# Trial And Error Is All You Need

**Jorge Vančo Sampedro**
202200473@alu.comillas.edu
Ingeniería Matemática e Inteligencia Artificial ICAI
Madrid

**Abstract**

The FICO dataset contains information about Home Equity Line of Credit (HELOC) applications made by real homeowners. A HELOC is a line of credit typically offered by a bank as a percentage of home equity (the difference between the current market value of a home and its purchase price). The fundamental task is to use the information about the applicant in their credit report to predict whether they will repay their HELOC account within 2 years.

## Exploratory Data Analysis

The used FICO dataset has the following input variables:

- ExternalRiskEstimate: A measure of borrower's riskiness based on consolidated external data sources.
- NetFractionRevolvingBurden: The proportion of an individual's current credit usage compared to their maximum allowed credit.
- AverageMInFile: The average duration, in months, of the trades in a borrower's credit file.
- MSinceOldestTradeOpen: The age, in months, of a borrower's oldest credit account.
- PercentTradesWBalance: Percentage of a borrower's trades that have balance.
- PercentInstallTrades: The percentage of a borrower's credit accounts that have fixed payment terms over a specified period.
- NumSatisfactoryTrades: Count of trades where a borrower has met obligations satisfactorily.
- NumTotalTrades: Number of Total Trades (total number of credit accounts).
- MSinceMostRecentInqexcl7days: Months since the last credit inquiry, ignoring the most recent week.
- PercentTradesNeverDelq: The percentage of a borrower's trades with no history of delinquency.

And the output variable:

- Risk Performance: Paid as negotiated flag (12-36 months). Class variable: 0 ('Good') or 1 ('Bad').

Firstly, the special cases where handled. Some columns had some special values assigned to represent the absence of data. The rows that had -9 where discarded because the entire row was made out of -9. However, it was not so clear what to do with the rows that had -7 or -8. Those values where turned into na values in order to impute them later on. It was not clear if those values would be important in making decisions later on, so one-hot encodings of those values were also stored for making comparisons.

The dataset feature distributions where studied to determine the preprocessing to use for each one of them. None of the variables showed to follow a normal distribution, most of them where skewed. Therefore, the box-cox transformation was used on NetFractionRevolvingBurden, AverageMInFile, MSinceOldestTradeOpen, PercentInstallTrades, NumSatisfactoryTrades and NumTotalTrades to make them more normal distributed, using the Anderson-Darling test to fit the best $\lambda$ parameter for the box-cox transformation. The box-cox transformation was chosen over the Yeo-Johsnon transformation because the dataset doesn't contain negative values, other than the special values that where discarded. To prevent problems when the columns have $0$s in them, $0.001$ is added to them if their minimum value is $0$.

At first, a dataset was made by scaling all the columns to being in the 0-1 range, another one was created by scaling every column and others were made out of a mixture of all due to the initial thought of scaling the percentages columns being the best option. At the end, it was decided to standarize all the columns because some models where unaffected and others, such as logistic regression or k-nearest-neighbors, improved. This makes sense because, when standarized, the distance between two points means the same in every feature.

Lastly, the knn imputer from sklearn was used to impute the missing values. A linear regression was first used to impute the values between the columns NumSatisfactoryTrades and NumTotalTrades due to their high correlation, of about 0.87. However, the knn imputer gave better results. To select the number of neighbors used, cross validation was performed. A custom knn imputer was coded to impute one column based on the others (as it could not impute rows with two missing values, the model was not used for imputation). With cross validation for k $\in \{5, 10, 20, 50, 100\}$ (the fitting process was very slow, so only this few values

of k were chosen), the number of neighbors for each column where $50, 100, 10, 10, 10, 50, 10, 5, 20, 20$, with a mean value of $28.5$. Instead of choosing 28 or 29, the value of 20 was chosen because it was one of the values used in the fitting process.

To study the interactions between variables, a linear regression was fitted to predict each column. The residuals where then plotted to search for non-linearities in them. No non-linearities where found, all residuals seemed to follow an almost normal distribution. The only structures found where linear structures when trying to predict the output variable, RiskPerformance. This structures seem to be caused by the discrete values the output variable takes.

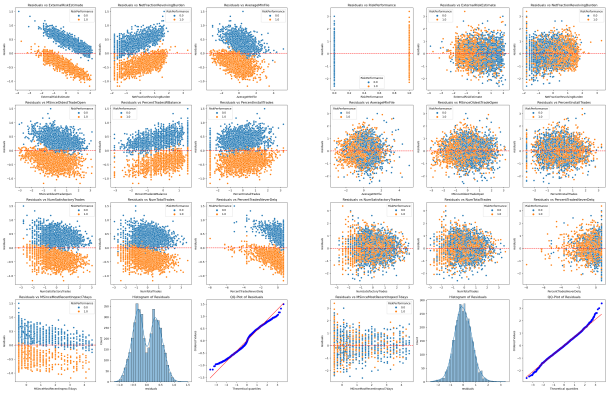

Figure 1: Residuals RiskPerformance.



Figure 2: Residuals PercentTradesWBalance.

To perform all this transformations and cleaning procedures, Pipeline and PipelineCV classes was programmed. Pipeline is used to divide the dataset into train and test, but it can also be used for the cleaning process. PipelineCV was specifically created for the cleaning process. It is mainly used during cross-validation, which will be further discussed later on.

## Hyperparameter selection

An extensive random grid search was performed to choose the best fitting hyperparameters for each model. Each model was trained and evaluated using 10-Fold cross-validation, with accuracy being the metric used to evaluate the models. To ensure no data-leakage, PipelineCV was used to clean and transform the data on each fold. This data-leakage was not considered when the cross-validation process started, so over $5200$ models had to be discarded due to this mistake. Every single model configuration was saved on a MongoDB database in order to visualize the best performance regions on the grid with the purpose of narrowing down the grid dimensions. The custom function used for the parameter search gets a dictionary of every model it has to train. Each model contains the range of possible values the hyperparameters can take.

In addition to the model hyperparameters, the datasets used also formed part of the random grid search, so as to determine the best method of cleaning the data for every model (e.g. random forest does not require as much preprocessing as logistic regression). In Figure 3, the accuracies obtained for each dataset are presented. To prevent really bad model hyperparameters from skewing this comparison, only the readings with higher accuracy than 0.70 are taken into account.

