UDACITY CAPSTONE PROJECT REPORT
ELO MERCHANT CATEGORY RECOMMENDATION

Sebastian Mack
Machine Learning Nanodegree
Udacity
`mack.seb@gmail.com`

January 7, 2019

Contents

List of Figures

1 Definition

1.1 Project Overview

As the field of study I selected an interesting topic from the finance industry that caught my attention when a new competition on the data science platform Kaggle had been announced. The initiator of this contest is one of the largest payment brands in Brazil called Elo (refer to `https://www.cartaoelo.com.br`) which has built partnerships with merchants in order to offer promotions or discounts to cardholders. Main objective of this project is to understand customer loyalty of the anonymized data that Elo is providing.

For this project the publicly available data for the "Elo Merchant Category Recommendation" Kaggle competition will be considered. It consists out of the following files:

(`https://www.kaggle.com/c/elo-merchant-category-recommendation/data`)

- **train.csv** - the training set
- **test.csv** - the test set
- **sample_submission.csv** - a sample submission file in the correct format - contains all card_ids you are expected to predict for
- **historical_transactions.csv** - up to 3 months' worth of historical transactions for each card_id
- **merchants.csv** - additional information about all merchants / merchant_ids in the dataset
- **new_merchant_transactions.csv** - two months' worth of data for each card_id containing all purchases that card_id made at merchant_ids that were not visited in the historical data.

In addition data field descriptions are provided in **Data_Dictionary.xlsx**.

Typically companies work with databases involving structured datasets which is also the case for our domain. There has been a lot of research and development in order to find algorithms and models for such type of data. In recent years especially a technique called stacking is becoming more popular and is achieving state of the art results within the field. For more information refer to [1, 2, 3, 4]

For me personally this problem is very interesting because it represents a real life problem that not only Elo but many other companies are facing right now. In this situation companies are already acquiring and collecting data with their existing services but still struggle to create a business value out of it. In this particular case we can see that by developing a model that can uncover the signal in customer loyalty, a new business value can be proposed. Both the consumer and the merchants could benefit from a well personalized experience. Furthermore the offering company could gain new clients with this value adding service.

1.2 Problem Statement

The problem that is to be solved can be described as a supervised machine learning problem because the model will be trained based on a given target feature. Since this target variable has a continuous value it can be further classified as a regression problem. A problem of this type can be solved and modeled with various approaches but in this study the most promising will be applied.

The main problem will be to build a model from the given data that can predict a loyalty score for each card_id given in the test data. Therefor the training data contains several features for each card_id and separate data files with additional information considering transactions of the card as well as information describing the respective merchants of the purchases.

In order to find a solution to the problem, I will use models and algorithms from supervised machine learning because in our training set we are already given our target variable in form of the loyalty score of each card owner. The specific technique which will be applied to get a model for this project is called stacking. The underlying idea behind stacked generalization was first introduced by a paper [1] from Wolpert. In other research [3] it was shown that super learning approaches provide both a fundamental theoretical as well as practical improvement to the construction of a predictor.

The principals of stacking have not only been proven their potential in acadamic world but also in real applications. In many of the recent kaggle competitions that involved structured data sets the winning models included stacking approaches.

Finally, I will outline the most important steps within my theorectical workflow in order to find a solution for the described problem. A structured approach will be helpful for a reasonable result and to have a scientific discussion on the final model. My planned workflow includes the following steps:

1. **Exploratory Data Analysis (EDA)**: As a first step I will explore the provided data and make an analysis. This includes summarizing properties and visualizing important outcomes. It will be also very useful to identify features that are relevant for the model and also to give hints for transformations that are required for fitting the data.
2. **Feature Engineering**: Within this step the knowledge gained from the previous step will be applied to clean the data set and to select important features as well as to define new ones.
3. **Train Baseline Model**: When the previous step is completed, the obtained transformed and extended dataset can be used to train the baseline model (or benchmarking model). It will be an implementation of an Ensemble model with gradient boosting for regression.
4. **Train Stacked Model**: The most important step is to train our stacked model which will be combined from several weak learners that still have to be selected.
5. **Tune paramters**: Since there are lots of parameters available in order to train a suffisticated model, it will be necessary to repeat some of the steps and fine tune the model until it is able to produce the desired scores.
6. **Evaluate Metrics**: In the last step, the results and scores of all generated models will be evaluated and compared to one another.

In case I am able to build a model with promising outcomes on the testing data, I will also make a submission file for the kaggle competition to get a score for the public leaderboard to test wether my model can compete with the ones from other kagglers.

1.3 Metrics

Since one objective of this project is also to produce a submission file for the ongoing kaggle competition, the choosen evaluation metric will be the Root Mean Squared Error (RMSE) as suggested by the rules.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2} \tag{1}$$

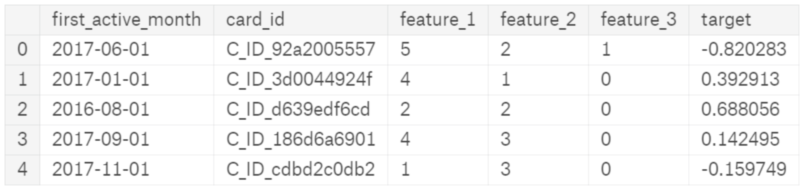
In our case we can calculate a score with equation (1) where $\hat{y}$ is the predicted loyalty score for each `card_id`, and y is the actual loyalty score assigned to a `card_id`.

2 Analysis

Before we begin to model the given problem, an analysis of the given data sets needs to be performed in order to understand which algorithms are going to be used in the next steps of the project. This chapter includes a data exploration which describes characteristic properties of the data as well as visualizations that help to summarize the most important outcomes. In another section of this chapter we discuss different algorithms that we intend to use for solving the problem given the specific domain. Finally, we provide a clearly defined benchmark result for comparing across performances obtained by our solution.

