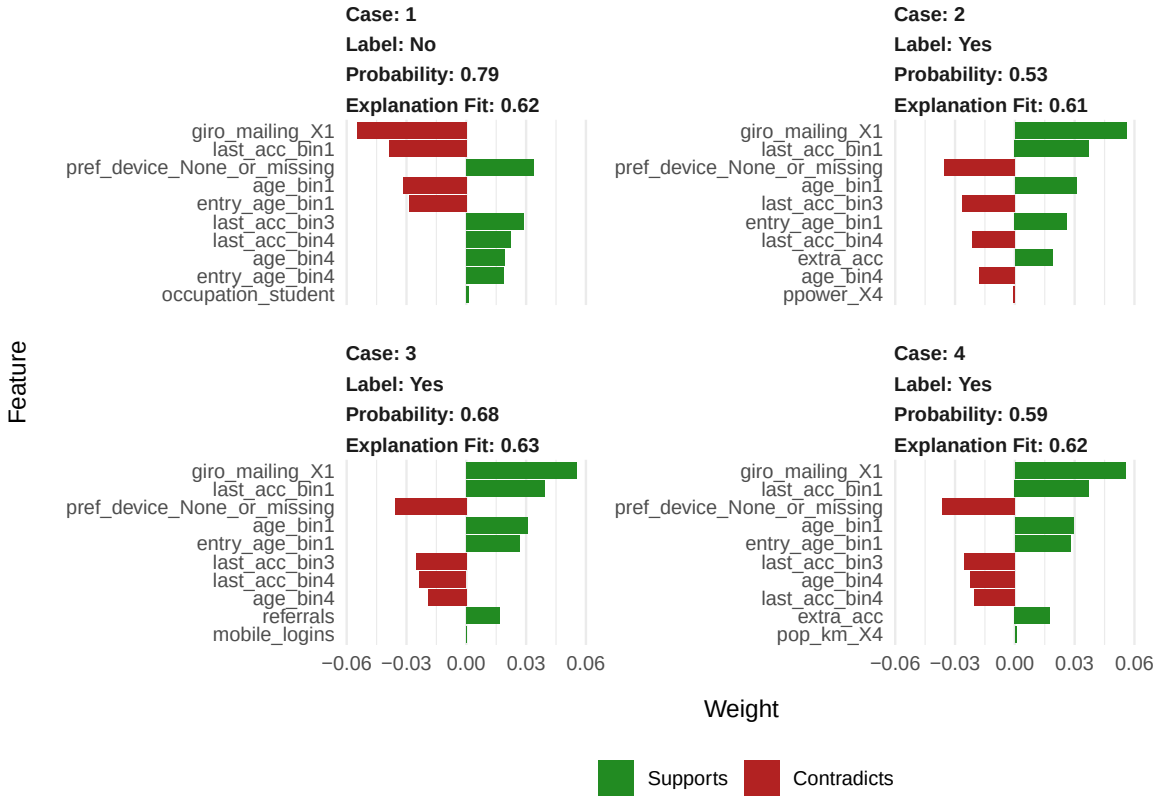

Figure 4: LIME feature importance of the ten most important features for each of the first four customers in the test data set

model's confidence in the prediction, and *explanation fit* reports the locally approximated model's R^2 with the ten most important features utilized. It is apparent that the features exhibiting the highest correlation with `xsell` also play the biggest role for those four customers (see figure 6 in the appendix for a graphical representation of correlation in the data set).

A red bar indicates that the particular value of this feature contradicts the model's conclusion, while a green bar signals supporting evidence to the model's prediction. Whether the customer received an advertisement mail about opening a checking account (`giro_mailing_X1`) shows the largest weight from the regression model in every one of the four cases. Also the group with the lowest days passed since the customer last opened an account (`last_acc_bin1`) shows a very large contribution to the prediction in those four cases. Additionally, the customers' age seems to have a big impact on cross-sell predictions—its binned version is consistently included as a top reason for

the predictions. However, note that no generalization on the entire data set is possible from this interpretation, since it only explains individual predictions.

A first step towards a general statement about the customer base's characteristics is looking at the distribution of features split by whether the customer opened a checking account or not. For `age`, the violin plot in figure 5 shows a striking difference in the age distribution. Younger customers tend to more often cross-buy. Looking at the correlation of all features with `xsell` in figure 6 in the appendix, the youngest age group shows the highest positive correlation after `giro_mailing_X1`. Receiving a mail with an offer to buy another product in every of the four individual explanations is the feature that has the highest influence on the model's prediction, while it also displays the highest correlation with cross-sell. On the contradicting side, a missing preferred device displays the highest negative correlation, while it also comes up as the third most important feature in the individual LIME explanations. Though a mere correlation does not imply causality, it can provide a first clue about general underlying mechanism and can validate the LIME results. Judging from the first four customers only, the locally fitted models' weights are in line with the overall correlations.

Combining the LIME explanations with the actual values for those customers would now enable to act on those results. The following section further elaborate on this issue and discusses the managerial implications that those methods bring with.

