Customer Churn Prediction in Banking Industry

Martin Pažický, Roman Berešík

1. Introduction

Customer churn is defined as the inclination of customers to stop using or purchasing a company's product in a given time period. It is a common measure of lost customers. Companies have started to realize that they should make an effort not only to seek new customers, but also to retain the existing ones. By understanding the factors that contribute to churn and accurately predicting which customers are at risk of leaving, banks can take proactive measures to mitigate churn and improve customer retention strategies.

Companies, especially banks, often collect lots of data about their clients. One of the options banks have, to fight against customer churn, is to use the data and train a machine learning model, which might predict when a customer is about to churn their services. Banks could then approach the client and try to resolve the reasons that led him to this decision.

In this project we are going analyze the data of a bank and try to predict customer churn using several machine learning models – random forest, logistic regression, and neural networks. Each model brings its own strengths and assumptions, allowing us to explore different approaches to churn prediction. Random forest, for instance, is an ensemble learning method that combines multiple decision trees to make accurate predictions. Logistic regression, on the other hand, is a classical statistical model that estimates the probability of a binary outcome based on input variables. Additionally, we will also experiment with more complex neural networks, such as deep learning architectures, to capture intricate patterns and nonlinear relationships in the data. We are going compare the methods and choose the best performing one.

2. Related Work

This topic got very popular due to the huge amounts of data banks have collected and many papers trying to predict customer churn have already been published. For example, in [1] authors use data mining tools such as Naïve Bayes or support vector machines to predict the churn behavior among Indian bank customers. Another paper [2] uses several classifiers to predict who is going to churn the bank and a comparison between the classifiers is conducted, of which Random Forest model comes out as the best.

Researchers have also looked into using neural networks in addition to conventional machine learning methods to solve this issue. A multi-layer perceptron (MLP) for instance, was used to forecast client attrition in a banking environment [3]. They showed that the MLP model performed comparably to other models, demonstrating the potency of neural networks for churn prediction.

Another study [4] also used a deeper learning model, namely a convolutional neural network (CNN), which has a more complicated neural network design, to predict customer attrition. The authors enhanced the churn prediction accuracy by utilizing the hierarchical feature extraction capabilities of CNNs to capture complex patterns and correlations in the data.

3. Exploratory Data Analysis

In this project we are going to use a dataset which describes customers of a bank based on their credit card activity and their demographic data (https://www.kaggle.com/datasets/sakshigoyal7/credit-card-customers). The data consists of 10127 records described by 21 attributes, of which 15 are numerical and the other 6 are categorical. The attributes of the data are described in table 1. Our goal will be to classify customers into 2 groups based on the *Attrition_Flag* attribute – the ones who left the bank and the ones that stayed.

Názov	**Typ**	**Popis**
CLIENTNUM	Numerical	Identificator
Attrition_Flag	Categorical	Customer churn (target var)
Customer_Age	Numerical	Age
Gender	Categorical	Gender
Dependant_count	Numerical	Number of dependents
Education_Level	Categorical	Education
Marital_Status	Categorical	Marital status (single/married/divorced)
Income_Category	Categorical	Annual income
Card_Category	Categorical	Card type
Months_on_book	Numerical	Length of relationship with the bank (number of months)
Total_Relationship_Count	Numerical	Number of products held
Months_Inactive_12_mon	Numerical	Number of inactive months in the last year
Contacts_Count_12_mon	Numerical	Number of contacts in the last year
Credit_Limit	Numerical	Credit limit on the card (maximum debt)
Total_Revolving_Bal	Numerical	Revolving balance (carried debt)
Avg_Open_To_Buy	Numerical	Available credit (average over the last year)
Total_Amt_Chng_Q4_Q1	Numerical	Change in the amount of transactions
Total_Trans_Amt	Numerical	Total amount of transactions (last year)
Total_Trans_Ct	Numerical	Total number of transactions (last year)
Total_Ct_Chng_Q4_Q1	Numerical	Change in the number of transactions
Avg_Utilization_Ratio	Numerical	Credit card debt to credit limit ratio

Table 1: Data description

During the exploration of the data, we noticed that the distribution of our target variable is imbalanced, and the positive outcomes (customers that stay in the bank) considerably outnumber the negative outcomes. We will have to apply proper methods in order to deal with this situation while training models.

