ECAC Data Mining Project 1

Case study about bank loans

Authored by João Álvaro Ferreira, João Augusto Lima and João Carlos Maduro

Domain Description

The domain for this project is about data collected on loans made in Czech bank(s). The loans are in the year gap 1993-1998. Also it is provided data on the clients and accounts who made those loans, in-depth information on the districts they live in inside the Czech Republic and transfers they have made in the year gap 1993-1996.

This covers 4501 accounts, 5370 clients, 355 loans without status, 329 loans with status, around 400 000 transfers and 77 Czech districts.

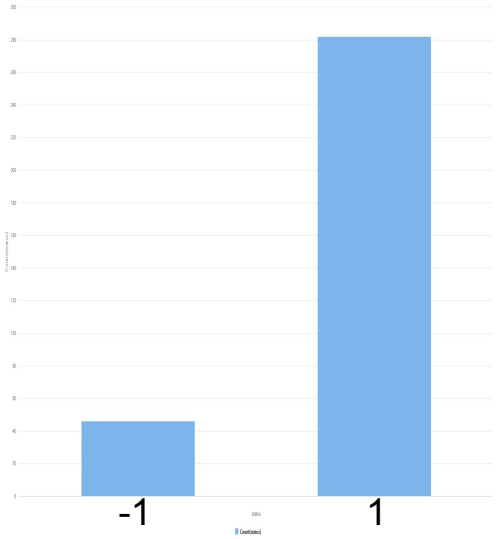
Business goal

1. Predict if a loan is going to be paid by the client.
2. Predict what loans are more or less risky/beneficial for the bank
3. Group client clusters/types based on their attributes, which can be a helpful tool for future strategies and predictions

Data mining goal

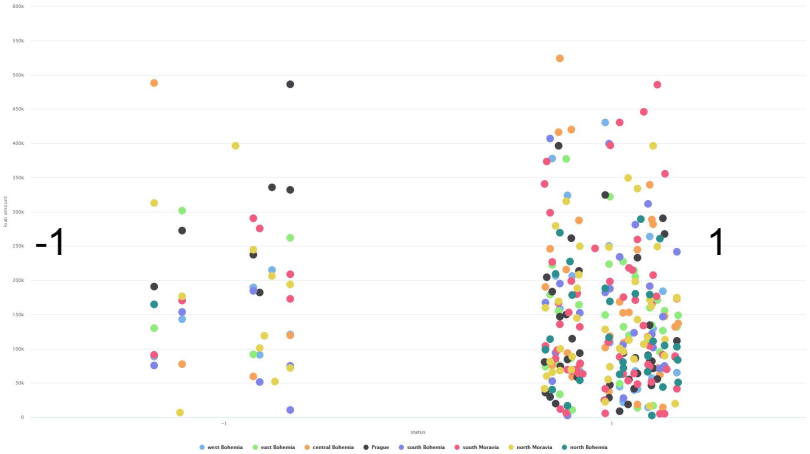
1. Make correct predictions about the status goal of the client (if they will or not pay the loan) using the data from past economic cycles.
2. Determine patterns, relations and behaviors of attributes within the dataset provided
3. Determine what attributes are relevant to group clients by, and create clusters based on them that accurately represent probable results from the client base

Simple data analysis



Imbalanced prediction class (282 to 46). This will cause some trouble to the algorithms, since it will prefer to choose status 1 instead of -1 in the majority of the cases.

This can be caused because the bank is actively giving a lot more loans than rejecting them, or can be caused by some registration error.



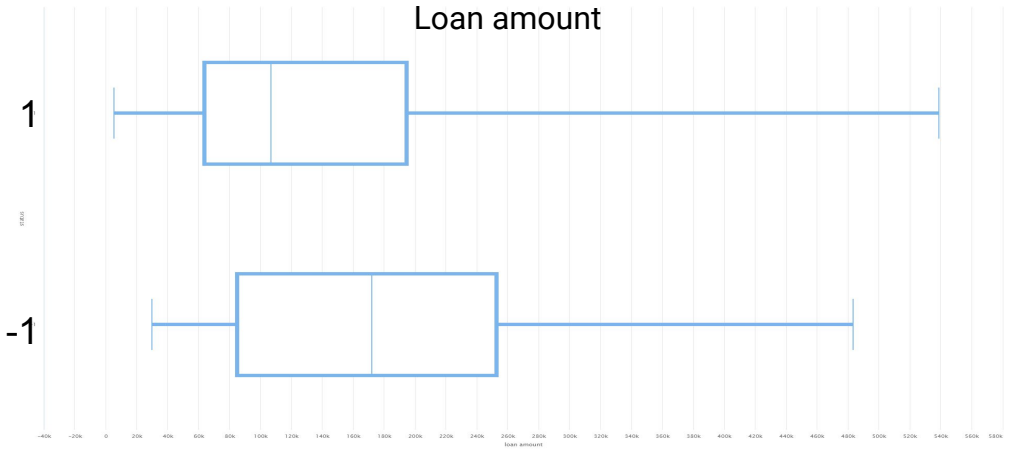
The bank is giving all types of amount of loans, but has a preference for giving low amount of loans.