2.1 Data Exploration

Within this section, we present basic statistics and information that can be extracted from the provided data sets. As can be seen in figure 1 which represents the first entries of the training data, it consists out of 6 distinct columns. We can find a column with 201917 unique card ids for each entry as well as a date column with the first active month of the respective card. In addition we are given 3 columns with features (1,2 and 3) that contain categorical values and also the target variable namely the loyalty score of the entry.



	first_active_month	card_id	feature_1	feature_2	feature_3	target
0	2017-06-01	C_ID_92a2005557	5	2	1	-0.820283
1	2017-01-01	C_ID_3d0044924f	4	1	0	0.392913
2	2016-08-01	C_ID_d639edf6cd	2	2	0	0.688056
3	2017-09-01	C_ID_186d6a6901	4	3	0	0.142495
4	2017-11-01	C_ID_cdbd2c0db2	1	3	0	-0.159749

Figure 1: Head of the training data

The target variable is a continuous value that ranges in the data set from a minimum value of -33.219281 to a maximum value of 17.965068. In figure 2 the probability density plot of the loyalty score is shown. We can see that the mean of the target variable is near 0 (-0.393636) and the calculated standard deviation equals 3.850500. It is important to note that the distribution is highly skewed or asymmetric (-6.72016) and has a positive kurtosis (55.031783).

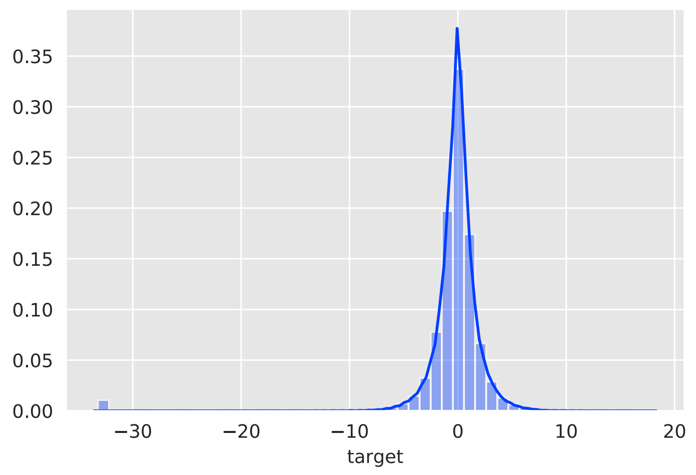


Figure 2: Distribution of the target variable (Loyalty score)

One observation that can be made from the boxplot (see figure 3) of the loyalty score is that there are many values that can be considered outliers. Although these points do not fall in 1.5 of the IQR we cannot simply remove them because they represent highly loyal or disloyal card owners and are especially from interest in our model.

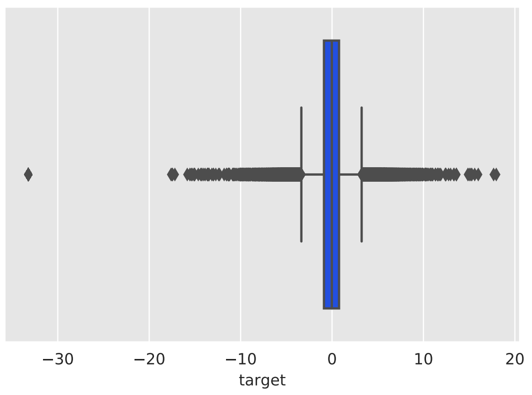


Figure 3: Boxplot of the target variable (Loyalty score)

The test data set contains exactly the same columns as the as the train data set except the target variable which is hidden for the leader board evaluation of the Kaggle competition. It includes 123623 entries with a similar relative distribution (value count) of categories compared to the training data.

Beside the train and test data sets we have additional information about the transactions of each card and about all merchants in different files but since this data needs to be aggregated and combined with the original sets we will discuss it in section 3.1.

2.2 Exploratory Visualization

This section provides visualizations that extract the most relevant characteristics about the data. All of the four features to analyze (`first_active_month`, `feature_1`, `feature_2` and `feature_3`) contain categorical values. In figure 4 we have a range of the first active month of the card from November 2011 to February 2018. On Exception is the month of January 2011 where no data points can be found in our set. A trend which can be observed in this figure is the increasing amount of active cards in the given time period.