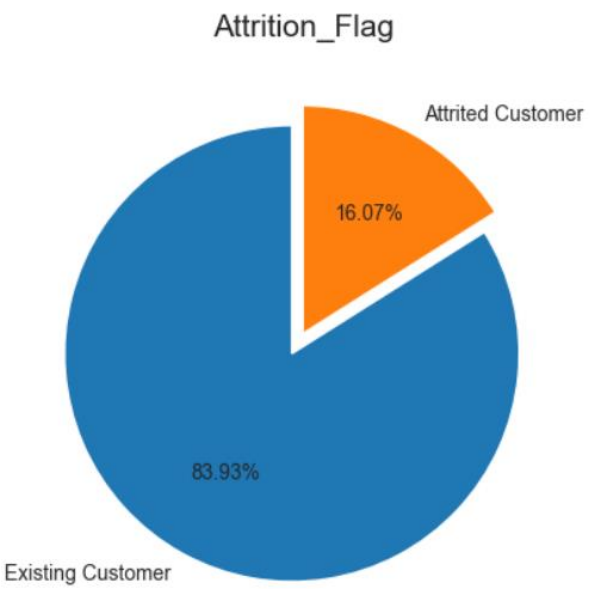

Figure 1: Attrition flag distribution

We have also examined the bivariate relationships between different columns of the data. We visualized the distributions of each numerical column in relation to the target variable using boxplots. In some variables a significant difference between the distributions with respect to the target variable could be spotted. We can for example see that the churners tend to have a smaller count of transactions than the non-churners (*figure 2*).

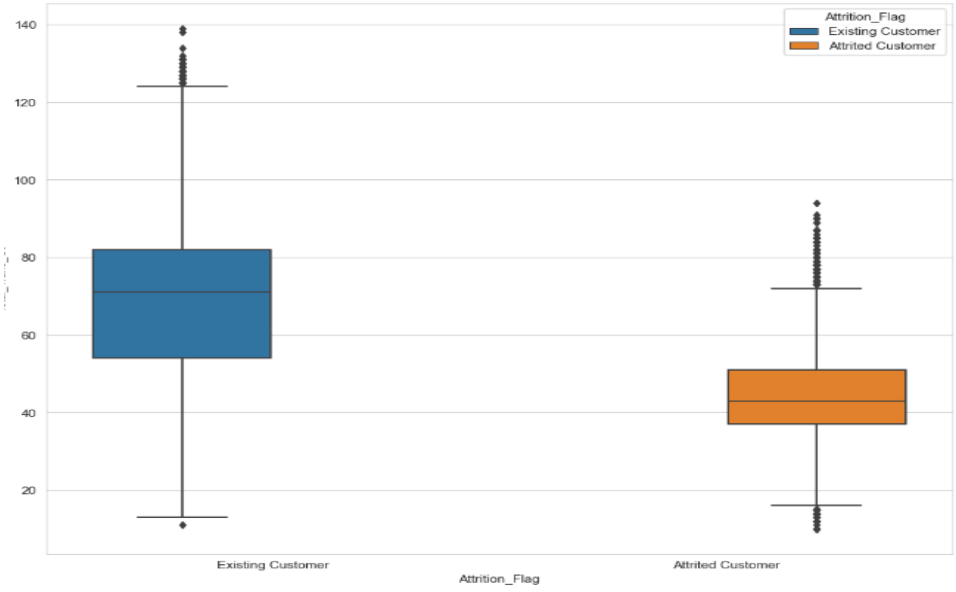

Figure 2: Transaction count distribution with respect to attrition flag

We have also looked into the relationships of categorical columns with the target variable. On *figure 3* we can see graphs with the percentage of attrited customers per category of a given variable. Each category has an annotation carrying the information about its support in the data (what percentage of records contains given value in the column) to emphasize the importance of the information. The red line in each graph depicts the percentage of attritied customers in whole dataset in order to help to portray the difference between attrition percentage of a given category and the average attrition percentage. We can for example see that the churn rate of customers holding a platinum card is remarkably high, however not many customers hold a platinum card. We can also spot that customers with income between 60 and 80 thousand dollars tend to stay in the bank.

