Introduction to Machine Learning (SS 2024)
Project: Programming Project

Author 1
Last name: Rieser
First name: David
Matrikel Nr.: 12141689

Author 2
Last name: Sillaber
First name: Cedric
Matrikel Nr.: 12211124

I. Introduction

The task of this project is to implement machine learning models to detect fraudulent transactions. The problem type is a binary classification task. Based on the timestamp, amount, and 28 additional features, the model should predict if a transaction is fraudulent or not. The dataset consists of 227,845 instances with 30 features and a flag indicating if the transaction is fraudulent. The dataset is highly imbalanced, with 0.173% of the transactions being fraudulent. There is no information about the 28 features, so it is not clear what they represent.

As the dataset is labeled, we can use supervised learning to solve this problem. We decided to use three methods for this problem. The first choice was to use a neural network. Neural networks can extract the most important features from the data and learn complex patterns. The second choice was a decision tree. To achieve even better results, we also tried a random forest classifier.

II. Implementation / ML Process

As mentioned in the introduction, the dataset is highly imbalanced. There are 227,845 data points, and less than 500 are fraudulent, which is about 0.173% of the dataset. So, when splitting the dataset into train and validation sets, we have to work with only a few fraud cases. A naive implementation of the neural network was not able to detect frauds correctly. With an accuracy of 99.83%, most likely, the frauds were not detected. To solve this problem, we decided to do simple data augmentation. As other methods were not adequate, we simply duplicated the fraudulent transactions. This was done as follows: The dataset was split into train and validation sets. Then the fraudulent transactions in the train set were duplicated randomly by a factor. Using this augmentation method, it was possible to give the frauds more weight in the train set, without changing the validation set. Additionally, we scaled the data using a normalization method. This functionality is achieved by calling scikit-learn's 'StandardScaler'.

For the problem, we used two methods: Multilayer Perceptron and Decision Tree.

A. Multilayer Perceptron

The neural network is based on 'pytorch''s 'nn.Module'. For the network, there are three layers: an input layer with 30 nodes, a hidden layer with a dimensionality of 94, and a learning rate of 0.0004301150216793739. The activation function is a ReLU function. The output layer has one node and a sigmoid activation function.

A neural network is suitable for this problem as it can learn complex patterns and extract the most important features from the data. We don't have any information about the features, so a neural network is a good choice. The programmer does not have to make any choices concerning the features. The dataset contains labeled data, so supervised learning and thus neural networks seem a good option.

The choice of hyperparameters was really important and made a difference. In this case, the hyperparameters were the factor used for the data augmentation, the number of nodes in the hidden layer, learning rate, and learning epochs. To find the best hyperparameters, we used a brute force method to train using randomly selected parameters. The parameters were randomly selected from a specified range. Given the results, the best hyperparameters were chosen. The final hyperparameters are: factor for data augmentation: 100, number of nodes in hidden layer: 94, learning rate: 0.0004301150216793739, learning epochs: 45.

B. Decision Tree

The Decision Tree is based on the 'sklearn' library.

Decision Trees are suitable for this problem as it can easily divide the multi-dimensional space given by the amount of features into distinct sections each with a given probability of being a fraud. We thought that frauds would usually have some feature that would spike in comparision to a normal transaction which the decision tree could very easily use to divide the space of frauds and non-frauds.

Choosing the hyperparameters is the most important part of training a decision tree, since the learning process determines the structure of the decision tree. We choose to ignore the ”minimum amount of samples per split/leaf”, ”maximum leaf nodes” and ”minimum weight fraction leaf” hyperparameters since it did not return any good results because of the small amount of frauds. We choose a brute-force approach after doing some experimenting on what values are even sensible to use for the remaining hyperparameters. The final hyperparameters are: Criterion: gini, Splitter: best, Max Depth: None, Max Features: 15, CCP Alpha: 0.0, Min Impurity Decrease: 1e-05, Random State: 25939. The train_test_split method was also supplied the same random state as the decision tree for this experimentation.

C. Random Forest

We further tested a modification of decision trees - the random forest classification. This method is based on the decision tree but uses multiple trees to improve accuracy. The accuracy was about 99.959%, and the model was able to detect the frauds. This is about the same as the accuracy of the decision tree.

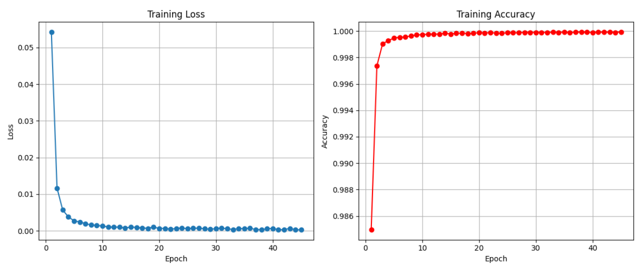
III. Results

During the training of the MLP model, the dataset was split into a training set and a validation set. The training set consisted of 70% of the data, while the remaining 30% was allocated to the validation set.