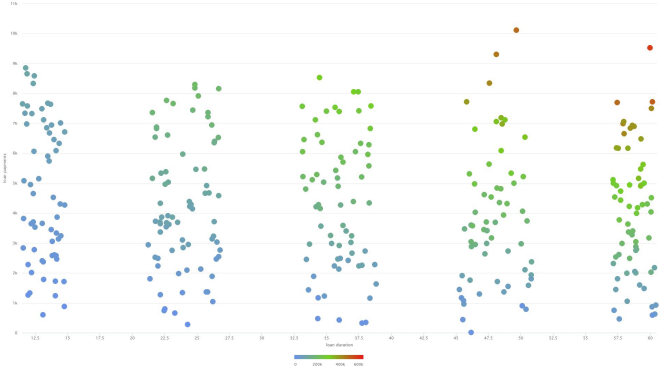
North Bohemia(colour blue) has only 1 rejected of a total of 29 requested loans and most of them are between the upper quartile (194196) and the lower quartile (63297)

Simple data analysis

Loan amount by status:

- Average is bigger on “-1” status (median on -1 is 171864 and on 1 is 106722);
- The bigger the loan amount ,more time it takes to pay it (loan duration) and the bigger is the amount of payments (loan payments). This is a common practice for banks and for the people that request loans.

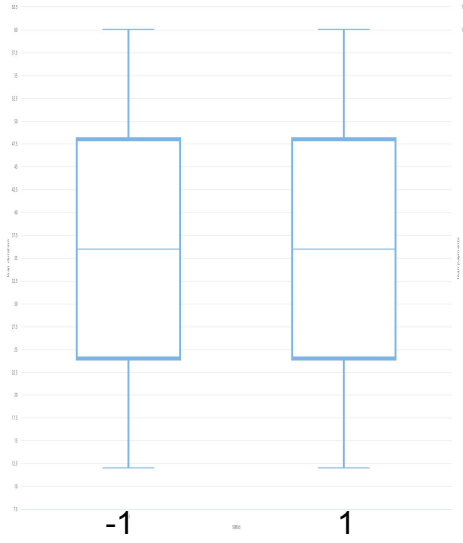




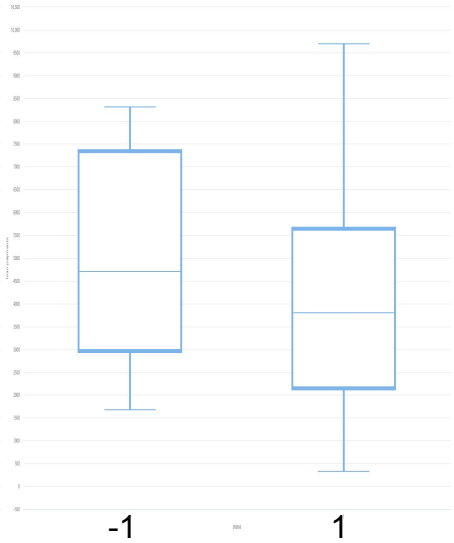
X axis - Loan duration (5 fixed values: 12, 24, 36, 48, 60, the jitter does not correspond to different values from the referred).
Y axis - Loan payments.
Colour - Loan amount