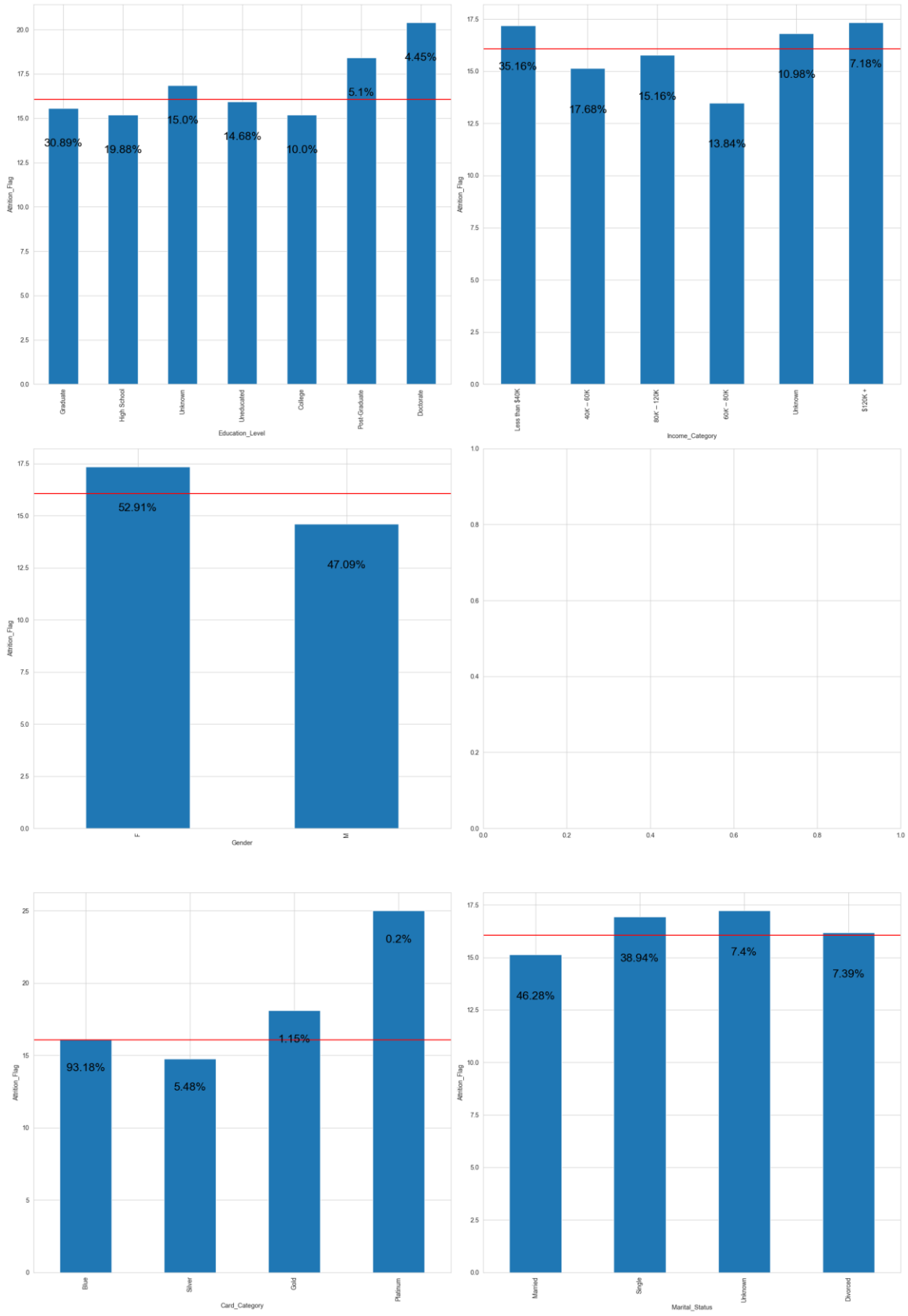

Figure 3: Attrition of customers in different classes of categorical variables

3.1. Handling Missing Values

The data did not contain any null values, however we noticed that some variables had *Unknown* values present. We considered those values missing and applied the imputation methods on these variables.

Missing values were only present in 3 variables, all of which are categorical.

Column	Missing values number
Education_Level	1519
Marital_Status	749
Income_Category	1112

Table 2: Missing values

To impute the missing values two approaches were used and evaluated. The first approach was to impute the values using decision tree. For each attribute with missing values, the tree was first trained on records with valid values in that column. The attribute that contained the missing values was chosen as the target variable, while the other attributes of the dataset were used as predictor variables. After the training phase, the records that contained missing value in the given attribute could be passed to the model, which decided what value will be used instead of null.

