Credit Card Approval Prediction

Negru Bogdan, Constantinescu Andrei-Eduard, Bădescu Mădălina,
Sabo Vlad-Andrei, Drobnitchi Daniel, Suto Robert-Lucian (411)

January 2023

1 Introduction

Credit score cards are used by financial institutions to assess the risk of lending money to credit card applicants. They use personal information and data provided by the applicant to predict the likelihood of default on credit card payments. Banks use this information to decide whether to issue a credit card to the applicant. Credit scores are a way to objectively measure risk.

Traditionally, credit card scores have been based on historical data. However, when there are significant economic shifts, past models may not be as accurate in predicting risk. These changes represent a challenge in the field and they are the reason for the development of many new models.

We have chosen to predict whether an applicant is a "good" or "bad" client, based on the historical data of their credits and also on their personal information (age, income, education etc.). The difficulty of this task comes from the fact that the definition of "good" or "bad" is not provided, so we need to use various techniques to create our labels.

2 Description of the Dataset

The Credit Card Approval Prediction created by Seanny [2] consists of two comma-separated values (CSV) files, both linked by an ID. The first file describes application records, containing different aspects of a customer who applies for a Credit card, such as:

- Gender
- Ownership of a car
- Ownership of a property
- Number of children
- Income category
- Marital status
- Start date of employment
- Way of living

and more relevant features that are taken in consideration when validating an application.

The second file, describes the credit score of a user, and is structured on three columns.

- The ID
- MONTHS_BALANCE - which reffers to the month of the extracted data, which is the starting point, backwards (0 is the current month, -1 is the previous month, and so on).
- STATUS - which has the following values:
  0: 1-29 days past due
  1: 30-59 days past due
  2: 60-89 days overdue
  3: 90-119 days overdue
  4: 120-149 days overdue
  5: Overdue/bad debts, more than 150 days
  C: paid off that month
  X: No loan for the month

3 Exploratory Data Analysis

The exploratory data analysis of the dataset was crucial for this project in order to visualize the dataset and choose the machine learning models. We explored the fundamental features of the data by plotting histograms, scattered plots, heat maps and other useful visual representations formats.

3.1 Checking for duplicates and missing data

The first step that we took was checking if the files contain any duplicate rows. This is done because identical entries can ruin the data split between train and test, if they are in different sets. After investigation, we observed that the first CSV contains 438510 unique ID's from the total of 438556, and the second one contains 45985 from the total of 1048574. We then proceeded to eliminate all the duplicates from both files.

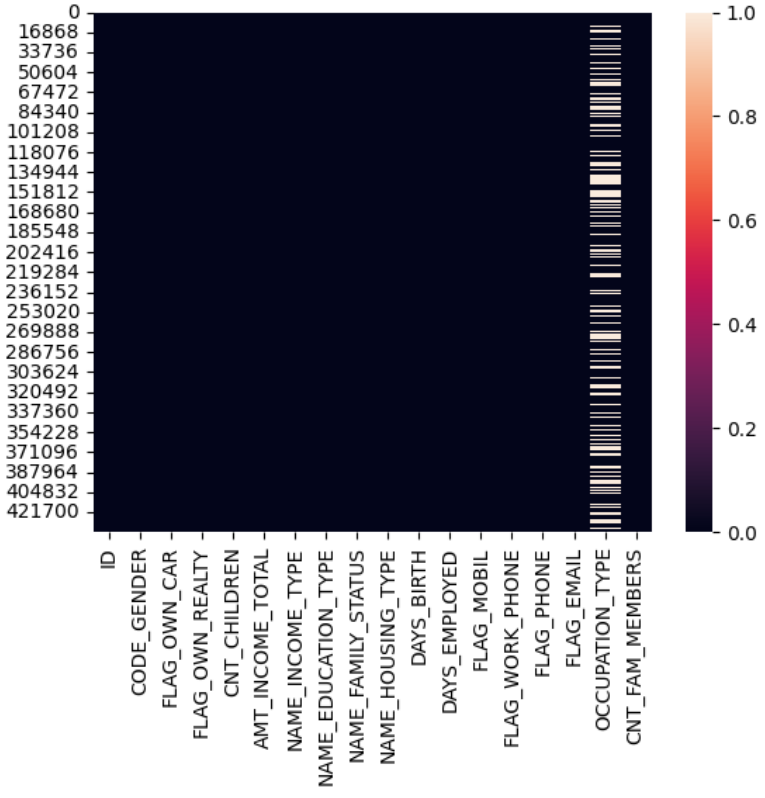

Figure 1: Heat map for the data in the first file (information about the applications)

Figure 2: Heat map for the data in the first file (information about the cred record of the applicants)

The second step in the initial analysis of the dataset was checking for missing values. In order to better visualize if there are entries missing in both files, we computed Figures 1 and 2, which represent heat maps of the data.

We can observe from these figures that the first file has missing entries for the occupation type of the applicants. Because of this, we eliminate this column from the data. For the second file, there is no missing information so we proceed with all columns.

3.2 Processing non-numerical columns

Machine learning algorithms require numbers for working. We analyzed the non-numerical columns in order to have a grasp of their importance for deciding which applicant is "good" or "bad" and, since the information they provided was relevant to our model, we decided to keep them.

In order to use these non-numerical columns, we transformed them with the help of Label Encoder, a preprocessing function. This transforms target labels into values between $[0, n_{classes-1}]$.

3.3 Outliers' elimination

Outliers are values in the dataset which are outside their expected range of values. If this situation occurs, it is best to remove the unlikely data, so that this information does not skew the results.

A simple way of visualizing the outliers of a dataset is to use scattered plot. We used this method on our information and the results are shown in Figure 3.

