Credit Card Fraud Detection

Authors: (45%) Henrique Sousa, 98324 and (55%) Vasco Regal, 97636 Computer Science Students from
Departamento de Eletrónica, Telecomunicações e Informática at University of Aveiro
Tópicos de Aprendizagem Automática
Professor: Pétia Georgieva

***Abstract*—The purpose of this work is to implement and compare machine learning models capable of identifying if a credit card transaction is fraud or not. In this paper we try to obtain a good result with the models developed in the same range of accuracy as the already defined models using a dataset provided by Kaggle. Some changes are discussed, based on the work of others that positively affected our work.**

***Index Terms*—Hyper Parameters Variation, Neural Network, Machine Learning, Naive Bayes, Logistic Regression, Support Vector Machines, Undersampling**

I. Introduction

Credit card payments are now the norm around the world. With this new, more electronic way of payment, there are also different forms of fraud. With this in mind, it is important to ensure that people are not charged for items they did not purchase.

Our project aims to train and test different machine learning models that can identify if a credit card transaction is real or if it is some sort of fraud in order to cancel the transaction and not charge the client. To help us train and test our model we will use a dataset [7] from Kaggle. All the code we developed can be found in our notebook [4].

II. State of the Art

Since this dataset is publicly available on Kaggle, there are several people trying to develop a good model that can identify whether a credit card transaction is a fraud or not. Looking at the most popular datasets, the one that stood out the most was Janio Martinez Bachmann's notebook [1]. In this notebook he uses various predictive models to see how accurate they are in detecting wheter a transaction is a normal payment or a fraud. He starts by creating a 50/50 ratio sub-dataframe of "Fraud" and "Non-Fraud" transactions to use it to train a Machine Learning Model. Janio first understands the data we are dealing with and does some preparation on the dataset the same way we had to do for our model. After the dataset has been prepared for training and testing models he uses four different classifiers and determines their accuracy. He tried Logistic Regression, KNeighborsClassifier, SVC and DecisionTreeClassifier and the accuracy scores were 95.0%, 93.0%, 92.0% and 88.0%, respectively. On Pierre-Alexis LE BORGNE's notebook [2] we found a SVM implementation with 99% accuracy to get these value Pierre tweaked some model parameters, using different kernels and class weights. Another notebook that we found out was Pavan Sanagapati's [5]. His notebook is based on Automated Hyper-parameter Tuning and presents different implementation methods for hyper-tuned parameter finding.

III. Dataset Preparation

The dataset we chose for our model was unbalanced. Using the original dataset provided by Kaggle would cause some Overfitting and Wrong Correlation issues. Since we want our model to be certain when a fraud occurs, using an unbalanced dataset would make it assume that in most cases there are no fraud. This means that we need to do some pre processing in order to make plausible conclusions.

As we can see in Fig. 1 our dataset has mostly non fraud cases. There are 284,315 non-fraud transactions

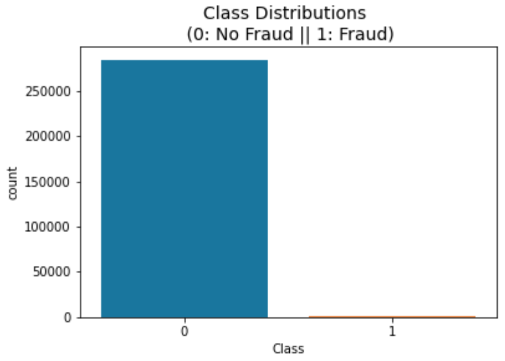

Figure 1: Class Distribution on original dataset

and 492 fraud examples, this represents only 0,17% of all the dataset. Because of that we need to normalize the data to generate a good proportion between frauds and non fraud cases to later develop a model that can properly find the patterns that form an actual fraud transaction

A. *Scaling and Distribution*

We need to create a sub sample of the dataset that offers an equal amount of Fraud and Non-Fraud cases. This will help our algorithm to better understand patterns and properly classify each entry. We will generate a subsample with a 50/50 ratio of fraud and non-fraud transactions. There are 492 cases of fraud transactions in our dataset. We need to choose 492 non-fraud cases which can be done randomly from the over thousands of cases present in the data given to create a new subsample with a balanced 50/50 ratio.

