Machine Learning Engineer Nanodegree
Capstone Project Report

Raphael Bürki
October 4th, 2018

I. Definition

Project Overview

The automotive retail business is under pressure. The earnings from selling cars are not enough to cover the cost of sales, so the industry relies heavily on recurring aftersales service revenues to make a profit.[1] But customers are generally free to choose where to service their cars once they have bought them and so it is crucial for individual car retail companies to turn their erstwhile buyers into loyal service customers.

The goal of this project is to help a large Swiss automotive retail and maintenance company to reduce its churn rate (→ the ratio of customers who switch away from one supplier to another in a given period) by building a *predictive model* that classifies customers in two groups:

- ACITVE: this is a loyal customer who will turn up for the next service event
- CHURN: this is a customer who will not shop up for the next service event

Such a model will enable the company to contact all customers approximately 3 months before the next mandatory service is due with a reminder mailing and – this is the clue! – an *offer of a substantial discount to those customers only that are identified to be about to churn*. This is an attempt to motivate these already nearly lost customers to turn up for the upcoming service event and thus stay loyal for longer.

Churn prediction with help of machine learning is a topic that is frequently addressed in academic research, but mostly in the context of specific use-cases in subscription based services / industries like telecoms.[2] Although the problem here cannot be solved in exactly the same way (because automotive service business is not subscription based and the service visits are rare events), they show that the main algorithms used for churn prediction are SVM, tree-based methods (often boosted) and neural networks.

Problem Statement

The main input data is a CSV-file (churnData.csv) containing anonymized historical data of 74'130 cars and their owners (for the rest of the paper we will just speak of instances). These cars have been bought from the company and been in mandatory service at least once in the period from 2006 to

[1] Every two years or 30'000 km of mileage a car must appear for mandatory service.
[2] Good examples are: Tarik A Rashid (2009), Classification of Churn and non-Churn Customers for Telecommunication Companies, in IJBB, Volume 3, Issue 5. T. Vafeiadis et al. (2015), A comparison of machine learning techniques for customer churn prediction, in Simulation Modelling Practice and Theory.

2017. The data was gathered by the company's BI team. Because of limited resources / effort and a rather old-fashioned IT-architecture that was a difficult process and various information from different sources has been consolidated in the file. There are 77 columns that can roughly be grouped in 4 areas:

- Features on a car's age, usage and service history
- Features on technical car specifications
- Features on a car's holder, either from the company's CRM-system or enriched / purchased demographic data from an external supplier
- Features on the company branch the customer is affiliated with

Most important, the cars have been classified in the binary target classes ACTIVE (active customers) and CHURN (lost customers). All cars which have not turned up for mandatory service after more than 27 months (24 months + a tolerance of 3 months) after their last recorded service event are classified as churn, because they should have turned up in the meantime. The other cars are still considered loyal as far as the company can know.

With help of this classification it is possible to apply different *supervised machine learning algorithms for binary classification* to this historical data. The goal is to test several possible classifiers, select the one that is best suited for this problem and train a model that can predict which of the future customers are most at risk for churn with help of a classification scoring.

METRICS

Of course, the model has to be be capable to detect the potential churners as good as possible, but it is crucial in this setting that no discounts are given to loyal customers who would have turned up anyway. That would be a financial disaster. This means that the false positive rate must be as low as possible (→ this means as few loyal customers that are offered a discount as possible). Therefore, the model's ability to *precisely* predict those customers that are about to churn is more important than the model's ability to *recall* all potential churners. As a reminder the definitions for precision / recall are:

- precision: true positives / (true positives + false positives) - how many of the model's identified CHURN instances are really CHURN instances
- recall (sensivity): true positives / (true positives + false negatives) - how many of all the CHURN instances in the data set are correctly identified as CHURN instances

In this project the **F-beta score** is used as main metric to compare the results of different models:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

It considers both precision and recall but gives more emphasis on precision if a value for beta of < 1 is chosen. In this project beta will be set to 0.5, a value that is commonly used to emphasize precision (see https://en.wikipedia.org/wiki/F1_score for reference).

II. Analysis

'PROLOGUE': DATA PREPARATION (GENERATION OF DISTANCE MEASUREMENTS)

Before proper exploratory analysis, one major transformation is applied to the dataset: The original features complete address data for the roughly 70 branches of the automotive company and complete address data for all instances. In this form this information is rather useless for predictive modelling. *That is why the decision was taken to calculate the distances from every instance address to the address of its respective branch* by passing the zip codes of the instances and the exact addresses of the branches to the GoogleMaps API and request the travel distance by car in meters and seconds.