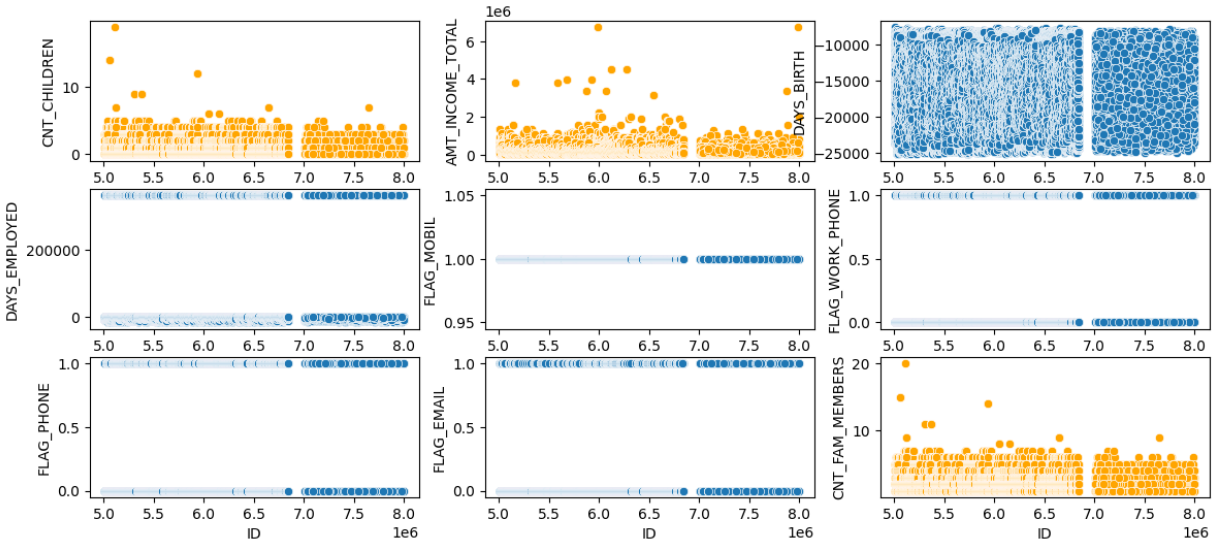

Figure 3: Scattered plots of the parameters for spotting outliers

In Figure 3, we have represented with the colour orange the scattered plots that have outliers. We can observe that there are unlikely values in the plots about the number of children, the total yearly income and the number of family members of applicants. Thus, we proceeded to delete the outlier entries from this paramteres.

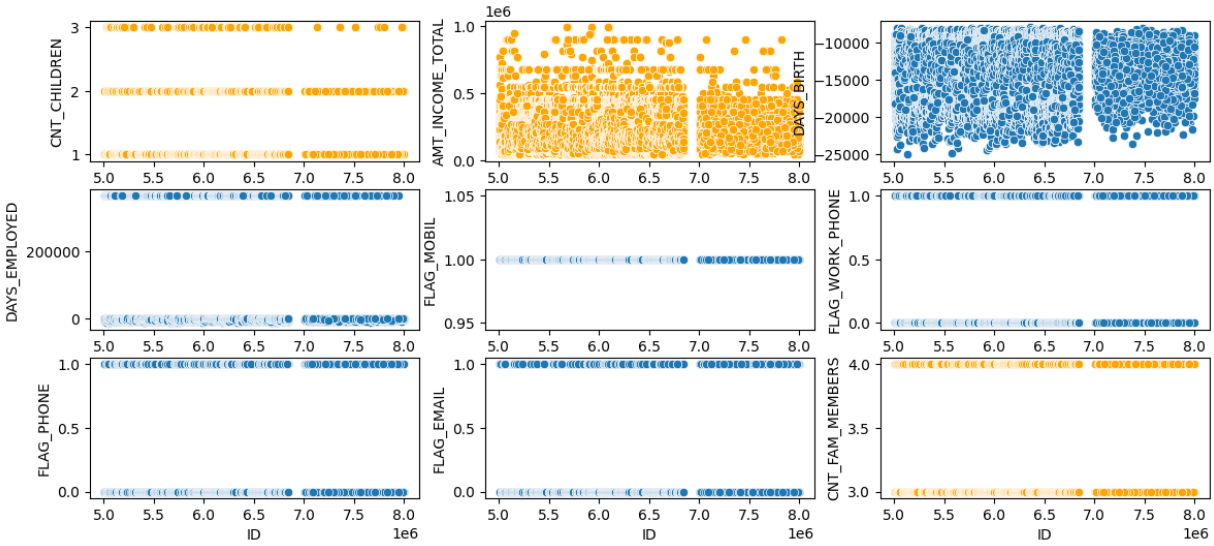

Figure 4: Scattered plots of the parameters after removing outliers

In Figure 4 we have computed the scattered plots for the same parameters as in Figure 3 after the removal of the unlikely data. We can observe that after this computation, the plots do not show any more outliers

3.4 Relationship between different features

At a first glanced, we tried to observe all of the features and how they could relate to each other, in order to better understand them. Here are some of the heat maps we generated to achieve this:

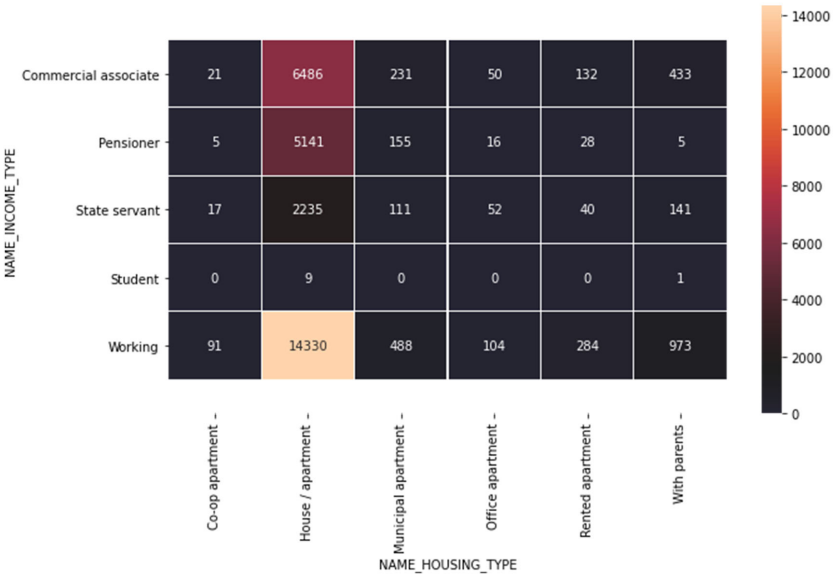

Figure 5: Heat map for representing the relationship between income type and housing

