Capstone Project

Luca Maiano

Machine Learning Engineer Nanodegree

Date: September 23th, 2018

Definition

Project Overview

As reported by BusinessInsider [1], last year, CreditCards.com found that credit card fraud was on the rise. Both number of frauds and types of credit card scams are more and more. These numbers seem unfortunately destined to grow even more in the future. For this reason, it is important to find a way to automatically recognise anomalies.
A lot of research has been done in order to find a solution to this problem. References [6] and [7] propose solutions using Artificial Neural Networks.

Problem Statement

The aim of the fraud detection system is to detect fraud accurately and before fraud is committed. The goal is to detect least and accurate false fraud detection. The most commonly techniques used fraud detection methods are Nave Bayes (NB), Support Vector Machines (SVM) and K-Nearest Neighbor algorithms (KNN). Each transaction has a set of unique features, such as the value of the transaction, the time at which it occured, issuer and recipient, an id and other sensitive data. The problem will be structured as a classification problem, where each transaction could be classified as "Normal" or "Fraud". With this project, I want to compare the performances of two models: Autoencoders and Random Forests.

Metrics

Precision and Recall are a common metric for this kind of problems; decent values of both help to obtain a precise system that recognise most of the bad transactions. Let us define TP, FP, TN and FN the number of true-positives, false-positives, true-negatives and false-negatives respectively.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

To solve the problem in the right way, we prefer a high value of recall (meaning that a high number of frauds are detected). We also take into account the ROC curve and confusion matrix, that will help us to evaluate the performances.
The aim of a fraud detection system is to correctly identify bad transactions with a low number (possibly zero!) false negatives. We can even accept some normal transaction that is recognised as fraud, but we need to correctly classify bad transactions. Precision and recall will help us to evaluate the behavior of the system. Particulary ROC curve will help us to fix the correct values of precision-recall and confusion matrix will be a useful way to measure the results precisely.

Analysis

Data Exploration

In order to reproduce this kind of problem, I found a useful dataset available on Kaggle [4]. The datasets contains transactions made by credit cards in September 2013 by european cardholders.

This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced and 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. Due to confidentiality issues, the authors cannot 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-dependant cost-sensitive learning. Feature Class is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Exploratory Visualisation

We have a highly unbalanced dataset (Figure 1). In fact, the number of normal transactions is huge w.r.t. frauds.

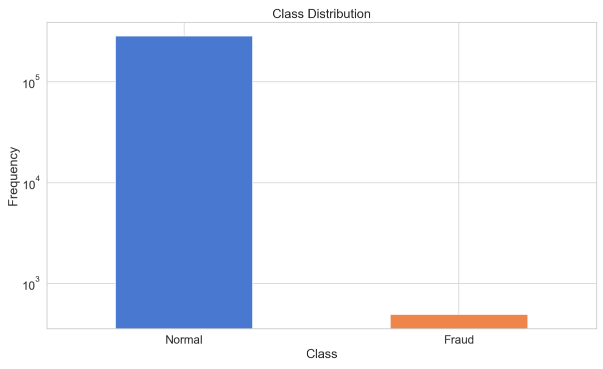


Figure 1: Distribution of classes.

The amount by percentage of transactions for each class is represented by Figure 2. We may assume that this feature is not so important. The distribution is almost the same for the two classes.

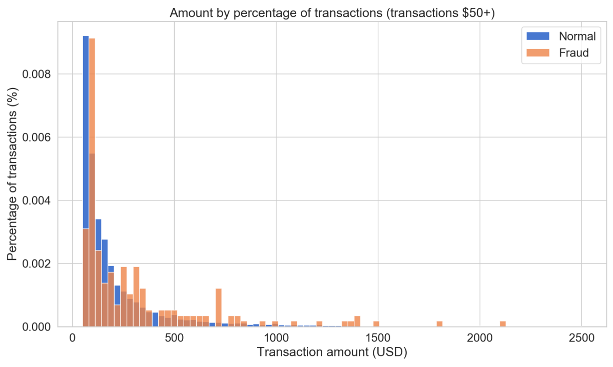


Figure 2: Amount by percentage of transactions.

In order to obtain a better understanding of the significance of each feature, we can analyse the density distribution of each w.r.t. class. The results of Figure 3, can help us to select the right features and to ignore the ones that are not relevant.