To perform this task, the original branch addresses had to be cleaned with help of another small dataset (branches.csv) that contains the actual physical locations of the branches. This step was necessary because the main data set contains a lot of (sometimes duplicate) administrative addresses for the branches that are not relevant to customers.

As the API request could not return all requested distances (there is a lot of bad data in the original set) the dataset was reduced to the 56'778 instances for which the distances could be calculated. Now it was ready for exploration and copied to a file called 'churnDataWithDistances.csv'.

All this initial preparation is documented in a separate jupyter notebook:
'1_churn_prepareDistanceData.ipynb'

DATA EXPLORATION

The prepared data set contains 82 columns (the original 77 + some cleaned address data that was necessary for the API request + the 2 added distance columns).

First step is *to set aside a test set*. To prevent 'data snooping bias' all EDA is performed only on the later training set. Stratified sampling is used for separation of the data into an 80%/20% split for training and test sets. So the imbalanced target class distribution of 35% of the instances labelled as CHURN and 65% labeled as ACTIVE can be preserved in both sets. Test and Training are then saved as for aCSV-Files for a clean reload later on (after the EDA).

EDA is performed in a structured and iterative process (see below for details) of

1. analyzing / visualizing and
2. immediate cleaning

The immediate cleaning helps to enable a better and more valid exploration.

For all steps of the analysis / visualization and especially for the cleaning functions have been defined and saved to separate helper files. This is especially important for the cleaning, so all cleaning steps can bee applied in identical form to the clean versions of the test and training sets when the proper feature preprocessing is applied after the EDA phase.[3]

[3] Generic plot functions for EDA are saved to a file called 'funEDA.py'. Generic cleaning functions that can be reused on any given dataset, were directly written in a separate file 'funCleaning.py'. Cleaning functions that are specific to this dataset were first written in the EDA ipynb and then saved to a separate file 'funChurn.py' for later reload during preprocessing.

The exact *data exploration steps* were as follows:

Part One: Structure

- Analysis of dataframe structure
    - Drop columns that are of no use for the predictive model
    - Fix datatypes that are wrong (for example transform object types to category)

Part Two: Univariate Analysis

- Exploration of numerical data (13 columns)
    - Removing of outliers, fixing mistakes and implausible data (e.g. age > 100 for drivers)
    - Transforming skewed data (with log10)
    - Dropping additional columns that cannot help further
- Exploration of categorical data (33 columns)
    - Fixing minor irregularities
    - Dropping additional columns that cannot help further
- Exploration of object / string data (4 columns)
    - Cleaning and transforming to category where possible
    - Dropping the rest of those columns
- Exploration of datetime data (2 columns)
    - Construction of a new numeric feature (duration between first and last event)
    - Dropping the rest of those columns
- Analyzing NaN
    - Filling (some) NaN where possible trough inference