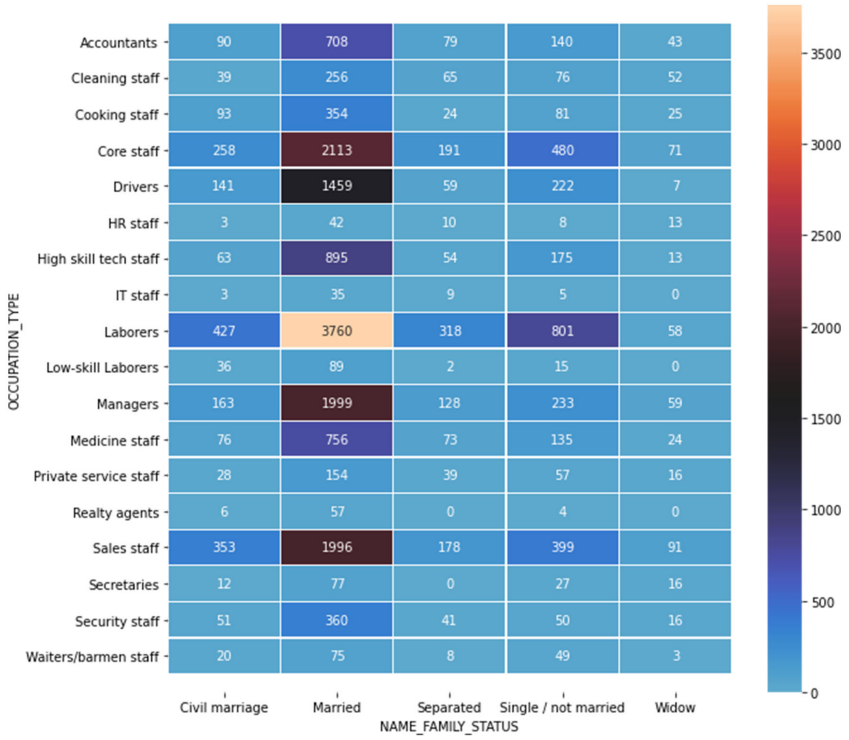

Figure 6: Heat map for representing the relationship between relation status and job

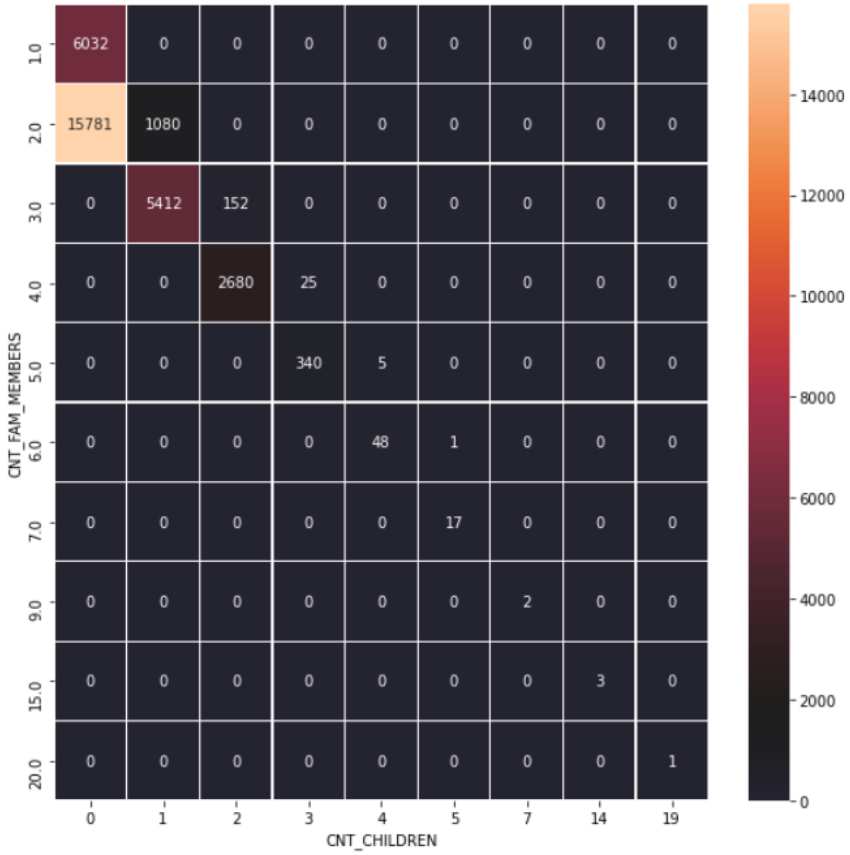

Figure 7: Heat map for representing the relationship between the number of family members and the number of children

This help us identify abnormal values (outliers), similar to the plots presented in Figure 3.

3.5 Distribution of total yearly income of applicants

A relevant feature for the credit car score prediction models is the total yearly income. The parameter highlights what are the financial possibilities of the applicants, thus, this information influences their credit card application. We have computed the distribution of this feature while splitting the dataset into train and test in Figure 8.