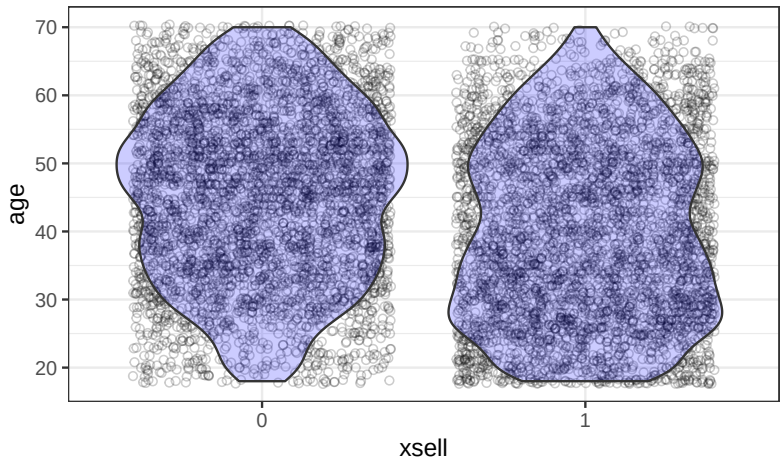

Figure 5: Individual variable assessment—violin plot of xsell vs. age

5 Managerial and Research Implications

This section first highlights managerial implications that can be derived from this study and then outlines future research areas in this field. Looking at the ANN's accuracy compared to a simple benchmark logit model one recommendation for practice is obvious: Do not use neural networks in this particular business setting. They are computationally expensive and provide no overall predictive advantage to way simpler models such as logit. Possible reasons for the disappointing ANN performance could be the lack of enough data—neural networks require larger data sets compared to well-specified simple regression-based models for accurately learning the underlying relationships (Chan et al. 1999). Also, neural networks are better suited for other tasks such as image recognition (Esteva et al. 2017) and speech (Hinton et al. 2012) or text recognition (Jaderberg et al. 2014). In those cases, a simple multivariate regression would not be suitable.

Nevertheless, LIME provides an important way to understand why the black-box ANN predicted cross-selling. This is important to first get an understanding which features are contributing by how much and in which direction. Out-of-the-box, this would not be possible with ANNs. More importantly however, LIME builds trust in the prediction of an otherwise mysterious algorithm. By fitting a locally reasonable and interpretable model, this new approach enables decision makers to assess the validity of the neural network's predictions beyond typical performance measures such as accuracy and AUC (Ribeiro et al. 2016).

Still, this paper lacks a generalizeable explanation beyond simple descriptive metrics such as correlation (see figure 6) or the distribution of observations by `xsell` as shown in figure 5. Though individual predictions can be explained by LIME, one cannot derive a truly general statement about the sample from it. In their research on LIME, Ribeiro et al. (2016) suggest submodular picking of individual observation-level explanations in order to derive generalizable insights into the black-box model. Simplified, they propose an algorithm that optimally picks the most representative individual observations for the entire sample. Unfortunately, the LIME-SP routine is so far only implemented in the original Python package and currently under development for the corresponding

keras R interface (Kavicky 2017).

This empirical analysis suggests which customer groups are likely to purchase another product. Most importantly, young customers exhibit a higher probability to purchase than old ones. The more recently a customer already cross-bought, the more likely it is that the bank can sell an additional product to this client again. Targeting those customer groups in particular could be a valid business strategy. However, more research would be necessary to establish a clear causal relationship between those variables. A neural network is definitely not the method of choice for this task—even though new developments such as LIME can greatly expand the trust and explainability of black-box models.

6 Conclusion

The first goal of this paper was to predict whether a bank customer opens another checking account based on their individual characteristics. A sophisticated hyper-parameter-tuned Artificial Neural Network was therefore introduced and its predictions compared to a simple logit model. Surprisingly, the ANN performed slightly worse than the benchmark logit model in terms of accuracy and only marginally outperformed it in terms of AUC. Due to the enormous computational resources required for finding a satisfactory ANN, this paper argues against the implementation of neural networks for predicting cross-sell in this particular business setting and for simpler methods such as a logit model.

The second research question—identifying underlying characteristics that drive cross-selling—was initially impossible to answer with a black-box ANN. However, this paper implemented a local approximation of the neural network with LIME by an easily interpretable regression model. This enabled detailed insights into which customer features support or prevent cross-selling on an individual customer level. Most importantly, sending mail about opening a new account is the most contributing feature for the four analyzed customers, followed directly by the customer's age. Younger clients are more likely to cross-buy, which is also supported on the global level by looking at correlations and individual scatter plots. Even though those two methods mostly answer the

research question, there is still more research to be done to establish causality. On a global level, the methodology implemented in this study only reports which features are associated with cross-selling. However, no reasonable statement can be made about which features *cause* an increased cross-selling probability across all customers.