B. *Random Under-Sampling*

In this phase we will remove data to better balance the dataset and avoid overfitting by our model. For this we have to determine how imbalanced is our class and how many instances are considered fraud transactions. We want to have an equal number of fraud and non-fraud transactions in our dataset so for that to happen we can use the 492 fraud causes and randomly choose another 492 non-fraud transactions to create a new balanced dataset. Bringing 284,315 non-fraud causes to only 492 makes us lose a lot of data so there is a risk that our classification models will not perform as accurate as we would like to.

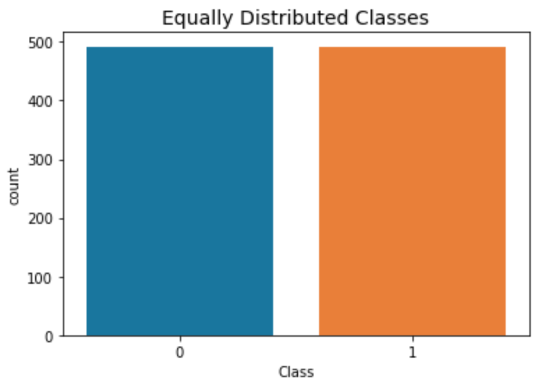

Figure 2: Class Distribution on generated dataset

C. *Correlation Matrices*

Correlation matrices are essential to understand our data. We want to learn if there are features that will heavily influence the decision if a transaction is or is not a fraud. For that, we need to use the subsample we generated above to understand which features have a high positive or negative correlation to fraud transactions. As we can see from figures 3 and 4 there is a big difference between the correlation matrix from the original dataset provided and the one we generated with 50/50 ratio. There are several features that will heavily influence our models decisions

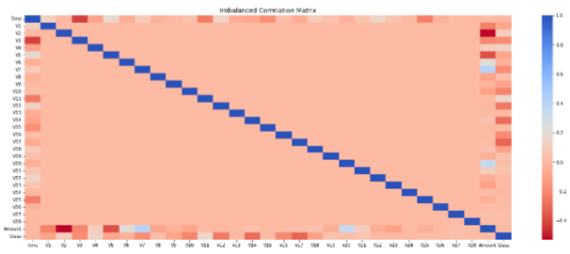

Figure 3: Imbalanced Correlation Matrix

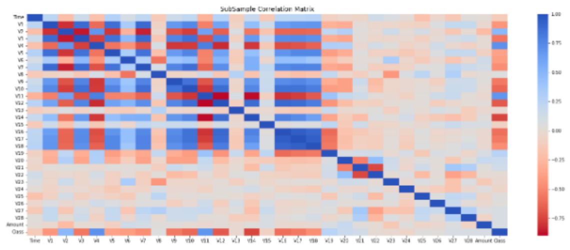

Figure 4: Subsample Correlation Matrix

D. *Anomaly Detection*

There can be some extreme outliers that have to be removed from features and doing this will positively impact the accuracy of our models. We used the Interquartile Range Method to determine what is considered an outlier, for that we separate our sorted from lowest to highest values in four equal sets, our 25th percentile will be the first value from the second set and the 75th percentile the last value from the third set. Now we can calculate the Interquartile Range value by subtracting the 25th percentile from the 75th. With this value in mind and a defined threshold we will get our lower limit by substracting the multiplication of the threshold by the Interquartile Range value from the 25th percentile and our upper limit will be the sum between the 75th percentile and the multiplication of the Interquartile Range value and the threshold defined. Now every value below the lower limit or above the upper limit will be considered an outlier and thus should be removed. Since we want to focus on extreme outliers we have to be careful to how far do we want the threshold for removing this outliers, because the lower the threshold is the more outliers it will remove.

IV. Dataset Analysis

Before starting training and testing machine learning models it's important to understand what we are dealing with. We built a dataset with 947 instances and 30 features, most of those features have no description due to privacy reasons and are named from V1 to V28. The *class* column defines if that set of values represents a fraud (*class* equal to 1) or not a fraud (*class* equal to 0)

V. Models

To get the most out of all the machine learning models used we tested them with the base model, with Hyper-parameter tuning and with K-Fold Cross-Validation on the hypertuned model. The results vary for each one of the types of models used. To facilitate the training and testing of the different models we created a function that trains and analyzes a specific model. Using this function we could get results for every different model just by calling the function with the model itself.