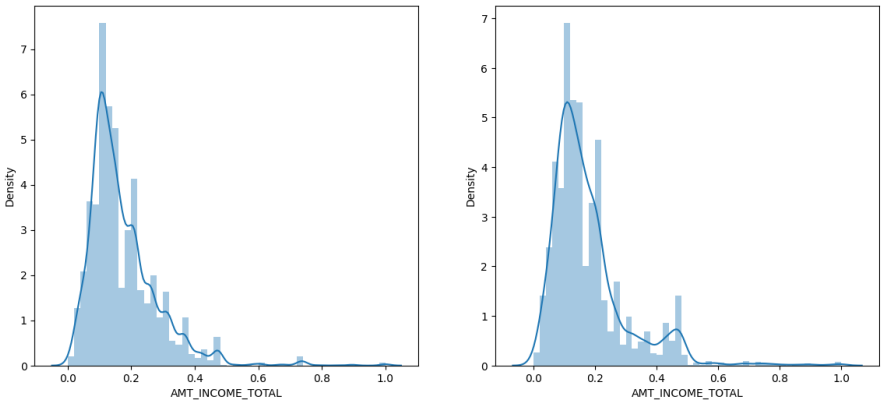

Figure 8: Distribution of total yearly income on the train (left) and test (right) sets

3.6 Distribution of days employed of applicants

The days employed parameter is relevant for the applicants' credit card score, because it showcases their work history and the numbers can represent a red or a green flag. In order to see the split between the sets used for training and testing in relation with the days employed parameter, we have computed Figure 9.

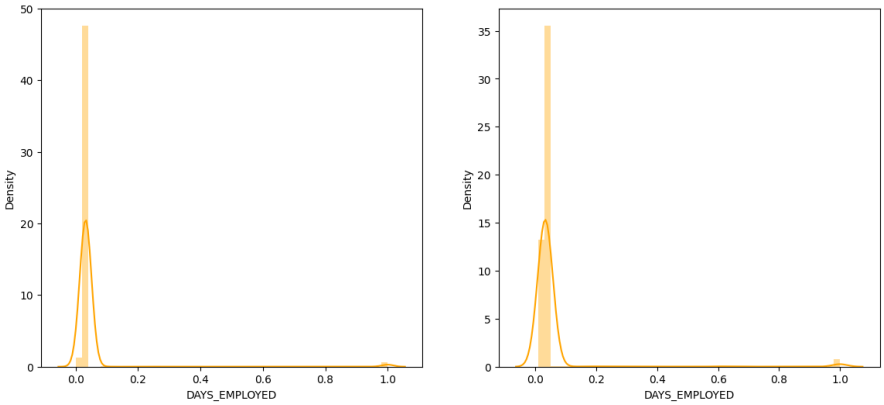

Figure 9: Distribution of the number of days employed on the train (left) and test (right) sets

3.7 Distribution of age of applicants

Maturity is one of the most important features when talking about this subject, therefore we looked into the age (represented in days from birth) of the applicants. We can see the distribution of this feature (on the train and test sets) in Figure 10

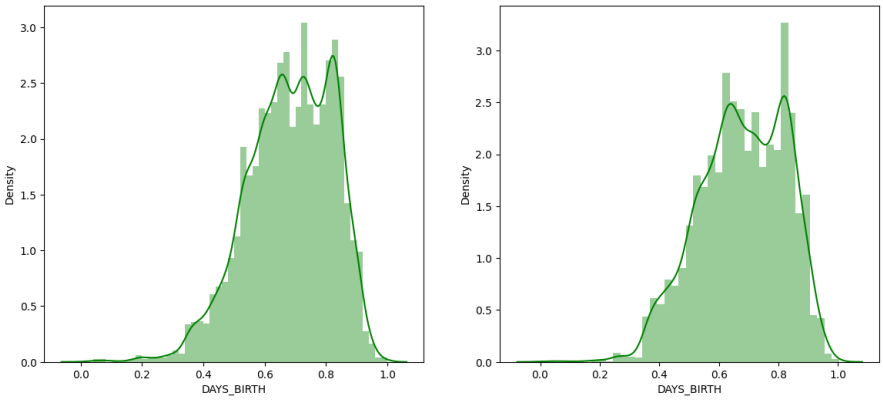

Figure 10: Distribution of days from birth on the train (left) and test (right) sets

3.8 Distribution of number of children of applicants

Family is also taken in consideration when applying for a credit card, especially direct family and children, hence, we also analyzed the distribution of the number of children. The distributions are as follows (Figure 11).

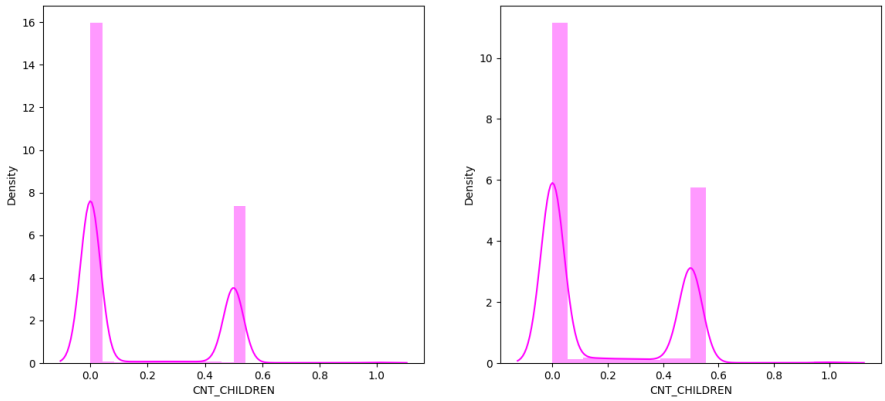

Figure 11: Distribution of number of children on the train (left) and test(right) sets

3.9 Distribution of family members of applicants

While talking about family, another relevant feature regarding credit card applications from this field, is the number of family members. To make sure our models are doing their jobs as intended, we analyzed the distributions of family members on the train and test sets. (Figure 12)

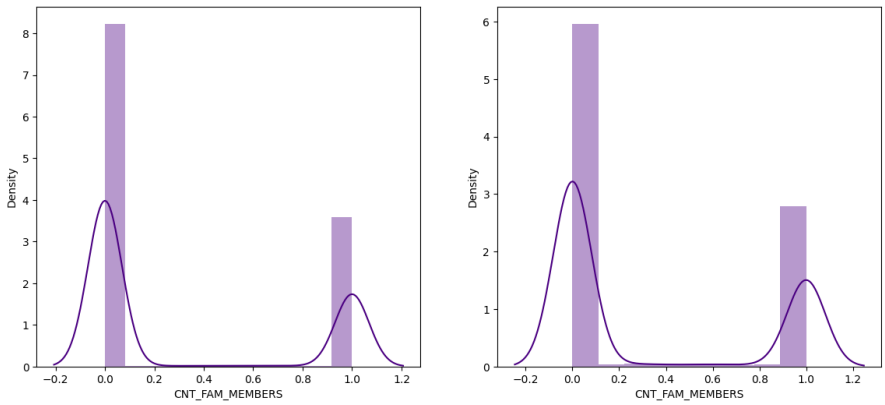

Figure 12: Distribution of family members on the train (left) and test(right) sets