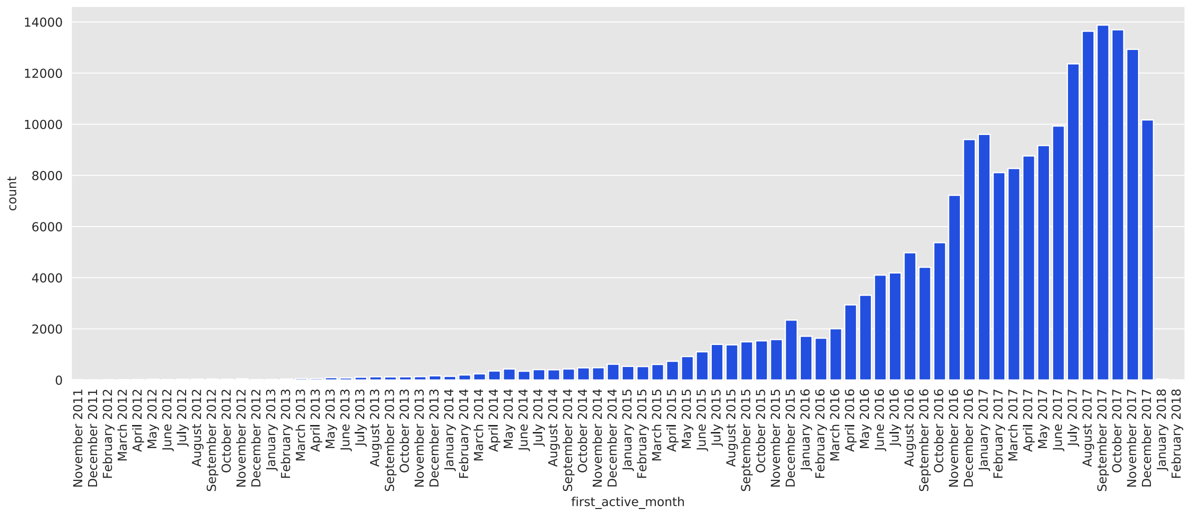


Figure 4: Plot of value count for each unique category

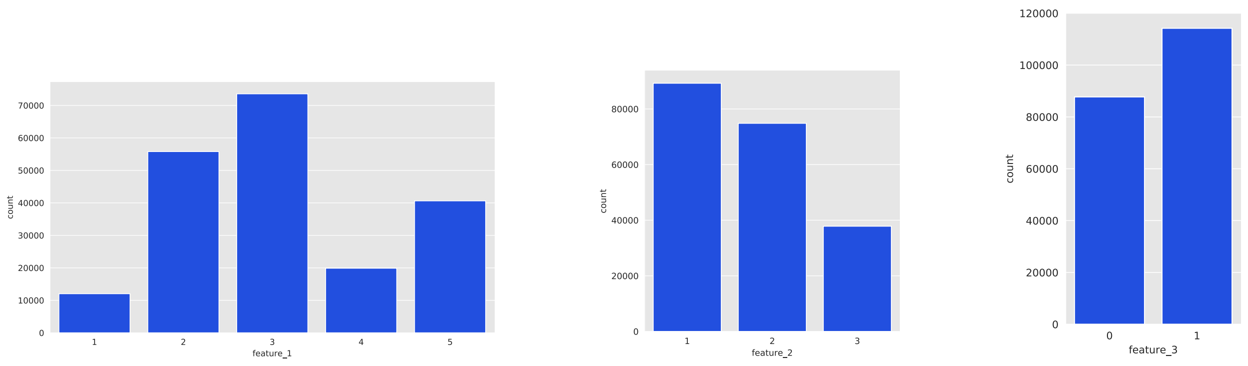


Figure 5: Plot of value count for each unique category: `feature_1`, `feature_2` and `feature_3`

As we can see from figure 5 we have five different categories for `feature_1`, three for `feature_2` and two for `feature_3`. The value counts for each of the categories demonstrate that some of them are more dominant in the data set than others. Within figure 6 the distribution of the target variable (loyalty score) is plotted for each category in form of violin plots. Across all features we can see that the distributions are comparable to one another and all of them have their peaks near 0.

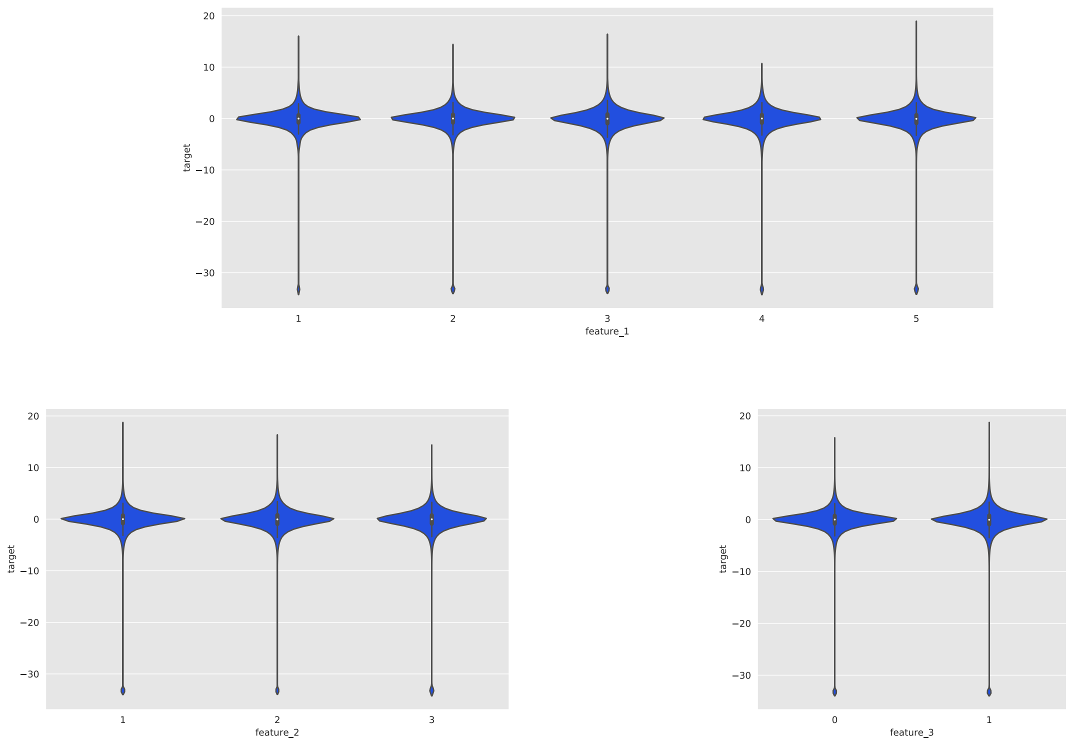


Figure 6: Violin plot for each unique category of target: `feature_1`, `feature_2` and `feature_3`

For more visualizations please refer to the appendix (6) which includes several plots of the testing data set and a detailed violin plot of the `first_active_month` feature.

2.3 Algorithms and Techniques

The technique that is being applied in order to solve the given problem is called *Stacking* or *Super Learning*. As it is decribed in [5], ensemble machine learning methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms. Many of the popular modern machine learning algorithms are actually ensembles. For example, Random Forest and Gradient Boosting Machine (GBM) are both ensemble learners. Both bagging (e.g. Random Forest) and boosting (e.g. GBM) are methods for ensembling that take a collection of weak learners (e.g. decision tree) and form a single, strong learner.

Stacking is a class of algorithms that involves training a second-level "metalearner" to find the optimal combination of the base learners. Unlike bagging and boosting, the goal in stacking is to ensemble strong, diverse sets of learners together.

For the implementation of this technique we use the `H2O` machine learning framework that provides an supervised ensemble machine learning algorithm for stacking. This Super Learner Algorithm includes the following steps for building a model:

1. Set up the ensemble
   (a) Specify a list of L base algorithms (with a specific set of model parameters)
   (b) Specify a metalearning algorithm
2. Train the ensemble
   (a) Train each of the L base algorithms on the training set
   (b) Perform k-fold cross-validation on each of these learners and collect the cross-validated predicted values from each of the L algorithms
   (c) The N cross-validated predicted values from each of the L algorithms can be combined to form a new N x L matrix. This matrix, along wtih the original response vector, is called the “level-one” data. (N = number of rows in the training set.)
   (d) Train the metalearning algorithm on the level-one data. The “ensemble model” consists of the L base learning models and the metalearning model, which can then be used to generate predictions on a test set
3. Predict on new data
   (a) To generate ensemble predictions, first generate predictions from the base learners
   (b) Feed those predictions into the metalearner to generate the ensemble prediction

2.4 Benchmark

For benchmarking purposes it was decided to use a Gradient Boosting Machine (GBM) algorithm as bench-marking model which can be compared to the main (stacked) model. The implementation we use is provided by the `H2O` machine learning framework. As it is described in [6], Gradient Boosting Machine for Regression is a forward learning ensemble method.

