Introduction to Machine Learning (SS 2023)
Programming Project

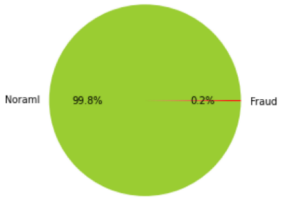
Author 1
Last name: Carp-Popescu
First name: Daniel-Sebastian
Matrikel Nr.: 12101821

Author 2
Last name: Terente
First name: Elizaveta
Matrikel Nr.: 12046958

Author 3
Last name: Goldin
First name: Emanuele
Matrikel Nr.: 12135754

I. Introduction

The main task of this project is to demonstrate the capability to identify fraudulent transactions. We will make use of the binary classifier we trained on the transactions dataset to detect fraud.Our dataset contains 227845 rows and 31 columns. Time, 27 features without particular name, Amount and last column : class. Class values can be 0 (usual transaction) and 1 (frauded transaction). Dataset is imbalanced : we have 227451 samples with normal transactions and only 394 with frauded transactions.



II. Implementation / ML Process

Random Forest

1) *Data preprocessing*
In order to be able to train the binary classification model for fraud detection, we split the dataset into features and the target variable. The target variable is the transaction class, that indicates whether a transaction is fraudulent or not. Next, a train-test split on the dataset has been performed, reserving 20% of it for testing, while maintaining at the same time the class distribution using stratified sampling. By doing this, it can be ensured that the test set will be representative of the two classes of transactions. Before starting to train the model, we also applied data normalization on the dataset, by using the "StandardScaler". This technique standardizes the features by setting the mean and the unit variance to zero. This helped to "get rid" of features with larger scales that would've dominated the learning process, now there are only consistent features in the dataset. Furthermore, to improve the prediction of our model, we made use of the class weights method. In order to do so, we computed the inverse of the class frequencies in the training set, meaning that the minority class (fraudulent transactions) got higher weights than the majority class.
2) *Classifier architecture*
The random forest classifier is a pre-built model from the *sklearn.ensemble* library. This model uses the concept of **ensemble learning** and as it was already mentioned before, it has decision trees at its base. Ensemble learning is a technique that combines multiple decision trees (in our case) to make prediction on data points. The random forest classifier creates multiple decision trees and each one of them is trained on a different subset of the training data and using a random subset of the features. During training/predictions, the output of each decision tree is combined in order to compute the final output. This approach is suitable for the given task, because it considers the "wisdom" of all the decision trees, meaning that the model can find complex relationships in the dataset, and it can make accurate predictions. Also, to be mentioned that it is more resistant to overfitting than a single decision tree, because of considering all of them.
3) *Hyperparameters*
For the random forest classifier, we use the "random_state" and "class_weights". The random_state parameter has been set to 42 and it wouldn't have mattered if it would've been a different value. This parameter has been used only for reproducibility purposes, allowing the random processes of this classifier to be built in a similar way, meaning that there will be always consistent results. For the class weights, we've computed the inverse of the class frequencies, resulting in 5.495683140892829e-06 for non-fraudulent transactions and 0.0031746031746031746 for frauds. The model performs no explicit hyperparameter search, instead it is using the default ones from *scikit-learn* for the number of decision trees, maximum depth, minimum number of samples for a split, minimum number of samples for a leaf and maximum number of features.