A Appendix

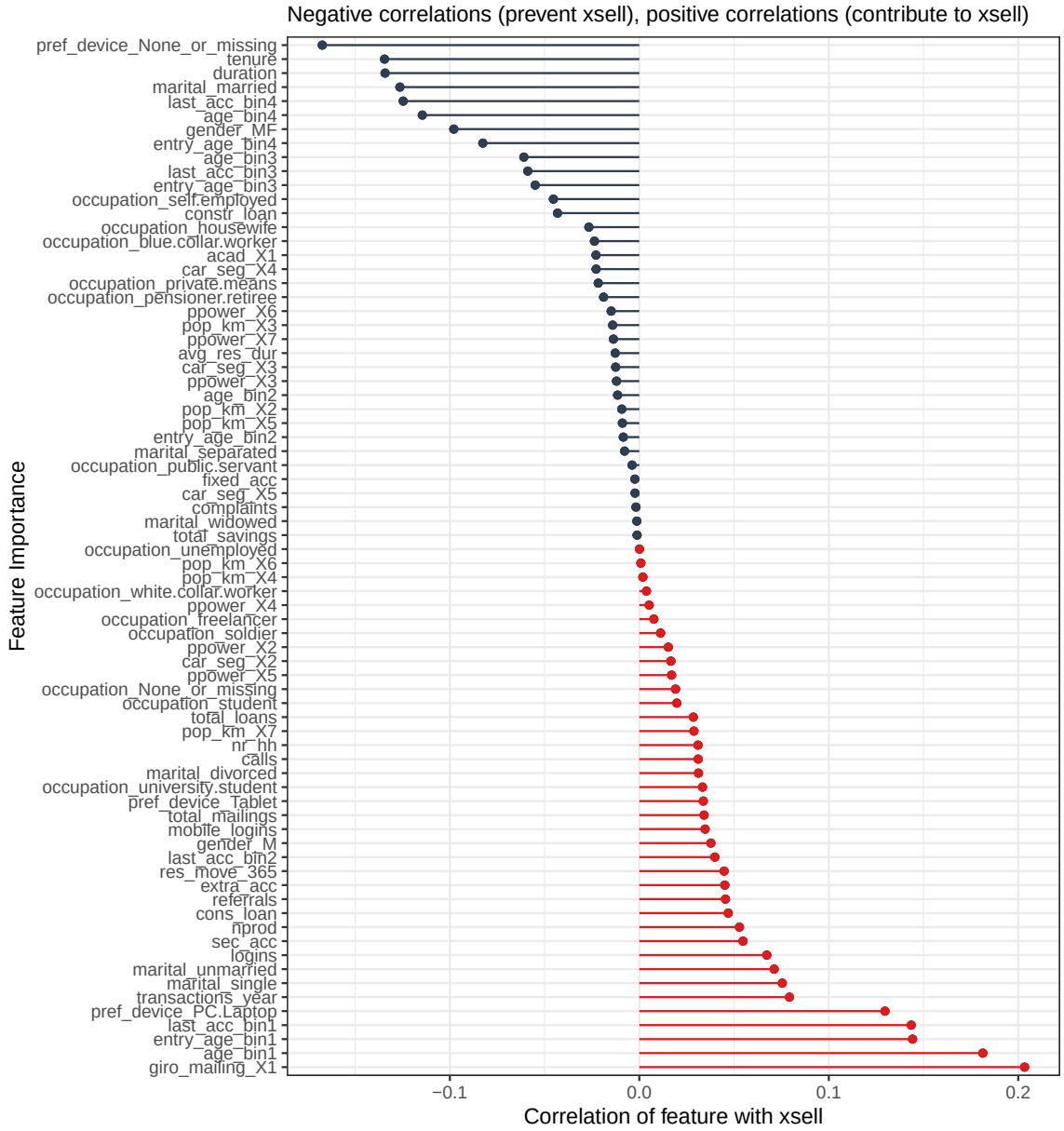

Figure 6: Correlation of feature engineered variables with xsell

B R Code

```r
##############################################
#####             Libraries             #####
##############################################

# clear workspace
rm(list = ls())

#install packages as required

# Exception: Installing Keras is a little tricky.
# You additionally need a python installation on your machine
#install_keras(method = "conda")

# Load libraries
library(keras)
library(lime)
library(tfruns)
library(tidyquant)
library(rsample)
library(recipes)
library(yardstick)
library(corrr)
library(randomForest)
library(missForest)
library(neuralnet)
library(caret)
library(dplyr)
library(corrplot)
library(pROC)
library(processx)
library(caret)
library(e1071)
library(stargazer)

# set same seed for R, Python, NumPy and Tensorflow
use_session_with_seed(42)

##############################################
##  Data cleaning and Feature Engineering  ##
##############################################

# read and check data
xsell_data_raw <- read.csv("xsell.csv", na.strings=c("", "NA"), stringsAsFactors =
    FALSE)
glimpse(xsell_data_raw)
summary(xsell_data_raw)

# shuffle rows
xsell_data_raw <- xsell_data_raw[sample(nrow(xsell_data_raw)),]

# create new variable tenure
xsell_data_raw$tenure <- xsell_data_raw$age - xsell_data_raw$entry_age

# data cleaning, replace NAs in char-variables with "None_or_missing"
xsell_data_raw <- xsell_data_raw %>%
  replace_na(list(pref_device = "None_or_missing"))
xsell_data_raw <- xsell_data_raw %>%
  replace_na(list(occupation = "None_or_missing"))

# all character columns to factor:
xsell_data_raw <- mutate_if(xsell_data_raw, is.character, as.factor)
#additional numeric variables that should rather be treated as factors
xsell_data_raw$car_seg <- as.factor(xsell_data_raw$car_seg)
xsell_data_raw$acad <- as.factor(xsell_data_raw$acad) # remove if strange results
xsell_data_raw$giro_mailing <- as.factor(xsell_data_raw$giro_mailing) # remove if
    strange results
xsell_data_raw$pop_km <- as.factor(xsell_data_raw$pop_km)
```

```r
xsell_data_raw$ppower <- as.factor(xsell_data_raw$ppower)

# Remove unnecessary data and clean data set
xsell_data_tbl <- xsell_data_raw %>%
  select(-X) #%>% #removes ID
  # if you don't want to run a random forest for NA imputation, you can do apply of the two easier fixes to NA's:
  #drop_na() #%>% # removes all NA's. Bad Solution! Improve! Removes 70% of observations
  #na.roughfix(xsell_data_raw) #replaces NA's: Numeric with median, factor with mode

# Impute NAs with a Random Forest
xsell_data_tbl$xsell <- as.factor(xsell_data_tbl$xsell) # transform to factor for random forest imputation
xsell_data_tbl <- rfImpute(xsell ~ . ,xsell_data_tbl, iter = 4, ntree=100)
xsell_data_tbl$xsell <- as.integer(ifelse(xsell_data_tbl$xsell == "0", 0, 1)) # transform back to numeric

glimpse(xsell_data_tbl)

# Split test/training sets
train_test_split <- initial_split(xsell_data_tbl, prop = 0.8)
train_test_split

# Retrieve train and test sets
train_tbl <- training(train_test_split)
test_tbl  <- testing(train_test_split)

# define features for binning
to_bin <- c("age", "entry_age", "last_acc")

# Create recipe
rec_obj <- recipe(xsell ~ ., data = train_tbl) %>%
  step_discretize(to_bin, options = list(cuts = 4)) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_center(all_predictors(), -all_outcomes()) %>%
  step_scale(all_predictors(), -all_outcomes()) %>%
  prep(data = train_tbl)

# Apply recipe to predictors (all vars excluding xsell)
x_train_tbl <- bake(rec_obj, new_data = train_tbl) %>% select(-xsell)
x_test_tbl  <- bake(rec_obj, new_data = test_tbl) %>% select(-xsell)
glimpse(x_train_tbl)

# define response variables for training and testing sets
y_train_vec <- pull(train_tbl, xsell)
y_test_vec  <- pull(test_tbl, xsell)

# visually check transformed data set with a histogram for each feature
x_train_tbl %>%
  gather(colnames, xsell) %>%
  ggplot(aes(x = xsell)) +
    geom_histogram() +
    facet_wrap(~colnames,
               scales = 'free',
               ncol = 9)+
  theme_bw()

