Introduction to Machine Learning (SS 2025)
Programming Project

Author
Last name: Moskaliuk
First name: Danylo
Matrikel Nr.: 12234444

I. Introduction

For this report I consider the task of classifying transactions to detect frauds in the financial dataset. The given dataset contains a total of 227.845 samples, each representing a single transaction with 28 anonymized signal components and raw features like Time and Amount. The target variable is binary, indicating whether a transaction is a fraud (1) or valid (0). The dataset is highly unbalanced: containing only 394 transactions that were labeled as fraudulent, which results in only 0.173% of the whole dataset.

Accordingly, this is a binary classification task, with each transaction corresponding to being fraud (Class = 1) or legitimate (Class = 0).

II. Implementation / ML Process

A. Data Preprocessing

I removed low-variance features (those with standard deviation $< 10^{-6}$) to improve model stability and reduce noise. Subsequently, I scaled the data using a normalization method, which was achieved using StandardScaler from sklearn.preprocessing, that helped us by centering the data (so the mean becomes 0) and scaling it (so the standard deviation becomes 1).

Moreover, I have split the data into train, validation and test sets. Out of the total samples, 5% were assigned to the test set, 20% to the validation set and the remaining samples to the train set respectively.

B. Handling Class Imbalance

As mentioned, the dataset is highly unbalanced. With that, a classifier can learn to ignore the fraud (minority) class entirely (predicting ”no fraud” for all samples) and still achieve more than 99% accuracy, with 0% recall on frauds.

To address this problem, I have decided to utilize class weighting during training, which would penalize misclassifications of minority fraud class more heavily, encouraging the model to pay closer attention to the rare fraud cases. In this case, I assign the class weight for the ”legit” transactions to 1, and for the ”fraud” it is a subject of hyperparameter search, starting from a minimum of 1.5.

C. Used Methods

I applied supervised learning techniques, namely a logistic regression and a multilayer perceptron to solve the given problem as the dataset is labeled.

1) Logistic Regression: This model represents a linear classifier based on the sigmoid function. It is implemented using scikit-learn’s LogisticRegression and optimizes a cross-entropy loss, which corresponds to a negative log-likelihood. It is based on the *liblinear* solver, which is the only solver in scikit-learn that supports both L1 and L2 regularization.

It is simple and interpretable and serves as a strong baseline in binary classification. It assumes linear separability, but with some modifications can provide us with good results having much less overhead in terms of complexity compared to other methods.

2) Multilayer Perceptron: This model represents a feed-forward neural network, a nonlinear classifier. It is based on PyTorch’s ’nn’ module and consists of an input layer, 2 hidden layers with *ReLU* activation functions applied after each hidden layer and an output layer with *sigmoid* function applied to output probabilities. The loss function applied is *BCEWithLogitsLoss*, which computes cross-entropy loss. Here I also utilize our fraud_class_weight hyperparameter to mitigate the class imbalance. Later, I also apply a sigmoid to the logits. I also utilized the Adam optimizer for training the model, due to its robustness in handling the sparse gradient. Training is performed using mini-batch gradient descent, number of epochs is a tunable parameter.

Neural networks are generally more powerful than Logistic Regression, as they can automatically extract important features from the data and learn more complex patterns, solving the curse of dimensionality. The main disadvantage is that they are more complex and contain more hyperparameters, which makes it harder to select a good model.

D. Hyperparameters

As the dataset is imbalanced and accuracy does not reflect the true performance of the models, I rely on the *ROC AUC* score on the validation set during hyperparameter tuning.

For each hyperparameter I define either a numerical range *[a,b]* or a discrete set of choices $\{c,d,e,...\}$.

1) Logistic Regression: To find the hyperparameters, I used randomized hyperparameter search with ParameterSampler from sklearn.model_selection with 50 randomly sampled combinations, which were then compared.

Among hyperparameters, I had: *C* stands for the inverse of regularization strength (final choice: 0.1336), *penalty* for the type of regularization (final: 'l2'), *fraud_class_weight* to address the class imbalance (final: 15.263).

2) Multilayer perceptron: To find the hyperparameters, I used randomized hyperparameter search with 60 trials. Each trial samples a new random set of hyperparameters, trains a model using it and evaluates the model on the validation set of the data.

Among hyperparameters, I had: *hidden_size* stands for the number of neurons in the hidden layer (final choice: 32), *batch_size* for the number of training samples per batch (final: 32), *learning_rate* controls the gradient's step size (final: 0.0007360), epochs for the number of training iterations over the dataset (final: 5), *fraud_class_weight* is used with *BCEWithLogitsLoss* to penalize fraud misclassifications more (final: 3.5440).