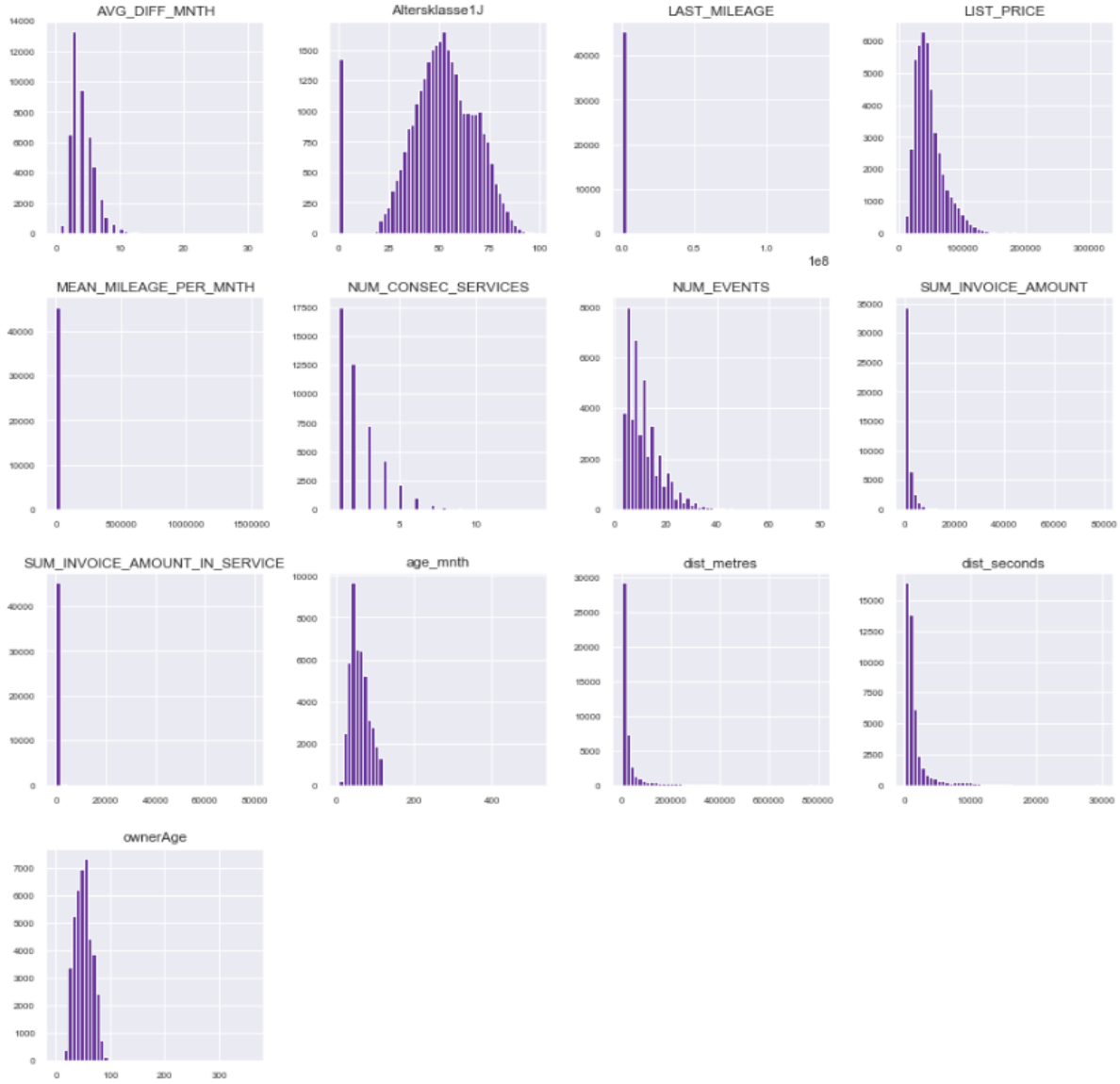
Part Three: Analysis of correlations / relationships

- Analysis of numeric feature correlations (specially to target variable)
- Analysis of correlation / relationship of categorical features to target variable
    - Dropping few rows with extreme correlations between themselves

Two examples why the immediate cleaning during the EDA process was helpful:

- The initial column count was somewhat intimidating, but 30 columns could be dropped right at the start either because they had only one value, were address data that was already substituted with distance data (see initial preparation), because they had far too many NaN, were irrelevant from a logical perspective or were redundant with other columns.
- Due to outliers (mostly from bad / implausible data) and skewed distributions the numerical columns were not meaningful in their initial form und not helpful in calculating good correlation scores. After outlier removal and log-transformation the results improved clearly (see illustrations below).

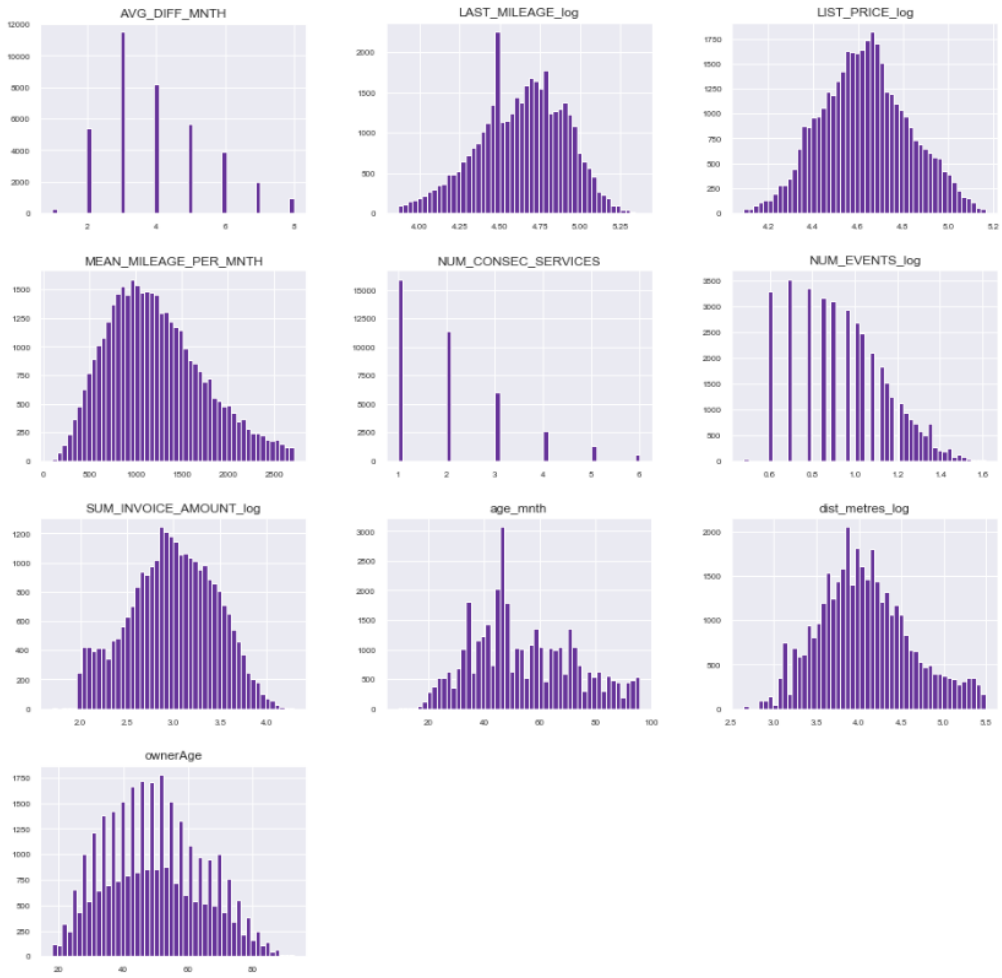
Fig 1a: Illustration of numerical data distribution before cleaning.