Table 1: Results for Cross Validation with a Gradient Boosting Machine (GBM) Benchmark

	mean	sd	cv_1	cv_2	cv_3	cv_4	cv_5	cv_6
mae	1.6049318	0.0098847	1.6034063	1.6325411	1.6202565	1.6075463	1.6059594	1.5916957
mrd	14.758833	0.2759703	14.457203	15.389267	15.149721	15.039012	14.56817	14.3106165
mse	14.758833	0.2759703	14.457203	15.389267	15.149721	15.039012	14.56817	14.3106165
r2	0.0044686	0.0004175	0.0037364	0.0044269	0.0051961	0.0053651	0.0044451	0.0046830
rd	14.758833	0.2759703	14.457203	15.389267	15.149721	15.039012	14.56817	14.3106165
rmse	3.8413863	0.0359411	3.802263	3.9229157	3.8922644	3.8780165	3.816827	3.7829375

- Benchmark score on test data (public Kaggle leaderboard): **3.921**

The benchmark model is trained with default parameters without any tuning of hyperparamters. As data input the training data from the competition is used without any feature engineering beside the transformation of dates and data types.

3 Methodology

In this chapter we discuss the methodology that has been applied to the given problem. It includes the steps for pre-pocessing the data in order to address any abnormalities or characteristics. Furthermore, it documents the implemented metrics, algorithms and techniques.

3.1 Data Preprocessing

The first thing to notice about the given data is that it is very clean presumably because it has already been extensively preprocessed by the competition host. In the training and testing data we have zero missing values and the additional data sets for transactions and merchants contain only a small fraction (below 0.1 %) which will be handled depending on the respective feature.

After loading the data sets, an important step is to define the data type for each column, because the framework pandas for structured data can handle it more efficiently. This has especially advantages for the usage of memory because data types like `float` allocate more space in memory than for example simple `int` or `bool`. In our situation, a clever assignment of data types resulted in memory savings up to several Gb of the RAM.

Feature engineering for this domain can be seen as probably the most important part to gain good results. A few basic features could be added to the training and testing data frames without aggregating data from additional sources. The following features have been created directly:

Table 2: New direct features for Train and Test data sets

Feature	Description
year	Year derived from the first_active_month feature
weekofyear	Week as integer derived from the first_active_month feature
month	Month as integer derived from the first_active_month feature
elapsed_time	Time difference in days between reference date and first_active_month date

In addition several other features have been aggregated from the `historical_transactions` and `new_merchant_transactions` data sources. A cleanup that was necessary for both files, is to map the given strings "Y" and "N" to binary values (1 and 0) in the "category_1" and "authorized_flag" columns. For the "category_2" column we used a lambda function to reduce strings (e.g. 1.00000) to their shorter integer format (1) without loosing any information.

The feature "purchase_date" can be used to create other features and in contrast to the date from the train and test data sets, the purchase dates also includes the time of the purchase. For this reason we can derive the features shown in table 3.

Table 3: New features for historical transactions and new merchant transactions

Feature	Description
purchase_year	Year as integer derived from the purchase_date feature
purchase_month	Month as integer derived from the purchase_date feature
purchase_weekofyear	Week of year as integer derived from the purchase_date feature
purchase_dayofweek	Day of week as integer derived from the purchase_date feature
purchase_hour	Hour as integer derived from the purchase_date feature
purchase_date	Date transformed to int64 * 1e-9
month_diff	Time difference in days between reference date and purchase_date date

In the given data sets for historical transactions and new merchant transactions several transactions can be found for each unique card_id (owner). Since our target variable is the loyalty score for each card owner in the train and test data, it is not simply possible to merge all of the transaction data into those frames.

Table 4: Aggregated features for Train and Test data sets

Feature	Aggregation Function
purchase_amount, installments	sum, max, min, mean, var
purchase_date	max , min
month_lag, purchase_month	max, min, mean, var
month_diff	mean
card_id	size
category (all 9), authorized_flag	sum, mean
purchase_year, purchase_weekofyear, purchase_dayofweek, purchase_hour, subsector_id, merchant_id, merchant_category_id	nunique

For the aggregation of features for each card id several functions are defined in table 4 and a complete list of features can be found in the appendix. Both the aggegated historical transaction features and the new merchant transactions are merged on the respective card id. Furthermore it is to mention that aggregation and merging had to be performed for each dataframe separately because otherwise we ran into memory issues.

3.2 Implementation

In the previous section, we described the engineering of features and construction of the input data of the model in form of train and test data sets. This input is used in the definition of our model architecture which will make use of Stacking.

As described in [7] the architecture makes use of two different types of data as it can be seen in the figures below. The so called "Level Zero" data (the input data) is used by the base learners. One the higher level we combine the predictions of the base learners into a new "Level One" data frame which is used by the super learner.

The following steps have been performed in the implementation phase:

1. Define matrix, X (input data: train and test), and response, y (target: loyalty score)
2. Specify L base learners (with model params)
3. Specify a metalearner
4. Perform k-fold CV on each of the L learners

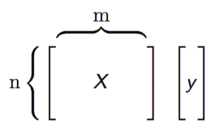


Figure 7: "Level Zero" data

1. Collect the predicted values from k-fold CV that was performed on each of the L base learners
2. Column-bind these prediction vectors together to form a new design matrix, Z
3. Train the metalearner using Z, y

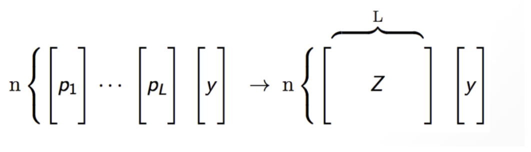


Figure 8: "Level One" data

For practical implementation purposes we use the following frameworks:

- H2OGradientBoostingEstimator (h2o.estimators.gbm)
- H2OGeneralizedLinearEstimator (h2o.estimators.glm)
- H2ORandomForestEstimator (h2o.estimators.random_forest)
- H2OStackedEnsembleEstimator (h2o.estimators.stackedensemble)
- numpy
- pandas
- matplotlib, seaborn

It is very important to notice that all base models must have the same cross-validation folds and the cross-validated predicted values must be kept because the H2OStackedEnsembleEstimator will use them to form the "Level One" data.