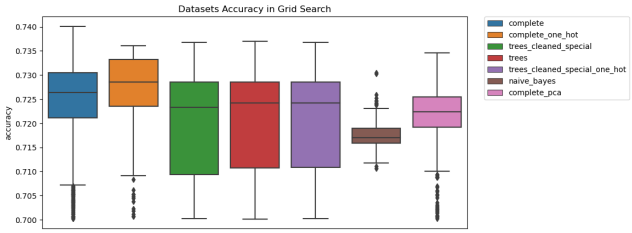


Figure 3: Accuracy obtained for each dataset

It can be seen that the datasets that are standarized (the ones that have complete in it) outperform the rest of the datasets. The datasets that have trees in their name are the ones that are neither standarized nor scaled. This is because tree based models do not require the distances between variables to be the same to work properly. Another important difference is that the datasets that have one_hot in their names have one-hot encodings of the special values described earlier. The fact that the complete_one_hot dataset is the best performing one out of the ones shown suggests that there is some kind of information that this absence of values is giving. However, the best values of accuracy where still given by the complete dataset, hinting that, in reality, they do not really give new information. Lastly, the complete_pca dataset was created by the most relevant pca components, the details are explained in the next section.

Before diving deeper into the models trained, it is of great help to talk briefly about the unsupervised learning techniques used.

## Unsupervised Learning

To understand the dataset, clustering and PCA was performed.

### PCA

Principal Component Analysis was used for reducing the number of columns in the dataset and understanding the data. As shown in Figure 4, with just 6 principal components, $91\%$ of the variance can be explained.

The number of components was chosen using the elbow method. It can be seen in Figure 4 that the increase of variance explained decreases after the first 6 principal components. This 6 components where used to create the $complete\_pca$ dataset.

The first two principal components are shown in Figure 5. It can be seen how the main characteristics of the customers ExternalRiskEstimate, NetFractionRevolvingBurden, AverageMInFile, MSinceOldestTradeOpen, PercentTradesWBalance, NumSatisfactoryTrades, NumTotalTrades.

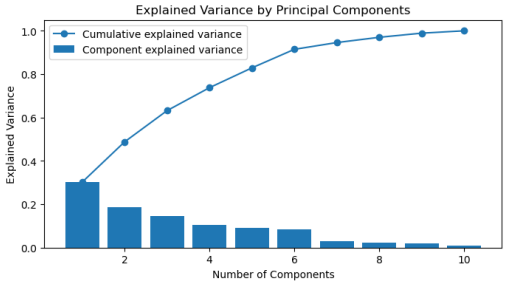


Figure 4: Explained variance by number of PCA components

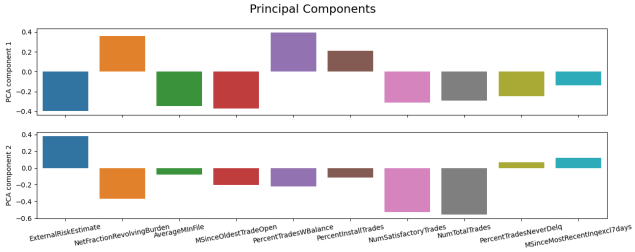


Figure 5: Two first Principal Components

Clustering was performed with KMeans, Gaussian Mixture Model and Hierarchical clustering. A KMeans class was coded for this experiment, it is worth mentioning that, for the sake of maximizing the distance between the cluster centroids at their initialization, the logarithm of the distance was used to prevent them from choosing points near one of the previous centroids. The furthest point from the clusters was the point with the largest sum of the logarithms of the distances to the centroids, which was then chosen as the next new cluster centroid.

It is interesting to note how the division is mainly made along the first principal component axis, as shown in Figure 6.

The number of clusters for KMeans was chosen using the silhouette. For the Hierarchical clustering, the largest height difference was used to determine the best cut. As the biggest cut was the last one (Figure 7), and KMeans had 2 clusters, 2 clusters where also used for Hierarchical clustering.

To see how well the clusters represent the target values, the confusion matrices of Figure 9 were made. The simple frontier the KMeans algorithm establishes does a great job dividing the dataset, having over 70% of accuracy on the training set. It is surprising how the Gaussian Mixture Model, even though it has the most similar separation to the original dataset, has the worst accuracy.

For the Gaussian Mixture Model, cross-validation was performed to obtain the best number of clusters. The metric used was BIC. Models with lower BIC are better than with larger BIC. The obtained number of clusters was 6; however, for visualization purposes, two clusters were used. Those 6 clusters are the ones that represent the entire data density the best. Those would be used for data generation.

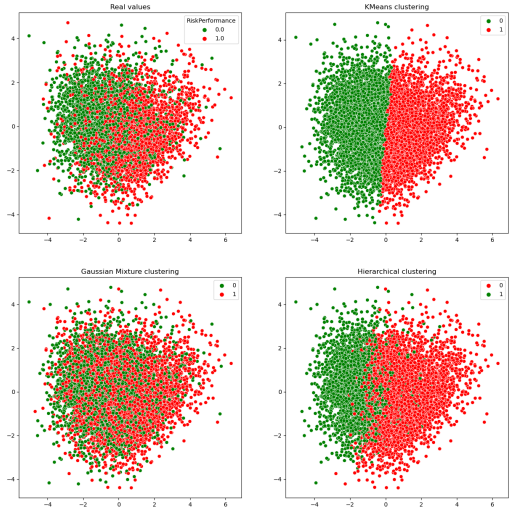


Figure 6: Clusters projected onto two Principal Components

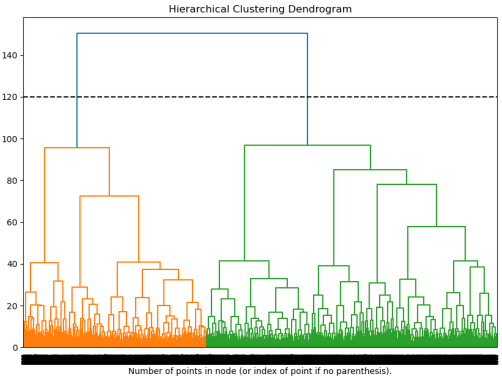


Figure 7: Hierarchical clustering

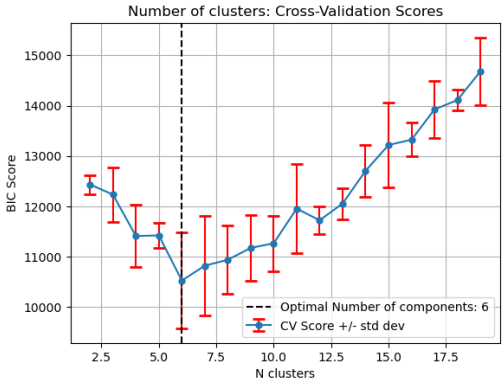


Figure 8: Gaussian Mixture Model Cross-validation

In this case, as the dataset has discrete variables and another model of some kind or some discretization technique would be needed, no data generation was performed.