Simple data analysis

Loan duration



Loan payments

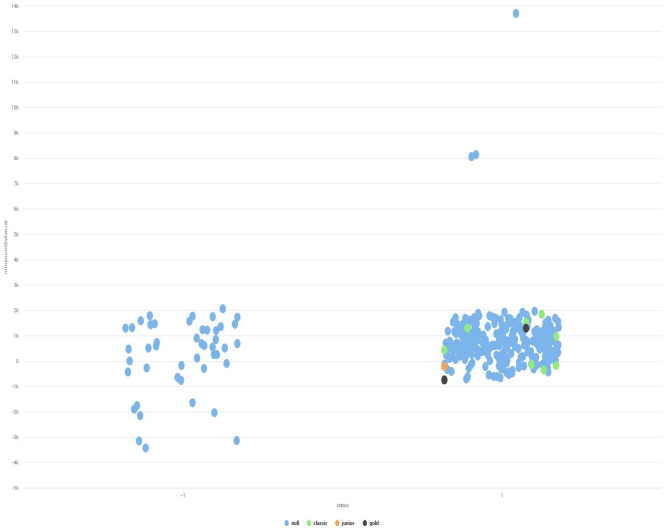


The loan duration (left graph) is the same in for both status. But the loan payments (right graph) differs in the status.

This can happen since the bank prefers less payments for a loan. We don't have more information about each loan but we can assume that it's more beneficial for the bank to make loans with less payments.

Exploratory data analysis

Status (X axis) by card type (Colour) by minimum balance of each user (Y axis)



Null represents no card (colour blue)

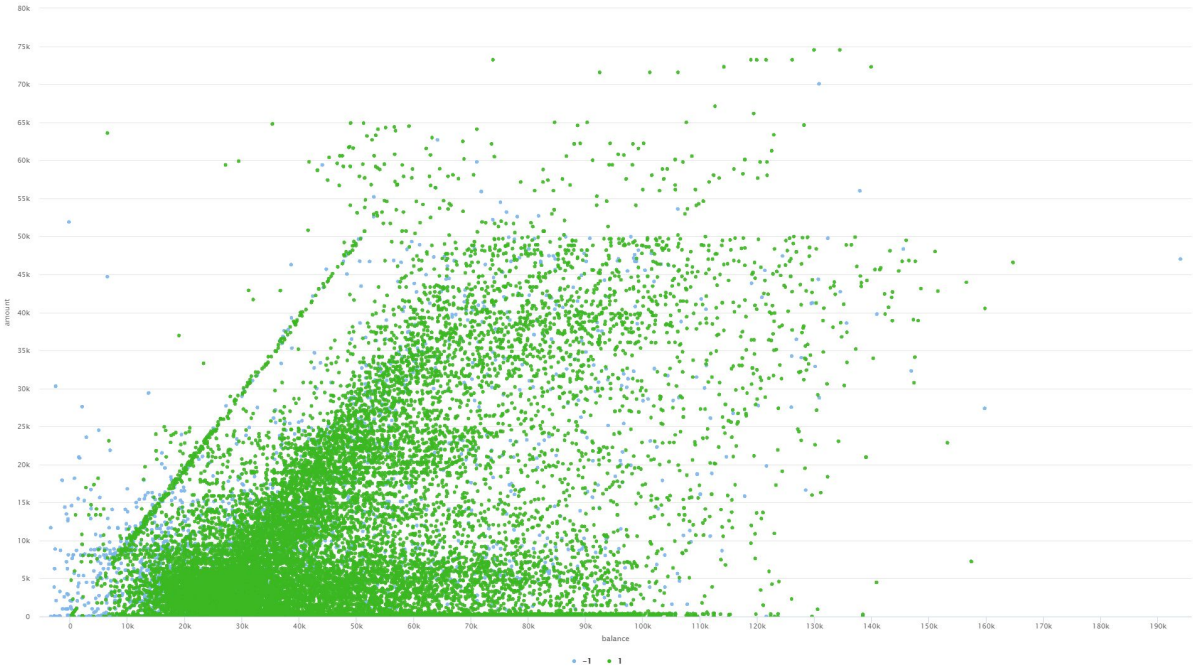
Status (X axis) by disposition type (colour) by maximum balance of each user (Y axis)

We can see that users with cards (colour different from blue) were all accepted for a loan.

From the disposition type we can also see that card with a disponent (colour green) all have been accepted for a loan.

From this 2 plots we can see that the attribute “balance” may have a few outliers (dots way in the top) but we don’t have enough information for this conclusion.