3.3 Refinement

Before training the stacked model we experimented with the benchmark model (H2OGradientBoostingEstimator) in order to tune the parameters in a grid search approach consider the following hyperparameters:

hyper_params =
"learn_rate": [0.01, 0.03],
- Specifies the learning rate. The range is 0.0 to 1.0
"max_depth": [4, 6, 8, 12, 16, 20],
- Specifies the maximum tree depth. Higher values will make the model more complex and can lead to overfitting).
"sample_rate": [0.6, 0.8, 1.0],
- Specifies the row sampling rate (x-axis).
"col_sample_rate": [0.6, 0.8, 1.0]
- Specifies the column sampling rate (y-axis).

This resulted in the in an optimal setup with learn_rate=0.01, max_depth=6, sample_rate=0.8 and col_sample_rate=0.8 which will be used for the gbm estimator in the stacked model.

Reported regression model metrics on train data
MSE: 13.187148162028853
RMSE: 3.631411318210711
MAE: 1.542065760164572
Mean Residual Deviance: 13.187148162028853

Reported regression model metrics on cross-validation data
MSE: 13.602099833569243
RMSE: 3.6881024705896177
MAE: 1.5562552491244939
Mean Residual Deviance: 13.602099833569243

4 Results

In this chapter we present the results of the combined super learning model and evaluate metrics that describe it. As a conclusion we will justify our solution by comparing it to the chosen benchmark model.

4.1 Model Evaluation and Validation

In tables 5, 6 and 7 we summarized the results from the 3 different Base Learners (GLM, DRF, GBM) by performing kfold cross validation (k=6). The fold assigment has been the same for all of the Base Learners in order to compare them and in addition this is a prerequesite for the stacking process.

By comparing the mean value for the rmse metric of the Base Learners, we can already apply a ranking between them across the cross validation data. Best results have been produced by the GBM (3.673) followed by the DRF (3.726) and GLM (3.830). When we look at the cross validation results among the 6 folds, it can be noticed that the models are robust against pertubations in the training data. We assume that especially the distribution of outliers in the folds could be the responsible for small changes.

In order to test if the models generalize well to unseen data, each one has also been evaluated against the public Kaggle leaderboard. The ranking among them is unchanged but we obtain slightly worse rmse scores: GBM (3.752) followed by the DRF (3.778) and GLM (3.906).

Table 5: Results for Cross Validation with a Generalized Linear Model (GLM) Base Learner

	mean	sd	cv_1	cv_2	cv_3	cv_4	cv_5	cv_6
mae	1.581836	0.0203363	1.5764347	1.5607811	1.6022342	1.628451	1.5383326	1.584782
mrd	14.688586	0.6209948	14.677058	14.209316	15.113588	16.225391	13.331208	14.574956
mse	14.688586	0.6209948	14.677058	14.209316	15.113588	16.225391	13.331208	14.574956
r2	0.0092557	0.0003340	0.0089441	0.0095686	0.0092110	0.0085077	0.0100180	0.0092848
rd	494312.56	20898.559	493927.03	478186.12	508617.6	546033.06	448635.16	490476.4
rmse	3.8308656	0.0807933	3.831065	3.7695246	3.88762	4.028075	3.6511927	3.8177161

- GLM Base Learner score on test data (public Kaggle leaderboard): **3.906**

Table 6: Results for Cross Validation with a Distributed Random Forest (DRF) Base Learner

	mean	sd	cv_1	cv_2	cv_3	cv_4	cv_5	cv_6
mae	1.6385423	0.0183705	1.6239307	1.6273823	1.6647147	1.6719296	1.595753	1.6475433
mrd	13.899682	0.556475	13.843744	13.750434	14.459413	15.067332	12.484625	13.792545
mse	13.899682	0.556475	13.843744	13.750434	14.459413	15.067332	12.484625	13.792545
r2	0.0622485	0.0088655	0.0652129	0.0415541	0.0520963	0.0792737	0.0728857	0.0624682
rd	13.899682	0.556475	13.843744	13.750434	14.459413	15.067332	12.484625	13.792545
rmse	3.7267144	0.0751078	3.7207184	3.7081578	3.8025534	3.8816662	3.5333588	3.7138317

- DRF Base Learner score on test data (public Kaggle leaderboard): **3.778**

Table 7: Results for Cross Validation with a Gradient Boosting Machine (GBM) Base Learner

	mean	sd	cv_1	cv_2	cv_3	cv_4	cv_5	cv_6
mae	1.5640444	0.0184761	1.5589854	1.5495439	1.5860294	1.6010317	1.5195141	1.5691615
mrd	13.504545	0.5703331	13.43026	13.37311	14.004401	14.745635	12.061675	13.412193
mse	13.504545	0.5703331	13.43026	13.37311	14.004401	14.745635	12.061675	13.412193
r2	0.0890768	0.0084026	0.0931331	0.0678547	0.0819251	0.0989318	0.1042942	0.0883222
rd	13.504545	0.5703331	13.43026	13.37311	14.004401	14.745635	12.061675	13.412193
rmse	3.6731944	0.0780653	3.6647317	3.6569262	3.7422454	3.8400044	3.4729922	3.6622663

- GBM Base Learner score on test data (public Kaggle leaderboard): **3.752**

The final model is, as mentioned before, a Super (Meta) Learner in form of another GLM model that predicts the target by combining the predictions of the Base Learners. It shows reasonable results and achieves the best metrics for rmse on the training data (refer to table 8). Furthermore, we can state that it generalizes well to unseen data with a final rmse of 3.727 on the test data set.