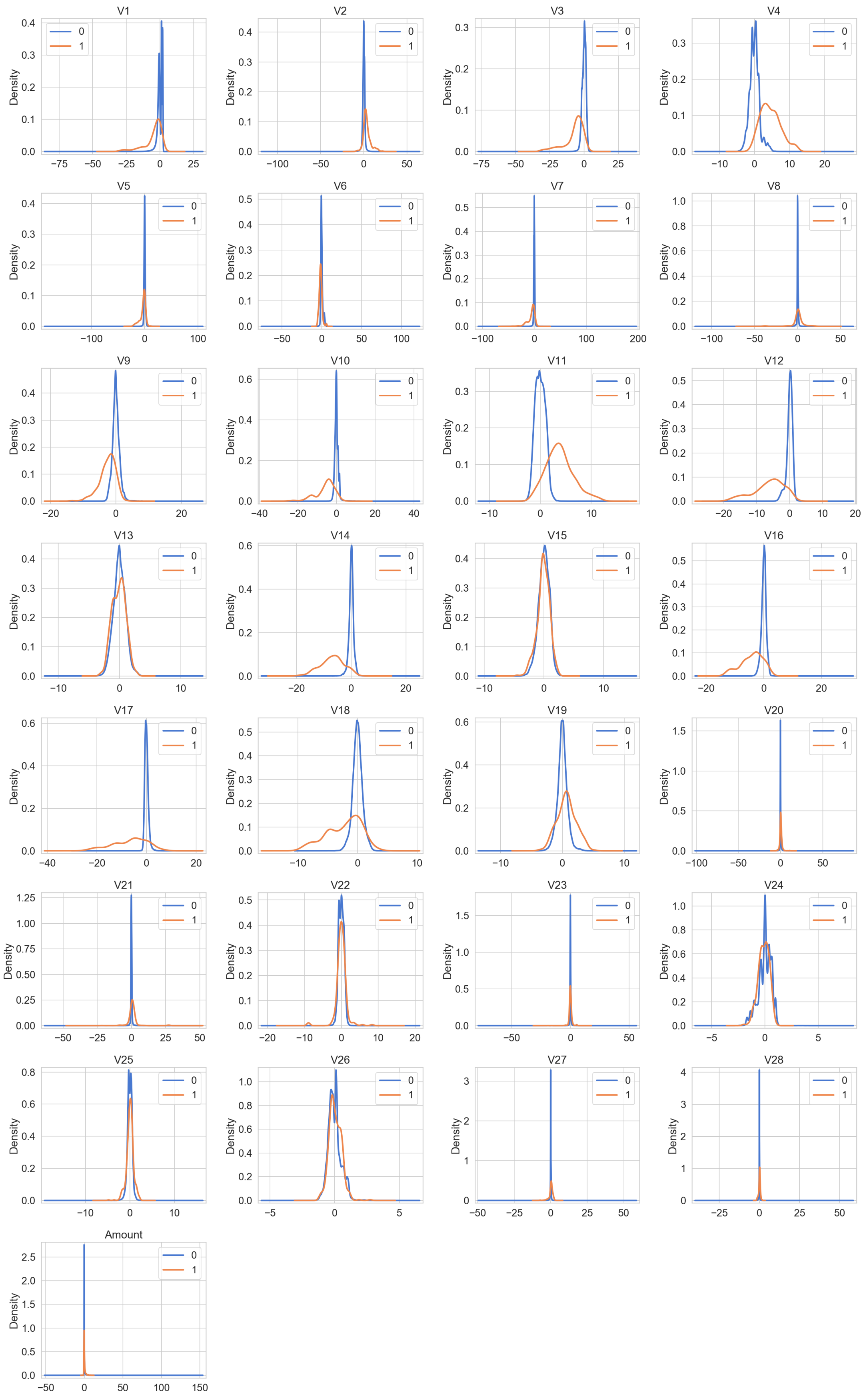


Figure 3: Density distribution of each feature.

Algorithms and Techniques

An Artificial Neural Network (ANN) is an information processing model that was inspired by the way the brain, process information. Each input element, is treated as a large number of highly interconnected processing elements (neurones) working together to solve specific problems. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones. This is true of ANNs as well. The network learns the weights of each connection in such a way that the output error is minimised. A neural network usually is made of an input layer, a variable number of hidden layers and the output layer. You can represent a neural network as a function of funcions. Each layer, represent a particular function, whose input is given by the previous layer and the output is delivered to the next layer. The information flows only from the input to the output.