# correlation matrix
correl_matrix <- cor(x_train_tbl,use="pairwise.complete.obs")
# correlation plot
corrplot(correl_matrix)

##############################################
# Building the Artificial Neural Network #####
##############################################

# I tuned the hyperparameters by trying out 324 different models
# See the scripts hyperpar_tuning_tf_runs.R and keras_nnet_architecture
# I then use the "best" hyperparameters below according to the results of those runs
```

```
132
133 # Setting up the ANN with Keras
134 model_keras <- keras_model_sequential()
135
136 # the hyperparameters inserted here were tuned with the hyperpar_tuning_tf_runs.R
      script
137 model_keras %>%
138
139   # First hidden layer
140   layer_dense(
141     units              = 64,
142     kernel_initializer = "uniform",
143     activation         = "relu",
144     use_bias           = TRUE,
145     bias_initializer   = 'zeros',
146     input_shape        = ncol(x_train_tbl)) %>%
147
148   # Dropout to prevent overfitting
149   layer_dropout(rate = 0.6) %>%
150
151   # Second hidden layer
152   layer_dense(
153     units              = 128,
154     kernel_initializer = "uniform",
155     use_bias           = TRUE,
156     bias_initializer   = 'zeros',
157     activation         = "relu") %>%
158
159  # Dropout to prevent overfitting
160  layer_dropout(rate = 0.6) %>%
161
162   # Output layer
163   layer_dense(
164     units              = 1,
165     kernel_initializer = "uniform",
166     activation         = "sigmoid") %>%
167
168   # Compile ANN
169   compile(
170     optimizer = 'adam',
171     loss      = 'binary_crossentropy',
172     metrics   = c('accuracy')
173   )
174
175 # print model architecture
176  model_keras
177
178 # Tensorboard can be useful to follow along the training of complex models as it
      trains
179 # Also, you can easily compare different models in one graph
180 # in this case, this is not necessary, hence commented
181 # launch TensorBoard (data won't show up until after the first epoch)
182
183 # tensorboard("logs/run_1")
184
185 history <- fit(
186   object           = model_keras,
187   x                = as.matrix(x_train_tbl),
188   y                = y_train_vec,
189   batch_size       = 100,
190   epochs           = 10,
191   validation_split = 0.3,
192   # include callback below if you want to use tensorflow
193   #callbacks = callback_tensorboard("logs/run_1", write_images = TRUE),
194   verbose          = 1
195 )
196
197 ## compare runs
198 ## In this example, I used three different callbacks for three different model
      specifications
```

```r
# tensorboard("logs")

# Plot the training/validation history of the Keras model
plot(history) +
  theme_bw()

##############################################
## Extract Predictions from the Keras ANN ####
##############################################

# Predicted Class
yhat_keras_class_vec <- predict_classes(object = model_keras, x = as.matrix(x_test_tbl
    )) %>%
  as.vector()

# Predicted Class Probability
yhat_keras_prob_vec  <- predict_proba(object = model_keras, x = as.matrix(x_test_tbl))
     %>%
  as.vector()


#####################################################
####       Model Comparison ANN/Logit      ##########
#####################################################

# run script that sets up a simple benchmark Logit model
source("benchmark_logit_model.R", echo = TRUE)

# compare ANN Model ROC curve to benchmark logit modek
roc_comp <- roc(estimates_keras_tbl$truth,estimates_keras_tbl$class_prob, percent=TRUE
    , plot=TRUE, print.auc=TRUE,
                print.auc.x = 90, print.auc.y = 85, grid=TRUE)
roc_comp <- roc(xsell_valid$xsell,xsell_valid$pred_logit, percent=TRUE, plot=TRUE,
                 print.auc=TRUE,grid=TRUE,  col = "red", print.auc.x = 90, print.auc.y
                      = 75, add = TRUE)
text(40,50, "Simple Benchmark Logit Model")
text(33,40, "Hyperparameter-tuned ANN with Keras", col = "red")

# compare model statistics (Confusion Matrix, Accuracy, Sensitivity, Recall, etc.)
conf_matrix_keras <- confusionMatrix(as.factor(yhat_keras_class_vec),as.factor(
    estimates_keras_tbl$truth),