Table 8: Comparison of results reported on train data

	Stacked Model	Benchmark Model	GLM	DRF	GBM
mae	1.522925	1.603647	1.582077	1.671510	1.541610
mse	12.019943	14.732459	14.657957	14.256069	12.859487
rmse	3.466979	3.838288	3.828571	3.775721	3.586012

- Super Learner score on test data (public Kaggle leaderboard): **3.727**

4.2 Justification

The overall results (refer to table 9) on the hidden test data of the competition can be seen as a confirmation for the quality of the final stacked model. At the moment of writing this report, we achieved a ranking in the top 50% of competitors which is a reasonably good result under the assumption that many of the other participants are using stacked approaches as well in order to train their models.

Table 9: Comparison of results reported on test data (public Kaggle leaderboard)

	Stacked Model	Benchmark Model	GLM	DRF	GBM
rmse	**3.727**	3.921	3.906	3.778	3.752

One remarkable aspect about the chosen model or architecture can also be seen once again for the testing data. Not only can we outperform our benchmarking model with Super Learning but also all of the other models that were used as Base Models. This supports the groundlying concept that each model for its own has advantages and disadvantages over other models. Basically in the data we could find areas where some model is better than its counterpart but there may be different areas where it is the other way around. For this reason stacked approaches can benefit from the diversity of their base models.

5 Conclusion

Finally, we use this last chapter to summarize the most important outcomes that have been produced by our project. A free-form visualization emphasizes interesting characteristics of the modeling results which are hinting to relevant features for predicting customer loyalty. Furthermore, we reflect on the entire process we used for this project and give a brief outlook for upcoming improvements to the model.

5.1 Free-Form Visualization

One of the most interesting parts about models like Gradient Boosted Machines (GBM) is the option to obtain the importance of features or variables from the created model. This is especially helpful in understanding the given problem or data set and also provides an insight how the model predicts the target variable.

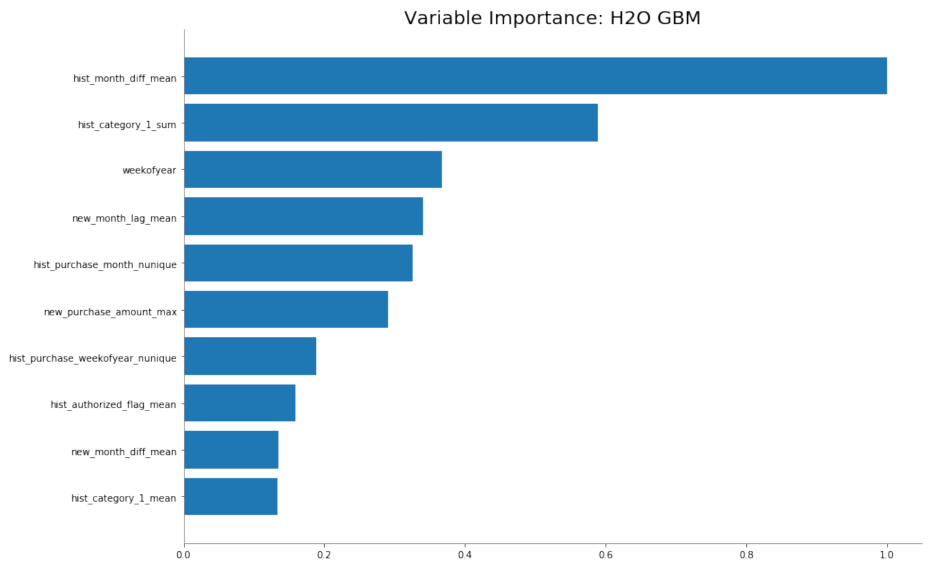


Figure 9: GBM Feature Importance Plot

As can be seen in figure 9, many of the 10 most important features are the ones that have been engineered. The feature with the highest scaled importance value is called "hist_month_diff_mean" and represents the mean of temporal differences (reference date - purchase date in months) for aggregated historical transactions of each card id. We can also see that the week of the year ("weekofyear") in which the customer was first active plays a big role in predicting his loyalty. Some of the features also proof some intuitions that could be made about the problem, e.g. that the mean of transactions that have been approved ("hist_authorized_flag_mean") or also the max purchase amount of new transactions ("new_purchase_amount_max") are highly relevant for the target variable. Although we can find categorical variables like "hist_category_1_sum, -mean", it is very interesting that the top 10 of features is dominated by temporal information.

5.2 Reflection

The process we followed for this project basically consisted out of the six phases shown in figure 10. As a first step the problem needed to be defined and described clearly with a given metric for evaluation. Second, we performed an analysis of the given data for the problem and identified its most important characteristics. Also several visualizations are provided to summarize relevant properties.

Next, many preprocessing steps to clean and transform values have been necessary before we could start to engineer new features for training. Feature engineering was an aspect that has proven to be extraordinary difficult and required a lot of trial and error as well as several days of tuning. In addition, because most of the features in the data had only a generalized naming convention like "feature_1" or "category_3" it was hard to understand the background.

Finally, the most interesting part of the project was the modeling phase followed be the evaluation. The idea of stacked ensembles is still fascinating that by combining different base learners into a super learner we can take advantages from all of them and create a more versatile and flexible model that generalizes well to unseen data. It was very pleasant to see that a super learning model approach is not only in scientific theory a good choice but also proves to be extremely effective in real world applications. Although the presented final stacked model could not quite compete in the kaggle leaderboard with the top models, it was demonstrated that it performs still way better than the benchmarking model and base learners.

