Aldo Hernández

# RANDOM FORESTS WITH PYTHON

**Abstract** *This document presents the implementation of a Random Forest classifier for credit card fraud detection using a dataset of European cardholder transactions from September 2013. Random Forest was selected due to its robust performance in classification tasks and its ability to mitigate overfitting, a common issue with individual decision trees. The highly unbalanced dataset, consisting of 284,807 transactions with 31 features (mostly PCA-transformed), was processed using a 70% training and 30% testing split. Our model implementation utilized 100 decision trees with bootstrap sampling and achieved exceptional performance with an f1-score macro average of 0.91, demonstrating the effectiveness of Random Forest algorithms for fraud detection even with unbalanced datasets. The results indicate that Random Forest is a viable and powerful approach for identifying fraudulent credit card activities in real-world financial data.*

**Keywords** random forests, python, trees, machine learning, classification

# 1. Introduction

Random Forest is a supervised machine learning algorithm for classification. It surges from the issue that decision trees tend to overfit when giving them enough depth [1], so the solution to this is to create several trees that work together.

This algorithm works as follows:

1. Select **k** features out of all **m** columns ($k < m$) and create a decision tree with those **k** features.
2. Create **n** trees changing the number of **k** features, we could also change the sample size for those trees (called *bootstrap sample*).
3. Take each of the **n** trees and test the same classification. We store the result of each tree, resulting in **n** outputs.
4. Calculate the total votes for each class and take the most popular as the final classification of the forest.

This forest is called random since the selection of **k** features is random, along with the sample size for each tree [1].

Some advantages of this algorithm are the following:

- Works well without hyperparameters tuning.
- Works well for classification and regression problems.
- Considerably reduces the risk of overfitting.

Of course this algorithm is not perfect, these are some of its disadvantages:

- Some very specific examples can make the random forest overfit.
- It has a bigger cost than creating a single decision tree model.
- It may need a lot of training time.
- It does not work well with small datasets.

# 2. Methodology

## 2.1. Before typing code

We must download the following .csv file containing the dataset that we will use. This dataset contains transactions made by credit cards in September 2013 by European cardholders. It is important to notice two things: the dataset is highly unbalanced, and almost all inputs are the result of a PCA transformation. Its size is 284807 rows with 31 columns, and the output can be 0 (if the client is "normal") or 1 (if the client committed fraud).

After downloading the required file, we import the following libraries and functions

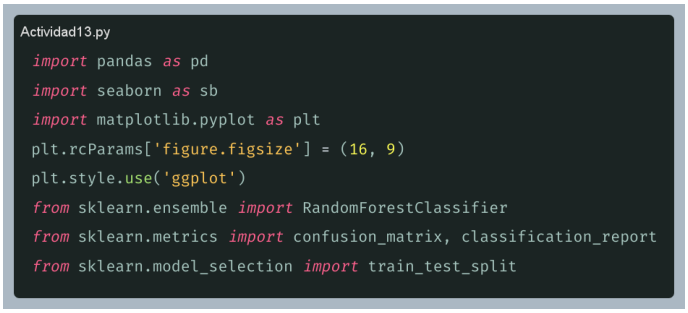


**Figure 1.** Essential imports.

## 2.2. Before creating the model

Now, we parse the .csv file as a Pandas dataframe and perform a holdout cross-validation where we will assign 70% of our dataset to the training set.


**Figure 2.** Splitting the dataset into training examples and testing examples.

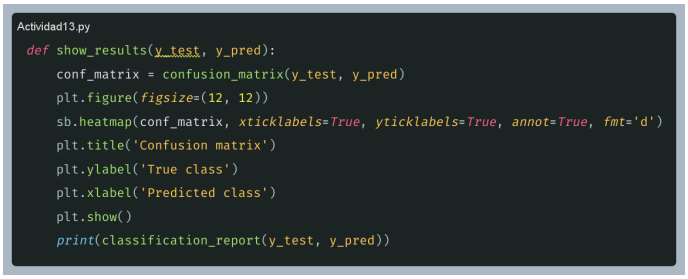


**Figure 3.** Function to show the model results.

Then, we define a function that we will use to print out the model results as shown in Figure 3.