Autoencoders are a particular class of neural networks that taken an input, "compress" that input down to core features and tries to reconstruct the original input from this squeezed representation. The input is usually mapped into a smaller or larger space from the first half of the network. This first phase is called encoding and produces a modified representation of the original input. The second part of the network "decodes" this representation, and tries to reconstruct the original input in the output layer. In order to set up an autoencoder, we need to select a certain number of layers and activation functions, the learning rate at which the network operates and the encoding dimension of each layer. This model is used for applications in which we want to recognise anomalies or "distortions" on data. We can use the autoencoder as follows: we train it to learn how to recognise normal transactions. During the first phase, the network encode the original input (i.e. a transaction) into a larger space; then in the decoding phase, tries to recognise the representation and to reconstruct the original input.

Decision tree learning uses a decision tree to go from observations about an item (represented as branches) to conclusions about the item's target value (leaves). Trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are an ensemble learning method for classification that operate by constructing a multitude of decision trees at training time trained on different parts of the same training set, with the goal of reducing the variance. In this way random decision forests correct for decision trees' habit of overfitting to their training set. You can set up a random forest calssifier adjusting the number of estimators, max depth, the minimum number of samples required to split an internal node or to be at a leaf node and a specific class weight. This kind of model fits perfectly with classification problems as the one we are trying to solve. Thus it will be used to learn how to classify transactions as normal or fraud.

The main challenge when it comes to modeling fraud detection as a classification problem comes from the fact that in real world data, the majority of transactions is not fraudulent. This brings us a problem: imbalanced data. There are a number of methods available to oversample a dataset. The most common technique is known as SMOTE: Synthetic Minority Over-sampling Technique. Take a sample for the minority class (frauds) for which we want to oversample from the dataset, and consider its k nearest neighbors (in feature space). Then take the vector between one of those k neighbors, and the current data point and multiply this vector by a random number which lies between 0 and 1. Add this to the current data point to create the new, synthetic data point. SMOTE will be used to balance the dataset as a pre-processing step.

Benchmark

In order to measure the quality of this experiment and to evaluate the results, we need some reference model. It seems really difficult to find others that conducted and shared the result of this kind of analysis. After some research, I found two other solutions on the exact same dataset that I selected. Reference [5] recognise frauds using logistic regression with 93.5% of accuracy while reference [6] uses KNN and Naive Bayes classifiers with 99.8% and 97.7% accuracy respectively. The best way to evaluate this project will be comparing the ROC curve that I will obtain with the ones below on Figure 4.

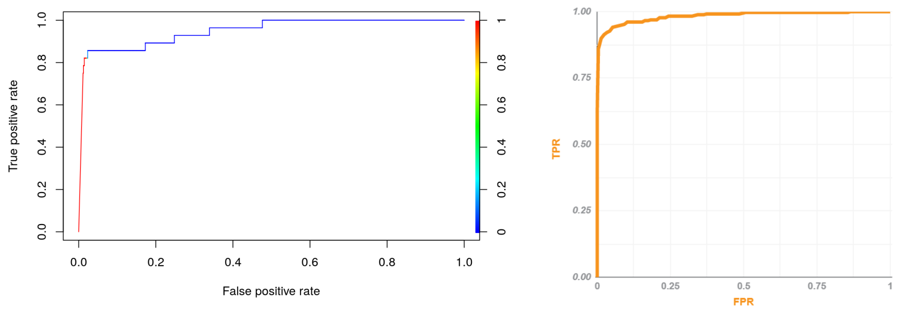


Figure 4: ROC curves from reference [5] and [6] respectively.

I will use those results as benchmark models. I do not expect to overcome the results achieved with KNN or Naive Bayes, but I think that the autoencoders will at least achieve the same results of model [5].

	Normal	Fraud
Normal	99,637%	0,189%
Fraud	0,022%	0,149%

Table 1: Confusion Matrix from reference [5].

In order to compare numerically the results, we also take into consideration the confusion matrices from references [5] and [6]. Notice that the values have been normalised as percentages because of the different size of the testing sets on which the results have been calculated.

	Normal	Fraud
Normal	97,570%	0,035%
Fraud	2,233%	0,161%

Table 2: Confusion Matrix from reference [6].