Issues with uncleaned data

- Massive outliers for most features, often because of bad data (e.g. car owners older than 300 years, cars with millions of mileage, …)
- Skewed distributions (e.g. dist_metres, list price, …)
- Lots of NaN (e.g. Altersklasse 1J, Sum invoice amount

Fig 1b: Illustration of numerical data distribution after cleaning.



Distributions after cleaning (outlier removal, (partial) log transformation, partial NaN removal) are more informative. It is clearly to see that customer loyalty wears out with time (→ number of consecutive services is decreasing with higher values). There is also a large peak for last mileage at 30'000 km, that is the value when the first service is due. The peak indicates that a lot of customers never return to service after this first time.

All EDA work is documented in a separate jupyter notebook:
'2_churn_EDA.ipynb'

EXPLORATORY VISUALIZATION

The most interesting part of EDA in this project is the analysis of correlations between predictive features and the target variable.

The *correlation between the numeric features and the target variable* (which was also transformed to a numeric column with values 1 for CHURN und values 0 for ACTIVE during cleaning) has iteratively been analyzed for various scales of outlier removal and of log transformation. The correlations for the final configuration are visualized in form of a heatmap:

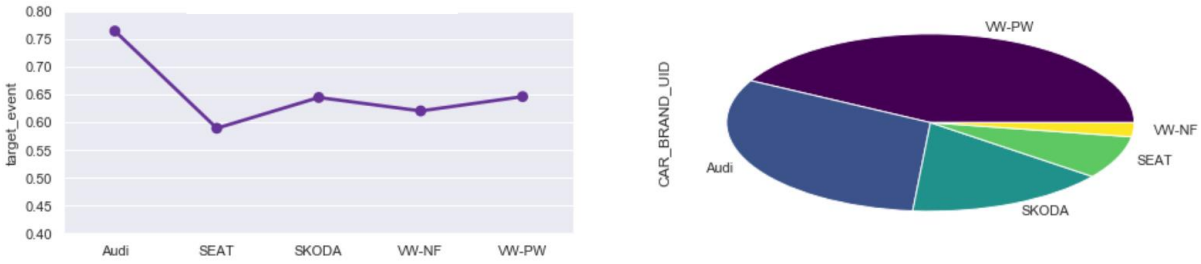
Fig 2: Illustration of correlations among the numerical features of the data set. The first row / column shows correlations between predictive features and target variable.

As can be seen from the plot *there are no numeric features with strong correlations to the target variable*. The strongest correlation is to 'age_mnth' (the age of a car in months). With a value of -0.25 it is negative, this means that older cars have a bigger probability for CHURN. The second-best correlation is (log-transformed) distance in meters, the feature that was initially constructed during data preparation. It has a negative correlation of only -0.16. The rest of the numerical features have absolute correlation strengths between 0.1 and near zero.

For the categorical features the relationship to the target variable is visually analyzed with help of point plots. These show the respective mean of the target values for the different categories. To remember: a mean of 0.65 is the average and means no effect can be seen. Higher means show that more active customers are in this categorial bin as on average, lower means show the contrary.

