FOM Hochschule für Oekonomie & Management Münster

university location Münster

part-time degree program

Wirtschaftsinformatik, 5. Semester

in the context of the course

Big Data & Data Science

on the subject

Prediction Challenge

Advisor: Prof. Sascha Schworm

Author: Michael Heichler
Matriculation Number: 48231
Ostbleiche 8
48231 Warendorf

Submission: February 28, 2020

Contents

[1] cf. *Hoeller*, *J.*, 2019.

[2] cf. *Hoeller*, *J.*, 2019.

[3] cf. *Hoeller*, *J.*, 2019.

List of Figures

List of Tables

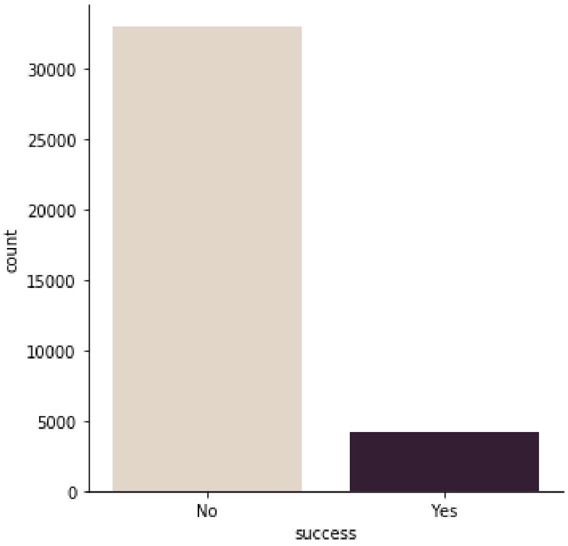
1 Introduction

The Prediction Challenge is splitted up in following tasks:

1. Cleaning up the given data set and detect „leaking“ data
2. Feature Engineering, containing automated and manually Feature Engineering
3. Choosing the right algorithm for the given problem (binary classification problem)

The data set contains multiple customers and the machine learning should learn out of a training set if the customer will churn.

Figure 1: Data Distribution



Additionally the data set is heavily biased which is shown in Figure 1.

1.1 Objective

Reaching a sustainable F1 Score that is better than to throw a pair of dice. Hereby reaching an overall F1 Score above 50 %.

2 Software details

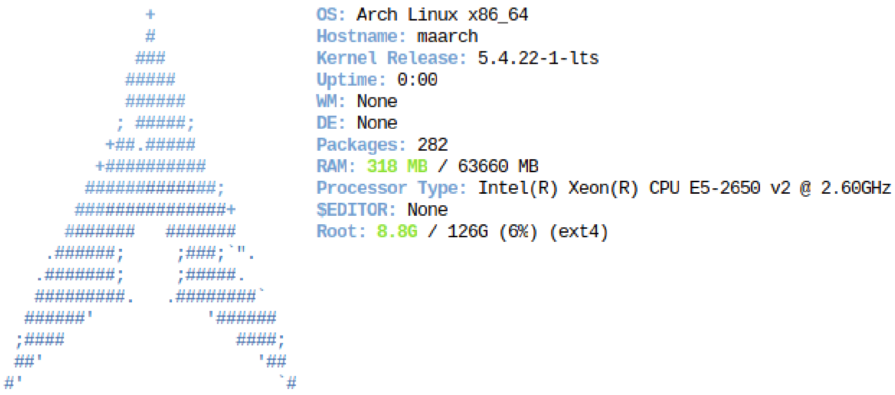
2.0.1 Own JupyterLab Environment

Table 1: Enviroment - Package Versions

Enviroment - Package Versions	
Pandas	1.0.1
Numpy	0.10.0
Scipy	1.15.2
sklearn	0.19.1-3
Matplotlib	3.0.1
Featuretools	0.13.2
LightGBM	2.3.1
JupyterLab	1.2.6
Jupyter-Core	4.6.1
Jupyter-Client	5.3.4
iPython	7.12.0

All packages are installed with the package manager Pacman and not with the Python package manager „pip“. The chosen Linux distribution is ArchLinux, Linux Kernel 5.4.22-1-lts, and Python Version 3.8.1.