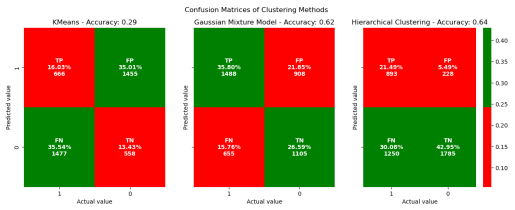


Figure 9: Confusion Matrices of Clustering methods

## Supervised Learning

Trying to predict the output variable is a relevant task in order to comprehend the dataset, this is why the use of supervised learning models was performed.

The chosen models for the experiment were the ones studied during the semester and some other new ones. Next, greater details are going to be given of each model.

### Logistic Regression

Even though it is a simple model, logistic regression did really well on this dataset.

Using lasso regularization, variable importance can be determined. It can be seen how ExternalRiskEstimate and NetFractionRevolvingBurden are the most important ones, as they are the last ones to dissapear and the ones with the higher absolute value. The least important features are PercentInstallTrades. The influence of each variable in the output of the model is explained later on.

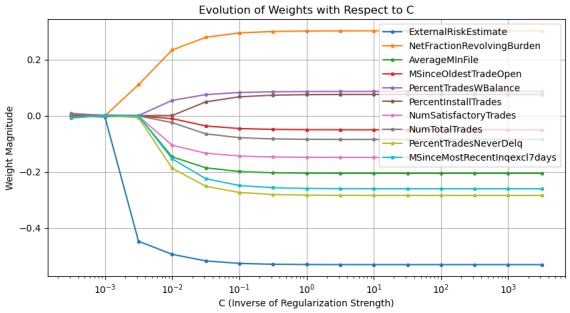


Figure 10: Lasso regularization on Logistic Regression

When plotted the weights of the best performing logistic regression with the first principal component, one can see that they are not identical. If they were, this would mean that the separating hyperplane is orthogonal to he first principal component. The differences shown in Figure 11 suggest that the hyperplane is defined by another vector, which surely contains information from the rest of the principal components.

### KNN

The hyperparameters used in the grid search were:

- k: The number of nearest neighbors used for the predictions.

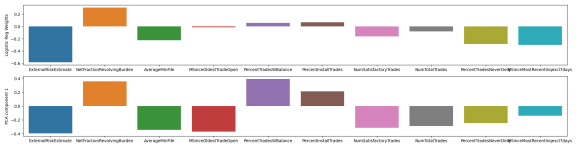

Figure 11: Logistic Regression weights and first PC

- p: The parameter p of the Minkowski distance. [1]

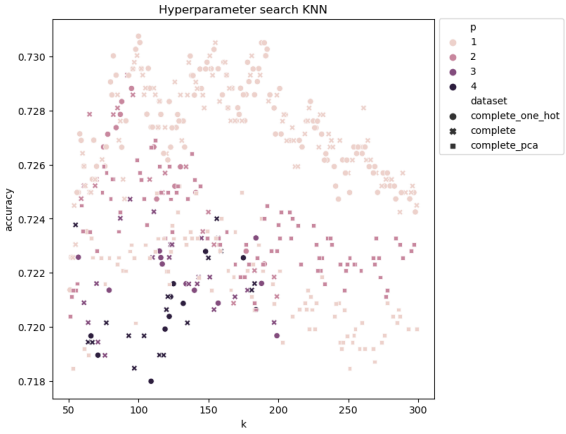


Figure 12: Accuracy based on k

Over 550 models of KNN were trained to determine that the best value of k and p are 100 and 1 respectively. As it can be seen in Figure 12.

### Decision Trees

A single decision tree was tested, but it performed poorly. Therefore, four different ensemble models were used: Random Forest, Gradient Boosting, Adaboost, XGBoost.

Random forest did really well. Looking at the variable importances in Figure 13, it can be seen that ExternalRiskEstimate and NetFractionRevolvingBurden are the most relevant features. And again, the least important features are NumTotalTrades and PercentInstallTrades.

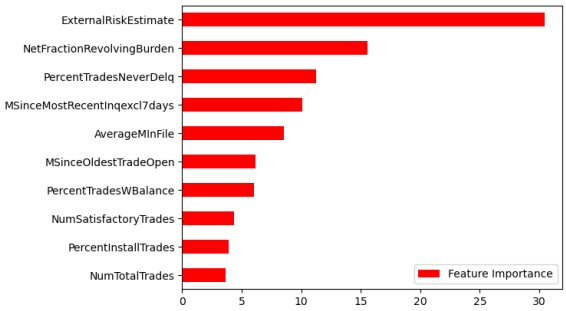


Figure 13: Feature importance on Random Forest

[1] Minkowski distance: $dist(x, y) = (\sum_{i=1}^{d} \mid x - y \mid^{p})^{\frac{1}{p}}$

Gradient Boosting and Adaboost did well, but not outstanding. However, the better version of gradient boosting, XGBoost, performed extraordinary.

Even though these tree methods do not require the data to be standarized, both XGBoost and Random Forest gave their best performance with the standarized dataset.

Comparing the variable importances from Random Forest (Figure 13) and XGBoost (Figure 14), it can be seen how ExternalRiskEstimate is the most important on both; however, the rest of the values change in order, though the relevance is still very little compared to the ExternalRiskEstimate.

