FOM Hochschule für Oekonomie & Management Essen

Place Muenster

Extra-occupational study programme on

Big Data & Data Science

5th semester

Term paper in

Prediction Challenge – Predicting new bank customers with machine learning

Supervisor:	Sascha Schworm
Author:	Tim A. Schalk
Student no.:	467429
In collaboration with:	*Sebastian Krakau*
Student no.:	*465522*
Submission date:	29.02.2020
Online Files on Github.com:	https://github.com/T1M0THY1337/big-data-prediction-challenge/tree/master/submission

I Table of contents

II **Index of figures**

1. Introduction

This paper demonstrates a practical application of the machine learning model XGBoost. A given data set serves as a practical basis. The data itself contains basic demographic information about numerous customers as well as data related to phone-based marketing calls during specific campaigns. The problem definition is, how to use new data sets to predict whether the person contacted will become a new customer of the bank. The task will be to find a suitable model, extend and cleanse the given data set and make predictions with the machine learning model.

2. XGBoost: Gradient Boosting Machine

For the practical execution the software library XGBoost (Extreme Gradient Boosting) is used. The algorithm handles regression and classification problems and provides better predictions by combining several weaker models.
The freely available library XGBoost provides a tree algorithm with gradient boosting. Gradient Boosting is one of the supervised learning algorithms that use a training data set to predict a target variable. Gradient (Tree) Boosting is an ensemble method besides Random Forest. This means that several learning methods are applied multiple times to a given data set. The models involved are simpler and weaker ones. Their estimates are combined or summed up to make better predictions. For model selection in XGBoost, decision tree ensembles are used, which consist of several classification and regression trees (CART). The tree algorithm with gradient boosting followed an additive strategy, which means that corrections are implemented by means of learned corrections and new trees are added subsequently. The support of a gradient descent algorithm ensures that the user-defined loss function is also minimized. The task of the newly added trees is also to predict remnants or errors of previous trees. The optimizations only take place at one level of the tree, as it is not possible to compare all possible trees completely. Another big advantage of XGBoost is the control and regulation of the model complexity, which is calculated by a regularization equation. XGBoost thus provides an intelligent overall package of gradient boosting and, thanks to optimizations for high computing power and performance, efficiently calculated predictions. Why exactly XGBoost is used in this case study is answered in the practical part of this paper.[1]

[1] https://xgboost.readthedocs.io/en/latest/tutorials/model.html

3. A practical elaboration

3.1 Overview of the given data set

The data set to be analysed includes 15 columns. The first column acts as identifier, the last column '*success*' as target feature. Consequently, 13 features remain, that have been sorted by the corresponding data type in the table available in the appendix A.

The features '*credit_default*', '*housing_loan*' and '*personal_loan*' would have the character of a boolean datatype, but the third category 'Unknown' does not give a binary variable anymore. A closer look at the labelled target feature shows that the data set is very imbalanced. The ratio of the category 'Yes' to 'No' is approximately 1:7.8, which means that there is a high bias due to the uneven ratio. The following graphic illustrates the problem.

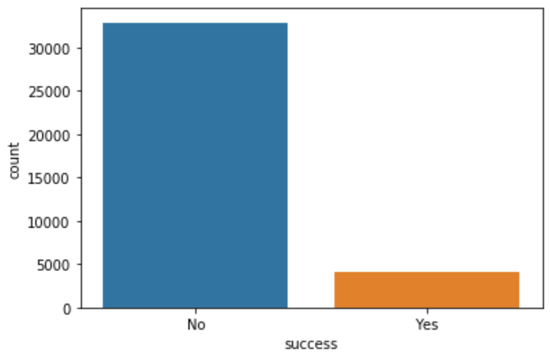


Figure 1: Imbalanced datas set

3.2 Learning task and Learning style

Based on the given data set it can be observed that the learning style is a Supervised Machine Learning model. The decisive point for this conclusion is the labelled target variable 'success'. Consequently, an input-output combination of features and labels exists. To determine the learning task, the target variable must be examined in detail, which is labelled binary ('Yes' and 'No'). It is evident that not a continuous numerical outcome is sought, but a discrete categorical one. Thus, it can be noted that this is a classification task.

3.3 Feature Engineering

During feature engineering, new features are generated from existing ones and additional external data records are added. First, new features are created from the feature '*date'*. The date stamp is used to derive the year, month, weekday as string, a weekend day as

Boolean and the season. Another group of newly created features can be recognized by the ending 'group'. These groups are grouped together from their original feature. An example is the feature '*age*'. The numerical values were used to create the categorical classes young, young_adult, old_adult and old.

```
# !python script
def getAgeGroup(inAge):
  if (inAge < 28):
    return 'young'
  elif (49 > inAge > 27):
    return 'young_adult'
  elif (61 > inAge > 48):
    return 'old_adult'
  else:
    return 'old'
```

The feature '*prime_rate*' contains data from an external data source.[2] The correct value was determined using the date stamp. Since the country of origin is not known, the national base rate was taken from the USA. If the assumption is wrong, however, the general influence of the USA on the world could create added value.