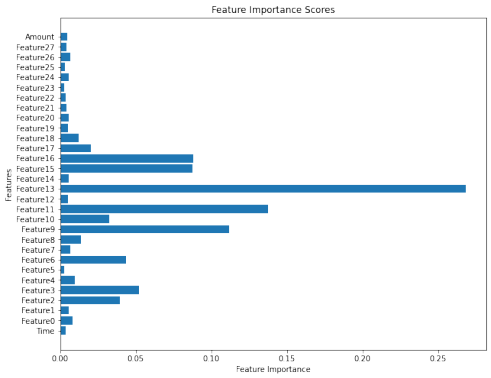


Fig. 1. Naive Bayes Feature Importance Score

Naive Bayes

1) *Data preprocessing*
   - The *StandardScaler* from *scikit-learn* is then used to perform feature scaling .
   - Removing outliers with Z score (threshold value defined as 3). For better performance i separate data into 2 subsets : 1st - all samples with Class 0 and 2nd - all samples with Class 1 and perform outliners remove on this 2 separate sets. Only then concatenate them back together. Otherwise instances of class 1 can be removed as outliners. As a result 32393 outliners of class 0 and 40 outliners of class 1 removed. This increases performance much!
   - To investigate which features are better to choose i used *RandomForestClassifier* from *sklearn.ensemble* and analyzed which features have most correlation with target classes running build-in function for *feature_importances_* (*see figure 1*). By running it multiple times it got clear that the most impact has feature 13. By trying different combinations of features (feature 13 with other features) it was determined that best performance gives ONLY feature 13. Adding other features lowers performance (slightly and not). I save top feature to a model as a parameter and then in *leader_board_predict_fn* i clean data set leaving only top feature). I am performing this operation on the set with already removed outliners!
   - I splitted our pre-processed train data into train set (70%) and test set(30%) to test classifier with *train_test_split* from *sklearn.model_selection*
2) *Classifier architecture*
   GaussianNB classifier from *sklearn.naive_bayes* is used. The architecture of the GaussianNB classifier is based on the Naive Bayes theorem, which is a probabilistic algorithm that uses Bayes' rule to estimate the probability of a sample belonging to a particular class. The "Gaussian" in GaussianNB refers to the assumption that the probability distributions of the features within each class are Gaussian or normal distributions.
   Classifier estimates the mean and standard deviation of feature(i have only one) for each class in the training data. During prediction, it calculates the probability of the sample belonging to each class based on the Gaussian (normal) distribution assumption. The class with the highest probability will be assigned to the sample.
3) *Hyperparameters*
   Default hyper parameters are being used + I am saving top feature in my model.

Logistic Regression

1) *Selection of the model*
   Logistic regression is often considered suitable for solving classification problems with highly imbalanced data. It is a simple and interpretable algorithm compared to more complex models like neural networks. It is based on the logistic function, which allows it to model the probability of belonging to a certain class given the input variables. It can handle imbalanced data reasonably well, especially when the class of interest is the minority class. By adjusting the class weights, you can control the trade-off between precision and recall to focus on correctly identifying outliers. Logistic regression provides coefficients for each input variable, which can help in understanding the relationship between the input variables and the likelihood of being an outlier. This can be valuable in identifying important features or factors contributing to outliers. Finally, it is computationally efficient, making it suitable for large datasets with a high number of features.
   We defined a pipeline which
   - Standardize the data
   - Extract the three more relevant features based on the RFE algorithm
   - Use logistic regression with fine-tuned hyperparameters (see *Fine tuning hyper-parameters* paragraph) as model
2) *Preprocessing* While examining the data-set, we saw that, except for time, each feature is represented by a float64 value. With the exception of the Amount feature, there is a lack of additional information regarding the meaning of each feature, as they are simply labeled as Feature i. Initially, our focus was on identifying any potential errors in the data-set, such as NaN or empty values, but we did not find any error. Subsequently, we analyzed the distribution of the features and observed that most of them closely approximate a Gaussian distribution, except for the Time feature. Since our objective is to detect outliers in the data-set and considering its imbalanced nature, we aimed for a preprocessing phase that does not disturb the outliers. Considering the (almost) Gaussian distribution of the features and the need to preserve the outliers, we used the StandardScaler from the sklearn library

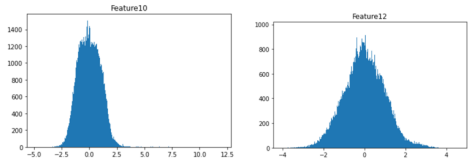


Fig. 2. Examples of features distribution

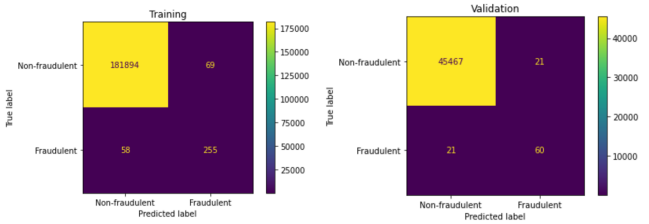


Fig. 3. F1 Score for the Logistic Regression model on the Training set (left) and on the Validation set (right)

to standardize the data-set. To prevent overfitting, we performed feature selection using Recursive Feature Elimination (RFE), assigning weights to the features and iteratively dropping the least significant ones until only three features remained. The choice of three features was based on their similar or potentially better performance compared to using all features.

3) *Fine tuning hyper-parameters*
To fine-tune the hyper-parameters of the model, we used GridSearchCV from sklearn using StratifiedKFold as the cross-validation method. The reason is that stratified K-fold ensure to represent properly the proportion of classes in each fold, extremely relevant in our case to ensure the presence of fraudulent transactions in each fold is represented in a meaningful manner. We tested a combination of the following hyper-parameters: - C: inverse of regularization strength (smaller values specify stronger regularization) - regularization: l1, l2 or none - solver: lbfgs (default), sag, or saga (last two suggested for large datasets) - class weights: probably one of the most relevant parameters in our case, since with the proper weights we can compensate for the imbalance of the data. The metric we selected to evaluate the better combination of hyper-parameters was the F1-score, which is the harmonic mean of precision and recall (formula): due to the fraudulent class being severely underrepresented, having 99,9% non-fraudulent transactions in the dataset, a dummy classifier that always assigns the class "non-fraudulent" would achieve an almost perfect accuracy during the training phase, which made these metrics not suitable to evaluate our model. F1-score keeps into account the precision and recall values, monitoring wrongly assigned labels (False positive/negative) and correctly assigned labels (True positive, true negative), making it a more reliable metric. The combination which perform the better was: C = 0.01; solver sag; regularization l2; class weights: 0.1 for class 0 (non-fraudulent), 10 for class 1 (fraudulent)

III. Results

The **Random Forest** classifier performs great on the dataset, with a precision of 97%. Recalling on frauds is at 75% and the f1-score (mean of precision recall) is at 84%. Meaning that the model is great at avoiding false positives. For non-fraudulent transactions all the values are at 100%.