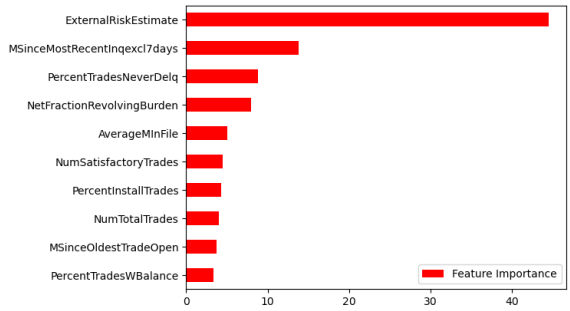


Figure 14: Feature importance on XGBoost

Using the shap values, the feature influences can be studied (Figure 15). It can be seen how higher ExternalRiskEstimate leads to lower RiskPerformance, and a lower value leads to a higher value in the output. This means that the higher value of ExternalRiskPerformance a customer has, the least likely that the customer is not going to pay. This is counter-intuitive to what someone would think, as having a bad estimate would suggest that the customer is problematic. This is specially concerning due to ExternalRiskEstimate being the most important feature. An explanation to this behaviour could be that the borrowers that have a higher ExternalRiskEstimate are more cautious because of the consecuences they had to face in the past.

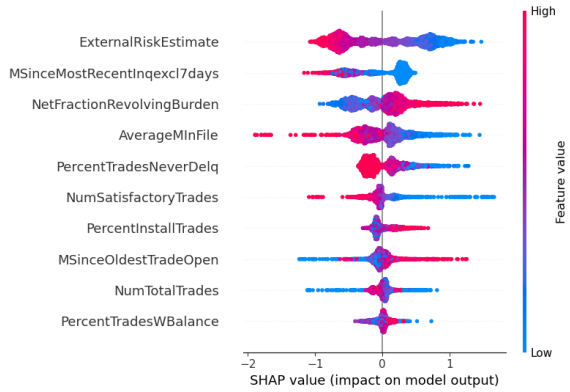


Figure 15: Shap values on XGBoost

Having higher NetFractionRevolvingBurden or MSinceOldestTradeOpen lead to higher risk. Whereas higher values of AverageMInFile, PercentTradesNeverDelq or NumSatisfactoryTrades lead to lower risk.

## Support Vector Machines

SVMs with linear, polynomial and rbf kernels were trained. Polynomial SVMs tended to take too long training and did not perform greatly, that is why not many of them were trained.

For the machines with linear kernel, only the hyperparameter C was searched for. In contrast, for the models with RBF kernel, the best gamma[2] hyperparameter had to be found too.

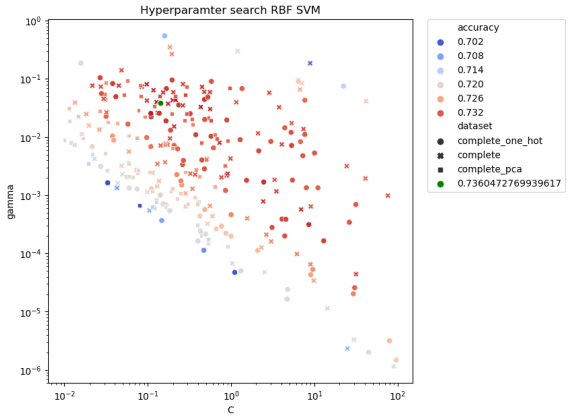


Figure 16: Hyperparamter search space for RBF SVMs

## Generative models

This section is composed by LDA, QDA and Naïve Bayes. These models work with the probability distributions of the data. Even though they are not the best performing models, they are a good way of establishing a base performance. All three models were programmed from scratch. In addition, Naïve Bayes was extended to approximating data distributions that do not seem to be normally distributed with bins. Neither LDA nor QDA have hyperparameters to search for; however, Naïve Bayes has alpha[3], which is responsible of smoothing the distributions and avoids assigning 0 probability to never seen events, and the anderson_statistic_threshold, which decides whether or not a distribution is normally distributed according to the Anderson-Darling test.

## Artificial Neural Networks

Artificial neural networks where designed and trained with the objective of getting the best performance on this dataset.

Firstly, a Multilayer Perceptron (MLP) was trained. The best model architecture and training hyperparameters were found using cross-validation. As the dataset has very little data points, it was seen that very little epochs were needed for the networks to overfitt the data. This is why the epochs where also a variable during cross-validation, in case early stopping was needed.

[2]The hyperparameter $\gamma = \frac{1}{\sigma^2}$ in the RBF kernel

[3]Alpha equal to 1 corresponds to Laplace Smoothing

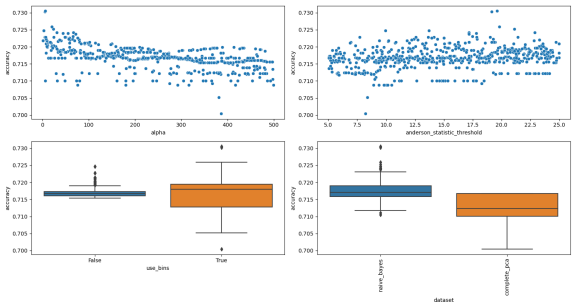

Figure 17: Hyperparamter search for Naïve Bayes

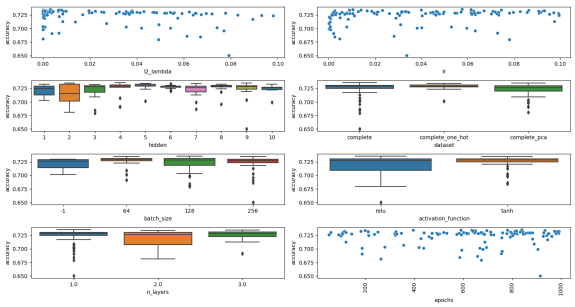

Figure 18: Hyperparamter search for MLP

The model did not perform remarkably; however, it performed consistently throughout the hyperparameter search space, which could indicate that a greater search is required.

To improve this base performance, two modified NNs were designed based on the Nearest Neighbors principle. The first model, called Neighbor NN, takes the n closest neighbors and concatenates their features, along side their value tag, to the feature vector of the input. This extended vector is then passed to a fully connected neural network. The added complexity led to it performing worse than the base MLP, this led to a second iteration of changes.

The training of this variations was complex. It had to be ensured that, during training, the point used for inference was not taken into account as a neighbor. This was done by skipping the neighbor with distance equal to zero.