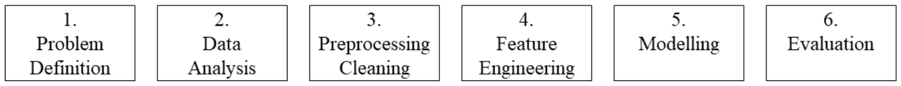


Figure 10: Project Process

5.3 Improvement

As can be seen from the public score of our final model in the Kaggle competition leaderboard, there is still a lot of room for improvements (at the moment top world participants reach RMSE scores around 3.647 and the competition is still ongoing). Although the provided model is able to predict the loyalty score of card owners quite acurately, at least in the following areas of the modeling improvements can be made in the future:

1. **Feature Engineering**: We have seen that especially aggregating new features from the historical transactions and new merchant transaction lead to high differences in the scoring metric. For this reason it would be a good idea to take an even closer look at how to combine and aggregate new features. In addition we have not taken the information about the different merchant ids where to purchase was made into account and could aggregate several other features from this file.
2. **Tuning of the Base Learners**: Until now only one of the Base Learner (GBM) could be tuned sufficiently due to time constraints and computational effort. It would make sense to also apply hyperparameter tuning for all of the other remaining Base Learner. An example is to use a randomized grid search approach to fine tune the relevant parameters
3. **Selection of the Base Learners**: In order to give our SUper Learner the chance to learn from a wide variety of models we can add more Base Learners to the final model. One candidate that has not been included yet is a Deep Learing Algorithm, since it is very difficult to find a proper architecture and fitting parameters. Furthermore beside only adding a single model for each algorithm, it would also be possible to add multiple models of the same class (grid of models with different paramters).
4. **Tuning of the Super (Meta) Learner**: As before also the Super Learner Parameters can be adjusted and tuned e.g. with a grid search approach.
5. **Selection of the Super (Meta) Learner**: In our model we used a Generalized Linear Model for ensembling of the Base Learner, but other models like Gradient Boosting Machines, Random Forest or even Deep Neural Networks are possible. It needs to be investigated if one of these approaches can lead to significantly better results than the present one.
6. **Stratified KFold**: In addition to the previous improvements, a training strategy where the cross validation is not performed on random folds but on stratified folds that preserve the percentage of samples for each class. The target variable of our class is not a category but with some tricks we can include in each fold a fair amount of outliers which will be a better representation for testing the kfolded data.

By applying some of these modelling approaches we are pretty confident that the model can be further improved and also a better ranking in the Kaggle leaderboard is reachable.

References

[1] David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[2] Leo Breiman. Stacked regressions. *Machine Learning*, 24:49–64, 07 1996.

[3] Mark Laan, Eric C Polley, and Alan Hubbard. Super learner. *Statistical applications in genetics and molecular biology*, 6:Article25, 02 2007.

[4] Erin E. LeDell. Scalable ensemble learning and computationally efficient variance estimation. *Doctoral Dissertation*, 2015.

[5] Stacked ensembles. `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html`. Accessed: 2018-12-19.

[6] Gradient boosting machine (gbm). `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/gbm.html`. Accessed: 2018-12-19.

[7] Stacking (super learner algorithm). `https://github.com/h2oai/h2o-meetups/blob/master/2017_01_05_H2O_Ensemble_New_Developments/h2o_ensemble_new_developments_jan2017.pdf`. Accessed: 2018-12-25.

6 Appendix

6.1 Additional Feature plots for test data

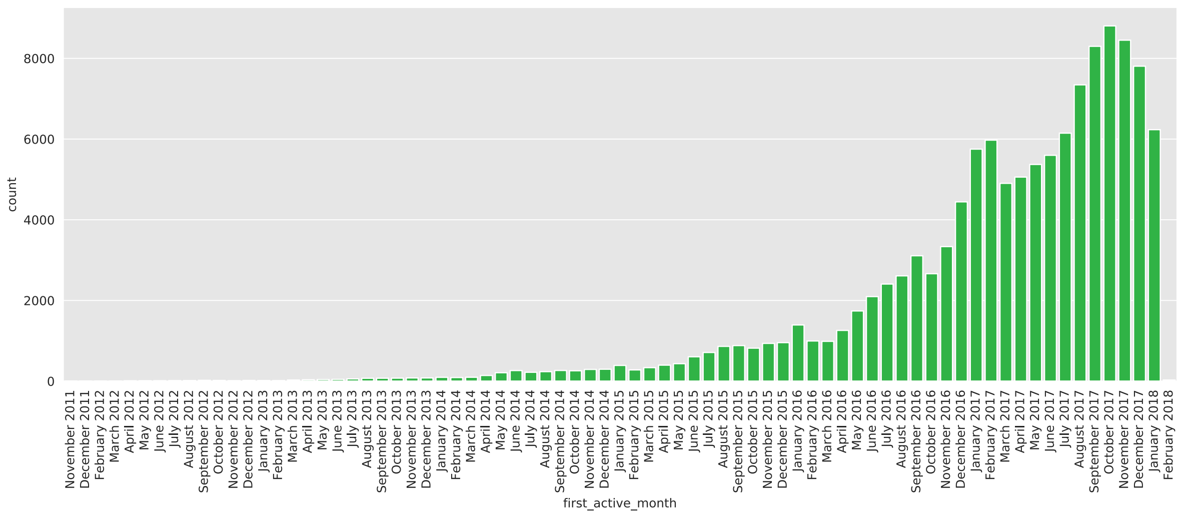


Figure 11: Plot of value count for each unique category: `first_active_month`

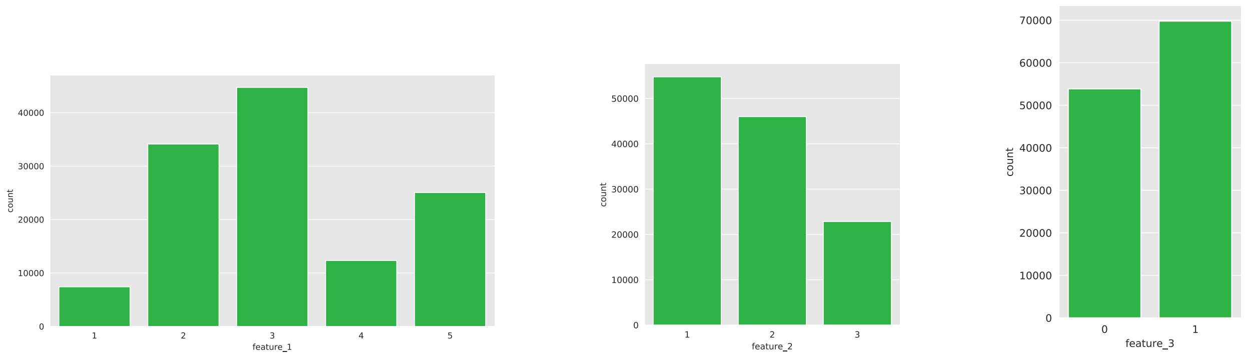


Figure 12: Plot of value count for each unique category: `feature_1`, `feature_2` and `feature_3`

