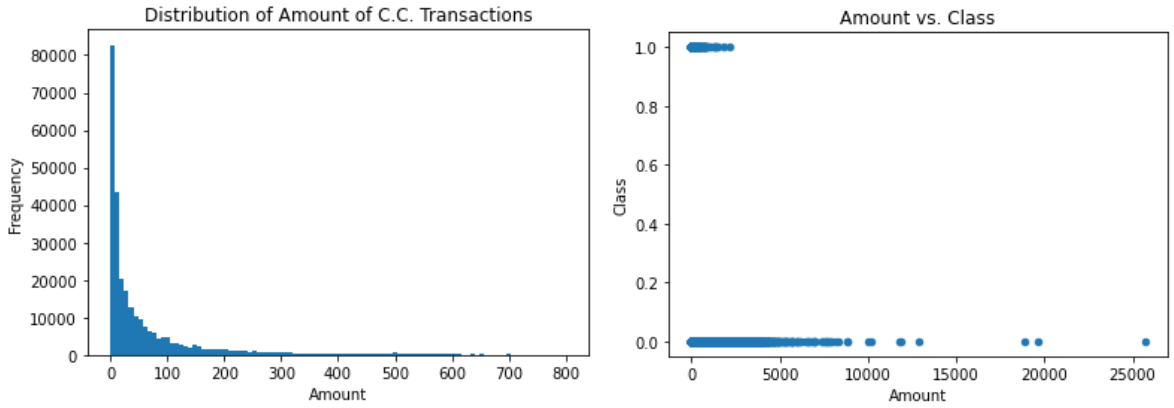
Emilio Padrón Molina

Credit Card Fraud Detection Report

Objectives and Description of the Dataset

For this report, I decided to analyze a dataset which contains information of credit card transactions made by credit cards in September 2013 by European cardholders, and predict whether a transaction was real or fraudulent (minimizing the amount of fraudulent transactions marked as non-fraudulent). This set was obtained from (https://www.kaggle.com/mlg-ulb/creditcardfraud), and given that it contains sensitive information, most features are encoded. Non-encoded variables include, the time lapse with respect to the first transaction of the set, and the amount of the transaction. Also, as one would probably guess, the class (fraudulent-1 and non-fraudulent-0) is available in the set. However, it is worth mentioning this is an unbalanced set given that only 0.172% of all transactions are fraudulent.

In general, the set contains 284807 rows (transactions) and 31 columns (30 features, from which 28 are encoded, and the target/class). Additionally, there aren't any missing values and all the features are floats. Given the lack of transparency of most features, one can mainly analyze the amount corresponding to each transaction. For this, a histogram to analyze the distribution of amounts, and a scatter plot between the amount and the class were built to obtain some insights:



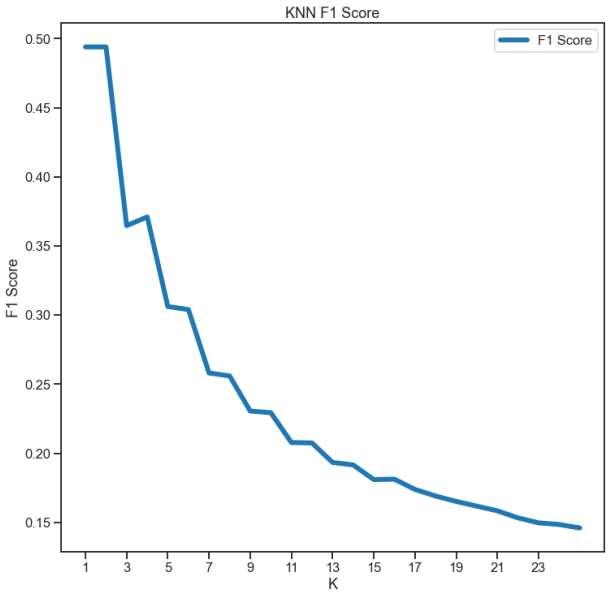
With these two figures, one can see that most transactions cover less than €100 and that the amount is not a reliable indicator of type of transaction as one would expect. This is because the range of the amount of fraudulent transactions cover only a small portion of the range of the majority of non-fraudulent ones. As a consequence, one needs more variables to have a good model, and thankfully, even though one cannot obtain direct insights from the encoded features, they can, nevertheless, be used for modelling.