3.10 Distribution of non-Numeric features

As mentioned above, there are also a considerable number of significant non-numerical features that are relevant for this problem. These give us detail about financial status of a person through various things, such as ownership of a car, ownership of a property, education level, income category, family status, housing type, or even general information, such as gender. For these features, the distributions are as follows:

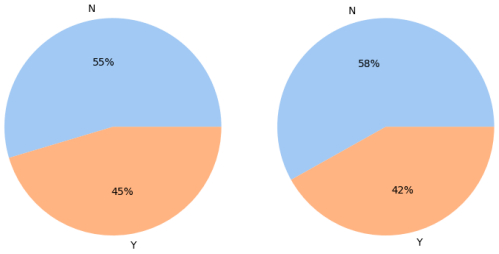

Figure 13: Distribution of car ownership on the train (left) and test(right) sets

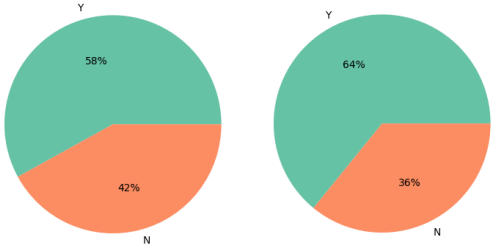

Figure 14: Distribution of property ownership on the train (left) and test(right) sets

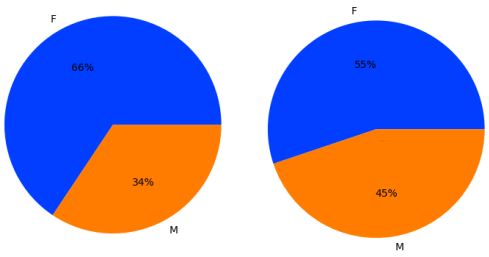

Figure 15: Distribution of genders on the train (left) and test(right) sets

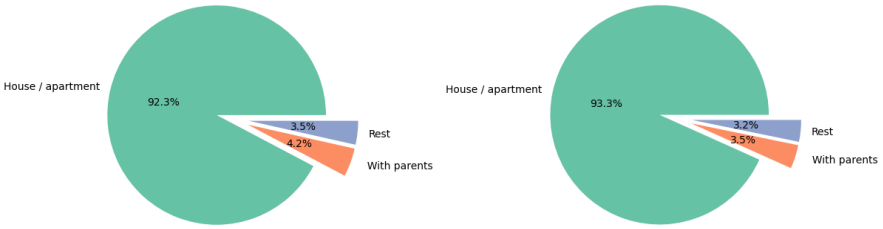

Figure 16: Distribution of housing types on the train (left) and test(right) sets

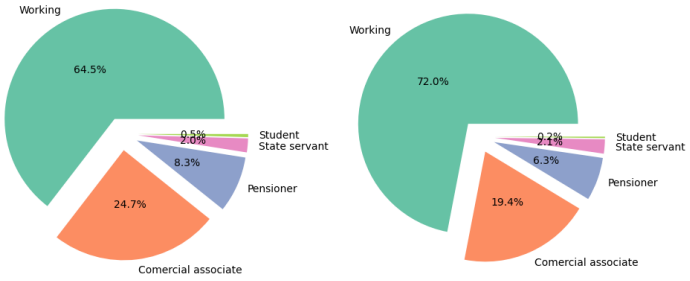

Figure 17: Distribution of income types on the train (left) and test(right) sets

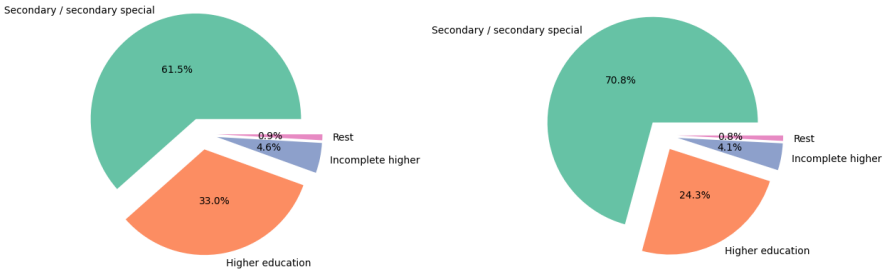

Figure 18: Distribution of education types on the train (left) and test(right) sets

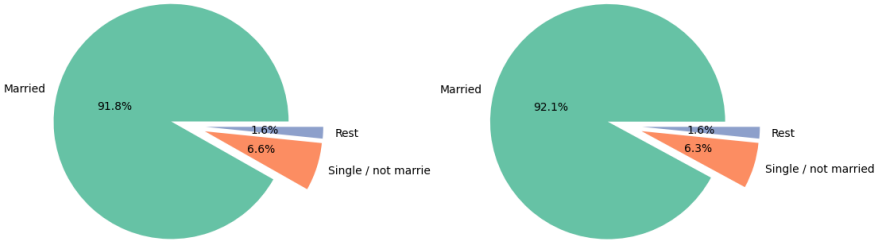

Figure 19: Distribution of family statuses on the train (left) and test(right) sets

3.11 Distribution of credit status of applicants

The credit status of each applicant is the feature that has the most weight when computing a credit card application prediction. This happens because the history of each requester regarding paying off his debts is the most telling in regards with his habits of meeting deadlines. We can observe the distribution between the test and train set of this feature in Figures 20 and 21.

In Figure 21 we denote with 0 the applicants which have the status C, X and 0. The 1 column is for applications with more than 30 days past due payments. This summation of data into 0s and 1s is done in order to make it easier to differentiate between the "good" and "bad" applicants.