The final accuracy achieved on the training set was 99.9999%, demonstrating excellent performance in learning from the data. On the validation set, the model achieved an accuracy of 99.9415%, indicating robust generalization to unseen data.

Moreover, the ROC AUC score obtained on the validation set was 96.62923%. This metric signifies the model’s ability to distinguish between fraudulent and legitimate transactions, with an accuracy of detecting approximately 96.62923% of fraudulent activities.

The figure below shows the training loss for the MLP. In the first few iterations, the loss is decreasing rapidly, but this trend slows down. This is the typical loss curve for an MLP.

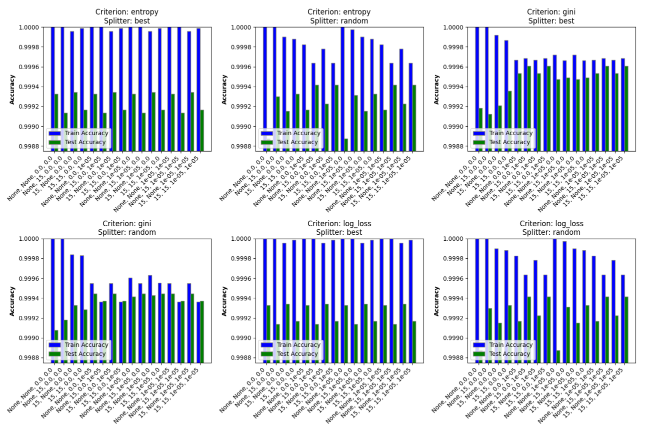


Using the test set on the JupyterHub, which consisted of different data, showed only ROC. The ROC for the train set was 99.2065%, and the test score was 98.7157%. This is a solid result.

The decision tree was also trained using a 30%/70% Train/Test split. The decision tree has a validation accuracy of 99.96%. The train accuracy is about 99.96865%. This indicates that the decision tree is able to accurately predict frauds over the whole dataset.

From the confusion matrixes of the test set we can see that it only mislabelled 4 of 116 frauds and 23 of 68238 non-frauds.

Following is a picture of the decision trees performance on training and test set with various different configurations of the hyperparameters. The plots are grouped by the ’Criterion’ and ’Splitter’ parameters and on the X-Axis you can see the choices for ’max_depth’, ’max_features’, ’ccp_alpha’ and ’min_impurity_decrease’ in that order respectively:



As you can see if no pruning and tree limitations are specified the tree can get a 100% accuracy on the training set by overfitting the training data perfectly. In the plot in the top right subplot you can see the best-performing models as bar 6, 8, 14 and 16. All of these hyperparameter configurations resulted in the same decision tree. They all have the ”gini”-Criterion, ”best”-Splitter and ”min_impurity_decrease” in common. In future experimentation one could focus in more on training using these values and varying the values of the other hyperparameters.

As a naive baseline predictor, we present a model that only outputs 0, meaning no fraud. Because the dataset is imbalanced and data augmentation does not affect the validation set, there are only a maximum of 0.15% frauds in the validation set. Thus, the naive predictor has an accuracy of at least 99.85%. For the ROC score the naive predictor has a 50%/50% chance to predict the correct classification. Both implementations are significantly better than the naive baseline predictor. The decision tree and random forest classification can detect frauds.

IV. Discussion

As mentioned above, our models are significantly better than a model that only predicts no fraudulent transactions.

A reason we think the decision tree perfomed so well is that frauds usually have something that makes them stick out. This enables the decision tree to learn which tresholds for which features indicate a fraud. This theory is also supported by the structure of the decision tree as it only uses about a third of the features defined in the dataset, indicating that these features spike when there is a high likelyhood of a fraud. The model is also remarkably small, coming in at a little more than 2KB in pure text form or 6KB as a pickle jar.

The neural network approach did not work as well as the decision tree on the train and validation set. However, the ROC for the dataset on the JupyterHub was really good and even better than the decision tree. This could have been improved by further hyperparameter optimization as well as experimenting with more hidden layers. The model size is also notably small with only 14 KB as a pickle jar.

We also tried removing the timestamp from the dataset since the time of a transaction likely does not hold any information about its fraudulence, but it did not improve the model's accuracy and caused problems with the test harness on JupyterHub.

For the decision tree we also tried varying the test/train split but this did not seem to improve the model accuracy and only elongated the searching process.

V. Conclusion

Overall, while the decision tree excels in both accuracy and model size efficiency, the neural network demonstrates a strong potential with superior ROC scores. Both models significantly outperform a naive baseline predictor. However, due to the limited number of fraudulent transactions in the dataset, the models are not yet perfect. Before deploying these models in a real-world application, further experimentation and optimization are necessary.

Our main takeaway was the contrast between traditional coding and machine learning. Machine learning involves lots of trial and error to identify the right model and optimal hyperparameters. While theoretical knowledge is essential for building a foundational understanding of the model and knowing which techniques to apply, experimenting with different configurations is crucial.