## 2.3. Creating the model

Finally, we create the model using the following hyperparameters:

- Generate 100 trees.

- Use different training sample sizes.
- The maximum quantity of features for each tree will be determined by the square root of the total features.
- Will use 8 cores of our CPU to make training faster.
- Emulate cross-validation in trees to avoid overfitting.

Actividad13.py

```
model = RandomForestClassifier(n_estimators=100,
                               bootstrap=True, verbose=2,
                               max_features='sqrt',
                               n_jobs=8,
                               oob_score=True)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
show_results(y_test, predictions)
```

**Figure 4.** Creation of the random forest classifier model.

## 3. Results

After training the model for almost a minute, we get the following heatmap for the model We can see very good results for this model with only 46 mistakes in all

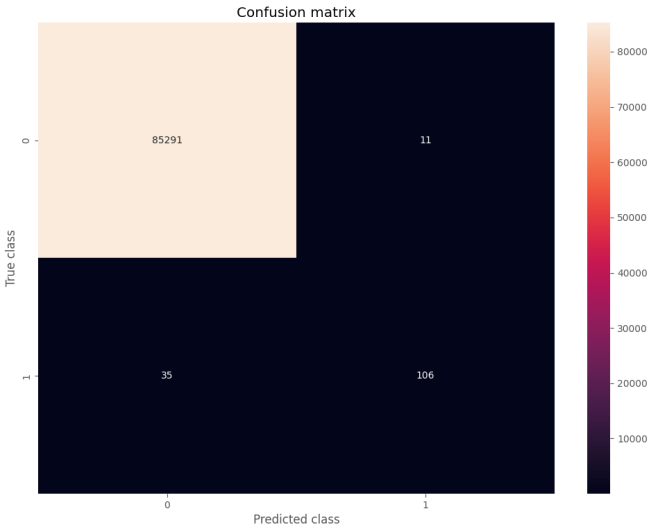


**Figure 5.** Heatmap for predictions vs real examples.

predictions made. Also, the f1-score macro average is 0.91, which is an excellent result. This means that we built a good model even if the dataset was very unbalanced.

## 4. Conclusions

The Random Forest classifier implemented in this study has demonstrated outstanding performance in detecting credit card fraud, achieving an f1-score macro average of 0.91 and making only 46 misclassifications across the entire test set. These results are particularly impressive considering the highly unbalanced nature of the dataset, which typically poses significant challenges for classification algorithms.

The success of our model can be attributed to several key factors. First, the inherent ensemble approach of Random Forest effectively mitigated the risk of overfitting that would likely have occurred with a single decision tree model. Second, the hyperparameter configuration—including 100 trees, bootstrap sampling, and limiting features to the square root of the total feature count—provided an optimal balance between model complexity and generalization capability. Finally, the parallel processing across 8 CPU cores significantly reduced the computational time required for training this complex model.

Our findings confirm the advantages of Random Forest highlighted in the introduction, particularly its strong performance without extensive hyperparameter tuning and its resistance to overfitting. Despite the computational cost being higher than a single decision tree, the substantial improvement in classification accuracy justifies this trade-off for critical applications like fraud detection.

Future work could explore additional techniques to further enhance model performance, such as addressing class imbalance through resampling methods, feature engineering beyond PCA-transformed variables, or comparing Random Forest performance against other ensemble approaches like gradient boosting. Nevertheless, this implementation demonstrates that Random Forest is a powerful and effective tool for credit card fraud detection, capable of delivering high-quality predictions even with challenging, real-world financial datasets.

## References

[1] Bagnato J.I.: *Aprende Machine Learning en Español*. Leanpub, 1.5st ed., 2020.

## Affiliations

**Aldo Hernández**
Universidad Autónoma de Nuevo León, San Nicolás de los Garza,
aldo.hernandezt@uanl.edu.mx

**Received**: ???
**Revised**: ???
**Accepted**: ???