Only one example is displayed here. (It was the most striking for me from a business perspective.), It shows that drivers of the Audi brand are significantly more loyal than those of the other brands in the survey:

Fig 3: Relationship between car brand and target variable. Values greater than 0.65 show that there are more active customers in this category than on average.[4]



ALGORITHMS AND TECHNIQUES

The plan is to evaluate the possible algorithms for this task in two steps:

1. "Quick & dirty" testing of 6 different classification algorithms (not much tuning).
2. Finer tuning and evaluation of the 2 or 3 best performing algorithms.

The six algorithms for step one are:

- LogisticRegression
- GaussianNB (Naïve Bayes)
- KNeighborsClassifier
- SVC
- RandomForestClassifier
- XGBClassifier

This is a mixture of classic parametric / non-parametric algorithms. Neural networks are out of scope for this project.

Short description of the listed algorithms:

- *Logistic Regression*: In Linear Regression, the relationship between the input variables (x) and output variable (y) is expressed as an equation of the form y = a + bx. Thus, the goal of linear regression is to find out the values of coefficients a (intercept) and b (slope). In logistic regression, the output is in the form of probabilities of the default class. As it is a probability, the output lies in the range of 0-1. The output (y-value) is generated by log transforming the x-value, using the logistic function h(x)= 1/ (1 + e^ -x). A threshold is then applied to force this probability into a binary classification.
- *Naïve Bayes*: NB calculates the probability that an event will occur (CHURN, ACTIVE), given that another event has already occurred (input feature values), with help of Bayes' Theorem. To calculate the probability of an outcome given the value of some variable, that is, to calculate the probability of a hypothesis(h) being true, given our prior knowledge(d), we use

[4] Each of these point plots is paired with a pie chart to show the quantitative category distribution. The smaller the individual categories the less the values can be trusted / the more outliers are produced.

Bayes' Theorem as follows: P(h|d)= (P(d|h) * P(h)) / P(d). This algorithm is called 'naïve' because it assumes that all the variables are independent of each other, which is a naive assumption to make in real-world examples.

- *k-Nearest Neighbors*: When an outcome is required for a data instance, the KNN algorithm goes through the entire dataset to find the k-nearest instances to it, or the k number of instances most similar to that record, and then outputs the mean of the outcomes (for a regression problem) or the mode (most frequent class) for a classification problem.
- *Support Vector Classifier (SVC)*: Support vector classifier learns a hyperplane to classify data into classes. It performs the so called 'kernel trick' to project the data into higher dimensions. Once projected into higher dimensions SVM figures out the best hyperplane which separates the data into classes. It attempts to maximize the margin (→ distance between nearest instances of the classes and the hyperplane), so that the hyperplane is just as far away from the nearest instance of class x as from the nearest instance of class y. In this way, it decreases the chance of misclassification.
- *Random Forest*: This is a method for bagging multiple weak learners together to generate a strong learner that can generalize well at the same time (→ low variance). The first step in bagging is to create multiple models with datasets created using the Bootstrap Sampling method. The second step is to create multiple models by using the same algorithm on the different generated training sets. Unlike in a decision tree, where each node is split on the best feature that minimizes error, in random forests, a random selection of features is chosen for constructing the best split. Thus, in bagging with Random Forest, each tree is constructed using a random sample of records and each split is constructed using a random sample of predictors.
- *Gradient boosting*: This technique also produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion: After each first weak learner, the next has to focus on the mistakes made, so the incorrectly classified points are subsequently weighted more and more. This is more elaborate than the bagging method, where the learners are trained differently but all have the same task.

BENCHMARK

Today the company has no means to identify the customers at risk. It knows from analysis of its customer base that overall it loses roughly 35% of customers from service event to service event. We find this ratio also in the data set: 35% of all instances are labelled as CHURN.

This means the following: If today a selection of random customers is offered a retention discount for their next service visit, 65% of these discounts go to customers that would have turned-up anyway. This 'naïve' approach must be beaten by far.

III. Methodology

DATA PREPROCESSING

Preprocessing and the whole rest of the project is documented in a new ipynb '3_churn_predictiveModelling.ipynb'.

First step is to load a fresh, clean version of the training (and test) set in its pre-EDA form. This to make sure that *all necessary cleaning operations defined in the EDA phase* are performed identically on both sets with help of a Pandas pipeline:

```
trainSet = (trainSet
 .pipe(funCleaning.fixDtypes, colsToCat=colsToCat)
 .pipe(funChurn.funCleanNumericals)
 .pipe(funCleaning.colsToLog10, colsToLog10=colsToLog10)
 .pipe(funChurn.funCleanCategoricals)
 .pipe(funChurn.funCleanStrings)
 .pipe(funChurn.funCleanMissingStates)
 .pipe(funCleaning.delCols, colsToDel=colsToDel)
);
```