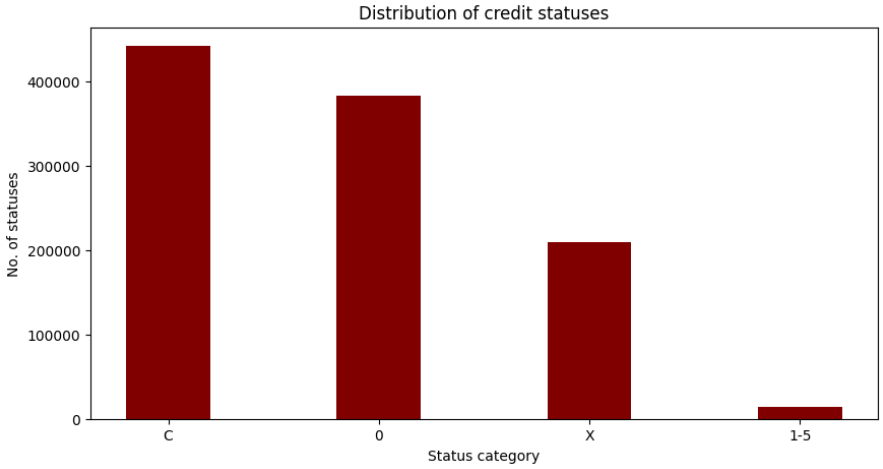

Figure 20: Distribution of credit status on the train (left) and test (right) sets

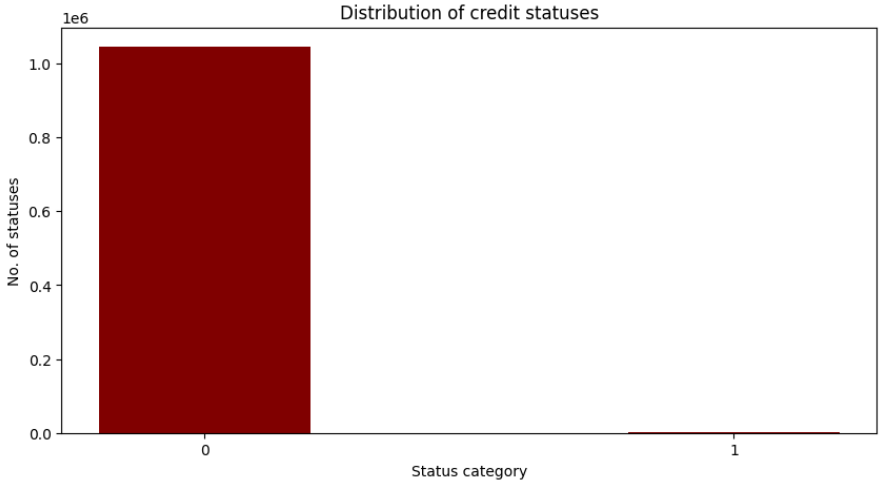

Figure 21: Distribution of credit status on the train (left) and test (right) sets

We can observe from Figures 20 and 21 that the credit card status data is not at all balanced, i.e. the applicants with status 0 far outweigh the ones with status 1. On further analysis we computed that the dataset contains 99,7% applicants with the label 0 and 0,3% applicants with the label 1. We solve this oversampling issue by using SMOTE (Synthetic Minority Over-sampling Technique [1]). This

function computes extra entries with status 1 in the dataset, based on the existing ones. SMOTE is used for both the test and train sets, making the number of applicants for each label equal.

Besides using the SMOTE function, we also wanted to try another method for balancing the dataset. We used RandomUnderSampler, which is a class to perform random under-sampling. It under-samples the majority class by randomly picking samples with or without replacement. When using this method, we are left with very few values for the train and test y sets. Y-train is left with 102 values and Y-test is left with 45. All of the models that we trained on the data processed with RandomUnderSampler performed worse then on the SMOTE balanced data, the accuracy being lower with 10 to 20 points. Thus, we decided not to add the results of the under-sampling procedure to the models and the documentation, but the code is present in the project's jupyter notebook file.

4 Machine Learning Approaches

The purpose of this analysis is to predict whether or not a client's credit application should be accepted or rejected based on the behaviour of similar users by classifying the currently recorded users into "good" and "bad" creditors. We implemented 6 machine learning models:

Algorithm	Parameters	Accuracy
Logistic Regression	solver = any, C=1000	0.753
K-Nearest Neighbor	neighbors = 5	0.807
Decision Tree Classifier	criterion=log_loss, split=2, leaf=3	0.874
Support Vector Classification	kernel = rbf, C = 500	0.872
Gradient Boosting	learning_rate = 0.1, max_depth = 5, subsample = 0.9	0.905
Random Forest	criterion=entropy, sample_split=5, estimators=50,samples_leaf=3	0.879

As for the dataset split, we decided on a 70% training - 30% testing split, using the training set for hyperparameter tuning and validation (with 5-fold validation being implemented).

4.1 Logistic Regression Classifier

A powerful method for binary classification, logistic regression [5] is a supervised learning method. What the algorithm does is to determine whether or not a data point is part of a class by assigning it a probability. As an extra preprocessing step, we normalised the data using a normaliser with the l2 norm, available in Scikit-learn. We also decided to tune the following hyperparameters:

- Regularisation parameter, C $\in$ $\{0.5, 1, 10, 100, 1000\}$
- Solver used by the model, solver $\in$ $\{lbfgs,\ newton-cg,\ saga,\ liblinear\}$

While seemingly perfect for the job, the accuracy provided by the model was roughly 75% after all the tuning. Not an awful result by any means, but not nearly what we were expecting.