3.4 Feature Selection And data cleansing

The data set is cleaned of unwanted data set instances before processing by the algorithm. Incorrect or unusable values could falsify or bias the result. For the manual feature selection, the ten least important features are determined. The ExtraTreesClassifier class is used for this purpose.

```
# !python script
model = ExtraTreesClassifier(random_state=x_random_state)
model.fit(X_dummie,y)
```

"This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset".[3] The searched result is then retrieved using the *feature_importances_* attribute. In order to be able to use the data set for pre-analysis, all categorical features are converted into indicator variables with the panda function *get_dummies*.

```
# !python script
X_dummie = pd.get_dummies(X, prefix_sep='_', drop_first=True)
```

[2] Date Source: https://www.kaggle.com/sohier/interest-rate-records
[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

The result can be found in the graphic in the appendix B, which shows the least twelve important features (the higher the value the better).

Based on this, the features ‘*dateWeekendDay*’, ‘*job*’, ‘*n_contacts_campaignGroup*’, ‘*age*’ and ‘*job*’ were removed for the final calculation.

```
# !python script
outDataframe.loc[outDataframe.isin(["Unknown"]).mean(axis=1) < 0.13]
```

This line cleans up record instances which contain the value Unknown for more than two features. These are considered to be incomplete.

```
# !python script
outDataframe[(outDataframe['credit_default'] != 'Yes')]
```

The evaluation of the importance of each feature showed that the feature ‘*credit_default*’ is the least important feature with zero percent. For this reason, it is excluded from the calculation with the above line.

```
# !python script
outDataframe['duration'].quantile(.97)
outDataframe.loc[(outDataframe['duration'] > upper_lim), 'duration'] =
      upper_lim
```

The upper code snippet deals with the feature 'duration', which has many different characteristic values. To eliminate possible outliers in this feature, the 97th percentile is calculated. All values that are outside of this percentile are overwritten with the limit of the 97th percentile.

3.5 Pipeline

The pipeline goes through 4 steps to learn the data set and make predictions.

```
# !python script
pipeline = Pipeline([
    ('preprocessor', preprocessor), # MinMaxScaler AND OneHotEncoder
    ('oversampling', smote_enn), # SMOTEENN
    ('feature_selection', SelectFromModel(LinearSVC(penalty="l1", dual=
False, random_state=x_random_state))),
    ('classifier', classifier)
])
```

In the first step (Preprocessor) the MinMaxScaler and the OneHotEncoder are applied to the data set. As a result of the OneHotEncoder, features with the data type String are packed into several features with the data type Boolean (0 or 1).

The second step addresses the problem with the imbalanced data set. The techniques of over- and under-sampling are used for this. The class SMOTEENN from the imbalanced-learn library provides this. In our example, the five closest neighbours of a data instance are edited. By specifying the parameter ratio with the value 0.7, the minority ('Yes') is increased to 70 percent of the total number of the majority ('No').

```
# !python script
sm = SMOTE(ratio = 0.7,k_neighbors = 5, n_jobs = -1)
smote_enn = SMOTEENN(smote = sm)
```

The third step in the pipeline is further support for feature selection. For this the function SelectFromModel is used, which returns the most important features from the dataset. A model, in this case the Linear Super Verctor Classifier, is passed for the function. The important thing here is the parameter penalty, because it tells the function to use Lasso Regression (L1) as a control method. Less important features are thus taken out of consideration.

```
# !python script
xgb.XGBClassifier(n_jobs=-1, random_state=x_random_state,
      objective='binary:logistic', scale_pos_weight=ratio)
```

In the last step of the pipeline the actual training of the XGBoost classifier to the data set begins. Through the previous steps, the algorithm gets a data set that is optimally adapted to it. The parameter scale_pos_weight tells the model what the ratio of negative classes to positive classes is. The info helps to handle the unbalanced data set correctly.

```
# !python script
search = RandomizedSearchCV(
pipeline, param_distributions=param_distributions, n_iter=x_rscv_n_iter
,scoring={'f1_score': f1_scorer}, n_jobs=-1,cv=x_rscv
,random_state=x_random_state,refit='f1_score',return_train_score=True)
```

The complete pipeline is controlled by the RandomizedSearchCV. The StratifiedKFold function informs the function at which points in training and test data records are to be split. Furthermore, the randomly generated hyper parameters and the scoring procedure are specified. The function runs through the data set several times and finally selects the most optimal parameters.