Next, two separate functions to remove NaN (with different strategies for different columns) and outliers are defined and applied.[5]

Then the target labels are separated from the rest of the sets and the necessary preprocessing for machine learning is performed with a small scikit-learn pipeline: One-Hot-Encoding for categorical columns, standard scaling for numeric ones:

```
# pipeline for standardization and one-hot-encoding
fullPipeline = ColumnTransformer([
        ('std_scaler', StandardScaler(), numCols),
        ('encoder', OneHotEncoder(), catCols),
    ])
```

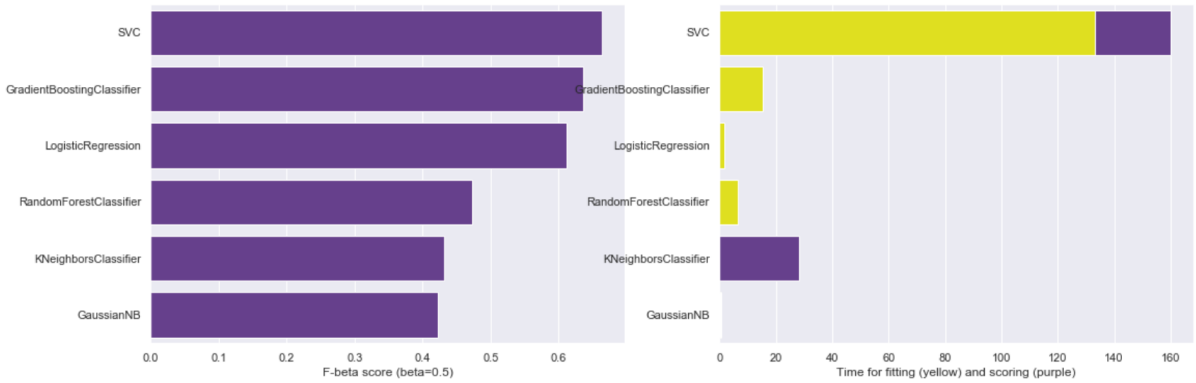
[5] These functions are not included in the initial pipeline because first I wanted to integrate them as tunable hyperparameters in the scikit-learn pipeline that will follow next. But unfortunately I had to give up on that (due to lack of know-how). So I had to test the effects of different levels of NaN handling and outlier removal manually on the longlist model scoring. Another reason for having those functions separate, is that I did not want to perform outlier removal on the test set (to get more robust results). Last but not least for NaN-removal I had to impute the values of the training set in the test set too.

IMPLEMENTATION

Step 1: Fit 6 longlisted algorithms to the preprocessed training set and compare the resulting F-beta-scores. For this first round the default parameters are not changed. Evaluation / testing is performed on the training set with 5-fold cross validation.

The resulting mean score values (and runtimes) are:

Fig 4: Comparision of training scores and runtimes for longlisted models



Three algorithms score an F-Beta-Score of 0.6 or higher and are selected for further tuning (reported in the next section):

- **SVC (mean F-beta-score: 66.44)**
- **XGBClassifier (mean F-beta-score: 63.71)**
- **LogisticRegression (mean F-beta-score: 61.16)**[6]

Step 2: Calculate Feature Importance with default XGB model and remove features based on thresholds 0.00000001, 0.001, 0.005, 0.01, 0.05. Calculate F-Beta scores for models with 5-fold CV. (See next section for results.)

Step 3: Reduce dimensionality with PCA based on retained variance levels of 0.94, 0.96, 0.98. Calculate F-Beta scores for models with 5-fold CV. (See next section for results.)

Step 4: Tune hyperparameters with grid search. Calculate F-Beta scores for models with 5-fold CV. (See next section for results.)

Grids are:

- LogReg = {'solver':['liblinear', 'saga'], 'C': [100.0, 200.0, 500.0]}
- XGB = = {'n_estimators':[200, 500, 1000], 'learning_rate': [0.01, 0.05, 0.1], 'max_depth' : [3, 6]}
- SVC = [{'kernel': ['poly'], 'C': [1, 10]}, {'kernel': ['rbf'], 'C': [1, 10, 100], 'gamma': [0.001, 0.01, 0.1]}]

LogReg is dropped for next steps.

Step 5: Calculate learning curves for best versions of SVC and XGB with 3-fold CV. (See results section for results.)

[6] F-beta-scores are multiplied *100 for better readability.

Step 6: Prepare test set with the same cleaning an preprocessing functions as the training set was prepared (see below for tipp).

Step 7: Evaluate two models on test data. (See results section for results.)

REFINEMENT

The scores reported in step 1 above are the 'initial solution' / base for further refinement. Before tuning model parameters, the different effects of *feature selection and dimensionality reduction* are tested.