The **Naive Bayes** classifier performs good on the dataset, with an Accuracy of 99% (but this value is high partly because of dataset imbalance), Precision of 1.0 which means that model avoids false positives very well(For non-fraudulent transactions all the values are at 100%), Recall of 85% (capturing of positive samples) , F1(balance between precision and recall) of 92% and AUC-ROC of 96%, which indicates the classifier's ability to distinguish between positive and negative instances.

The **Logistic Regression** showed worse performance compared to the other models we tested, indicating it may not be the best one to approach the problem. We achieved an F1 Score of 80The drop in the score value may be caused by the regularization, which does not apply during the validation phase.

IV. Discussion

The GaussianNB **Naive Baye**s Classifier demonstrates great performance with a high accuracy of approximately 99.97% and a precision score of 1.0, indicating a low rate of false positives. It effectively captures positive instances with a recall score of approximately 85.47% and achieves a balanced F1-score of approximately 0.92. The AUC-ROC value of 0.96 indicates its strong ability to distinguish between positive and negative instances. Overall, the classifier exhibits robust performance, accurately classifying the majority of instances and effectively capturing positive cases in this particular classification task. The training of a model may require some time(up to 5 minutes), primarily due to the random forest's top feature search process. However, once the model is trained, it can be utilized swiftly and efficiently for making predictions or performing inference tasks. I tested classifier with big amount of different features and combinations of the features(detemined by random forest) to find the best combination (feature 13). I tried to apply weight to classes,because its highly imbalanced, but it didn't make a difference. I tried everything i wanted.

The generalization of **Logistic Regression** exceeded our expectations. We attempted various approaches, but unfortunately, they did not yield the desired outcome. Initially, since based on the correlation matrix all the features appeared to be relevant, we trained the model using all the features, but this led to overfitting and

poor generalization. In an attempt to address this, we experimented with bagging using a 40-60 and an 80-20 split. However, the results were marginally worse, indicating that further fine-tuning of the ensemble through grid search might be necessary. Interestingly, the class weights that yielded the best results were not what we anticipated. We expected a higher weight for class 1.

The **random forest classifier** with class weights outperforms the one where undersampling has been used. Based on the classification report from sklearn.metrics, the undersampling model has a precision of 5% on fraudulent transactions, which is very poor, unlike the class weights method with a precision of 97%.

V. Conclusion

Final result for **Naive Bayes** is Train Dataset Score: 95% on Train Dataset and 97% on Test Dataset i consider as successful. Classifier performs better on the test set than on the training set, it suggests that the classifier generalizes well to unseen data.This scenario indicates that the model has not overfit the training data and can effectively capture the underlying patterns in new, unseen instances. The improved performance on the test set demonstrates that the classifier has successfully learned the relationships between the features and the target variable, allowing it to make accurate predictions on previously unseen data. **Logistic regression** proves to be a valuable choice for tackling classification problems with highly imbalanced data, such as outlier identification, even though it may not possess the same level of power as more complex models. Interestingly, more information does not always lead to improved performance, as the inclusion of all features can result in assigning the majority class label to all instances, illustrating the need for careful feature selection and regularization. The **random forest** algorithm seems to achieve good results, especially when having very imbalanced datasets. Making use of a simple technique like applying class weights, definitely makes it a suitable model for such a problem, unlike the undersampling technique that lead to classifying almost each transaction as being a non-fraudulent one, which at first sight might look like it performs well because of the very high scores.

The main **takeaway** of this project is that machine learning algorithms, such as Random Forest, Naive Bayes and Logistic Regressio can effectively detect fraudulent transactions. First two classifiers demonstrated strong performance in identifying fraudulent instances, with high precision and recall scores.Despite demonstrating lower performance compared to the other models, the Logistic Regression model still achieved more than 90 final score percent. This indicates that while it may not have performed as well as the other models, it still exhibited a high level in classifying fraudulent transactions. The models were able to generalize well to unseen data, indicating their potential for real-world fraud detection applications. Preprocessing techniques contributed to the improved performance of the classifiers much. Overall, this project highlights the effectiveness of machine learning in tackling fraud detection challenges and emphasizes the importance of selecting appropriate algorithms and preprocessing techniques for optimal performance.

TABLE I

ROC AUC score of tested models

Classifier	Train Set	Test Set
Random Forest	99,35%	95,27%
Naive Bayes	94,78%	97,41%
Logistic regression	92,03%	91,99%