Methodology

Data Preprocessing

The first thing to notice is that ”Time” does not represent the hour at which each transaction occurred. This feature only enacts the sequence by which the transaction appeared. This feature is not as representative, thus it will be removed.

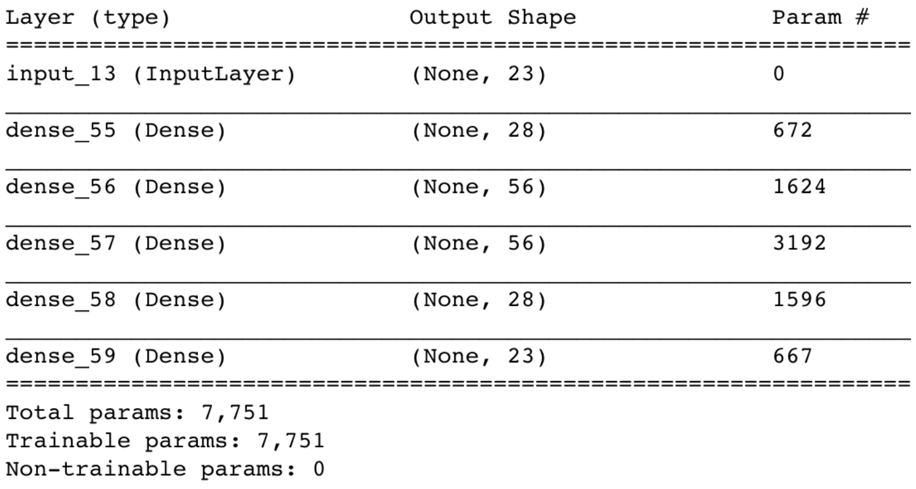
Most of the data result from the product of a PCA analysis. Due to confidentiality issues, the authors cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA. Notice that "Amount" is the only feature that has not been modified. Hence we normalise this feature in order to scale w.r.t. the others.
Now we can judge the importance of each feature from Figure 3. From this visual exploration, we can remove the following features: "V13", "V15", "V22", "V23", "V26" and "Amount".

Implementation

The classifiers were trained on the preprocessed training data. This was done in a Jupyter notebook (titled Autoencoders Credit Card Frauds Detection) using Python 3. The following libraries where used:

- keras: to implement the autoencoder;
- sklearn: to implement the random forest and metrics;
- imblearn: to implement SMOTE oversampling;
- mathplotlib: to plot all the curves;
- pandas and numpy: in order to easly process the data.

First the data has been loded, and splitted into training and testing (20% of the dataset) sets. The training has been still splitted to use the 10% of it for validation. The autoencoder has been developed using keras Model class. It requires a bit of effort to design a network that learns without overfitting. You need to experiment a bit with the number of layers and neurons if you want the loss function to converge. From Figure 5 you can give a look at the structure of the network. It has four hidden layers. During the encoding phase, we map the input to 28 and 56 neurons, then we try to reconstruct the original input with the decode phase (i.e. two hidden layers of 56 and 28 neurons respectively and 23 neurons on the output).



```
Layer (type)                 Output Shape              Param #
=================================================================
input_13 (InputLayer)        (None, 23)                0
_________________________________________________________________
dense_55 (Dense)             (None, 28)                672
_________________________________________________________________
dense_56 (Dense)             (None, 56)                1624
_________________________________________________________________
dense_57 (Dense)             (None, 56)                3192
_________________________________________________________________
dense_58 (Dense)             (None, 28)                1596
_________________________________________________________________
dense_59 (Dense)             (None, 23)                667
=================================================================
Total params: 7,751
Trainable params: 7,751
Non-trainable params: 0
```

Figure 5: The autoencoder, made of 4 hidden layers and the output.

The model has been trained on training and validation sets for 100 ephocs setting the batch_size to 128 and learning rate to 0.001. The loss function has been plotted using mathplotlib and the system has been evaluated on the testing set using the metrics above. At the end a threshold has been fixed to 2.2 according to the results obtained during the evaluation. At the end we plot the confusion matrix.

As we mentioned above, we want to compare the autoencoders with Random Forests. In order to increase the performances of this model, we also apply some data augmentation. We have divided the data into training, validation and testing sets as above. Thus we augmented the training set using SMOTE function from imblearn. The random forest has been trained and implemented using the standard sklearn library. Still all the evaluation metrics have been used as as previously explained. Notice that data augmentation highly increase the performances. In fact, I experimented also with some other techniques like undersampling, but they did not provide the same results.