The second idea was to compress all the information coming from the nearest neighbors. This architecture was inspired by Graph Neural Networks (Figure 19). The model takes the n nearest neighbors, compresses their information using a fully connected layer and sums the resulting vectors. The output of that is then concatenated to the feature vector of the input and passed down to a FCNN.

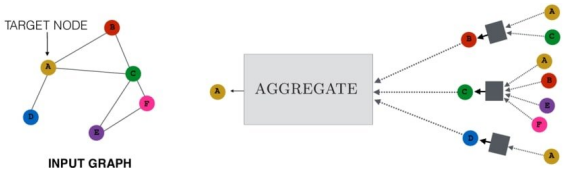


Figure 19: Image of GNN from CS224W Stanford Course

An experiment was done training the model compressing the information into two dimensions. The result of applying the fully connected layer to all the dataset is shown in Figure 20. Note that this does not learn to separate based on the inputs because information of the tag is given.

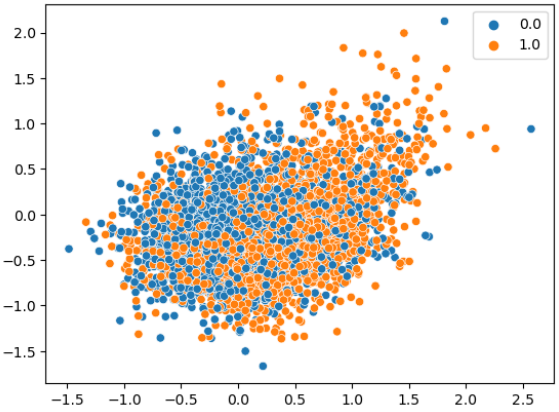


Figure 20: Image of compressed dataset using Compressed Neighbor NN

The same can be seen in Figure 21, where a deep version of NeighborNN was used. However, there was not enough time to perform cross-validation on this version of the model.

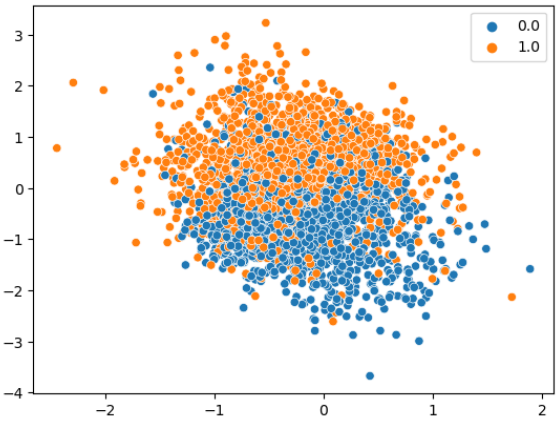


Figure 21: Image of compressed dataset using Deep Compressed Neighbor NN

When the cross-validation process was not done properly, that is to say, data-leakage was happening because the validation set was standarized before the split, this model outperformed the base NN. However, when that was fixed, it did not manage to get as good results as the base architecture. This implies that with more data, where the mean and variance estimates would be more robust and precise, this model could perform better than the other ones. This is just speculation however.

With all this models, the simpler versions with less parameters performed best.

| Models | Accuracy CV | Precision CV | Recall CV | Specificity CV | F1 Score CV | Test Accuracy |
|---|---|---|---|---|---|---|
| Logistic Regression | **73.60 ± 0.71** | 74.10 ± 0.86 | 74.98 ± 0.88 | 72.15 ± 0.75 | 74.52 ± 0.78 | **73.82** |
| KNN | 73.07 ± 0.83 | 73.98 ± 1.08 | 73.66 ± 0.82 | 72.48 ± 0.94 | 73.80 ± 0.90 | 73.53 |
| Decision Tree | 71.17 ± 0.56 | 72.28 ± 1.03 | 71.77 ± 1.35 | 70.49 ± 1.64 | 71.90 ± 0.73 | 70.74 |
| Random Forest | 73.36 ± 0.84 | 73.16 ± 1.01 | 76.25 ± 0.94 | 70.27 ± 0.95 | 74.66 ± 0.92 | 73.53 |
| Gradient Boosting | 72.86 ± 0.76 | 72.82 ± 0.93 | 75.39 ± 1.07 | 70.11 ± 0.86 | 74.06 ± 0.91 | 73.05 |
| Ada Boost | 73.31 ± 0.83 | 74.27 ± 1.00 | 73.75 ± 0.99 | 72.75 ± 1.13 | 73.99 ± 0.91 | 72.67 |
| XGBoost | **74.01 ± 0.80** | 73.52 ± 1.02 | **77.47 ± 0.81** | 70.34 ± 0.99 | **75.43 ± 0.85** | 73.44 |
| Linear SVM | 73.53 ± 0.68 | 73.66 ± 0.87 | 75.71 ± 0.69 | 71.21 ± 0.80 | 74.66 ± 0.73 | 73.63 |
| RBF SVM | 73.60 ± 0.75 | 73.42 ± 0.96 | **76.46 ± 0.79** | 70.58 ± 0.90 | **74.89 ± 0.79** | 73.44 |
| LDA | 73.12 ± 0.73 | **74.60 ± 0.92** | 72.61 ± 0.72 | **73.69 ± 0.91** | 73.57 ± 0.74 | 73.05 |
| QDA | 73.22 ± 0.81 | **74.98 ± 1.01** | 72.13 ± 0.86 | **74.39 ± 0.98** | 73.51 ± 0.85 | 73.44 |
| Naïve Bayes | 73.05 ± 0.70 | 73.19 ± 0.95 | 75.28 ± 0.66 | 70.67 ± 0.90 | 74.20 ± 0.76 | 72.76 |
| MLP | 73.22 ± 0.98 | 73.42 ± 1.07 | 75.33 ± 1.12 | 71.02 ± 1.01 | 74.33 ± 1.00 | **74.11** |
| Neighbor NN | 73.24 ± 0.69 | 73.65 ± 0.87 | 74.89 ± 0.81 | 71.51 ± 0.81 | 74.24 ± 0.74 | 73.34 |
| Neighbor Compressed NN | 73.36 ± 0.61 | 74.26 ± 0.97 | 74.06 ± 0.77 | 72.69 ± 0.96 | 74.11 ± 0.66 | 72.67 |
| Model Ensemble | 73.60 ± 0.75 | 73.42 ± 0.96 | **76.46 ± 0.79** | 70.58 ± 0.90 | **74.89 ± 0.79** | |

Table 1: Models Metric values

### Model Ensemble

Using the best performing models, a model ensemble was created to see if it could get higher performances, with the hope of some models correcting the errors of others. This model uses the hyperparamters obtained from the random grid search and also uses the best pipeline for each model. Cross-validation was performed to select the number of models to use in the ensemble. The starting point was 2 models; however, the decreasing trend and the fact that it performed worse than the best performing one.