A. Logistic Regression

Logistic regression is a process of modeling the probability of a discrete outcome given some input variables. Usually, this model is used with binary outcomes, something that can take two values, in our case Fraud or non-Fraud. To use the Logistic Regression model on our dataset we used the *sklearn.linear_model.LogisticRegression* library [6], to get the best out of this model we used hyper tuning and got the following parameters combination:

- **solver**: *liblinear* - liblinear is a good choice for small datasets like ours. It is a linear classification that supports logistic regression and linear support vector machines. With the help of a Coordinate Descent algorithm it successively performs approximate minimizations along coordinate directions or coordinate hyperplanes to solve optimization problems.
- **max_iter**: *100* - There will be a maximum of 100 iterations taken for the solvers to converge
- **C**: *1* - A high *C* value tells the model to give more weight to the training data and a lower value indicates that the model should give complexity more weight at the cost of fitting the data.
- **class_weight**: *balanced* - This will use the values of *y* to automatically adjust weights inversely proportional to class frequencies in the input data as $n\_samples/(n\_classes * np.bincount(y))$
- **penalty**: *l2* - This adds a L2 penalty term. L2 adds "squared magnitude" of coefficient as penalty term to the loss function [3]. $\lambda \sum_{j=1}^{p} \beta_j^2$

We then tried the Base Model, the model with Hypertuned Parameters and with K-Fold Cross-Validation and got the result on Fig. 5 As we can see the accuracy

	Base	HyperTuned	K-fold CV
Accuracy	0.921	0.921	0.921
F1 Score	0.915	0.915	0.915

Figure 5: Logistic Regression Results

and F1 Score doesn't change for the different strategies we opt to try and 6 is the same for the three methods too. Looking at the confusion matrix we can confirm that there are a low number of instances wrongly identified and we can consider our model as being accurate.

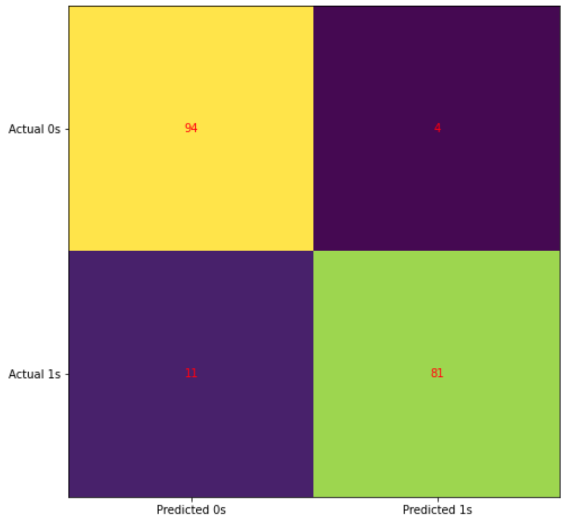

Figure 6: Logistic Regression Confusion Matrix

B. Support Vector Machine

SVM is a non-probabilistic binary linear classifier, which maps data entries to points in space based on classifiers, creating a gap between categories. Then, new entries are mapped to the same space and their category is defined by which side of the gap was the entry mapped to. The implementation of this model was achieved with *sklearn.svm.SVC* [6]. For model parameters, hypertuning was used to find the best combination .

- **C**: *100* - Regulization value.
- **kernel**: *rbf* - Radial Basis Function, a very popular kernel. The kernel decides how will the Kernel matrix be formed.
- **gamma**: *0.01* - On the rbf kernel, this value will be used to calculate the classification coefecient

We tested the base model, the hypertuned model and the K-Fold best estimator:

	Base	HyperTuned	K-fold CV
Accuracy	0.911	0.915	0.915
F1 Score	0.907	0.912	0.912

Figure 7: SVM Results

From the hypertuned model we generated the confusion matrix 8

C. Naive Bayes Classifier

Naive Bayes Classifier is a probabilistic machine learning model that's used for classification task. It is

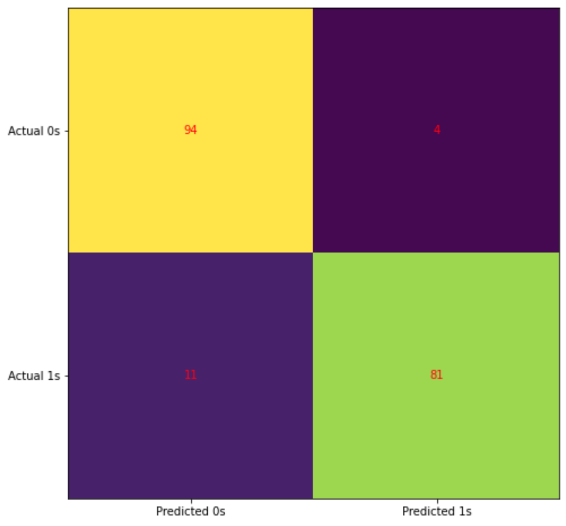

Figure 8: Hypertuned SVM Confusion Matrix

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Figure 9: Naive Bayesian equation