Before training the tree, it was also necessary to temporarily deal with missing values in the predictor variables. Since the attributes with missing values are exclusively categorical, the most frequent value occurring in the given attributes was chosen as a temporary replacement. Decision tree is also not able to work with categorical variables, therefore we transformed categorical variables into numerical using one hot encoding.

Another approach we used was imputation using k-nearest neighbors algorithm (KNN). To be able to use KNN, similarly as in previous approach, some preprocessing needed to be done. The same preprocessing steps as in the decision tree imputation were applied, with one extra step, which was normalization of numerical values using min-max scaling, to prevent one variable from having significantly higher impact on the result, than others. Otherwise, the process was the same as in the first approach (temporary imputation using the most frequent value, target / predictor variables choice), the only change was in the choice of model which was used.

To assess these two approaches, splitting of the dataset to training and test data was used. The training data were used to build the models, then the predictions of model were evaluated on the test data. The evaluations and splits were done for each variable which contained missing data separately, and the scores were then averaged for each method. As a metric to evaluate the results of different models accuracy was used, since in this case there is not a big difference between the cost of false positives and false negatives.

Assessment results can be seen in *table 3*. Each row represents one method and the numbers in the cells represent the test accuracy score of a particular method for corresponding column. We can see that the decision tree has performed better on all the variables. Therefore, we decided to proceed with the dataset imputed by the decision tree method.

	Education_Level	Marital_Status	Income_Category	Average
Decision Tree	0.360627	0.553838	0.601775	0.505413
KNN	0.302555	0.513326	0.557404	0.457762

Table 3: imputation methods evaluation

3.2. Data Transformation

In order to use the data for training the selected machine learning models, various data adjustments had to be made. The models are not able to deal with categorical data directly, therefore we needed to transform them to numerical data first. Applying domain knowledge allowed us to spot certain hierarchical relationships between the values of various features. To preserve the relationships in the transformed features ordinal encoding method was applied. The mentioned hierarchies are depicted in *table 4*.

Column	Order
Card_Category	Blue < Silver < Gold < Platinum
Income_Category	Less than $40K < $40K - $60K < $60K - $80K < $80K - $120K < $120K +
Education_Level	Uneducated < High School < College < Graduate < Post-Graduate < Doctorate

Table 4: Ordinal relations in categorical variables

To deal with other categorical columns one-hot encoding was used – a popular technique used to represent categorical variables as binary vectors. This technique has one drawback which is increasing the dimensionality of a dataset, especially if there are categorical variables with a large number of unique categories [5]. This, however, is not our case and the increase in the number of variables is minor.

3.3. Outlier Detection

To find and handle observations in a dataset that significantly stray from the norm or predicted patterns, outlier detection is a key approach in data analysis. Outliers are data points that are far outside the bulk of observations, and by identifying them, analysts can learn important information about potential abnormalities, errors, or odd behaviors in the data. Outliers can result from a number of things, including incorrect data entry, sensor issues, or actual extraordinary events. By ensuring that data patterns and trends are founded on accurate and representative data, identifying and correcting outliers helps to improve decision-making processes, increase the accuracy and reliability of statistical studies, and prevent biased conclusions.

In our approach we went through all the numerical columns and used the function to assign the skewed values to quantiles. The function uses a logarithmic transformation as well as the 5th and 95th percentile. Logarithmic transformation helps address skewed data by reducing the impact of extreme values and making the distribution more symmetrical. The 5th and 95th percentile approach effectively

identifies and handles outliers, ensuring that statistical analyses and models are not unduly influenced by extreme observations. These techniques strike a balance between addressing outliers and preserving the integrity of the data.

3.4. Feature Selection

Feature selection is a technique that can help to reduce the training time of a model. Even though we don't have too many features in our dataset and the training is quite fast, feature selection might also improve the performance of a model by reducing noise or getting rid of irrelevant data [6].

We have chosen a filter feature selection method from *sklearn* library – *SelectKBest.* It is a method used to select the k most informative features from a given dataset. It operates by scoring the features based on their individual relationship with the target variable and selecting the top k features with the highest scores. The scoring of relationships was done using ANOVA (Analysis of Variance) statistical method. ANOVA (Analysis of Variance) is one of the statistical tests that tries to decode the correlation among the various features of data. The main reason why we chose this method is that studies the statistical differences between both numerical and categorical sets of features of the data, and our target variable happens to be categorical, while other features are numerical [7].