Figure 2: System environment



```
              +                OS: Arch Linux x86_64
              #                Hostname: maarch
             ###               Kernel Release: 5.4.22-1-lts
            #####              Uptime: 0:00
            ######             WM: None
           ; #####;            DE: None
          +##.#####            Packages: 282
         +##########           RAM: 318 MB / 63660 MB
        #############;         Processor Type: Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
       ###############+        $EDITOR: None
      #######   #######        Root: 8.8G / 126G (6%) (ext4)
    .######;     ;###;`".
   .#######;     ;#####.
   #########.   .########`
  ######'           '######
 ;####                 ####;
 ##'                     '##
#'                         `#
```

Important modules for the Binary Classification task such as hyper-parameter tuning are:

- RandomizedSearchCV - Randomised search on hyper-parameters, implements a fit and a score method
- GridSearchCV - Exhaustive search over specified parameter values for an estimator

3 Structure of the source code

The source code is based on the Minimum Working Example[4] from the GitRepository of Prof. Sascha Schworm but heavily modified.

A huge help in these analysis were the GitRepository from the user joehoeller[5]. It is almost the same data structure, but with changes in the data set / feature count.

Firstly, the standardised module import can be found in the first lines of code. Secondly, before changes were made, data set analysing took place.

Here the data set is analysed solely on the bias. Hence the dataset is imbalanced - a huge amount of customers which a not churned against a small set of successfully churned customers - it is most likely, that the model will not train in a correct way.

Before changes were made on the dataset (for example changing column types from numerical to categorical), a more in-depth analysis were made.

3.1 Categorical analysis

Beforehand checking, if any data is missing is a must. Then the graphical output of values distributed across categorical attributes gives an clear view what categorical features are relevant and which are not helpful for the classification problem.

Firstly, no missing data („unknowns“) were reported about the marital status. The bank has a good understanding about the marital status of the customers. Furthermore no or very few customers are illiterate and defaulted on a loan. The bank probably does not want to extend an offer to a customer with a bad credit. Really negative are the high number of unknown values. The same conclusion can be found in the GitRepository from joehoeller[6]

A more in depth analysis of the categorical features reveals that proportions of customers with or without a housing loan are somewhat similar. Therefore this feature does not help about understanding the problem at hand. Only few customers have a personal loan. Striking is the fact, that double as many cellular as landline phone calls took place. The understanding if previous phone calls were a success is missing. Most of the previous conversations failed. In the current campaign most calls were not successful, too.

[4] cf. *Schworm*, *S.*, 2019.

[5] cf. *Hoeller*, *J.*, 2019.

[6] cf. *Hoeller*, *J.*, 2019.

The last step of the categorical analysis shows, how much each categorical feature weights on the outcome of the call. The most impact of the outcome are the features „previous_conversion“ and „n_contacts_before“.

3.2 Numerical analysis

After the standard check if data is missing, the analysis starts with a graphical chart which shows the occurrence on a scale, hereby the y-axis show the number of occurrences and the x-axis the values of the features.

The data shows, that the customer age is mostly between 20 and 60, only a few customers are over 60. Additionally the customers were often neither contacted about the ongoing nor about the previous campaign.

Final step is a outlook on the feature behaviour. The data provides that the average age of the customer is 38, contacts were made during the current campaign at least two times. Days since last contact are mostly „-1“ - hereby most of the customer were not contacted before or no valid data is available. The manipulation of the data due to the feature days_since_last_contact is not a fact that can be overlooked. It manipulates the data because few customers have an instance of days_since_last_contact associated that are not properly taken into account. The consideration is to zero the „-1“ values of the feature or give it a high number like „999“.

3.3 Transformation of „days_since_last_contact“

Firstly, to make it a little bit easier to follow the guidelines from the GitHub Repository of joehoeller[7] the feature „days_since_last_contact“ were modified. The „-1 “ values are changed into „999“.

Secondly a new feature is created out of „days_since_last_contact“ which is called „days_since_last_contact_cat“ and it is a categorical version of the feature.

3.4 Feature Engineering

For the improved feature engineering „Featuretools“ assists the search. It is an immense package which uses Deep Feature Synthesis, short DFS, for automated feature engineer-

[7] *Hoeller, J.*, 2019.

ing. Therefore using raw data combined with known variables about your data to build meaningful features for machine learning and predictive modelling.