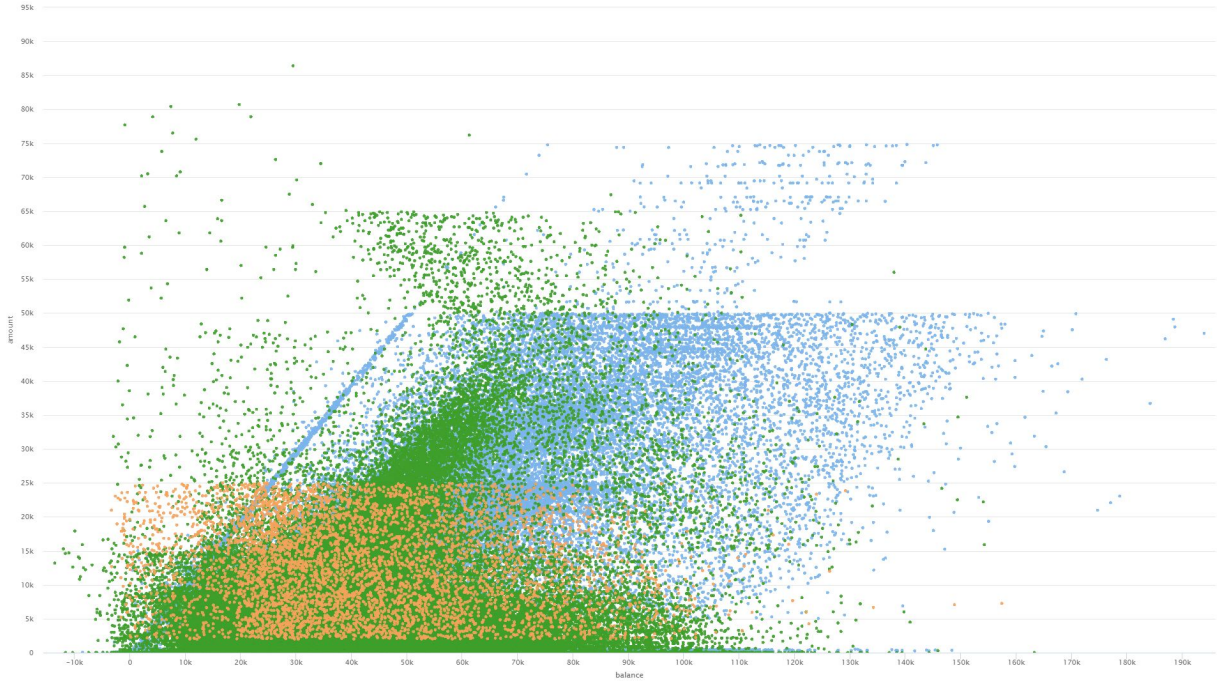
Exploratory data analysis



This graph represents all the transactions made by balance (X axis) and amount (Y axis) and by status (Colour: blue-1;green 1).

From this graph we can see different information such as the amount of money a user has after each operation. From this we assume there are no outliers.

Exploratory data analysis



But this is the graph of all transactions and with the type of operation made (orange for withdrawal in cash; green for withdrawal; blue for credit)

Problem definition

The problem we are attempting to solve is determining what loans, from a large list will be paid or not. To do so via data mining, we must divide it into two:

- A descriptive task, that classifies loans with attributes gathered from the information provided (ie. client info, account info, region info, transactions from the same account, etc.)
- A predictive task, that given on the information gathered and classified in the descriptive task will make a prediction on the status of the loan