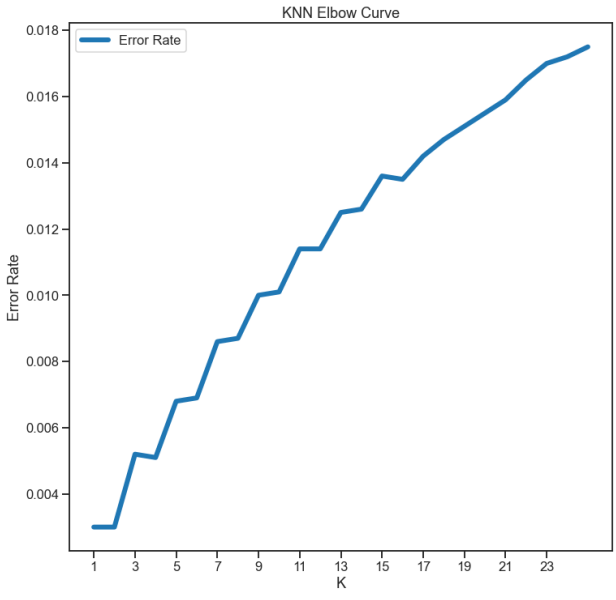
Data Engineering

Given that the set is unbalanced, I decided to use a stratified shuffle split to separate the data into training and testing (15% of the data) sets, and then, scale the data using the MinMaxScaler (only fitted with training data). Afterwards, the training set is put into a resampling process. First, the SMOTE algorithm can be implemented to add observations that corresponded to fraudulent transactions; this increased the number of class-1 samples from 418 to 48333. Next, a Random Undersampling algorithm can be implemented to drop observations from non-fraudulent transactions without a bias; this resulted in a decrease of class-0 samples from 241667 to 48333. This achieves the goal of the balancing of the training set, making it ready for modeling.

Models

With the new training data, I decided to build four different models and optimize their specific hyperparameters. The first model to be constructed is the K-Nearest Neighbors (KNN) for which its most important hyperparameter is the number *k*. To determine the best value of *k*, one can build an iterative algorithm that evaluates the model's performance for different values of this hyperparameter and use the elbow's method to make a final decision. By looking at the graphs below, one can see that the rate of error change significantly slows around *k*=20; therefore, a new model with this hyperparameter was constructed to later see its true performance using the testing set.





For the second model, I decided to build a Logistic Regression. For this type of classifier, the most important hyperparameters are the inverse of regularization

strength and the penalty; therefore, these result in *C*=1 and a L2 penalty, after using the GridSearchCV object.

Next, a Support Vector Machine (SVM) model is coded. In this case, after optimizing the kernel, gamma, and inverse of regularization parameters using GridSearchCV once more, the best model results in *C*=10, *gamma*=1, and an "RBF" kernel.

The last model I decided to construct is an ensemble model consisting on the Logistic Regression and KNN models previously built. For this specific model, one can use the VotingClassifier object with the hyperparameter "voting" set to "hard" (given that the KNN model is not based on probabilities) to try to have a more improved model.

Errors and Scores

After constructing all models, one can put them in action using the testing set and analyze their specific accuracy, precision, recall, and F1 score. The results of these models are summarized in the following table:

:

	Accuracy	Precision	Recall	F1
KNN (k=20)	0.996278	0.300469	0.864865	0.445993
Logistic Regression	0.994289	0.212838	0.851351	0.340541
SVM	0.996934	0.340782	0.824324	0.482213
Voting Classifier	0.998783	0.605769	0.851351	0.707865

Insights and Analysis

Taking into account that the objective of these models is to minimize the amount of fraudulent transactions marked as non-fraudulent, the most relevant score of all is the recall. In other words, the best model for our purpose is the one that has the highest recall. Therefore, one can see from the previous table that, even though all the models have a good prediction power, the KNN model is the best of all given that it has the highest recall score. Nonetheless, both the Logistic Regression and the Voting Classifier are a close second.

Additionally, it is worth noting that the Voting Classifier, composed by the KNN and Logistic Regression model, has a higher precision than the rest of the models. In this case, the precision show how well the model performs in avoiding marking non-fraudulent transactions to be fraudulent. Even though this is not the most important, considering the small difference between its recall and the KNN's, one could also argue that this is the best model. However, to have a definite answer, more testing data would be needed.

Finally, models can always be improved in a way or another. For this one in specific, I consider that the best way to this is by obtaining more data of the fraudulent class. By doing this, one improves the quality of oversampling and decreases the amount of dropped data from undersampling. Also, one could try building a decision tree model and its variations (such as Random Forests) to see if there is a significant improvement in performance.