                              positive="1", dnn = c("Prediction", "Actual"))
conf_matrix_logit <- confusionMatrix(as.factor(xsell_valid$predict),as.factor(xsell_
    valid$xsell),
                                     positive="1", dnn = c("Prediction", "Actual"))
conf_matrix_keras
conf_matrix_logit

ConfMatK <- as.data.frame.matrix(conf_matrix_keras$table)
stargazer(ConfMatK, head = FALSE, title = "Table", summary = FALSE)


#####################################################
#### Evaluate Feature Importance with LIME #########
#####################################################

# Setup
class(model_keras)

#Setup lime::model_type() function for keras
#This specifies that the task at hand is a classification task
model_type.keras.engine.sequential.Sequential <- function(x, ...) {
  return("classification")
  }


# Setup lime::predict_model() function for keras
predict_model.keras.engine.sequential.Sequential <- function(x, newdata, type, ...) {
  pred <- predict_proba(object = x, x = as.matrix(newdata))
  return(data.frame(Yes = pred, No = 1 - pred))
```

```
  }


# Test the predict_model() function
predict_model(x = model_keras, newdata = x_test_tbl, type = 'raw') %>%
  tibble::as_tibble()

# Run lime() on training set
explainer <- lime::lime(
  x              = x_train_tbl,
  model          = model_keras,
  bin_continuous = FALSE
)

# Run explain() on explainer
explanation <- lime::explain(
  x_test_tbl[1:4, ],
  explainer      = explainer,
  n_labels       = 1,
  n_features     = 10,
  kernel_width   = 0.5,
  feature_select = "forward_selection"
)

# Plot feature importance
plot_features(explanation, ncol = 2) #+
  labs(title = "LIME Feature Importance Visualization",
       caption = "Test data set, first four customers")

plot_explanations(explanation) +
  labs(title = "LIME Feature Importance Heatmap",
       subtitle = "Hold Out (Test) Set, First Four Cases Shown") +
  theme_bw()

# Feature correlations to xsell
corrr_analysis <- x_train_tbl %>%
  mutate(xsell = y_train_vec) %>%
  correlate() %>%
  focus(xsell) %>%
  rename(feature = rowname) %>%
  arrange(abs(xsell)) %>%
  mutate(feature = as_factor(feature))

# Correlation visualization
corrr_analysis %>%
  ggplot(aes(x = xsell, y = fct_reorder(feature, desc(xsell)))) +
  geom_point() +
  # Positive Correlations - Contribute to xsell
  geom_segment(aes(xend = 0, yend = feature),
               color = palette_light()[[2]],
               data = corrr_analysis %>% filter(xsell > 0)) +
  geom_point(color = palette_light()[[2]],
             data = corrr_analysis %>% filter(xsell > 0)) +
  # Negative Correlations - Prevent xsell
  geom_segment(aes(xend = 0, yend = feature),
               color = palette_light()[[1]],
               data = corrr_analysis %>% filter(xsell < 0)) +
  geom_point(color = palette_light()[[1]],
             data = corrr_analysis %>% filter(xsell < 0)) +
  # Vertical lines
  # geom_vline(xintercept = 0, color = palette_light()[[5]], size = 1, linetype = 2) +
  # geom_vline(xintercept = -0.05, color = palette_light()[[5]], size = 1, linetype =
    2) +
  # geom_vline(xintercept = 0.05, color = palette_light()[[5]], size = 1, linetype =
    2) +
  # Aesthetics
  theme_bw() + # theme_tq replaced because in conflict with randomForest package
  labs(#title = "Cross Sell Correlation Analysis",
       subtitle = paste("Negative correlations (prevent xsell),",
                        "positive correlations (contribute to xsell)"),
```

```
        x = "Correlation of feature with xsell",
        y = "Feature Importance")

# individual variable assessment: age
train_tbl %>%
  ggplot(aes(x=as.factor(xsell), y=age))+
  geom_jitter(shape = 1, alpha = 0.2) +
  geom_violin(fill="blue", alpha = 0.2) +
  labs(x = "xsell")+
  theme_bw()

# individual variable assessment: giro mailing
table(as.factor(train_tbl$xsell), train_tbl$giro_mailing)

# individual variable assessment: logins
train_tbl %>%
  ggplot(aes(x=as.factor(xsell), y=logins))+
  geom_jitter(shape = 1, alpha = 0.5) +
  geom_violin(fill="blue", alpha = 0.2) +
  ylim(0,10) +
  theme_bw()

table(train_tbl$xsell, train_tbl$gender)
```