Solver	C	5-fold accuracy	Test Accuracy
lbfgs	0.1	0.657	0.580
lbfgs	1	0.721	0.692
lbfgs	10	0.761	0.738
lbfgs	100	0.765	0.750
lbfgs	1000	0.765	0.753
newton-cg	0.1	0.657	0.580
newton-cg	1	0.721	0.692
newton-cg	10	0.761	0.738
newton-cg	100	0.765	0.750
newton-cg	1000	0.765	0.753
saga	0.1	0.657	0.580
saga	1	0.721	0.692
saga	10	0.762	0.738
saga	100	0.765	0.750
saga	1000	0.765	0.753
liblinear	0.1	0.653	0.570
liblinear	1	0.714	0.677
liblinear	10	0.758	0.734
liblinear	100	0.765	0.749
liblinear	1000	0.765	0.753

We obtained the best performance with a regularisation parameter equal to 1000, the solver used for this task turning out not relevant in the end. Increasing the regularisation parameter above 100 also yielded very small improvements on the overall accuracy (as can be seen in the C=1000 examples), thus we can conclude that further increasing it would not help improve the model.

4.2 K-Nearest Neighbors

A simple classification algorithm, k-nearest neighbors is [6]a supervised learning method. This algorithm considers each labeled data point as part of a vector space. From there, the class of a new data point is determined by choosing the most prevalent class amongst the k closest labeled data points according to some chosen metric. We have chosen to use the usual euclidean distance as our metric, leaving the number of neighbors k as the only hyperparameter to be tuned.

Despite the simplicity of the algorithm, it obtains a serviceable but not outstanding accuracy of 82% for small numbers of neighbors, with the accuracy of our model decreasing quickly as the number of neighbors increases, clearly observable in the table bellow:

Number of Neighbors	5-fold Accuracy	Test Accuracy
3	0.965	0.805
5	0.970	0.807
7	0.961	0.798
10	0.952	0.787
20	0.925	0.784
200	0.758	0.657
250	0.742	0.645
500	0.697	0.663

4.3 Gradient Boosting

A boosting algorithm [3]is based on the asumption that a set of weak models can create a single powerful model. It works by taking a base model then building a second one who performs better in the areas where the first model performed poorly. The 2 models are combined, then the process is repeated multiple times. Gradient boosting bases target outcomes on the on the gradient of the error with

respect to the prediction, each new model aiming to minimize said prediction error. We have the following choices of boosters:

4.3.1 Decision Tree

The decision tree booster has the following hypeparameters:

- Learning Rate: the size of the shrinkage used in updating the model
- Maximum Tree Depth
- Subsampling ratio: ratio of the dataset used in training (subsampling will happen on every interation of the boosting)

This booster shows great performance, with a testing accuracy of around 99% and a validation accuracy of around 90% , as shown in the table bellow (the hyperparameters are listed in the same order as above).

4.3.2 Linear

The linear booster has only one hyperparameter, the feature selector, which influences the feature selection and the ordering method. This model performs extremely poorly, as shown bellow.

4.3.3 DART

The DART booster is similar to the decision tree booster, the only difference being that the model will perform dropouts. It has the following hyperparameters:

- Drop Rate: the percentage of the previous trees to be dropped
- Normalization algorithm: whether the new trees will have the same weight of each of the dropped trees (the tree algorithm) or of the sum of dropped trees (the forest algorithm)

Hyperparameters	5-fold Accuracy	Validation Accuracy
0.1, 5, 0.1	0.983	0.89
0.1, 5, 0.5	0.989	0.902
0.1, 5, 0.9	0.989	0.905
0.1, 6, 0.1	0.987	0.882
0.1, 6, 0.5	0.992	0.904
0.1, 6, 0.9	0.992	0.903
0.1, 7, 0.1	0.988	0.882
0.1, 7, 0.5	0.992	0.896
0.1, 7, 0.9	0.993	0.902
0.5, 5, 0.1	0.983	0.89
0.5, 5, 0.5	0.989	0.902
0.5, 5, 0.9	0.989	0.905
0.5, 6, 0.1	0.987	0.882
0.5, 6, 0.5	0.992	0.904
0.5, 6, 0.9	0.992	0.903
0.5, 7, 0.1	0.988	0.882
0.5, 7, 0.5	0.992	0.896
0.5, 7, 0.9	0.993	0.902
0.9, 5, 0.1	0.983	0.89
0.9, 5, 0.5	0.989	0.902
0.9, 5, 0.9	0.989	0.905
0.9, 6, 0.1	0.987	0.882
0.9, 6, 0.5	0.992	0.904
0.9, 6, 0.9	0.992	0.903
0.9, 7, 0.1	0.988	0.882
0.9, 7, 0.5	0.992	0.896
0.9, 7, 0.9	0.993	0.902

Hyperparameters	5-fold Accuracy	Validation Accuracy
Feature selector: cyclic	0.567	0.547
Feature selector: shuffle	0.567	0.547

Hyperparameters	5-fold Accuracy	Validation Accuracy
Drop rate: 0.1, Normalization : tree	0.885	0.818
Drop rate: 0.1, Normalization : forest	0.921	0.858
Drop rate: 0.4, Normalization : tree	0.837	0.786
Drop rate: 0.4, Normalization : forest	0.855	0.794
Drop rate: 0.7, Normalization : tree	0.781	0.755
Drop rate: 0.7, Normalization : forest	0.822	0.778

4.4 Support Vector Classification

A Support Vector Classifier, or SVC for short, [7]is a supervised learning algorithm that relies on building hyperplanes between different data points in order to separate them. The planes are chosen based on their distance to their support vectors and the amount of correctly classified points. From here stem the concepts of hard margins(minimum distance to the data points) and soft margins(allows some points on the wrong side of the margin or hyperplane), which are controlled by the regularization parameter. Harder margins tend to overfit, while lower margins tend to misclassify more often. Another thing of note about SVC is, in case the data is not separable, it can be transposed into a plane in which that becomes possible through a technique called kerneling. These mentioned hyperparameters have been tuned with values from the following sets:

- Regularisation parameter, C $\in$ $\{0.1, 0.5, 1, 10, 100, 500\}$
- Kernel used by the model, kernel $\in$ $\{linear,\ poly,\ rbf,\ sigmoid\}$

As can be seen in the table on the next page, both the linear and sigmoid kernels were suboptimal in solving this classification problem, especially when compared to the radial basis function and polynomial kernels. Increasing the regularisation parameter beyond 100 also caused the polynomial kernel model to overfit. From these, we can conclude that the best approach with SVC would be to use the rbf kernel and a strong regularisation parameter (increasing it over 500 may lead to minor improvements before finally overfitting, but no dramatic spikes in accuracy are expected).