To apply the K best features method, we needed to choose a proper value for K. We first obtained scores for all the features ordered from highest to lowest and then evaluated where the score drop is huge (after which point the features have very low importance scores) and decided to stick with the features before the drop. We also plotted a chart to be able to better detect the drop visually. The process is similar to finding the best k for k-means using the elbow method. With this method we decided to continue with k equal to 9. The point is highlighted with a red dot on *figure 4*.

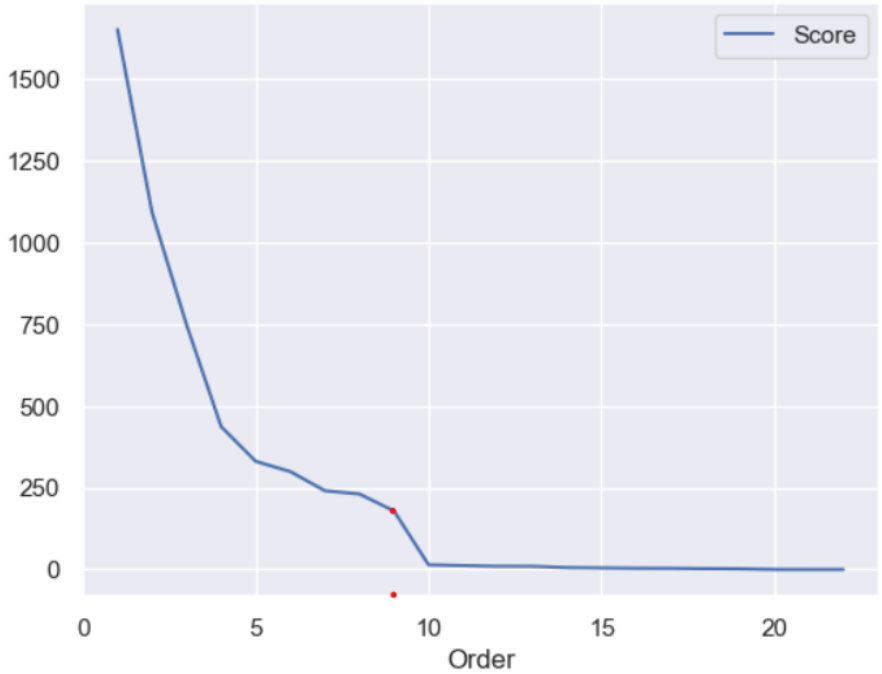

Figure 4: Feature importances.

4. Classification

For classification tasks we decided to compare the following approaches: Logistic Regression, Random Forest, Multi-Layer Perceptron (MLP), and more complex neural networks built using PyTorch.

Focusing on enhancing the F1 metric becomes essential as our dataset is unbalanced (one class having much less occurrences than the other). The F1 score combines recall and precision to provide an impartial assessment of a model's performance. Accuracy alone might be deceiving when dealing with unbalanced datasets since a classifier may obtain high accuracy by only forecasting the majority class and completely disregarding the minority class. We try to balance properly detecting instances from both groups while decreasing false positives and false negatives by giving the F1 metric priority. By taking this method, we can make sure that our model is not biased toward the majority class and that it accurately reflects the characteristics of the minority class, producing classification results that are more accurate and insightful.

4.1. Logistic Regression

4.1.1. Logistic Regression v1: fitting the model (no oversampling, no hyperparameter tunning)

In our pursuit of finding an appropriate baseline model for our classification task, we carefully considered various options and ultimately decided to employ Logistic Regression. Logistic Regression is a well-established and widely used statistical technique that is particularly suited for binary classification problems. Its simplicity and interpretability make it an ideal choice for establishing a benchmark performance level. By fitting a logistic regression model to our labeled training data, we can assess its ability to discriminate between the two classes and establish a baseline accuracy for comparison with more advanced algorithms.

4.1.2. Logistic Regression v2: fitting the model (with oversampling, scaling and hyperparameter tunning)

We tried to improve performance of Logistic Regression with help of feature scaling, handle imbalanced data using SMOTE and simple hyperparameter tuning for following parameters:

- **C** - Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization).
- **solver** - Algorithm to use in the optimization problem.

With these changes we were able to slightly improve F-score of Logistic Regression model (see Figure 5).

