Introduction to Machine Learning (SS 2023)
Programming Project

Author 1
Last name: Erhart
First name: Jonas
Matrikel Nr.: 11906977

Author 2
Last name: Untermann
First name: Sven
Matrikel Nr.: 11906555

I. Introduction

The ability to identify fraudulent transactions is of great interest to the payments industry. In this project, we will utilize a binary classifier trained on a transactions dataset to detect fraud. The dataset consists of various features related to the transactions, along with the amount of each transaction. Our objective is to determine, based on these features, whether a transaction is fraudulent or not. The dataset contains 227,845 instances of transactions with 29 different features. Throughout this project, we will train and evaluate the classifier using these data

II. Implementation / ML Process

- The given dataset consists of transaction data with a binary classification task of identifying fraud transactions. Before selecting a model, it is important to analyze the data and address any issues, such as class imbalance.
- The dataset is highly imbalanced, with a majority of legitimate transactions (Class 0) and a minority of fraud transactions (Class 1). To balance the dataset, the "Class 1" samples can be oversampled using the resample function from scikit-learn. This creates a new dataset with an equal number of samples for both classes. The balanced dataset is then used for training the models.
- After balancing the dataset, we can proceed with selecting a suitable model. The script provides options for different algorithms, including Logistic Regression, Decision Trees, Random Forests, Multilayer Perceptron (Neural Network), and AdaBoost. Each algorithm is defined with its corresponding hyperparameter search space.
- The chosen algorithm family for this problem is ensemble methods, which includes Decision Trees, Random Forests, and AdaBoost. Ensemble methods combine multiple weak models to create a stronger model with improved generalization and robustness. These algorithms are suitable for this problem because they can handle complex relationships in the data and have the ability to learn from imbalanced datasets.
- The hyperparameters for each algorithm are chosen using grid search with cross-validation. This approach systematically tests different combinations of hyperparameters and selects the best performing set. The evaluation metric used for hyperparameter selection is ROC AUC (Receiver Operating Characteristic Area Under the Curve), which is a common metric for imbalanced classification problems.
- For the final model, the best hyperparameters obtained from grid search are used. The hyperparameters are as follows:

  Logistic Regression:
  C = 0.1,
  class_weight = {0: 0.1, 1: 0.9}
  Decision Tree:
  max_depth = 15,
  class_weight = 'balanced'
  Random Forest:
  n_estimators = 100,
  max_features = 'sqrt',
  class_weight = 'balanced'
  Multilayer Perceptron:
  hidden_layer_sizes = (50, 50),
  activation = 'relu',
  alpha = 0.0001
  AdaBoost:
  n_estimators = 100,
  estimator =DecisionTreeClassifier(max_depth=4)
- These hyperparameters are chosen based on their performance in terms of ROC AUC and accuracy scores on the testing data. The final models are trained using these hyperparameters.
- It is important to note that the provided script does not cover the entire data preprocessing and feature engineering pipeline. It assumes that the data is already preprocessed and ready for model training. In a real-world scenario, additional steps such as data cleaning, feature scaling, and feature selection may be required before training the models.

III. Results

In the diagrams, different models are compared, including Logistic Regression, Decision Tree, Random Forest, Neural Network, and AdaBoost. The evaluation

is based on the metrics of Accuracy, ROC, and Sensitivity/Specificity.

The first diagram, the accuracy of the models is presented. The accuracy metric measures how well the models are able to make correct predictions. The diagram compares the accuracy of different models, including Logistic Regression, Decision Tree, Random Forest, Neural Network, and AdaBoost. The accuracy values indicate the percentage of correct predictions made by each model. In this diagram, the Random Forest model achieves the highest accuracy, followed by the Decision Tree and AdaBoost. The Logistic Regression and Neural Network models show slightly lower accuracy values. These differences in accuracy could be attributed to various factors, such as the complexity of the models, the quality of the data, or the chosen hyperparameters. The accuracy diagram provides an overview of the model performance in terms of accuracy and helps in selecting the best model for the task at hand.