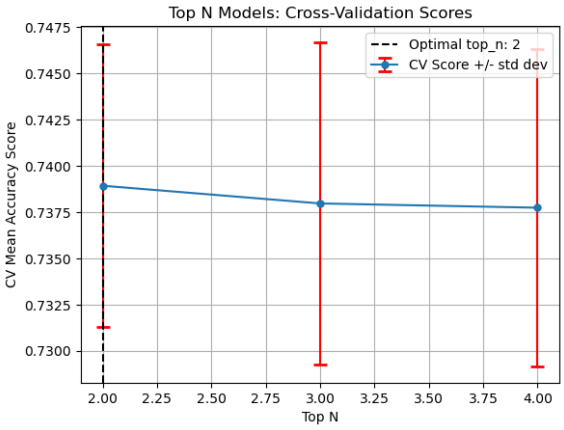


Figure 22: Cross-validation on Model Ensemble

## Model results

After training over 7770 models properly, the result is that the best version of each model lie in just 1% difference of accuracy, except for the decision tree, which is around 2 to 3% lower.

With the model hyperparameters chosen, cross-validation for each model was again performed. But this time, more metrics were stored for each fold. This way, a mean of each metric can be shown. With all this metrics, a more general view of the models can be taken into account, and it enables more flexibility on the model choice depending on the desired behaviour.

XGBoost is the best performing model out of the selected ones. It has the highest value in three out of the five metrics chosen to represent. However, one might choose to use QDA if the objective is to maximize de Specificity, as a bank might want to do to reduce the number of borrowers who do not pay back properly.

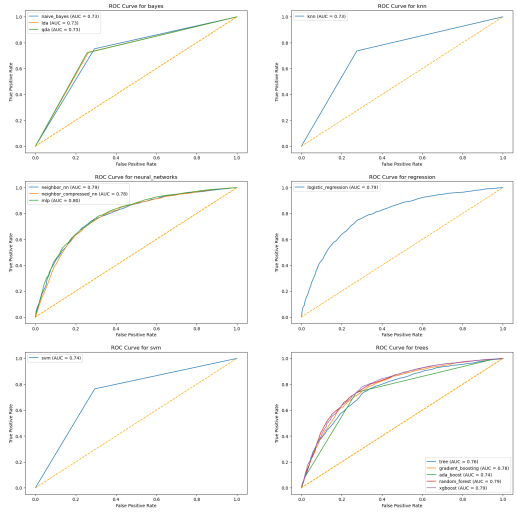

Figure 23: Model mean ROC curves in cross-validation

## Inference

To use the selected model, inference.ipynb was created. It loads the pretrained model and the pipeline, performs the transformations to the data and displays the inference. Shap

was used for the visualization and explanation of the resulting inference, as shown in Figure 24. It can be seen, that for this example, many variables push the output to be low, whereas there are not so many features that indicate the contrary.

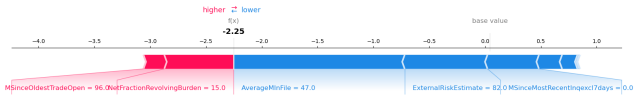


Figure 24: Inference example

## What to improve

Even though I am very proud of the work I have done, there are some mistakes that I want to correct. First of all, only the accuracy of the model was stored when performing grid search, it would have been much better to store all the metrics. This would have made it easier to analyze the results and start making better conclusions early on. In hindsight, a deeper study of the model hyperparameters should have been done, as there were times where no progress was made due to not searching correctly. For example, the training of SVMs was stopped because, with large Cs, they got stuck.

Performing train test iterations on various seeds to get more robust test accuracies. However, the single test accuracy reading would be left, as it is the most realistic metric of the model, as the model was not biased towards that test set.

Comparing PCA against LDA in data reduction would have been a good idea, as it is known that LDA reduces better the dataset when it comes to separating the data points.

Being more organised with the code would have made much more simple the process and would have made it possible to iterate much faster and try new things with ease. The beginning was very slow because I had no idea of what to do with the dataset. Having the cleaning process in one jupyter notebook made it impossible to iterate over ideas. It was not until PipelineCV was programmed that I started making progress.

# Appendix

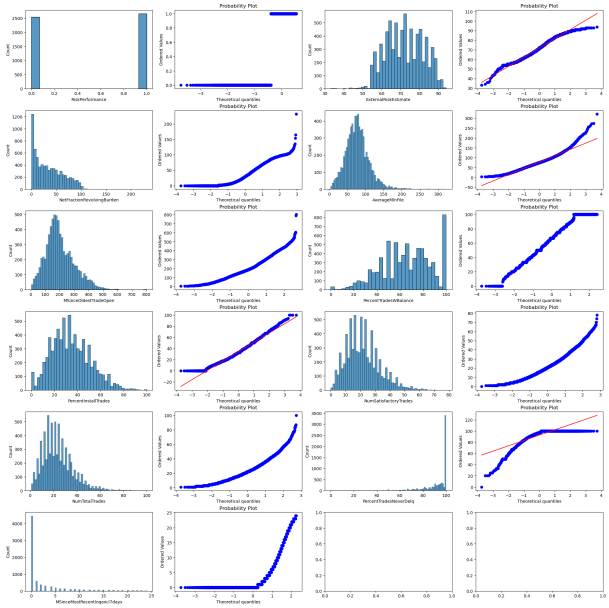

Figure 25: Original data distributions.