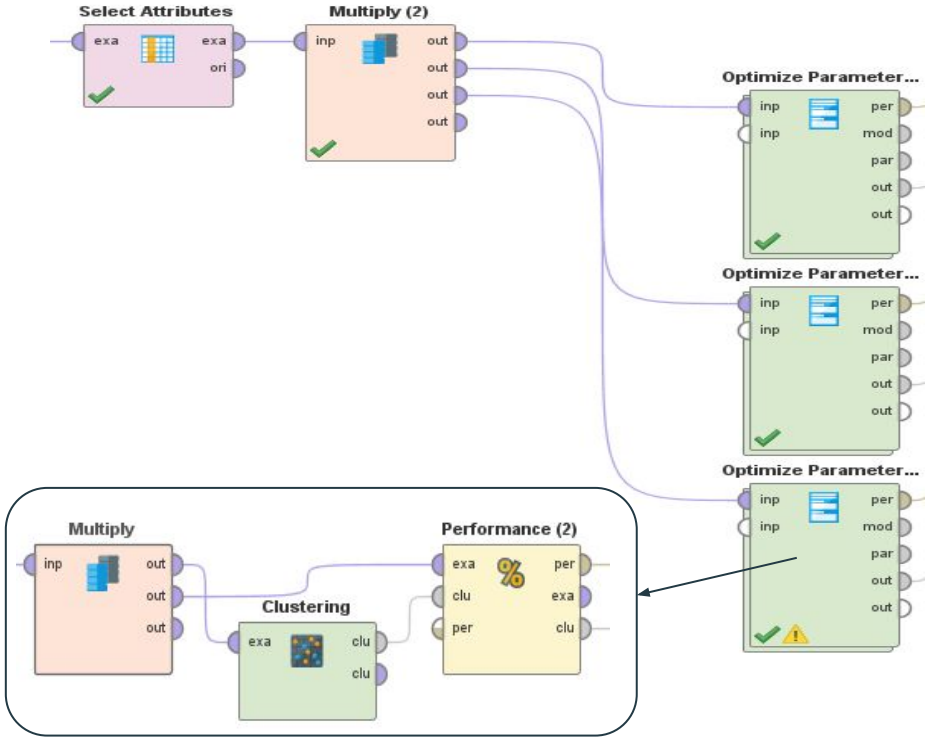
Descriptive task

For the descriptive task we wanted to group the different types of loan (of each client), which in result characterize the different type of clients.

For this we used several algorithms:

- k-Means (divide the data into k clusters that are similar)
- k-Medoids (similar to k-means)
- DBScan (density-based algorithm)

For the iterations of the algorithm we applied different attributes (we applied the most relevant attributes for loans).
We also have parameter tuning to each of the algorithms for 3 different performance measures: the cluster count index, the distance of the centroids and the cluster density.



Descriptive task

Attribute	cluster_0	cluster_1
loan amount	222180	129408
loan duration	60	24
loan payments	3703	5392
average_salary	8843	8110
average(balance)	55230.444	30869.781
average(amount)	11725.495	4550.303

From DBScan guided by density cluster performance

Attribute	cluster_0	cluster_1
loan amount	96442.742	296792.850
loan duration	31.210	50.250
loan payments	3545.863	6026.650
average_salary	9540.544	9757
average(balance)	42712.967	45090.680
average(amount)	9163.463	10682.299

From K-means guided by the distance performance (average within centroid distance)

This clusters tell us two things:

- The loan amount is associated with the duration and payments (the bigger the amount is the bigger the others are);
- The loan amount is connected with the average balance, amount and salary.

All of these clusters are expected. This clusters are similar to the conclusions from the data analysis where clients that make bigger loans are more inclined to have a better economic state (They have more salary and make larger amount of transactions).
Also the relation with loan amount, duration and payments is expected since bigger loans take more time to pay and more payments. This is a common practice for banks.

Data preparation - Data quality

- **Accuracy:**
  - There is little information about the loan. For example what's the use of the loan.
  - There is no information about the client expenses and their current job.
- **Completeness:**
  - Missing table/relation permanent order;
  - Several relations don't have the corresponding total amount of rows (credit_card, transaction);
  - In the relation transaction several attributes have missing values on them;
  - In the relation district, there are also missing values;
- **Consistency:**
  - Joining the tables account, district, client and disposition gives duplicated rows since district is not the same in the account and client;

Data preparation - Data quality

- **Conformity:**
  - Year format is YYMMDD except for the birth_number of the client since it is mixed with it's sex;
  - The type of card is junior, classic or gold;
  - In the relation loan, status should be A/B/C but it's -1 and 1 representing accepted or denied;
  - In the relation transaction, the type should be the symbol +/- but the actual value is a string;
- **Timeliness:**
  - All data of loans is between 1993 and 1999 (not included 1999)
- **Uniqueness:**
  - We can only assume the uniqueness by the id of each table. To detect if one person had 2 accounts we needed more information (for example: the client's name)

Data preparation - Preparation of data

1. Feature engineering
2. Filling missing data with mean value
3. Separation by year
4. Redundant attributes
5. Outliers Detection and Interpretation
6. Forward feature selection
7. Sampling: SMOTE (with ratio 0.2) + Undersampling (with ratio 0.5)

Data preparation - Feature engineering

Features created:
operation percentage - the percentage of card uses by account
crime rates - ratio of crimes committed by year per inhabitant
minimum balance by account - the minimum balance of a transaction by account
average balance by account - the average balance of a transaction by account
average amount by account - the average amount of a transaction by account
Transaction count - the total number of transactions by account

Features selected:
Minimum balance by account, type of transaction, average balance by account and average amount by account

Experimental setup - python pipeline

Our python pipeline first asks the user to choose an algorithm. Afterwards, in the following order, it:

1. Loads the data from the CSV files
2. Splits the data in years, from 93 through 96
3. Sends the training data for pre-processing
4. Collects valuable attributes from cross-referencing with the other tables (transaction, client, district, etc.)
5. Missing data is resolved (via mean algorithms) and outliers are identified
6. Forward feature selection algorithms are applied to select the relevant attributes
7. SMOTE is applied to the minority class (-1) and Undersampling is applied to the majority class (1). After this, a Standard Scaler is used to normalize the data.
8. The machine learning model fits to the train dataset and tests itself with the test dataset.
9. The data gets evaluated (confusion matrix, accuracy, precision and ROC) and sent to be recorded in a CSV file with the loan id and the status

Experimental setup - metrics, algorithms hyper-tuning, outliers

The metrics used to evaluate the results are AUC (Area under the curve), accuracy, precision and a confusion matrix.