Listing 1: Main analysis

```
#####################################
#### Hyperparameter tuning ##########
#####################################

# run the external script with defined flags as default
training_run("keras_nnet_architecture.R")

# !!CAUTION!! Combines every single combination (3^(#tuned parameters)), thus long runtime
# run various combinations of dropout1 and dropout2

# Only Dropout tuning
runs <- tuning_run("keras_nnet_architecture.R", runs_dir = "dropout_tuning", flags = list(
  dropout1 = c(0.2, 0.4, 0.6),
  dropout2 = c(0.2, 0.4, 0.6)
))

# run combinations of dense units, epochs and batch size
runs <- tuning_run("keras_nnet_architecture.R", runs_dir = "runs/20190608_neurons_epochs_batch_tuning", flags = list(
  #dropout1 = c(0.2, 0.4, 0.6),
  #dropout2 = c(0.2, 0.4, 0.6),
  dense_units1 = c(8, 64, 128),
  dense_units2 = c(8, 64, 128),
  epochs = c(5, 10, 30),
  batch_size = c(50, 100, 150)
))

# only number of neurons per layer tuning
runs <- tuning_run("keras_nnet_architecture.R", runs_dir = "number_neurons_tuning", flags = list(
  dense_units1 = c(8, 32, 64),
  dense_units2 = c(8, 32, 64)
))


#####################################
#### After-tunig processing ##########
#####################################

# check latest run
latest_run()
```

```
40
41 # compare runs in interactive RStudio viewer
42 compare_runs()
43
44 # View the run with the highes val_acc
45 view_run("runs/20190608_neurons_epochs_batch_tuning/2019-06-08T08-56-30Z")
46
47 # compare runs in data frame
48 runs1 <- ls_runs(runs_dir = "runs/20190607_dropout_neurons_epochs_tuning", order =
       metric_val_acc)
49 runs2 <- ls_runs(runs_dir = "runs/20190608_neurons_epochs_batch_tuning", order =
       metric_val_acc)
50
51 # combine both runs in one df
52 runs <- rbind(runs1, runs2)
53
54 # extract important information only for printing with stargazer
55 runs_print <- runs %>%
56   arrange(desc(metric_val_acc)) %>%
57   select(metric_val_acc, metric_acc, flag_dense_units1, flag_dropout1,
58          flag_dense_units2, flag_dropout2, flag_epochs, flag_batch_size)
59
60 stargazer(runs_print[1:5,], summary = FALSE)
61 stargazer(runs_print[323:324,], summary = FALSE)
62
63 # move all runs into run/archive
64 clean_runs()
65
66 # extract and save certain model
67 copy_run_files("runs/2019-06-08T00-38-43Z", to = "20190607-best-model")
```