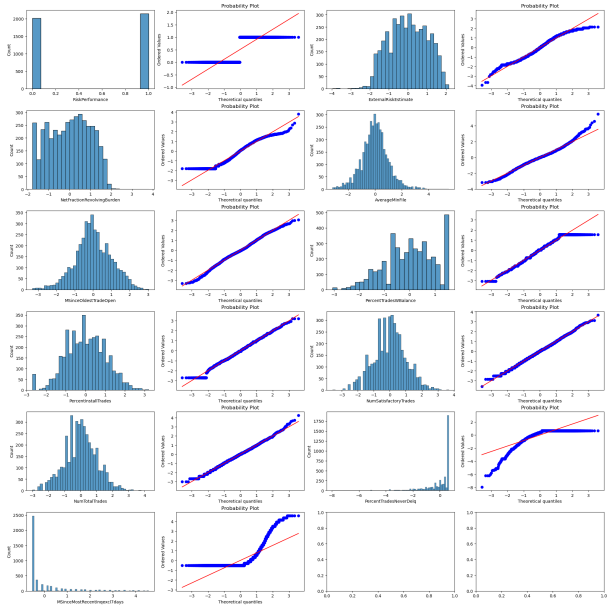

Figure 26: Standarized and transformed data distributions.

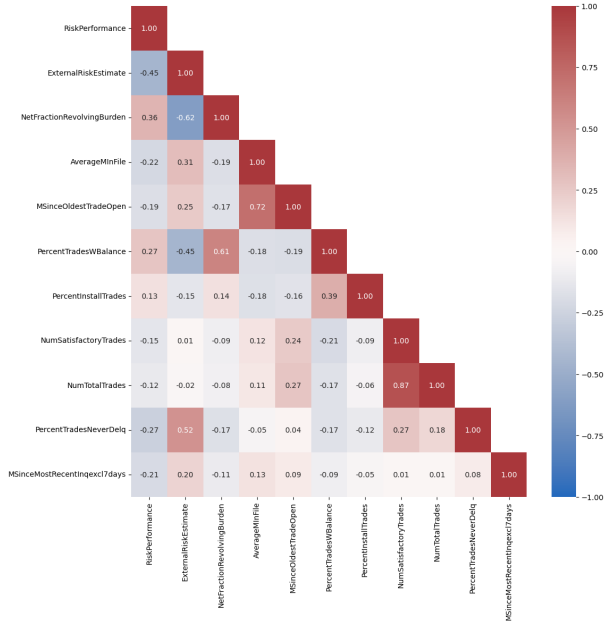


Figure 27: Correlation matrix.

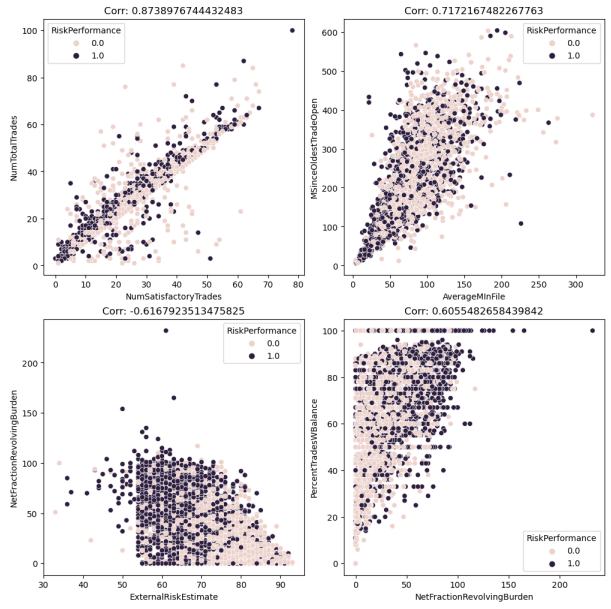


Figure 28: Most correlated variables.

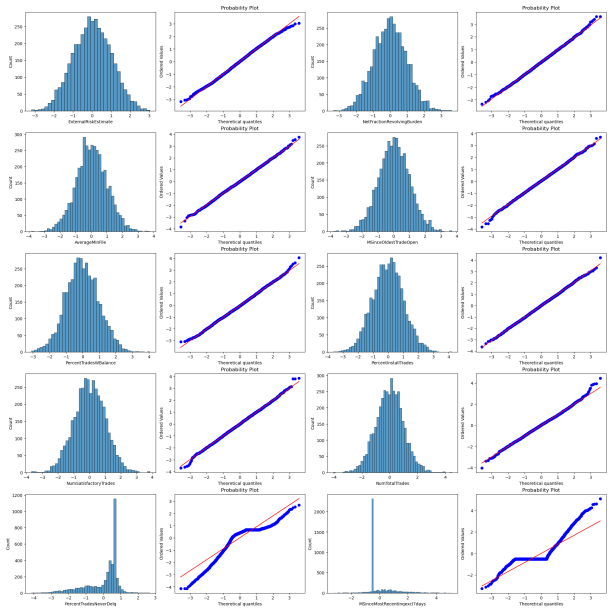

Figure 29: Data samples generated with Gaussian Mixture Model.

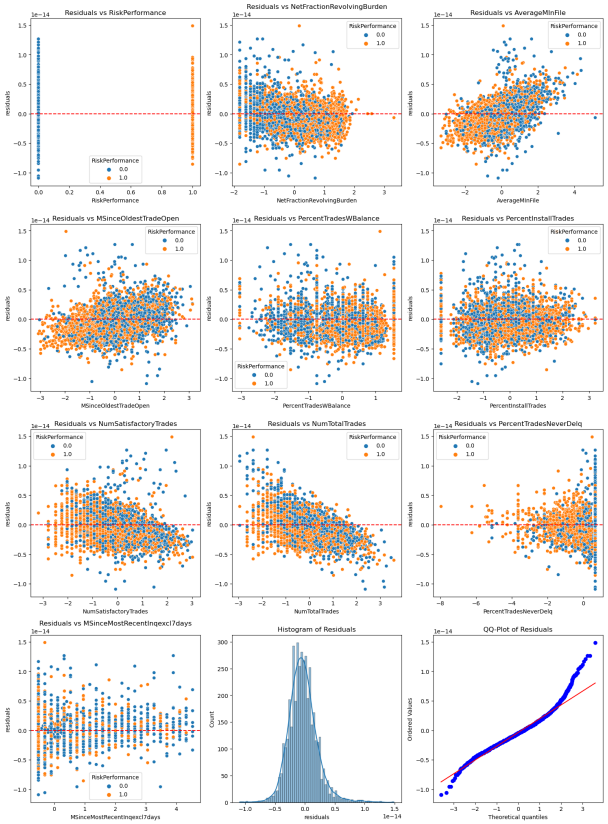

Figure 30: Residuals of predicting ExternalRiskEstimate.