III. Results

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0 (Legit)	0.9996	0.9995	0.9995	45490
1 (Fraud)	0.7143	0.7595	0.7362	79
ROC AUC Score (Validation): 0.9636				

TABLE I
Validation Set Evaluation Metrics for Logistic Regression

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0 (Legit)	0.9997	0.9994	0.9996	170588
1 (Fraud)	0.7251	0.8407	0.7786	295
ROC AUC Score (Validation): 0.9865				

TABLE II
Train Set Evaluation Metrics for Logistic Regression

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0 (Legit)	1.00	1.00	1.00	45490
1 (Fraud)	0.85	0.76	0.80	79
ROC AUC Score (Validation): 0.9775				

TABLE III
Validation Set Evaluation Metrics for MLP

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0 (Legit)	1.00	1.00	1.00	170588
1 (Fraud)	0.86	0.86	0.86	295
ROC AUC Score (Validation): 0.9956				

TABLE IV
Train Set Evaluation Metrics for MLP

I trained and validated 2 types of classifiers: Logistic Regression and a MLP. The performance was measured using

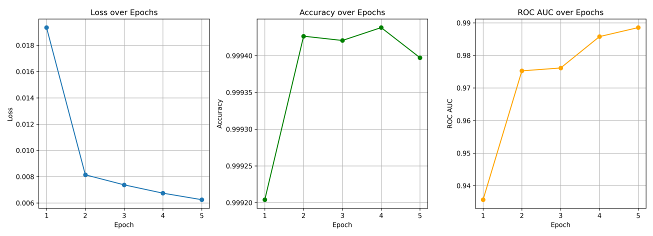

Fig. 1. Training metrics of MLP over epochs

metrics, that are suitable for unbalanced classification tasks, namely *ROC AUC*, *Precision*, *Recall*.

Table I demonstrates, that Logistic Regression is capable of detecting legitimate transactions with a high precision and recall. Fraud detection on the validation set is also reasonably strong with ∼71% precision, meaning most fraud predictions are correct and ∼75% recall, meaning the majority of frauds get captured.

Table II in turn shows, that Logistic Regression performs even better on the minority class on the train set as expected. Nevertheless, the validation performance is not far behind.

Observing the performance of MLP, we can see, that it outperforms logistic regression across most metrics, especially in fraud precision, F1-score and ROC AUC.

Table VI and Table IV demonstrate, that it performs perfectly on legitimate transactions and is at the same time robust in fraud detection both on the train and validation sets. The F1 score of 0.80 and the ROC AUC of 0.9775 outperform logistic regression (0.736 and 0.9636, respectively) on the same validation data split.

Figure 1 demonstrates the training process of the MLP over the selected 5 epochs, which shows that the model is learning effectively from the imbalanced data. The training loss decreases steadily over the epochs, from ∼0.019 to ∼0.006. The ROC AUC also improves and even significantly, from ∼0.93 to ∼0.99. The training process is converging and overall quite stable.

IV. Discussion

These results demonstrate that both the logistic regression and MLP models were effective at classifying and distinguishing fraudulent transactions from legitimate and are much better than a naive model that just marks every input as legitimate, with MLP slightly outperforming logistic regression across most evaluation metrics.

The performance achieved could be explained by some important factors. First of all, I used the class weighting, which turned out to be essential, as without it a given model would just turn into a naive classifier, almost always predicting the majority class and failing to detect frauds. Secondly, the use of ROC AUC instead of, for example, accuracy as the optimization metric contributed as well, as accuracy is not really an appropriate metric for the given problem. Moreover, using MLP, as we can see, enabled me to capture more complex patterns, which contributed to improved results for fraud detection.

I also tried several potential improvements, which I discarded in the end. First, I tried oversampling the fraud class, which led to overfitting and poor generalization for my implementation. I also tried to train the MLP without class weights, which resulted in a very poor recall for fraud transactions, which confirms the necessity of this technique. In order to try to achieve even better results, one could also try to experiment with deeper MLP architectures and with threshold tuning relative to precision-recall metrics, which might further improve the balance between false positives/negatives.

V. Conclusion

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0 (Legit)	0.9997	0.9996	0.9996	11373
1 (Fraud)	0.7727	0.8500	0.8095	20
ROC AUC Score (Validation): 0.9781				

TABLE V
TEST SET EVALUATION METRICS FOR LOGISTIC REGRESSION

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0 (Legit)	1.00	1.00	1.00	11373
1 (Fraud)	0.94	0.85	0.89	20
ROC AUC Score (Validation): 0.9929				

TABLE VI
TEST SET EVALUATION METRICS FOR MLP

The MLP achieved a *ROC AUC* of 0.9929 on the unseen test set, with a fraud class precision of 0.94 and recall 0.85. This demonstrates that the model managed to generalize in a positive way. The logistic regression also performed competitively, with *ROC AUC* of 0.9781, making it a good baseline.

Testing the models on the JupyterHub, I could only observe the *ROC AUC* scores there. The MLP achieved the *ROC SCORE* of 0.9918 for the train and 0.9903 for the test dataset, which confirms the results observed on our test set from the original data. The logistic regression model got *ROC AUC* of 0.9814 for the train set and 0.9857 for the test set, which also confirms the results.

Overall, I consider these as solid results, considering the extremely imbalanced dataset. The main takeway of this project is that handling class imbalancing (with a help of class weighthing) and choosing appropriate evaluation metrics (ROC AUC instead of accuracy) are essential for solving such problems as a fraud detection one. Relatively simple and intuitive model, such as a logistic regression can perform well with proper hyperparameter tuning, but neural networks such as MLP can even further improve the results by capturing more complex patterns in features.