Listing 2: Hyperparameter tuning routine

```
1  ##########################################
2  ### keras neural network architecture ####
3  ##########################################
4
5  # to be called by hyperpar_tuning_tf_runs.R
6
7  # setting up flags for hyperparameter tuning
8  FLAGS <- flags(
9    flag_integer("dense_units1", 8),
10   flag_numeric("dropout1", 0.6),
11   flag_integer("dense_units2", 64),
12   flag_numeric("dropout2", 0.6),
13   flag_integer("epochs", 30),
14   flag_integer("batch_size", 128)
15 )
16
17
18 # Setting up the ANN with Keras
19 model_keras <- keras_model_sequential()
20
21 model_keras %>%
22
23   # First hidden layer
24   layer_dense(
25     units              = FLAGS$dense_units1,
26     kernel_initializer = "uniform",
27     activation         = "relu",
28     use_bias           = TRUE,
29     bias_initializer   = 'zeros',
30     input_shape        = ncol(x_train_tbl)) %>%
31
32   # Dropout to prevent overfitting
33   layer_dropout(rate = FLAGS$dropout1) %>%
34
35   # Second hidden layer
36   layer_dense(
```

```
    units               = FLAGS$dense_units2,
    kernel_initializer  = "uniform",
    use_bias            = TRUE,
    bias_initializer    = 'zeros',
    activation          = "relu") %>%

  # Dropout to prevent overfitting
  layer_dropout(rate = FLAGS$dropout2) %>%

  # Output layer
  layer_dense(
    units               = 1,
    kernel_initializer  = "uniform",
    activation          = "sigmoid") %>%

  # Compile ANN
  compile(
    optimizer = 'adam',
    loss      = 'binary_crossentropy',
    metrics   = c('accuracy')
  )


history <- fit(
  object           = model_keras,
  x                = as.matrix(x_train_tbl),
  y                = y_train_vec,
  batch_size       = FLAGS$batch_size,
  epochs           = FLAGS$epochs,
  validation_split = 0.3,
  verbose          = 0
)
```

Listing 3: Flagged neural net architecture for hyperparameter tuning

```
##############################################
###   Apply model to out-of-time sample    ###
##############################################

# import oot data set
xsell_oot <- read.csv("xsell_oot.csv", na.strings=c("","NA"), stringsAsFactors = FALSE
    )

# create new variable tenure
xsell_oot$tenure <- xsell_oot$age - xsell_oot$entry_age

# data cleaning, replace NAs in char-variables with "None_or_missing"
xsell_oot <- xsell_oot %>%
  replace_na(list(pref_device = "None_or_missing"))
xsell_oot <- xsell_oot %>%
  replace_na(list(occupation = "None_or_missing"))

# all character columns to factor:
xsell_oot <- mutate_if(xsell_oot, is.character, as.factor)
#additional numeric variables that should rather be treated as factors
xsell_oot$car_seg <- as.factor(xsell_oot$car_seg)
xsell_oot$acad <- as.factor(xsell_oot$acad) # remove if strange results
xsell_oot$giro_mailing <- as.factor(xsell_oot$giro_mailing) # remove if strange
    results
xsell_oot$pop_km <- as.factor(xsell_oot$pop_km)
xsell_oot$ppower <- as.factor(xsell_oot$ppower)

# extract ID (X)
X <- xsell_oot$X

# Remove unnecessary data and clean data set
xsell_oot_data_tbl <- xsell_oot %>%
  select(-X) %>% #removes ID
```

```
  # if you don't want to run a random forest for NA imputation, you can do apply of the two easier fixes to NA's:
  #drop_na() #%>% # removes all NA's. Bad Solution! Improve! Removes 70% of observations
  na.roughfix() #replaces NA's: Numeric with median, factor with mode

# Impute NAs with a Random Forest
# can't do here, because there is no xsell
#xsell_oot_data_tbl <- rfImpute(xsell ~ . ,xsell_oot_data_tbl, iter = 4, ntree=100)

# apply recipe to normalize, etc.
x_valid_tbl <- bake(rec_obj, new_data = xsell_oot_data_tbl)

# Predicted Class
yhat_keras_class_vec_oot <- predict_classes(object = model_keras, x = as.matrix(x_valid_tbl)) %>%
  as.vector()

xsell_oot_incl_pred <- cbind(X, xsell_pred = yhat_keras_class_vec_oot)
write.csv(xsell_oot_incl_pred, "xsell_oot_predicted.csv", row.names = FALSE)
```

Listing 4: Using the ANN to predict the previously unseen out-of-time data set

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2015), 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems'.

Bengio, Y. (2000), 'Gradient-based optimization of hyperparameters', *Neural computation* **12**(8), 1889–1900.

Benítez, J. M., Castro, J. L. & Requena, I. (1997), 'Are artificial neural networks black boxes?', *IEEE Transactions on neural networks* **8**(5), 1156–1164.

Bergstra, J. & Bengio, Y. (2012), 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research* **13**(Feb), 281–305.

Chan, H.-P., Sahiner, B., Wagner, R. F. & Petrick, N. (1999), 'Classifier design for computer-aided diagnosis: Effects of finite sample size on the mean performance of classical and neural network classifiers', *Medical physics* **26**(12), 2654–2668.

Chen, J. H. & Asch, S. M. (2017), 'Machine Learning and Prediction in Medicine — Beyond the Peak of Inflated Expectations', *The New England journal of medicine* **376**(26), 2507–2509.

Chollet, F. & Allaire, J. (2017), 'R Interface to Keras'.

