***Credit Card Fraud Detection***

Machine learning project

Name: Febin Varghese

Data overview

The datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where I have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

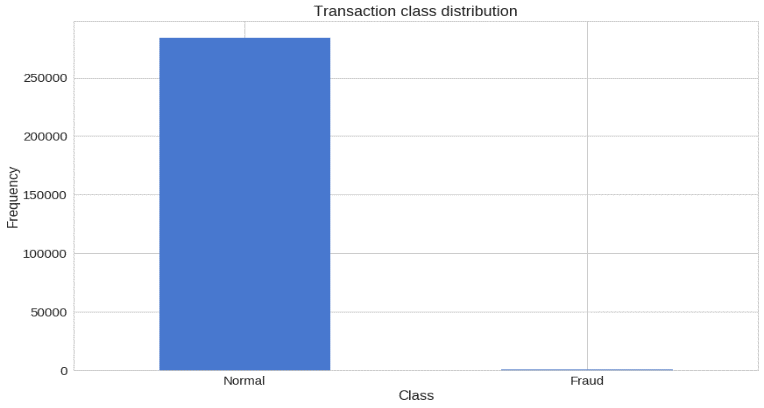
It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, they didn’t provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

**Introduction**

In this project I will use various predictive models to see how accurate they are in detecting whether a transaction is a normal payment or a fraud. As described in the dataset, the features are scaled, and the names of the features are not shown due to privacy reasons. Nevertheless, I can still analyze some important aspects of the dataset.

# Exploration

Here I have 284807transactions different and 31 columns, 2 of which are Time and Amount. The rest are output from the PCA transformation. without any null values



I have a highly imbalanced dataset on our hands. Normal transactions overwhelm the fraudulent ones by a large margin.

# Building the model (neural network)

My Autoencoder uses 4 fully connected layers with 14, 7, 7 and 29 neurons respectively. The first two layers are used for our encoder, the last two go for the decoder. Additionally, L1 regularization will be used during training:

train me for 100 epochs with a batch size of 32 samples and save the best performing model to a file. The Model Checkpoint provided by Kera’s is handy for such tasks. Additionally, the training progress will be exported in a format that Tensor Board understands.

# Evaluation

The reconstruction error in my training and test data seems to converge nicely

The ROC curve plots the true positive rate versus the false positive rate, over different threshold values. Basically, I want the blue line to be as close as possible to the upper left corner. While the results look pretty good, I must keep in mind of the nature of our dataset. ROC doesn’t look very useful for us(acu0.9602)

# Prediction

model is a bit different this time. It doesn’t know how to predict new values. But I don’t need that. To predict whether a new/unseen transaction is normal or fraudulent, I can still calculate the reconstruction error from the transaction data itself. If the error is larger than a predefined threshold, I will mark it as a fraud (since our model should have a low error on normal transactions). (threshold = 2.9)

I created a very simple Deep Autoencoder in Keras that can reconstruct what non-fraudulent transactions looks like. Initially, I was a bit skeptical about whether this whole thing is goanna work out, bit it kind of did. Think about it, I gave a lot of one-class examples (normal transactions) to a model and it learned (somewhat) how to discriminate whether new examples belong to that same class. Isn’t that cool? Our dataset was kind of magical, though. I really don’t know what the original features look like.

### **Random Forest Classifier**

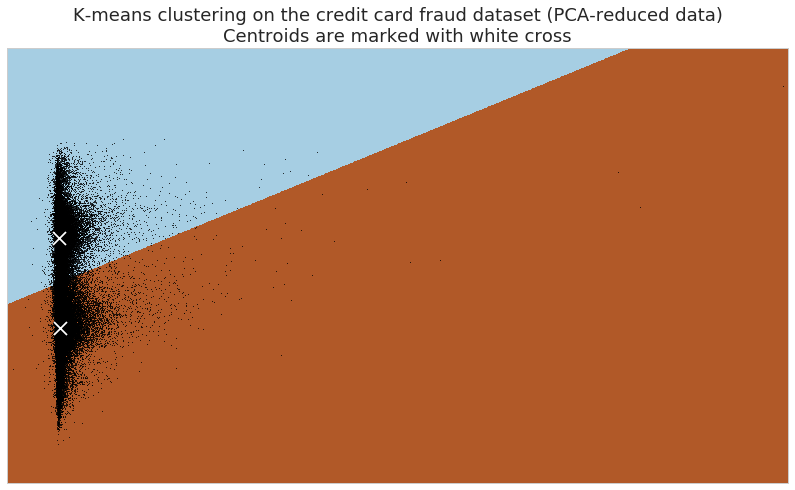
precision recall f1-score support

0 1.00 1.00 1.00 142161

1 0.85 0.53 0.65 243

avg / total 1.00 1.00 1.00 142404

Here I have precision 0.85 and recall 0.53



Accuracy: 0.452913700830966

False negative rate (with respect to misclassifications): 0.0011279877088235866

False negative rate (with respect to all the data): 0.000617106621128454

False negatives, false positives, mispredictions: 58 51361 51419

Total test data points: 93987

The K-means clustering model produced a low accuracy of 54.27%. Of the wrongly predicted transactions, 99.75% were false positives, giving only 0.24% false negatives, or 0.11% of the validation set. However, the false negative rate was only so low due to the extremely low proportion of frauds in the dataset. 112 of the 176 frauds were misclassified as non-frauds, giving this a true accuracy rate of 36.36%. Therefore, K-means would not be the preferred model for this dataset, as it did not correctly predict frauds and it also produced a lot of false positives

**Logistic Regression**

The logistic regression gave the best results. logistic regression gave a great accuracy rate of 99.88%, with 0.079% of the validation set being false negatives (or 0.49% of the number of misclassifications). The logistic regression with oversampling gave an interesting result, as they performed worse than the logistic regression. The accuracy was 98.01%, with 1.56% of the validation set being false negatives (or 3.12% of the misclassifications). Lastly, the logistic regression with balanced weights achieved the best results: although the accuracy was 97.5%, just 0.011% of the validation set resulted in false negatives (or 0.44% of the misclassifications).

Conclusion

Logistic regression outperformed the K-means and neural network and Random Forest. i believe that it is because of how the decision boundary changed with the class weights features. The neural network and random forest were next, and K-means performed the poorest. i believe this is since clustering relies entirely on the similarities and differences of features of the dataset. Since fraud transactions can look very similar to regular transactions, it is difficult to put them into a separate group based on features alone.