	F-score	Precision	Accuracy	Recall
logistic_regression_test_v1	0.771079	0.815692	0.889417	0.742318
logistic_regression_test_v2	0.775123	0.744167	0.848989	0.848905

Figure 5: Logistic regression performance comparison

4.2. Multi Layered Perceptron (MLP)

4.2.1. MLP v1: fitting the model (with oversampling and scaling)

After using Logistic Regression as a baseline for classification, we opted for a Multi Layered Perceptron (MLP) due to its ability to capture complex relationships in the data. MLP's multi-layer structure and non-linear activation functions enable it to model intricate patterns and extract hierarchical representations, making it a suitable choice for tasks involving non-linear data or high-dimensional datasets. By leveraging MLP's capabilities, we aimed to enhance the model's predictive performance and accommodate the complexities present in our classification problem.

4.2.2. MLP v2: fitting the model (with oversampling, scaling and hyperparameter tunning)

Results from MLP strongly depend on the setting of its hyperparameters. Therefore, besides scaling (that is also important to apply before training neural networks) and oversampling we will use random search to select best hyperparameters namely:

- **hidden_layer_sizes** - The ith element represents the number of neurons in the ith hidden layer.

- **solver** - The solver for weight optimization.

- **alpha** - Strength of the L2 regularization term. The L2 regularization term is divided by the sample size when added to the loss.

- **activation** - Activation function for the hidden layer.

After these modifications we were able to achieve the best results so far with F-score higher that 86% (see Figure 6).

	F-score	Precision	Accuracy	Recall
mlp_test_v1	0.776269	0.745352	0.850575	0.847886
mlp_test_v2	0.862125	0.850697	0.922315	0.874990

Figure 6: MLP performance comparison

4.3. Random forest

4.3.1. Random forest v1: fitting the model (no oversampling, no hyperparameter tunning)

An ensemble learning technique called a random forest classifier mixes different decision trees to produce predictions. It is renowned for its adaptability and dependability when handling classification and regression problems. By randomly choosing subsets of attributes and samples from the training dataset, the method creates several decision trees. The input data is individually classified by each decision tree, and the combined outputs of all the trees yield the final forecast. Random forest is a preferred option for many applications because of this ensemble approach's improvement of generalization and decrease of the risk of overfitting.

4.3.2. Random forest v2: fitting the model (with oversampling)

For a random forest classifier, oversampling can be advantageous when working with unbalanced data. Our dataset was rebalanced by oversampling the minority class, resulting in a more representative sample size for the classifier to train from. In addition to addressing the issue of class imbalance, this increases the classifier's capacity to correctly categorize instances from both classes.

4.3.3. Random forest v3: fitting the model (with oversampling and hyperparameter tuning)

Our classifier already produces feasible results, however there are some hyperparameters that can be used to train the model. We haven't taken advantage of those and used the default values so far. Therefore, in this part we used cross-validation implementation - GridSearchCV, which tries all possible combinations of provided hyperparameters values namely:

- **max_depth** - The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **max_features** - The number of features to consider when looking for the best split.
- **min_samples_leaf** - The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
- **min_samples_split** - The minimum number of samples required to split an internal node.
- **n_estimators** - The number of trees in the forest.

The Random Forest classifier has shown superior performance compared to Logistic Regression and Multi-Layer Perceptron on the provided dataset. It excels in handling complex and high-dimensional data by constructing an ensemble of decision trees and aggregating their predictions. It effectively captures non-linear relationships, interactions, and feature importance, which can be crucial in achieving accurate predictions. Additionally, Random Forest mitigates overfitting by using bootstrap sampling and feature randomization. Also, it is a commonly used method when we deal with unbalanced data, which is our case.

	F-score	Precision	Accuracy	Recall
rf_test_v1	0.914900	0.934580	0.955608	0.897815
rf_test_v2	0.914424	0.912815	0.953230	0.916053
rf_test_v3	0.917321	0.919405	0.955212	0.915271

Figure 7: Random Forest performance comparison.

4.4. Neural Networks with PyTorch

We made the decision to implement our own neural network architecture using PyTorch instead of relying on the multi-layer perceptron (MLP) from the scikit-learn library for several reasons. Firstly, PyTorch provides a more flexible and intuitive framework for building and training neural networks. With PyTorch, we have fine-grained control over the network architecture, allowing us to customize each layer

and easily experiment with various activation functions, optimizers, and loss functions. This flexibility is essential when working with complex datasets or tackling challenging tasks.