based on the Bayes theorem. First the Naive Bayes algorithm converts the data set into a frequency table, then it creates a likelihood table by finding the probabilities, next it uses 9 to calculate the posterior probability for each class where the one with the highest probability is the outcome of the prediction. Similar to the other models we used *sklearn* library, specifically *sklearn.naive_bayes.GaussianNB* [6]. For this model the ideal parameters found by hypertuning were:

- **var_smoothing**: *1e-09* - *var_smoothing* will artificially add the defined value to the distribution's variance. This essentially widens the Gaussian curve that serves as a *low pass* filter allowing only the samples close to its mean to pass, so it makes it account for samples that are further away from the distribution mean.

	Base	HyperTuned	K-fold CV
Accuracy	0.895	0.895	0.895
F1 Score	0.884	0.884	0.884

Figure 10: Naive Bayes Results

Looking at 10 we can see that similarly to Logistic Regression working on our model parameters doesn't seem to affect much the accuracy and F1 Score of our model. The Confusion Matrix (Fig. 11) and Classification report are the same for the three ways we tried for the Naive

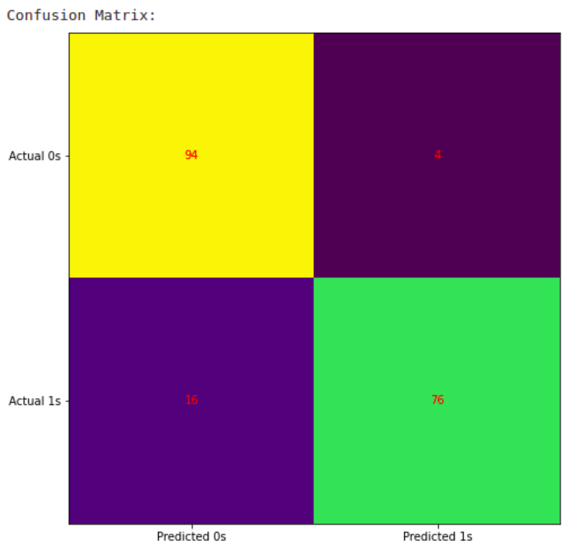

Figure 11: Naive Bayes Confusion Matrix

Bayes Classifier.

D. Neural Network

A neural network is a simplified model of the way the human brain processes information. It works by simulating a large number of interconnected processing units that resemble abstract version of neurons. The best set of parameters found for this model were:

- **activation**: *relu* - This parameter will define the activation function for the hidden layer. The *relu* choice will use the rectified linear unit function, that returns $f(x) = max(0, x)$
- **alpha**: *0.0001* - This defines the L2 penalty (regularization term) parameter.
- **hidden_layer_sizes**: *(12, 12)* - It will have 12 hidden layers with 12 hidden units each.
- **learning_rate**: *invscaling* - This will gradually decrease the learning rate at each time step 't' using an inverse scalling exponent of 'power_t'. $effective\_learning\_rate = learning\_rate\_init/pow(t, power\_t)$
- **learning_rate_init**: *0.001* - This controls the step-size in updating the weigths.
- **max_iter**: *1000* - The solver will iterate until it converges or if it reaches *1000* iterations. Since our solver will be *adam* this will count the number of epochs and not the number of gradient steps.
- **solver**: *adam* - The solver for weight optimization will be *adam*. 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba.

From 12 we can see that the model with K-Fold Cross Validation was the one that got the best accuracy and F1

	Base	HyperTuned	K-fold CV
Accuracy	0.900	0.910	0.911
F1 Score	0.895	0.901	0.910

Figure 12: Neural Network Results

Score for the Neural Network Model and it got the 13 confusion matrix.