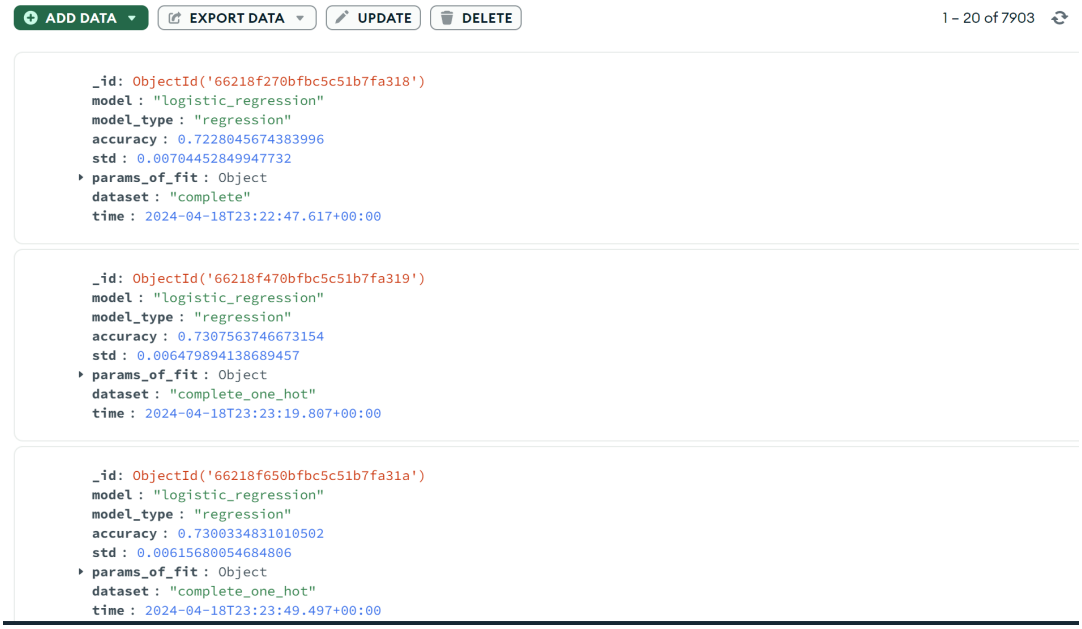


Figure 31: Example of model parameters uploaded to MongoDB during cross-validation

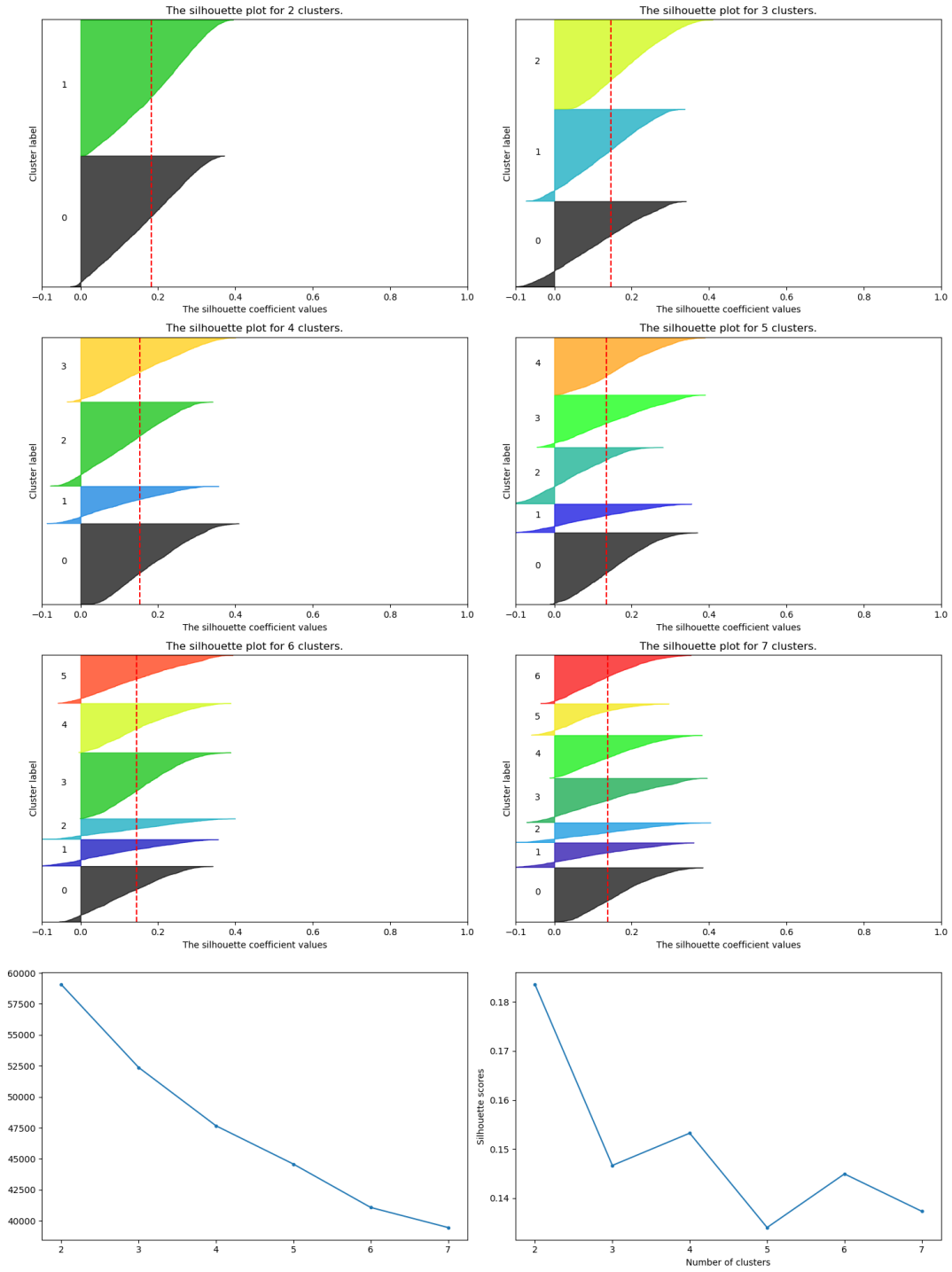


Figure 32: Silhouette KMeans