Dayhoff, J. E. & DeLeo, J. M. (2001), 'Artificial neural networks: Opening the black box', *Cancer: Interdisciplinary International Journal of the American Cancer Society* **91**(S8), 1615–1635.

Domingos, P. M. (1999), MetaCost: A General Method for Making Classifiers Cost-Sensitive, *in* 'Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', San Diego, CA, pp. 155–164.

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M. & Thrun, S.

(2017), 'Dermatologist-level classification of skin cancer with deep neural networks', *Nature* **542**(7639), 115–118.

Fawcett, T. (2006), 'An introduction to ROC analysis', *Pattern Recognition Letters* **27**(8), 861–874.

Felvey, J. (1982), 'Cross-selling by computer', *Bank Marketing* **7**, 25–27.

Glorot, X., Bordes, A. & Bengio, Y. (2011), 'Deep Sparse Rectifier Neural Networks', *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics* **15**, 9.

Hansen, L. K. & Salamon, P. (1990), 'Neural network ensembles', *IEEE Transactions on Pattern Analysis & Machine Intelligence* (10), 993–1001.

Hastie, T., Tibshirani, R. & Friedman, J. (2017), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics, 2 edn, Springer-Verlag, New York.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B. & Sainath, T. (2012), 'Deep Neural Networks for Acoustic Modeling in Speech Recognition', *IEEE Signal Processing Magazine* **29**.

Hornik, K. (1991), 'Approximation capabilities of multilayer feedforward networks', *Neural Networks* **4**(2), 251–257.

Jaderberg, M., Simonyan, K., Vedaldi, A. & Zisserman, A. (2014), Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition, *in* '2011 International Conference On Computer Vision'.

Jordan, M. I. & Mitchell, T. M. (2015), 'Machine learning: Trends, perspectives, and prospects', *Science* **349**(6245), 255–260.

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.

Kamakura, W. A., Wedel, M., de Rosa, F. & Mazzon, J. A. (2003), 'Cross-selling

through database marketing: A mixed data factor analyzer for data augmentation and prediction', *International Journal of Research in Marketing* **20**(1), 45–65.

Kavicky, R. (2017), 'Local Interpretable Model-Agnostic Explanations (R port of original Python package)'.

Keaveney, S. M. (1995), 'Customer Switching Behavior in Service Industries: An Exploratory Study', *Journal of Marketing* **59**(2), 71–82.

Kingma, D. P. & Ba, J. (2014), Adam: A Method for Stochastic Optimization, *in* 'International Conference on Learning Representations 2015', San Diego, CA.

Li, S., Sun, B. & L. MONTGOMERY, A. (2011), 'Cross-Selling the Right Product to the Right Customer at the Right Time', *Journal of Marketing Research* **48**, 683–700.

Liaw, A. & Wiener, M. (2002), 'Classification and Regression by randomForest', *R News* **2**(3), 18–22.

Olden, J. D. & Jackson, D. A. (2002), 'Illuminating the "black box": A randomization approach for understanding variable contributions in artificial neural networks', *Ecological modelling* **154**(1-2), 135–150.

Pedersen, T. L. (2018), 'Package 'lime': Local Interpretable Model-Agnostic Explanations'.

Pedersen, T. L. & Benesty, M. (2018), 'Understanding lime', https://cran.r-project.org/web/packages/lime/vignettes/Understanding_lime.html.

Reichheld, F. F. & Sasser Jr, W. E. (1990), 'Zero defections: Quality comes to services', *Harvard Business Review* **68**(5), 105–111.

Ribeiro, M. T., Singh, S. & Guestrin, C. (2016), "Why Should I Trust You?": Explaining the Predictions of Any Classifier, *in* '22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', San Francisco, CA, USA, pp. 1135–1144.

Ruder, S. (2016), 'An overview of gradient descent optimization algorithms', http://sebastianruder.com/optimizing-gradient-descent/index.html.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting', *Journal of Machine Learning Research* **15**, 1929–1958.

Werbos, P. J. (1990), 'Backpropagation through time: What it does and how to do it', *Proceedings of the IEEE* **78**(10), 1550–1560.

Zhang, Z. & Sabuncu, M. (2018), 'Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels', *Advances in Neural Information Processing Systems 31* pp. 8778–8788.

Statutory Declaration

Ich versichere hiermit, dass ich die vorliegende Arbeit selbsändig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Wörtlich übernommene Sätze oder Satzteile sind als Zitat belegt, andere Anlehnungen, hinsichtlich Aussage und Umfang, unter Quellenangabe kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist nicht veröffentlicht. Sie wurde nicht, auch nicht auszugsweise, für eine andere Prüfungs- oder Studienleistung verwendet.

I herewith declare that I have completed the present term paper independently, without making use of other than the specified literature and aids. Sentences or parts of sentences quoted literally are marked as quotations; identification of other references with regard to the statement and scope of the work is quoted. The thesis in this form or in any other form has not been submitted to an examination body and has not been published. This thesis has not been used, either in whole or part, for another examination achievement.

Frankfurt am Main, July 15, 2019

Lukas Jürgensmeier