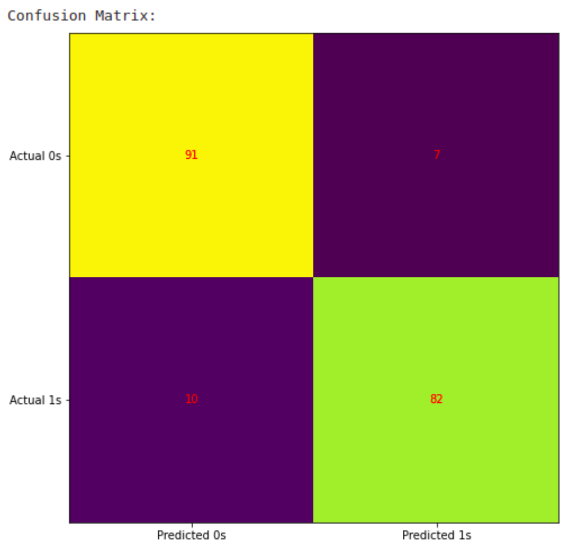

Figure 13: K-Fold CV Neural Networks Confusion Matrix

VI. Model Comparison

To compare the models developed, a boxplot 14 was created with the cross validation scores of each model. By analyzing the boxes, we conclude that although Bayes Network has the best maximum score, SVM is the most consistent performance-wise.

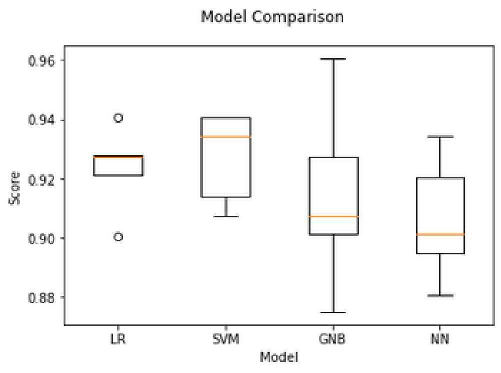

Figure 14: Model Score Comparison

VII. Conclusion

With this article we were able to see that working on the data we have can prove to be useful for the accuracy of our model. It was possible to analyze four different machine learning models: Logistic Regression, Support Vector Machines, Naive Bayes and Neural Networks. All these models were tested in three different ways: the base model, the hypertuned model and the model with K-Fold Cross Validation. We conclude that each model has different accuracies and F1 Scores and that these depend not only on the type of model but also on the processing that is done to train these models.

VIII. Novelty and contributions

Looking at Janio Martinez Bachmann's approach [1] he got the better results when using the SMOTE technique to oversample the dataset. The model's accuracy for Logistic Regression and applying SMOTE was 96.9%, 4.5% more than what our Logistic Regression model was able to achieve. With that said we can conclude that our approach wasn't the most effective one. However, a 90% plus accuracy is still considered good for a machine learning model so our model is usable in a real scenario. Our data pre-processing and unbalancement fix was highly adapted from his work.

Kaggle user Pierre-Alexis LE BORGNE published an SVM implementation with an accuracy of 0.99 [2]. To achieve these results, some model parameters were tweaked which we did not try searching for in hypertuning, like using different kernels and class weights. We can although confirm our conclusions that SVM is the most accurate model, since this is the best results for this dataset in the website.

On Hyper-Tuning, we studied Pavan Sanagapati's notebook [5] which presents implementations of different methods for hyper-tuned parameter finding. We decided to adapt the Grid Search implementation, due to its reliability.

In discussion with our colleagues who also addressed this topic for their article, we found that their approach was in the same page as ours. Their best model was also SVC. However, to test their models they used a much larger dataset than us with more than 200,000 samples and obtained an accuracy & F1 Score of 97,69% & 98,68% for Logistic Regression Model, 99,98% & 99,98% for SVC and 99,95% & 99,95% for the MLP Classifier.

References

[1] Janio Martinez Bachmann. *Credit Fraud - Dealing with Imbalanced Datasets*. URL: https://www.kaggle.com/code/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets/notebook.

[2] Pierre-Alexis LE BORGNE. *Credit card dataset: SVM Classification*. URL: https://www.kaggle.com/code/pierra/credit-card-dataset-svm-classification.

[3] Anuja Nagpal. *L1 and L2 Regularization Methods*. URL: https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c.

[4] Henrique Sousa & Vasco Regal. *Credit Card Fraud Detection*. URL: https://github.com/VascoRegal/creditcard-fraud-detection/blob/main/main.ipynb.

[5] Pavan Sanagapati. *Automated Hyperparameter Tuning*. URL: https://www.kaggle.com/code/pavansanagapati/automated-hyperparameter-tuning.

[6] *scikit-learn*. URL: https://scikit-learn.org/stable/index.html.

[7] Machine Learning Group - ULB. *Credit Card Fraud Detection*. URL: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.