By implementing our own architecture with PyTorch, we can harness the full potential of deep learning techniques and adapt them to our specific needs, ultimately leading to improved performance and more accurate models.

The biggest challenge we encountered with our custom architecture was overfitting, where the model performed exceptionally well on the training data but struggled to generalize to unseen examples. To address this issue, we employed multiple regularization methods to improve the model's generalization ability.

- **Dropout** - By randomly eliminating some of the neurons during training, or setting them to zero, Dropout is a regularization strategy that aids in preventing overfitting. The nn.Dropout module is used in the code to apply dropout after a few specific levels. Self.dropout1 is applied following the first fully connected layer, self.dropout2 follows the second fully connected layer, and self.dropout3 is applied prior to the final sigmoid activation, for instance. Dropout encourages the network to acquire more robust and generalizable properties, reducing the model's reliance on particular neurons.

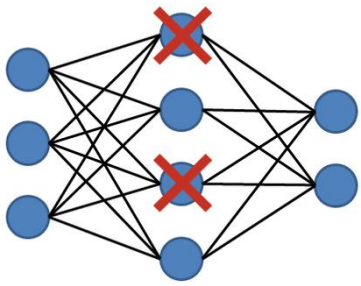

Figure 8: Dropout in neural network.

- **L2 Regularization (Weight Decay)** - Weight decay, commonly referred to as L2 regularization, is a regularization method that modifies the loss function by adding a penalty term depending on the squared magnitude of the model's weights. The loss function is explicitly given L2 regularization in the code. L2 regularization is added by calculating the L2 norm (Euclidean norm) of each parameter tensor using torch.norm(param, 2). This is done after the forward pass and the first loss computation. The loss is then increased by the L2 regularization term multiplied by a lambda value (l2_lambda). As a result, the model is encouraged to weigh less, and overfitting is reduced.
- **Learning Rate Scheduler** - To enhance convergence and avoid overshooting or becoming stuck in local minima, a learning rate scheduler modifies the learning rate during training. The ReduceLROnPlateau scheduler is employed in the code. When the loss stops getting better after a specified number of epochs (patience), the learning rate is decreased. It continuously monitors the loss value. Scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=1, verbose=True) initializes the scheduler. By executing scheduler.step(running_loss) inside the training loop, the scheduler is updated with the running loss value. The model may converge

more quickly and arrive at a better answer with the help of this dynamic change of the learning rate.

```
Epoch 27/40, Train Loss: 0.17008334898336955, Test Loss: 0.2160453349351883
Epoch 00028: reducing learning rate of group 0 to 1.0000e-04.
Epoch 28/40, Train Loss: 0.1686473369467039, Test Loss: 0.24254420399665833
```

Figure 9: Example of learning rate reducing in our training process.

- **Batch Normalization** - The activations of each layer in the neural network are normalized using the batch normalization procedure, which involves subtracting the batch mean and dividing by the batch standard deviation. The training process is stabilized and expedited by this normalization stage. Using the nn.BatchNorm1d module, batch normalization is implemented in the code after a few fully linked layers. Self.bn1 and self.bn2, for instance, are applied after the first fully linked layer and the second fully connected layer, respectively. By introducing some noise to the activations, batch normalization helps to lessen the internal covariate shift and offers regularization effects.

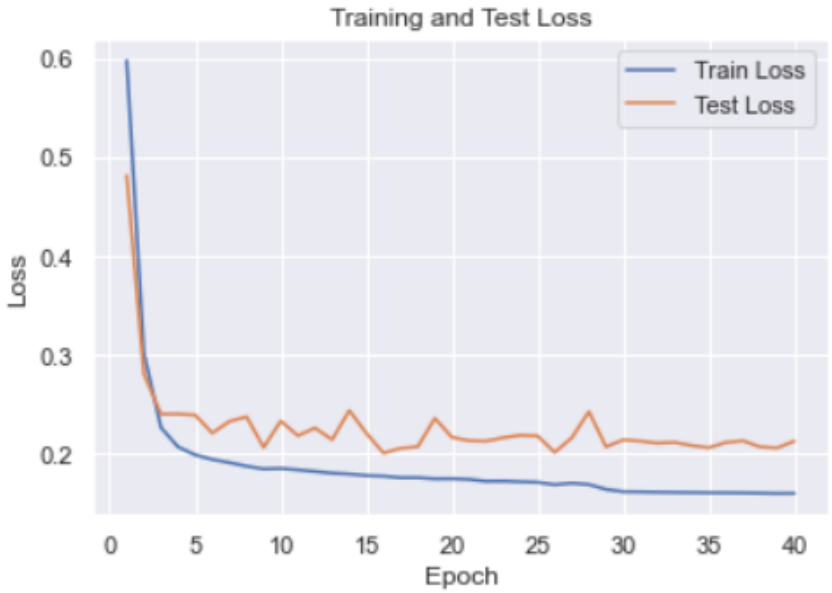

Figure 10: Visualization of train and test loss during the training.