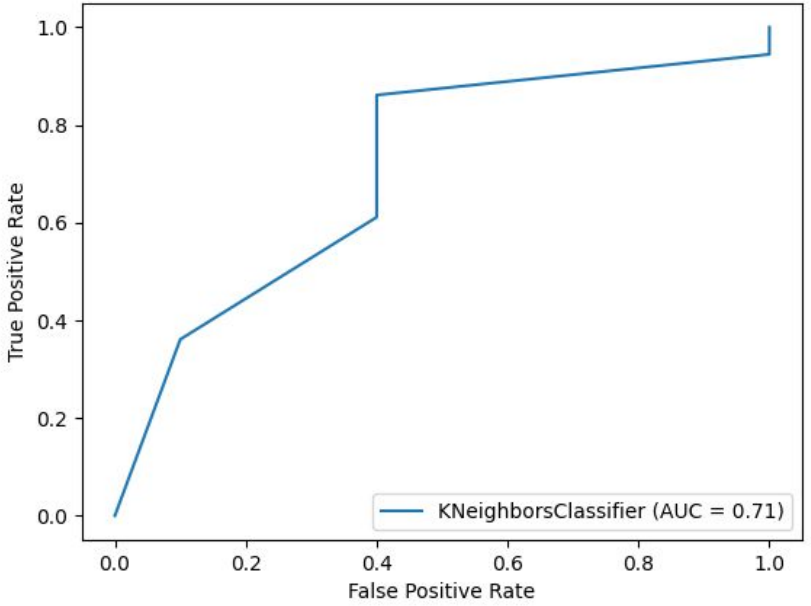
Outliers are identified and analysed, but they're not removed as they have useful information (or at least, what we class as outliers - the 0.001%) and removing them brings down our metrics.

In the case of the Support Vectors Machine, a hyperparameter tuning algorithm is applied. The algorithm starts with an initial parameter grid, and generates different outputs with different params using a Randomized Search with Cross-Validation. Then we build a parameters grid, using the best values from the previous step, for a Grid Search with Cross-Validation.

K-Nearest Neighbors

AUC	0.79
Accuracy	0.83
Precision	0.87
Recall	0.83
f1-score	0.85
Best K	5

	True -1	True 1
Pred -1	6	4
Pred 1	10	62



K-Nearest Neighbors analysis

To choose the best value of k neighbors, it was used an algorithm that would test the value of AUC of the same model, but using different values of k in the range [1, 40].

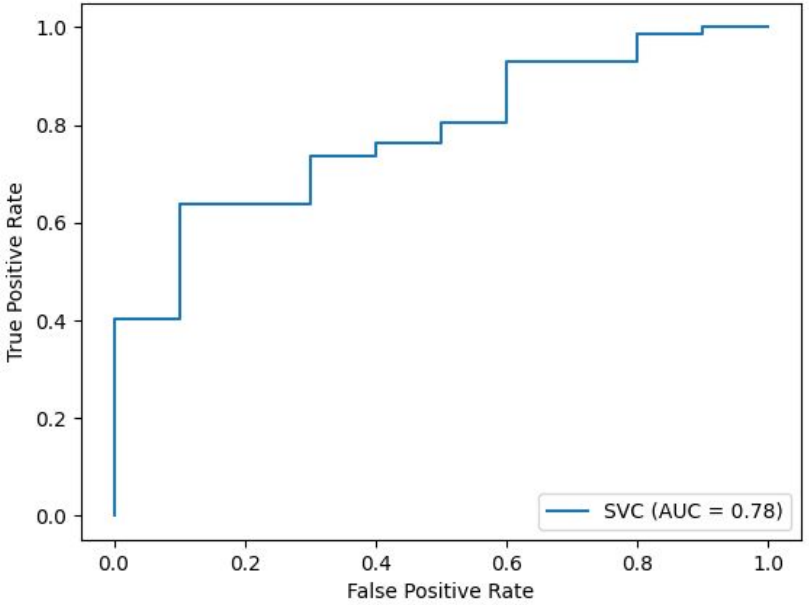
The chosen value of 5 nearest neighbors, proved to be effectively the most accurate value to apply in KNN, due to the results of the tests and due to the analysis made in the “Exploratory data analysis” step.