The structure of the classes made it easy to generate exactly these features that the data set additionally needs. To build features with Featuretools it is necessary to create an Entity Set in which the data frame is inserted. The variable types must not manually added because Featuretools can detect the types automatically. Due to hiccups the variables and the specific types are manually added, also it is a good practice and helps to get a clean code.

Before Featuretools can work correctly, it is important to split up the entity set into small entity zones.

This is done with this code snippet:

```
es = es.normalize_entity(base_entity_id=’dataset’, new_entity_id=’
    days_since_last_contact’, index=’days_since_last_contact’)
es = es.normalize_entity(base_entity_id=’dataset’, new_entity_id=’previous_conversion’
    , index=’previous_conversion’)
```

The chosen interesting dataset parts in which Featuretools should look for correlation with other features are

- days_since_last_contact_cat
- previous_conversion

The order is not important during the zone setup.

With the code snippet

```
# Threshold for removing correlated variables
threshold = 0.7  # Absolute value correlation matrix

corr_matrix = df.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Select columns with correlations above threshold
collinear_features = [column for column in upper.columns if any(upper[column] >
    threshold)]
df_flt = df.drop(columns = collinear_features)
df_flt.shape
```

all features were checked if the correlation is above the absolute value of 0.7 and therefore a viable choice for the training model.

Furthermore more features were created out of feature combinations. First the „duration“ column is sorted out because of leaky data values which means the model will not train correctly on this model and the F1 score will be bugged. From the data set and the analysis from it the following chapters will show up the key points of importance. All the questions that will follow are kindly copied from the joehoeller Github Repository[8]

3.4.1 Feature: Was the customer contacted about the last campaign?[9]

Through a subclass the features missing data („999“) of „days_since_last_contact “were given a new shape without changing it's data. The new shape of the array from now is „-1,1“and will be given to a FunctionTransformer

```
add_pcontacted_last_campaign = FunctionTransformer(ft_pcontacted_last_campaign,
    validate=False)
```

Hereby we have a new feature which answer the question if the customer was contacted about the last campaign with a boolean. (YesNo)

3.4.2 Feature: Was the customer part of the last campaign?[10]

Next all „Inexistent“ from the feature „previous_conversion“ was reshaped and afterwards a new feature created through the call

```
add_pcampaign = FunctionTransformer(ft_pcampaign, validate=False)
```

This created a feature for each row which answer in a Boolean matter. (YesNo)

3.4.3 Feature: Was the customer contacted more than ten times?[11]

Like in the previous chapters first the „n_contacts_before “ were reshaped and then a new feature was created

```
add_previous = FunctionTransformer(ft_previous, validate=False)
```

which answers the question of this sub chapter in a boolean matter.

[8] cf. *Hoeller*, *J.*, 2019.
[9] cf. *Hoeller*, *J.*, 2019.
[10] cf. *Hoeller*, *J.*, 2019.
[11] cf. *Hoeller*, *J.*, 2019.

3.4.4 Feature: Transforming Features into categorical ones

Next hint from the joehoeller GitRepo[12] were, that a new feature can be created by transforming the already existing feature „n_contacts_before " thus the new feature will be of a categorical type.

The procedure is the same like in the previous chapters. First the definition which tells how to reshape the array, and then a call with following code

```
add_campaign_gte10 = FunctionTransformer(ft_campaign_gte10, validate=False)
```

which creates a new feature called „add_campaign_gte10".

3.4.5 Feature: Binning „age"

Data binning or also called „Discrete binning" is a pre-processing technique to reduce minor observation errors.[13] Continuous variables are spliced up in groups (bins) without consideration of target information.[14]

First step is the creation of a binning pipeline. (the template from the GitRepositry joehoeller[15] is used) The pipeline defines how to handle submitted values. Firstly, a logarithm comes into place, second the class KBinsDiscretizer() make continuous data into bins.

Lastly a pipeline gives the new numeric feature „ft_campaign_to_previous into the binning pipeline and will be scaled with the RobustScaler class.