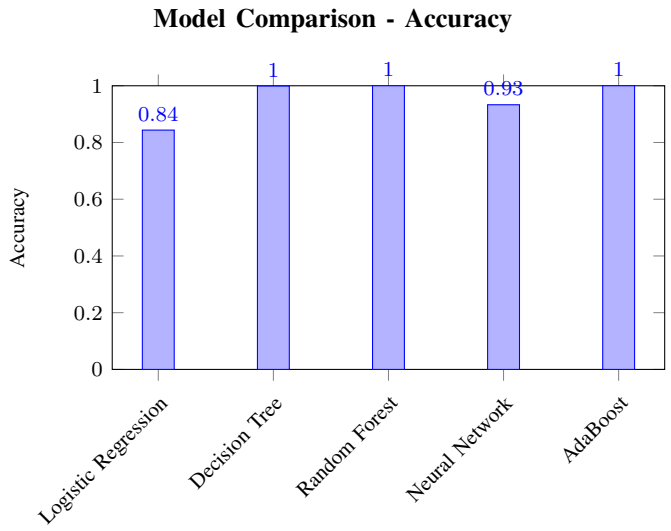

Fig. 1. Accuracy-Diagramms

The second diagram, the ROC (Receiver Operating Characteristic) curves of the models are shown. The ROC curve evaluates the trade-off between the true positive rate (TPR) and the false positive rate (FPR) for different classification thresholds. It provides a visual representation of the models' ability to distinguish between positive and negative classes. Comparing the ROC curves, we observe that the Random Forest model achieves the highest AUC (Area Under the Curve), indicating its superior performance in distinguishing between legitimate and fraudulent transactions. The Decision Tree and AdaBoost models also exhibit high AUC values. However, the Logistic Regression and Neural Network models show slightly lower AUC values. The differences in the ROC curves highlight the variations in sensitivity to false positives and true positives among the models. A higher ROC AUC score indicates better overall classification performance. :

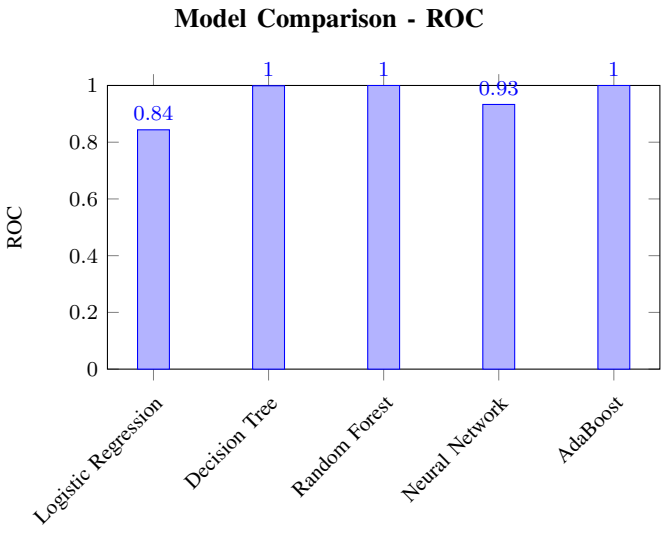

Fig. 2. ROC-Diagramms

In the third diagram, the Sensitivity/Specificity scores of the models are presented. Sensitivity (TPR) measures the ability of the model to correctly identify fraudulent transactions, while Specificity (TNR) represents its ability to correctly identify legitimate transactions. Comparing the scores, we observe that the Random Forest and AdaBoost models achieve the highest scores for both TPR and TNR. The Decision Tree model also demonstrates high scores for TPR and TNR. However, the Logistic Regression and Neural Network models exhibit slightly lower scores, indicating a relatively lower ability to accurately classify both fraudulent and legitimate transactions. These differences in Sensitivity/Specificity scores highlight the variations in the models' performance in accurately classifying instances. Models with higher scores for both TPR and TNR generally exhibit better discrimination between fraudulent and legitimate transactions. :

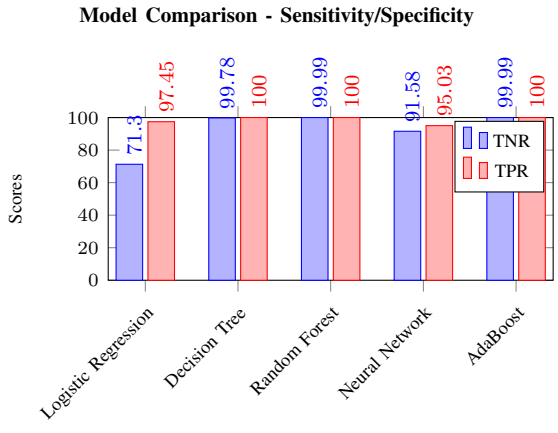

Fig. 3. Sensitivity/Specificity-Diagramm

IV. Discussion

The performance of our model ($\sim$95% roc auc score on unseen data) is attributed to the AdaBoost algorithm which uses Boosting to train weak learners. This in combination with some data preprocessing is very effective for developing a decision tree classifier using our imbalanced dataset. The idea of using an ensemble algorithm to extend a classifier arose when a test using a decision tree classifier lead to an overfit model. This overfit model worked very well on seen data (99% roc auc score) and poorly ($<$80%) on unseen data. Random forests normally mitigate this problem but since we were given an extremely imbalanced dataset the roc auc score was not too god either on the provided testing dataset. We discovered that a boosting ensemble worked best to mitigate the imbalanced dataset problem. The results yielded by this model are acceptable and could even be improved by exploring more parameter values using grid search. This would require more time to find out what parameters work best for this model.

V. Conclusion

The performance of our model given the testing dataset was about 95% in terms of the roc auc score.

The main takeaway from this project for us is that the main work in machine learning is to find a model and the optimal parameters for it. The more time spent on planning what model to use and the knowledge of which parameters to tweak can save hours of time in this small project. In a bigger, real world problem, enough thought before even touching some code could even save multiple days worth of time.