- Feature selection (FS) is based on different thresholds of feature importance weights calculated with the XGB model
- dimensionality reduction is performed with PCA for different percentages of retained total variance

The results show that:

- SVC benefits from both: best score for FS: 68.28, for PCA: 67.48
- XGB is not so much affected: best score for FS: 63.86, for PCA: 61.53
- LogRegression suffers from less features: best scores for FS: 60.25, PCA: not even tested

This was considered for *fine tuning of the hyperparameters of these three classifiers with grid search*. SVC was fitted to a training set with reduced dimensionality (96% of the original variance preserved, 231 features compressed to 66 components). The other two algorithms were fitted on the unreduced training set.[7]

The refined final model configurations and corresponding best mean F-beta-scores for 5-fold CV were as follows (hyperparameter changes due to grid search are marked in red):

XGB: top score: 69.08

GradientBoostingClassifier(criterion='friedman_mse', init=None, learning_rate=0.05, loss='deviance', max_depth=6, max_features=None, max_leaf_nodes=None,min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=500, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)

SVC: top score: 67.82

SVC(C=10, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

[7] Note on feature importance / feature selection: It was interesting to see that the feature importance are what had to be expected after the EDA. But for transformation of the training set I decided against feature selection and pro dimensionality reduction with PCA because the former would have had to be done manually (and that is, by the way, not so easy after one-hot-encoding) and the latter can be fitted to the test set automatically.

LogisticRegression: best score 61.19

LogisticRegression(C=500.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

After finetuning of the hyperparameters the score of a more complex version of XGB has overtaken SVC. Logistic Regressions will no longer be considered as potential solution to the problem.

IV. Results

Model Evaluation and Validation

It is interesting to see if the gains in the scores that can be seen after tuning are not only achieved due to higher variance / overfitting (especially of the XGB model). For this purpose, learning curves are calculated (based on 3-fold CV on the training set):

Fig 5a: Learning curves for XGB model

For the XGB model the two curves do not converge well and the learnings from more data do not help much to improve the validation score. This is a sign for high variance / overfitting and the good results of this model cannot be trusted.

Fig 5b: Learning curves for SVC model

Relative to the XGB model the SVC model has more bias but much less variance. The results of this model for new data will be more reliable and robust.

This in mind, the final evaluation on the test set shows the following results:

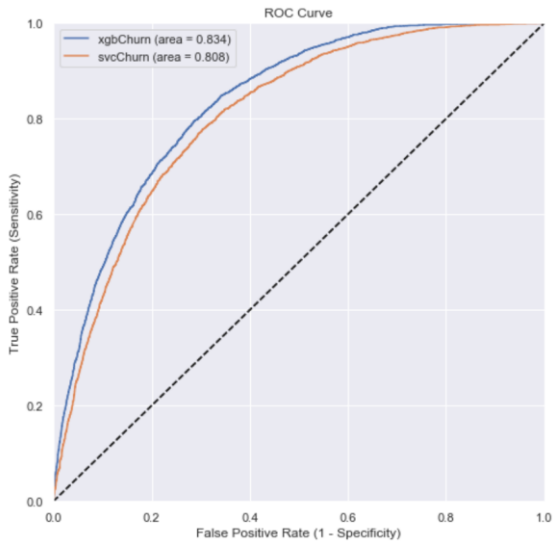
- **XGB - F-beta-score: 64.73**
- **SVC - F-beta-score: 62.11**

The expected drop in the score for the XGB model is there but (less expected maybe) the same holds true for SVC also. So XGB scores still higher.

JUSTIFICATION

Both model's results are significantly better than the benchmark (random approach). This can be shown with a plot of the area under the ROC-Curve. The dotted line is the result of the benchmark / naïve approach:

Fig 6: ROC-Curves



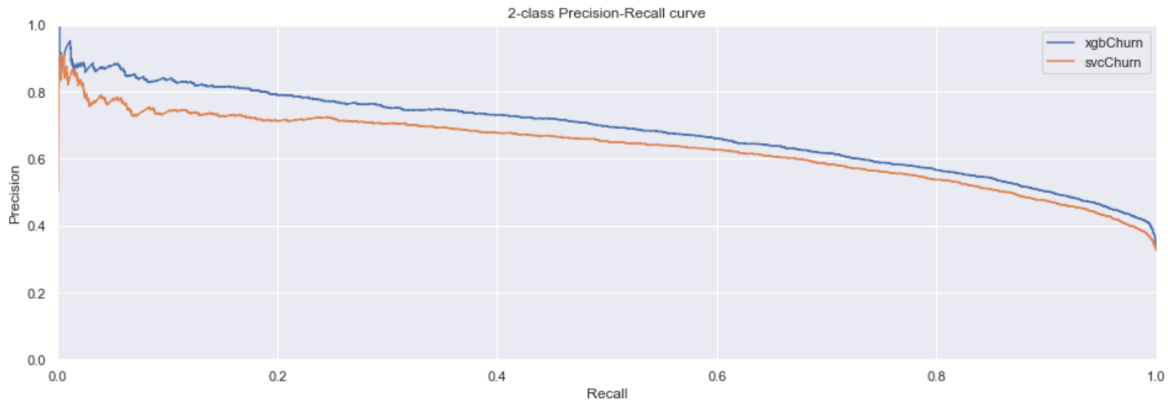
As already told in the metrics section, a high precision is crucial for the implementation of this solution. Unfortunately, the average precision scores are probably not high enough for that:

- **XGB – Precision of CHURN predictions: 69%**
- **SVC – Precision of CHURN predictions: 63%**

One possible solution can be to only contact customers with a certain probability threshold for being classified as CHURN. If for example only customers with CHURN probability >= 90% are contacted the precision is > 85% for both models – that should be acceptable, but it means that only about 5% of all predicted CHURNERS will be contacted.

The underlying idea can be illustrated with help of the precision-recall-curves:

Fig 7: Precision-recall curves

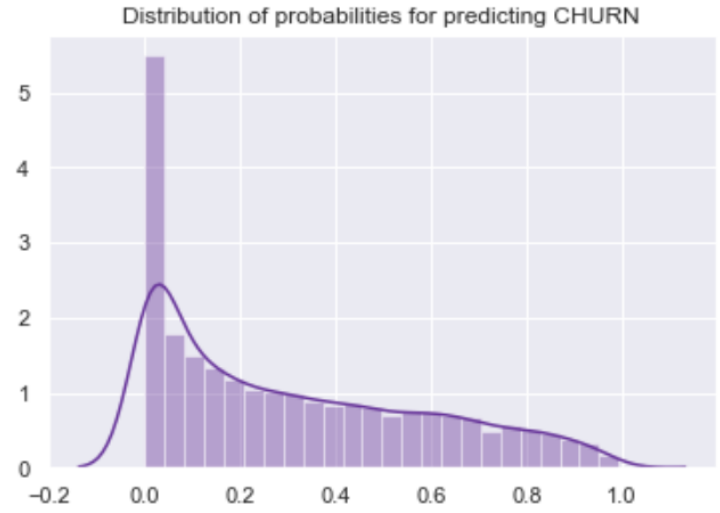


No matter which of the two models is chosen as the final solution, the results are not good enough yet to have solved the problem.

V Conclusion

FREE-FORM VISUALIZATION

Some important visualizations have already been shown in the past sections. Here I would like to show one additional interesting finding, although it was not analyzed more deeply in the project. And I only visualized it for the XGB model: It is the distributions of probabilities for classifying an instance as CHURN. In reverse it can be read as the distributions of probabilities for classifying an instance as ACTIVE of course:



For a lot of instances, the model is very confident to predict that they are NOT CHURN / that they are ACTIVE. But unfortunately for us, it has much more problems to say the contrary with high probability. Maybe this is something to think about when trying to improve the model.

REFLECTION

The process of this project can be summarized with the follow main steps:

1. Transforming address data into distance feature with help of GoogleAPI request
2. Split in test and training sets with stratified sampling
3. EDA with definition and testing of data cleaning functions
4. Preprocessing of data with pipelines
5. Quick-check of 6 different classifying algorithms
6. Tuning of 3 best performing algorithms
7. Evaluation and validation of two best models
8. Conclusion that problem is not solved yet

First it has to be said, that I learned extremely much in this project and I explored a lot of things / options. But sometimes I felt that for every question answered another two were raised. The most difficult part for me was probably the decision how to clean / preprocess data. I would have liked to build some custom transformers for outlier and NaN treatment, log transformation, and so on, so that I could have them included different options for each in the grid search process for model evaluation / testing. On the other hand I had performance issues, especially when performing lots of fittings for the SVC model and to test many more options would probably have been too time consuming.

IMPROVEMENT

- After the correlation analysis during EDA it was to assume, that the model probably would not be very satisfying. There was a lot of more or less randomly assembled data in the set and I think the biggest improvement will be achieved in selecting some data that is better suited to the problem solution (for example more service history details).
- Cleaning / preprocessing was kind of a compromise to suit all models in the longlist and identical for all models (with exception of the extra dimensionality reduction for SVC). Maybe XGB would score better without log transformation, outlier removal and the like. This should be tested.