6.2 Violin plot first active month

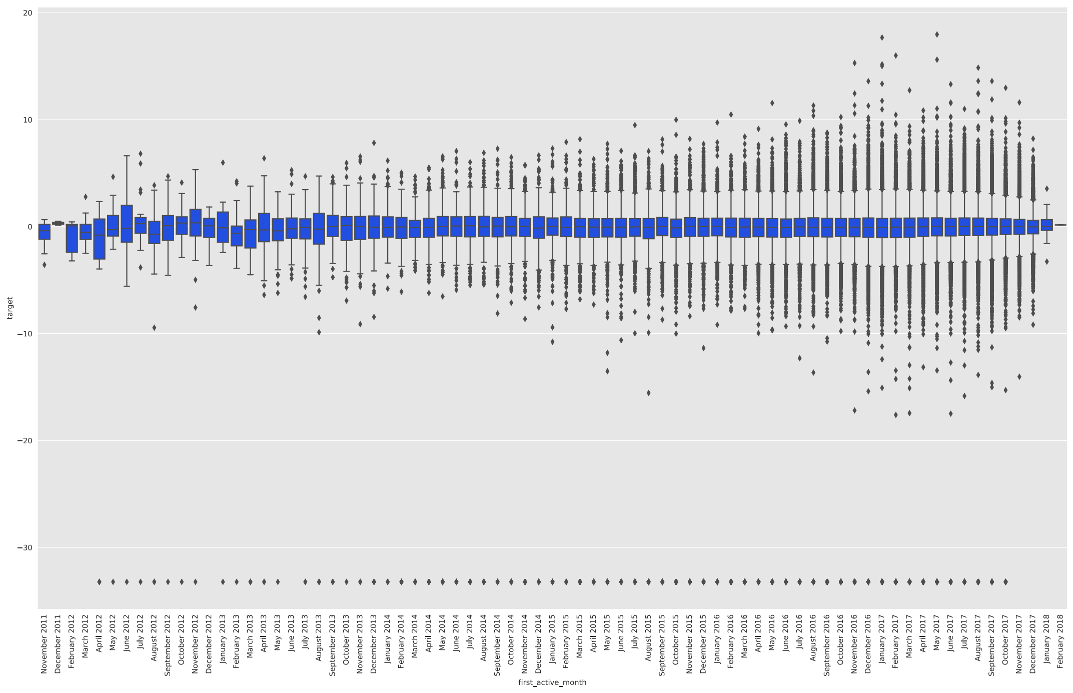


Figure 13: Violin plot for each unique category of target: `first_active_month`

6.3 List of Features

1. feature_1
2. feature_2
3. feature_3
4. hist_purchase_date_max
5. hist_purchase_date_min
6. hist_month_diff_mean
7. hist_card_id_size
8. hist_purchase_amount_sum
9. hist_purchase_amount_max
10. hist_purchase_amount_min
11. hist_purchase_amount_mean
12. hist_purchase_amount_var
13. hist_installments_sum
14. hist_installments_max
15. hist_installments_min
16. hist_installments_mean
17. hist_installments_var
18. hist_month_lag_sum
19. hist_month_lag_max
20. hist_month_lag_min
21. hist_month_lag_mean
22. hist_month_lag_var
23. hist_authorized_flag_sum
24. hist_authorized_flag_mean
25. hist_purchase_weekend_sum
26. hist_purchase_weekend_mean
27. hist_category_1_sum
28. hist_category_1_mean
29. hist_category_2_1_sum
30. hist_category_2_1_mean
31. hist_category_2_2_sum
32. hist_category_2_2_mean
33. hist_category_2_3_sum
34. hist_category_2_3_mean
35. hist_category_2_4_sum
36. hist_category_2_4_mean
37. hist_category_2_5_sum
38. hist_category_2_5_mean
39. hist_category_3_A_sum
40. hist_category_3_A_mean
41. hist_category_3_B_sum
42. hist_category_3_B_mean
43. hist_category_3_C_sum

44. hist_category_3_C_mean
45. hist_purchase_year_nunique
46. hist_purchase_weekofyear_nunique
47. hist_purchase_month_nunique
48. hist_purchase_dayofweek_nunique
49. hist_purchase_hour_nunique
50. hist_subsector_id_nunique
51. hist_merchant_id_nunique
52. hist_merchant_category_id_nunique
53. new_purchase_date_max
54. new_purchase_date_min
55. new_month_diff_mean
56. new_card_id_size
57. new_purchase_amount_sum
58. new_purchase_amount_max
59. new_purchase_amount_min
60. new_purchase_amount_mean
61. new_purchase_amount_var
62. new_installments_sum
63. new_installments_max
64. new_installments_min
65. new_installments_mean
66. new_installments_var
67. new_month_lag_sum
68. new_month_lag_max
69. new_month_lag_min
70. new_month_lag_mean
71. new_month_lag_var
72. new_authorized_flag_sum
73. new_authorized_flag_mean
74. new_purchase_weekend_sum
75. new_purchase_weekend_mean
76. new_category_1_sum
77. new_category_1_mean
78. new_category_2_1_sum
79. new_category_2_1_mean
80. new_category_2_2_sum
81. new_category_2_2_mean
82. new_category_2_3_sum
83. new_category_2_3_mean
84. new_category_2_4_sum
85. new_category_2_4_mean
86. new_category_2_5_sum
87. new_category_2_5_mean

88. new_category_3_A_sum
89. new_category_3_A_mean
90. new_category_3_B_sum
91. new_category_3_B_mean
92. new_category_3_C_sum
93. new_category_3_C_mean
94. new_purchase_year_nunique
95. new_purchase_weekofyear_nunique
96. new_purchase_month_nunique
97. new_purchase_dayofweek_nunique
98. new_purchase_hour_nunique
99. new_subsector_id_nunique
100. new_merchant_id_nunique
101. new_merchant_category_id_nunique
102. year
103. weekofyear
104. month
105. elapsed_time
106. hist_purchase_date_diff
107. hist_purchase_date_average
108. hist_purchase_date_uptonow
109. hist_first_buy
110. new_first_buy
111. card_id_total
112. purchase_amount_total