With the best value k, this model was able to predict more true -1s (the minority class) than any other model, despite not having the biggest value of AUC.

Support Vector Machine

AUC	0.78
Accuracy	0.80
Precision	0.84
Recall	0.80
f1-score	0.82

	True -1	True 1
Pred -1	4	6
Pred 1	10	62



Support Vector Machine

Despite its nature as a probabilistic binary linear classifier, we were not able to get better results as we were expecting.

Even using the parameter grid generated by the hyperparameter tuning process (only RandomSearch or RandomSearch + GridSearch), SVM was not able to get better results than KNN.

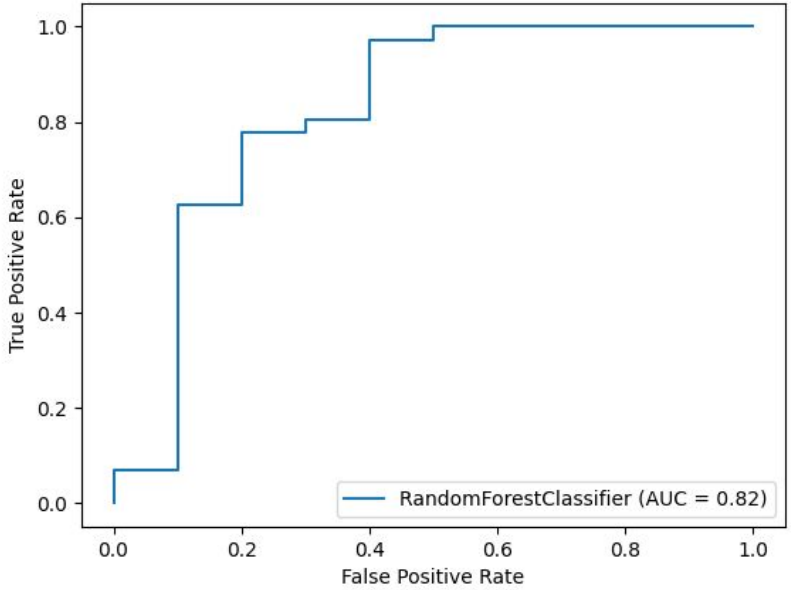
We were not able to figure what went wrong, because in theory this model should have better results.

These chosen parameters were: C=1.0, gamma='scale', kernel='linear'.

Random Forest

AUC	0.82
Accuracy	0.93
Precision	0.93
Recall	0.93
f1-score	0.91

	True -1	True 1
Pred -1	4	6
Pred 1	0	72



Random Forest analysis

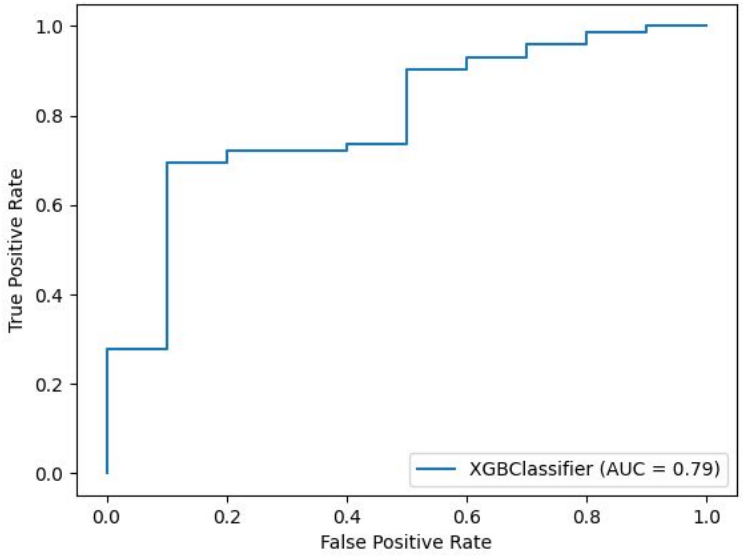
By using an ensemble learning method like Random Forest, We were able to achieve our best results in all metrics.