Kernel	C	5-fold accuracy	Test Accuracy
linear	0.1	0.633	0.575
linear	0.5	0.645	0.569
linear	1	0.646	0.566
linear	10	0.648	0.553
linear	100	0.657	0.532
linear	500	0.658	0.527
poly	0.1	0.872	0.720
poly	0.5	0.916	0.767
poly	1	0.931	0.783
poly	10	0.956	0.795
poly	100	0.974	0.823
poly	500	0.978	0.808
rbf	0.1	0.860	0.753
rbf	0.5	0.921	0.783
rbf	1	0.933	0.792
rbf	10	0.959	0.821
rbf	100	0.974	0.855
rbf	500	0.979	0.872
sigmoid	0.1	0.380	0.432
sigmoid	0.5	0.393	0.429
sigmoid	1	0.396	0.431
sigmoid	10	0.397	0.431
sigmoid	100	0.396	0.431
sigmoid	500	0.397	0.430

4.5 Decision Tree Classifier

A decision tree classifier[8] is a type of supervised machine learning model used for classification problems. It works by creating a tree-like model of decisions and their possible consequences. The tree is constructed by breaking down the data into smaller subsets, and at each step, a decision is made about which feature to split the data on, based on which split will result in the greatest reduction in impurity. The process is repeated recursively until a stopping criterion is met, such as a minimum number of samples in a leaf node.

Each internal node in the tree represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a class label. The final class label for a new sample is determined by traversing the tree from the root to a leaf, making the decisions at each internal node based on the values of the sample's features.

We have tuned the following hyperparameters for this model:

- Criterion $\in$ $\{gini,\ entropy,\ log\_loss\}$
- Minimum samples split, min_samples_split $\in$ $\{3,\ 5\}$
- Minimum samples leaf, min_samples_leaf $\in$ $\{3,\ 5\}$

Criterion	min_samples_split	min_samples_leaf	5-fold Acc.	Test Acc.
gini	3	3	0.984	0.795
gini	3	5	0.982	0.805
gini	5	3	0.979	0.810
gini	5	5	0.979	0.813
entropy	3	3	0.982	0.875
entropy	3	5	0.983	0.875
entropy	5	3	0.982	0.876
entropy	5	5	0.980	0.875
log_loss	3	3	0.982	0.874
log_loss	3	5	0.983	0.876
log_loss	5	3	0.980	0.876
log_loss	5	5	0.979	0.876

4.6 Random Forest Classifier

[4]The random forest classifier consists of a large number of individual decision trees that operate as an ensemble, each individual tree in the random forest outputs a class prediction and the class with the most votes becomes the final prediction. It is a supervised learning algorithm that produces even without hyperparameter tuning great results.

One big advantage of this alogrithm is that it can be used for both regression and classification problems. Being a derivative of the decision tree classifier, it uses almost the same hyperparameters in a training batch.

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

The main limitation of random forest is that a large number of trees can make the algorithm too slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained.

We have tuned the following hyperparameters for this model:

- Criterion $\in \{gini,\ entropy\}$
- Minimum samples split, min_samples_split $\in \{3,\ 5\}$
- Minimum samples leaf, min_samples_leaf $\in \{3,\ 5\}$
- N Estimators $\in \{50,\ 1000\}$

Criterion	min_samples_split	min_samples_leaf	N Estimators	5-fold Acc.	Test Acc.
gini	3	3	50	0.987	0.855
gini	3	5	50	0.987	0.854
gini	5	3	50	0.990	0.830
gini	5	5	50	0.984	0.821
entropy	3	3	50	0.985	0.863
entropy	3	5	50	0.987	0.873
entropy	5	3	50	0.984	0.879
entropy	5	5	50	0.988	0.868
gini	3	3	1000	0.987	0.845
gini	3	5	1000	0.987	0.850
gini	5	3	1000	0.990	0.853
gini	5	5	1000	0.984	0.856
entropy	3	3	1000	0.985	0.874
entropy	3	5	1000	0.987	0.868
entropy	5	3	1000	0.984	0.865
entropy	5	5	1000	0.988	0.876

5 Conclusion and future work

Modeling on a credit card approval prediction is a significant task in Machine Learning, because it requires a big amount of data. Exploring our data set, we noticed that our set is unbalanced to the small number of denied applications, but there still were some important features/characteristics in the data, that one would take in consideration while reviewing a similar application. We implemented 6 conventional machine learning models (Logistic Regression, K-nearestt Neighbor, Decision Tree, SVC, Gradient Boosting, Random Forest), out of which, the Gradient Boosting model performed the best.

An intriguing line of future research would be to address the imbalance in the dataset in order to attain optimal outcomes. This could be accomplished through the acquisition of additional data or the experimentation of alternative methods, such as oversampling, to mitigate the imbalance. It would also be interesting to resolve this problem from a different perspective, and maybe analyse approved applications to see if they truly should have been approved or spot fraudulent approval.

References

[1] The imbalanced-learn developers. SMOTE. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html, 2014-2022. [Accessed 10.01.2023].

[2] Seanny. Credit Card Approval Prediction. https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction, 2022. [Accessed 10.01.2023].

[3] sklearn. sklearn.ensemble.GradientBoostingClassifier Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html. [Accessed 9.01.2023].

[4] sklearn. sklearn.ensemble.RandomForestClassifier Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed 9.01.2023].

[5] sklearn. sklearn.linear_model.LogisticRegression Documentation. https://imbalanced-https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. [Accessed 9.01.2023].

[6] sklearn. sklearn.neighbors.KNeighborsClassifier Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. [Accessed 9.01.2023].

[7] sklearn. sklearn.svm.SVC Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Accessed 9.01.2023].

[8] sklearn. sklearn.tree.DecisionTreeClassifier Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.htmll. [Accessed 9.01.2023].