From graph in figure 10, we can see that our model is still overfitting. We added multiple regularization techniques mentioned above. Still, after implementing them we can see that the model has problems with overfitting. This can be mainly because of our dataset. We have only about 10000 samples in total so in training data only 7500 samples. Neural networks usually need much more data to be trained on. We could simplify the architecture, but we already test simple architecture with multi-layer perceptron. In this case, we also wanted to demonstrate a more complex solution although it did not perform the best.

	F-score	Precision	Accuracy	Recall
custom_NN_test	0.868284	0.838625	0.920333	0.911156

Figure 11: Custom neural network performance evaluation.

5. Evaluation

In some circumstances, such as when working with unbalanced datasets and a short sample size, random forest models have the potential to outperform neural networks even if it is a much less complex approach. Random Forest models can perform well in our situation, where the dataset has less samples and is unbalanced, due to their inherent qualities.

Additionally, Random Forest's effectiveness in unbalanced datasets is influenced by its capacity for feature selection and outlier handling. However, because they need a lot of training data to generalize properly, neural networks, despite being strong and capable of learning complex patterns, may suffer with sparse data and unbalanced classes.

The top-performing model, Random_Forest_v3, demonstrates an impressive F-score of 0.917321, indicating a high harmonic balance between precision and recall. It achieves a precision of 0.919405 and a recall of 0.915271, suggesting its ability to accurately identify positive instances while minimizing false positives.

Model Name	F-score	Precision	Accuracy	Recall
Random_Forest_v3	0.917321	0.919405	0.955212	0.915271
Random_Forest_v2	0.914900	0.934580	0.955608	0.897815
Random_Forest_v2	0.914424	0.912815	0.953230	0.916053
Custom_Neural_Network	0.868284	0.838625	0.920333	0.911156
Multi_Layer_Perceptron_v2	0.862125	0.850697	0.922315	0.874990
Multi_Layer_Perceptron_v1	0.776269	0.745352	0.850575	0.847886
Logistic_Regression_v2	0.775123	0.744167	0.848989	0.848905
Logistic_Regression_v1	0.771079	0.815692	0.889417	0.742318

Figure 12: Models sorted based on F-score

6. Bibliography

[1] M. Kaur, K. Singh and N. Sharma, "Data Mining as a tool to Predict the Churn Behaviour among Indian bank customers," *International Journal on Recent and Innovation Trends in Computing and Communication,* 2013.

[2] M. Rahman and V. Kumar, "Machine Learning Based Customer Churn Prediction in Banking," *Proceedings of the 4th International Conference on Electronics, Communication and Aerospace Technology, ICECA 2020,* 2020.

[3] M. R. Ismail, M. K. Awang, M. N. A. Rahman and M. Makhtar, "A Multi-Layer Perceptron Approach for Customer Churn Prediction," *International Journal of Multimedia and Ubiquitous Engineering,* 2015.

[4] K. A. Amuda and A. B. Adeyemo, "Customers Churn Prediction in Financial Institution Using Artificial Neural Network," 2019.

[5] M. A. B. J. G. S. E. Pau Rodríguez, "Beyond one-hot encoding: Lower dimensional target embedding," *Image and Vision Computing,* vol. 75, pp. 21-31, 2018.

[6] K. C. S. W. F. M. R. P. T. J. T. H. L. Jundong Li, "Feature Selection: A Data Perspective," *ACM Computing Surveys,* vol. 50, pp. 1-45, 2017.

[7] U. G. U. Moorthy, "A novel optimal feature selection technique for medical data classification using ANOVA based whale optimization," *Journal of Ambient Intelligence and Humanized Computing,* vol. 12, pp. 3527-3538, 2021.

[8] H. C. Leung and W. Chung, "26th Americas Conference on Information Systems, AMCIS 2020," 2020.