AML Project 2 report

Adam Czerwoński, Adam Narożniak, Jędrzej Ruciński

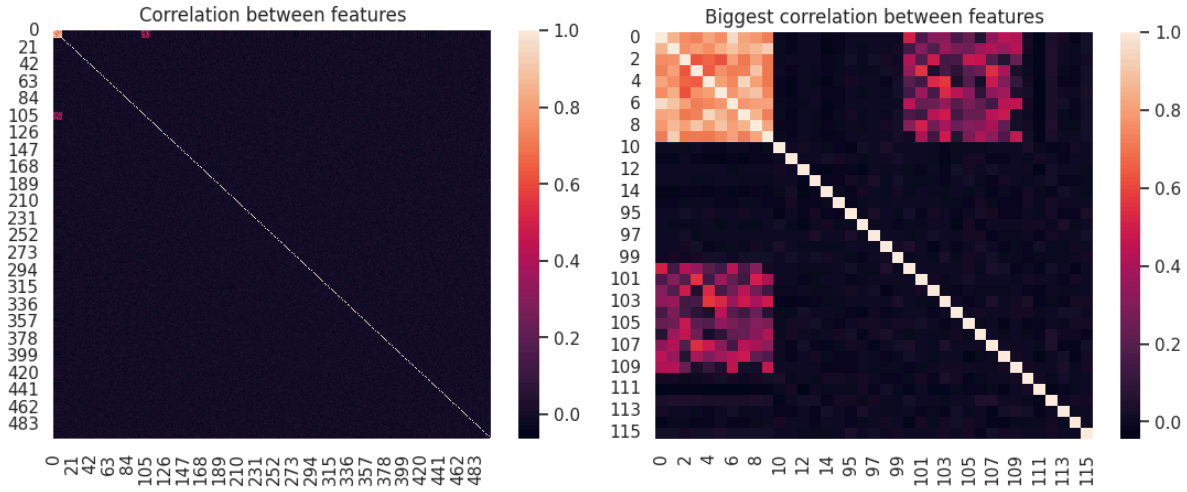
Methodology

This project sets out to tackle an intriguing task, which aims to represent a simple real-life scenario. Very often in business scenarios we are faced with a challenge in which we need to maximize customer outreach but minimize the number of resources used for the task.

We are given a training set and a test set both representing data about 5000 clients. Each of these data sets consists of 500 predictor variables along with a y-label vector for the training set. We must use the training set to create a model that will allow us, with good certainty, to select 1000 out of the 5000 customers in the training set as potential users of our fictional offer. For each selected customer that takes advantage of the offer, we are paid 10 euros. This means that there is a potential 10000 Euro for us to earn. The catch, however, is that for every variable we use to create our model, we must pay 200 euros. This creates a very interesting function to optimize which will be the basis of our modeling.

Data Exploration and Analysis

In order to get a better understanding of the data that we are dealing with we need to take a closer look at the training features. As a start, we look at the pairwise correlations between all features. We see that there is no apparent correlation between the variables apart from a few small groups at which we take a closer look here.

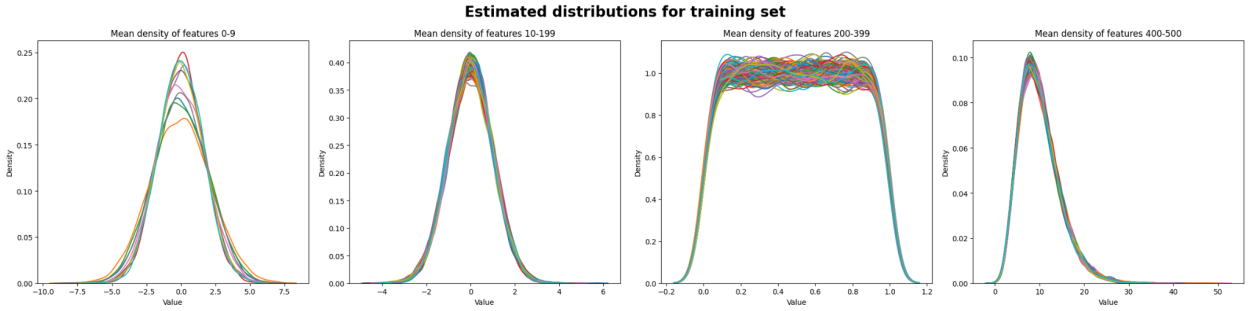


We may infer from this that features 0-9 have a strong correlation with each other along with a

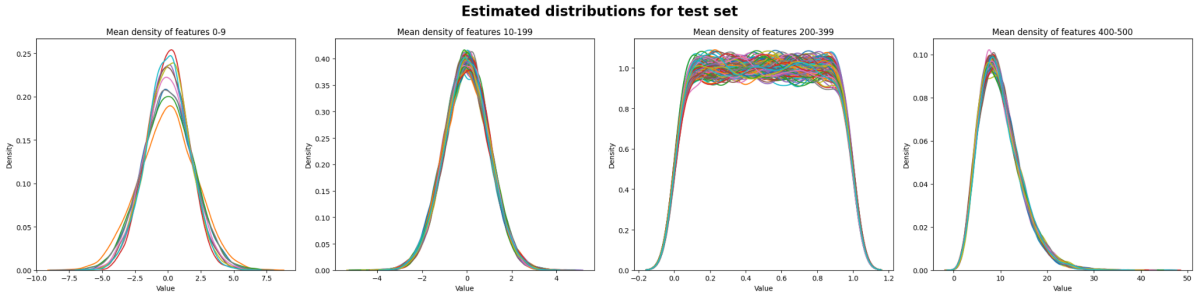
slightly weaker, but still visible, correlation with features 100-109. All of these features, on the other hand, show no visible correlation to the target variable y.