3.6 Performance Measure: F1 Score

A performance measure is used in machine learning to get an indication of how well your model fits the data set and eventually can make the correct predictions. In the present case study, the so-called F1 score will be the best choice. Although the accuracy is a very common and well-known method, it faces the problem of a high bias when the data set is imbalanced. As mentioned above, this is severely the case in the present data set, which is why this performance measure is not applicable. Two further classification metrics are recall and precision. However, due to the lack of background information on the data set, it is not possible to determine whether a false negative prediction or a false positive prediction is associated with higher costs. The F1 Score provides on the one hand a balance between Recall and Precision and on the other hand a compatibility with uneven class distribution, whereby the second point offers a great advantage.

```
# !python script
f1_scorer = make_scorer(f1_score, average='binary', pos_label='Yes')
```

For performance measurement, a callable object is created with the make_scorer function from the scikit-learn library. The scikit-learn function f1_score is passed as parameter, which calculates the F1 score. Furthermore, it is specified that it has a binary data type and the category 'Yes' is to be interpreted as a true value.

```
# !python script
search.cv_results_['mean_train_f1_score'][search.best_index_] * 100
search.cv_results_['mean_test_f1_score'][search.best_index_] * 100
```

The F1 score calculation is assigned to the model used as scoring. The results of the score can be retrieved after the calculation using the corresponding functions of the machine learning model. As several runs are made, the mean value of the best run is used as output.

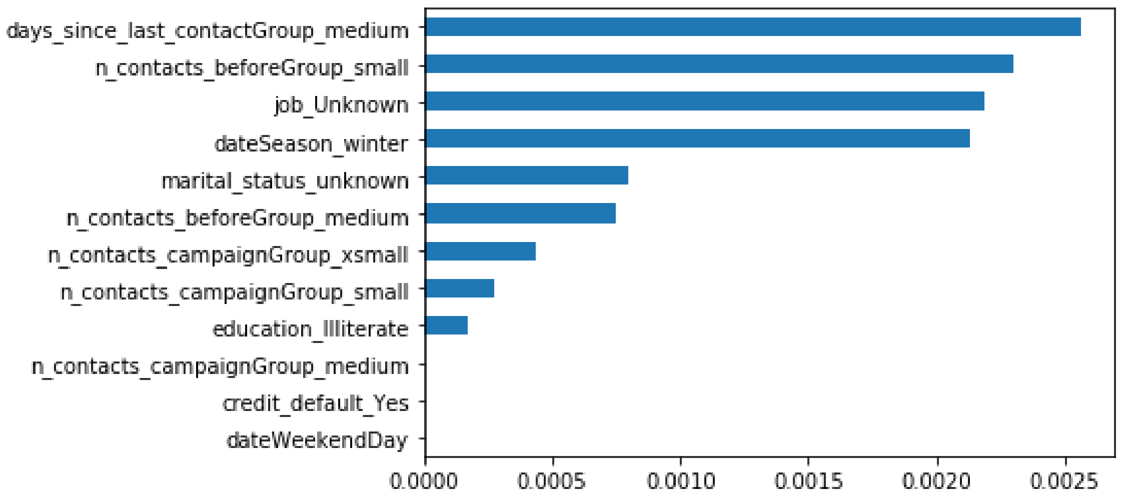
4. Validation of the results

The scoring model F1 Score gives a value in the range from 60% to 62%. The variation is caused by the embedded randomness. The result is only partially satisfactory but could be useful to a certain extent. An improvement in the result could probably be achieved by further enhancing feature engineering. Due to insufficient background information about the data set, an embedding of further external data is made difficult. For example, the country of origin could be important. For further research purposes it is valuable to know the importance of individual features. Thus, for further observations or analyses, the most important features in this thesis should receive attention like the duration feature, for instance.

III Appendix A: Overview of the Features

Date stamp	String	Integer
date	marital_status	age
	education	n_contacts_campaign
	job	days_since_last_contact
	credit_default	n_contacts_before
	housing_loan	previous_conversion
	personal_loan	
	communication_type	

IV Appendix B: Graphic: The 15 least important features



Declaration in lieu of oath

I hereby declare that I produced the submitted paper with no assistance from any other party and without the use of any unauthorized aids and, in particular, that I have marked as quotations all passages which are reproduced verbatim or near-verbatim from publications. Also, I declare that the submitted print version of this thesis is identical with its digital version. Further, I declare that this thesis has never been sub-mitted before to any examination board in either its present form or in any other similar version. I herewith disagree that this thesis may be published. I herewith consent that this thesis may be uploaded to the server of external contractors for the purpose of submitting it to the contractors' plagiarism detection systems. Uploading this thesis for the purpose of submitting it to plagiarism detection systems is not a form of publication.

Muenster, 29.02.2020	[illegible]
(Location, date)	**(Genuine signature)**