3.5 Oversampling

Via the sklearn class „resample" the first oversampling step takes place. Here the minority data, values of the „Yes", group are approximately 3.59195 times randomly multiplied to get 15.000 „Yes" samples.

This is a simple method which comes before the stratified splitting into training and test data. This is due to the reason that after the split SMOTE will take care of the remaining minority values after splitting.

[12] cf. *Hoeller, J.*, 2019.

[13] cf. *pitney bowes*, Spectrum Technology Platform, 2017.

[14] cf. *pitney bowes*, Spectrum Technology Platform, 2017.

[15] cf. *Hoeller, J.*, 2019.

SMOTE stands for „Synthetic Minority Over-sampling Technique“ in „which the minority class is over-sampled by creating „synthetic“ examples rather than by over-sampling with only [copied replacement] data.“[16]

Due to the mix of random oversampling and synthetic oversampling the goal to prevent a high failure rate and or low precision score is archived.

3.6 Splitting between Training and Test data

The choosen cross-validator for splitting up training and test set is the so called „StratifiedShuffleSplit()“ cross-validator. With it it is possible to return stratified and randomized folds. „The folds are made by preserving the percentage of samples for each class.“[17]

3.7 The final model

The chosen framework for this task is „LightGBM“. A relative new, lightweight and fast framework based on the Gradient Boosting Model. After various comparison with other frameworks like „xgboost“ LightGBM was the clear winner in consideration of speed and scoring.

The scoring model and a good indication is the F1 score, which is a function of Precision and Recall.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{1}$$

The Hyperparameter tuning took place via a randomised search with RandomizedSearchCV. With the pipeline

```
rf = Pipeline(steps=[('ft_union', ft_union),
                     ('sampling', smo),            ### SMOTE ###
                     ('classifier', lgbm)])
```

the construct is completed and the model is ready for tuning and lastly fitting it against „X_train“ and „y_train“:

```
# RandomizedSearch
grid = RandomizedSearchCV(rf, params, scoring=scorer, verbose=1, cv=10, n_jobs=-1,
    n_iter=15, return_train_score=True)
grid.fit(X_train, y_train)
```

[16] *Chawla, N. V.* et al., 2002, pg. 328.

[17] *sklearn*, n. d.

The search for hyperparameters were made at first with „GridSearchCV“. But due to it high complexity and time consumption, and a near identical score compared to the „RandomizedSearchCV“ the choice of the Hyperparameter Tuning technique was easy.

4 Fazit

Summing up this challenge is, that a thoroughly done data analysis is key for a good learning result. Without it the leaky data values like the feature „Duration“ hadn't been found. Thanks to the Github repository of joehoeller[18] a lot of key features have been developed and implemented. With Featuretools some of automatic help in feature engineering was provided.

With nearly 72 features the training of the model and the final test results were at 99,69 % on the training set and 92,39 % on the testing set. With more time that overfitting could be resolved with some feature tweaking and hyperparameter tuning. It is very likely that, with the additionally provided hyperparamters for imbalanced learning that comes with the xgboost classifier, the xgboost repository would've been better. Due to the time and given extent of the challenge the concentration lied on the feature engineering part and in-depth testing.

[18] cf. *Hoeller*, *J.*, 2019.

Bibliography

Chawla, *N. V.*, *Bowyer*, *K. W.*, *Hall*, *L. O.*, *Kegelmeyer*, *W. P.* (2002): SMOTE: Synthetic Minority Over-sampling Technique, in: Journal of Artificial Intelligence Research, 16 (2002), pp. 321–357

Hoeller, *Joe* (2019): Machine Learning for Predictive Lead Scoring, `https://github.com/joehoeller/Machine-Learning-For-Predictive-Lead-Scoring`, s.l., 2019

Pitney bowes (Spectrum Technology Platform, 2017): Spectrum Technology Platform - Machine Learning Handbuch, 1st ed., Stamford CT, 2017

Schworm, *Sascha* (2019): Prediction Challenge - Minimum Working Example, `https://github.com/saschaschworm/big-data-and-data-science/blob/master/notebooks/prediction-challenge/minimum-working-example.ipynb`, s.l., 2019

sklearn (n. d.): StratifiedShuffleSplit, `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html`, Accessed: 2020-02-28, s.l.