After this, we decided to estimate the distributions of features in both the training and test sets. For this, we used kernel density estimation and we can split the features into four groups based on their estimated distribution.

Let us start with the training set, and show the KDE plot for all features.



From this, we may infer a normal distribution for the first 200 features (with a different density for the first 10 features. The second 200 features present a uniform distribution. The final 100 features present a distribution that could have been normal, but has been slightly skewed towards the left, so it is difficult to assume its distribution type. Now, for the most important part, we need to see if the test data set presents a similar distribution.



As we may see, everything aligns with the training set, which allows us to conduct experiments on the train data set that will be representative of the test data set and allows us to take a more informed approach.

Objective function and model evaluation

Because of the previously specified way of calculating the profit of our campaign, we have come up with our own objective function. We have used this function for evaluation of all of our models and feature selection methods. It's defined by the following formula:

$$profit = c \times 10 - 40 \times p$$

where p is the number of features used and c is obtained by firstly taking 200 observations that our model classified as positive with the highest probability and then checking how many of them were predicted correctly. In the end, our model was supposed to output the 1000 most likely to be positive observations from the test set with 5000 rows. When we were doing cross-validation with 1000 observations in the validation set we wanted to preserve this ratio and that's why we used 200 observations for calculating c and scaled the cost of features to 40. It's important to point out that this is not our predicted score on the test set, we only use this function for comparing different models.

Feature + Model selection

For feature selection combined with model evaluation, a custom grid search with 5-fold cross-validation was designed. In this way, we were able to measure mean performance and standard deviation on all pairs of feature selectors and models (all with customizable parameters).

For feature selection methods we considered: selection based on the importance of parameters (SelectFromModel class in scikit-learn) - LogisticRegression, Random Forest; and secondly, we considered ANOVA F-value and mutual information (f_classif and mutual_info_classif). We used this method and parameterized the number of features that we wanted to keep.

The models we considered in this grid search were: RandomForest, DecisionTree, LogisticRegression, SVC, LDA, QDA, and NaiveBayes (some of the parameters of these methods were treated as variables too).

After getting these results, we were able to choose the most promising methods and consider more hyperparameters of these specific methods. The best methods were: Random Forest as the feature selector and the QDA, NaiveBayes, and SVC as the models.

We were also able to investigate if there are common sets of features that were giving the best results. Our findings were that the best results were obtained using the feature number [100, 102, 103, 105] and [100, 102, 105].

In the final step, we decided to check both of these sets with custom ensembles of the methods that produced the most promising results.

	RF	DT	LogReg	LDA	QDA	SVC-rbf
SelectFromModel(estimator=LogisticRegression(), max_features=3)	868.000000	846.000000	900.000000	902.000000	886.000000	842.000000
SelectFromModel(estimator=LogisticRegression(), max_features=5)	794.000000	768.000000	824.000000	824.000000	810.000000	808.000000
SelectFromModel(estimator=LogisticRegression(), max_features=10)	660.000000	610.000000	650.000000	646.000000	638.000000	608.000000
SelectFromModel(estimator=LogisticRegression(), max_features=20)	234.000000	166.000000	216.000000	216.000000	178.000000	214.000000
SelectFromModel(estimator=LogisticRegression(), max_features=30)	-202.000000	-232.000000	-158.000000	-160.000000	-164.000000	-184.000000
SelectFromModel(estimator=RandomForestClassifier(), max_features=3)	1206.000000	972.000000	1042.000000	1046.000000	1356.000000	1334.000000
SelectFromModel(estimator=RandomForestClassifier(), max_features=5)	1246.000000	920.000000	964.000000	958.000000	1340.000000	1340.000000
SelectFromModel(estimator=RandomForestClassifier(), max_features=10)	1058.000000	768.000000	708.000000	684.000000	1166.000000	1128.000000
SelectFromModel(estimator=RandomForestClassifier(), max_features=20)	676.000000	356.000000	258.000000	274.000000	592.000000	618.000000
SelectFromModel(estimator=RandomForestClassifier(), max_features=30)	226.000000	-52.000000	-210.000000	-190.000000	122.000000	128.000000

XGBoost and LightGBM with Hyperopt

XGBoost and LightGBM are very popular and high-performing models for classification tasks. The biggest challenge when working with them is choosing the right set of hyperparameters. If we were to consider a large range of them, then a simple grid search would take far too much time. That's why we have used a Python package called Hyperopt that uses the Tree of Parzen Estimators algorithm for minimizing a user defined function, which was the profit defined earlier in this report.

Both of those models were trained on two feature sets - [100, 102, 103, 105] and [100, 102, 105].

Ensemble

Our last attempt to improve the profit was ensembling the best models and using majority soft voting for the predictions. From the previous tests, five models stood out - NaiveBayes, QDA, SVM with Gaussian kernel, XGBoost, and LightGBM. Because it was still not clear which of the two feature sets mentioned before was better we have decided to test them both. We have tried all of the possible combinations of the best models. Below are the profits of the best five ensembles.

ensemble	profit
SVM + QDA + NB	1382
QDA	1380
NB	1378
SVM + QDA	1376
QDA + NB	1374

features 100, 102, 105

ensemble	profit
LGBM + NB	1390
LGBM + NB + QDA	1384
NB + QDA	1384
LGBM + NB + QDA + SVM	1380
LGBM + NB + SVM	1380

features 100, 102, 103, 105

Naive Bayes combined with LightGBM gave the best results and were used for our final predictions on the test set.