The efficiency of this method improved even further after balancing the dataset and after some feature engineering. The increase in performance was bigger compared to the other methods

XGBoost

AUC	0.79
Accuracy	0.72
Precision	0.83
Recall	0.72
f1-score	0.76

	True -1	True 1
Pred -1	5	5
Pred 1	18	54



XGBoost analysis

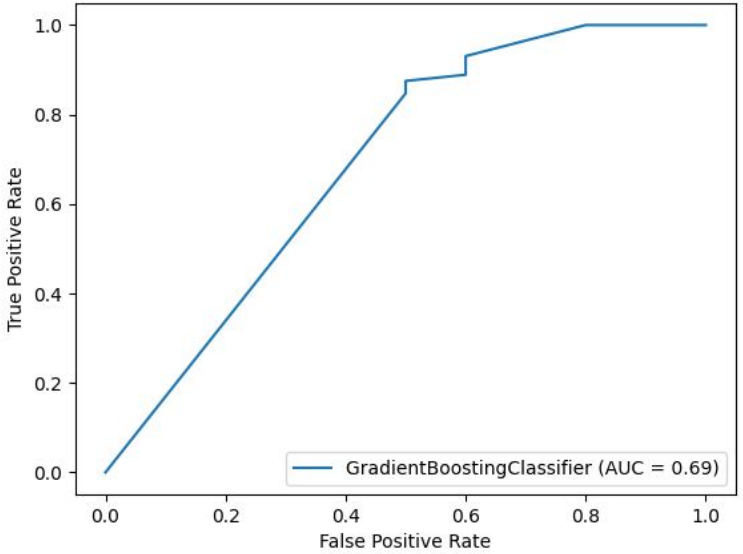
After the success, we tried another ensemble algorithm, this time based in the gradient boost technique. We choose XGBoost, due to the increasing popularity in recent years.

Despite not having better results than Random Forest, still proved to be a good model and had better results than the first two ones that we experimented with

Gradient boosting

AUC	0.69
Accuracy	0.83
Precision	0.86
Recall	0.83
f1-score	0.84

	True -1	True 1
Pred -1	5	5
Pred 1	9	63



Gradient boosting analysis

For our last experimentation, we tested a simple implementation of the gradient boost technique, but without the balancing techniques of XGBoost.

Our initial thought was that our initial sampling strategy would be conflicting with the balancing strategy of XGBoost. But the results of this experiment proved otherwise, and were worse than the previous methods.

Kaggle Competition

For the kaggle competition, we made an effort to take advantage of the 2 submissions per day to ensure our developments were in the right track, even though the public rating for our results isn't the "full picture"

We had a rating of 0.81913 on our best submission at the end of the competition, which was 14th place in the leaderboards at the time.

Getting so focused on the the Kaggle competition was a bad choice, as it led us to "keeping" strategies that gave us good results in the public score and discarding strategies that lowered that score, even if the strategies themselves were overall correct.

The daily submission cycle also prevented heavy code refactoring, which would be helpful for our project.

Yearly Split

We quickly came to the conclusion that is valuable to split the years into separate datasets and train them separately due to yearly trends and common factors. We noticed general trends, such as more loans being fulfilled from 95 compared to 94, but a drop in 96.

However, our implementation of it was too simple and led to worse results metric-wise, so we did not expand upon it enough. A yearly split like this needs to be built into the project from the start and have an impact on its structure, and our implementation simply divided the data at the start and nothing else.

Instead, we should've done subsequent testing and training year by year (training 94, testing on 95, training on 95, testing on 96, etc.) and digged into what tactics would be most effective on a per-year basis.

Conclusions, limitations and future work

According to our results, the best algorithm is the Random Forest, with an AUC of around 0.82 and all other values in the 0.90s.

Feature selection and sampling are key to obtaining better results, with user-created ratios often being useful attributes.

The data provided wasn't the best, having relatively small amounts once the necessary filters were applied.

The Kaggle Competition caused a lot of tunnel vision, which would be best avoided.

In the future, we would've spent more time fitting the hyper-parameters to our algorithms and would've implemented a clean slate for a more perfected, less messy pipeline with easier data manipulation.

Tools

- Git SCM: For collaboration between the group and version control.
- RapidMiner: To rapidly test the models or pipelines, and to make statistical analysis.
- Python and respective libraries (Pandas, Numpy, Scikit-learn, Plt): For more complex and detailed data mining pipelines.
- Google Slides: To build the presentation and respective report.

Inter-group Evaluation

João Álvaro Ferreira	João Augusto Lima	João Carlos Maduro
0	0	0

Thank you, that was all