Refinement

The implementation required a lot of refinement. I started with a simple net, trying to compress the original input and to extract the most relevant information. I set up an autoencoder with the encoding layer with 14 and 7 neurons. Then I set the batch_size from 32 (by default) to 128. Initially I used the entire set of features except "Time". When I removed less important features, the accouracy of the network increased.

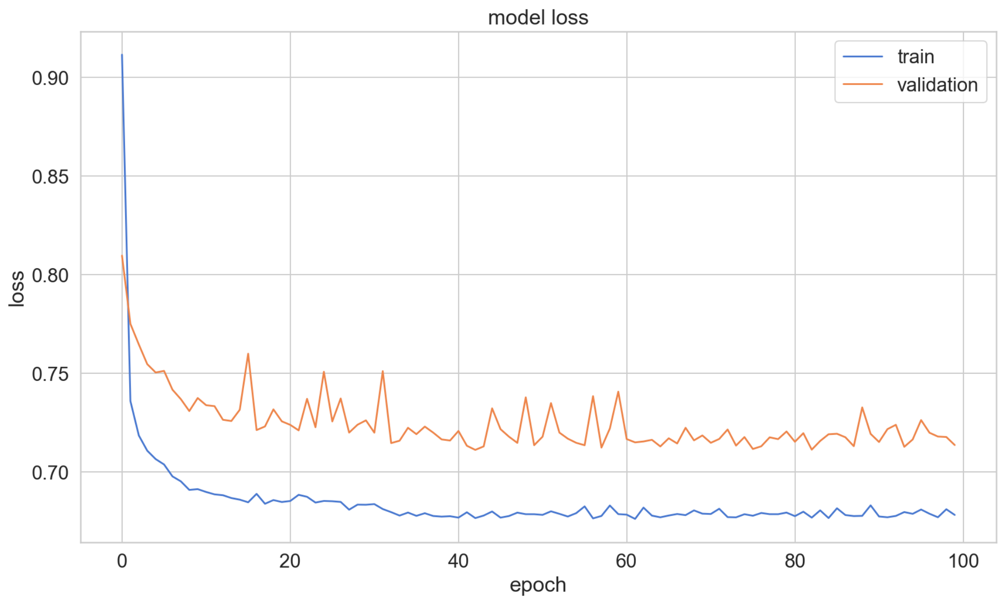


Figure 6: Loss function of the autoencoder.

I tried also to apply Random Forests without data augmentation. Obviously the results were not as good. I have also experimented with under-sampling techniques, but it didn't help.

Results

Model Evaluation and Validation

The results achieved with both models are quite interesting. The autoencoder is less precise, but recognize correctly a good percentage of fraudes. The random forest model on the other hand achieves higher results with a better precision. Both models have been trained and validated on a subset of the dataset, while the 20% of the dataset has been reserved for testing. The evaluation of the model has been done on this subset in order to garantee the models robustness. Figure 7 show the precision-recall curves for both models.

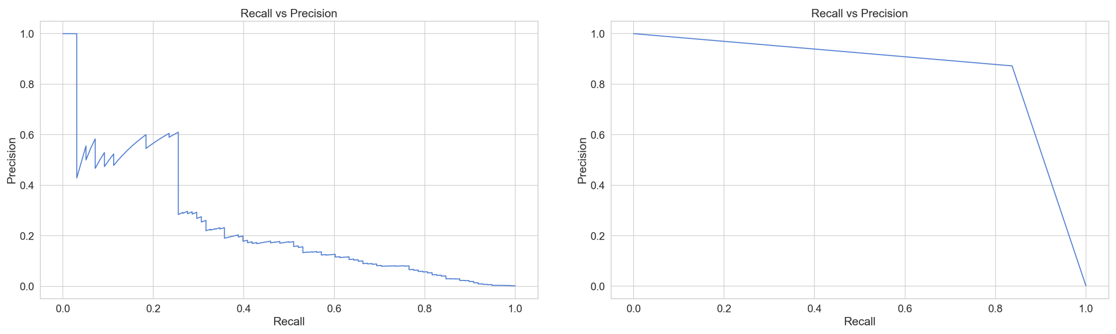


Figure 7: Precision-Recall curves of the autoencoder (left) and random forest (right).

The confusion matrix can help us to have a better idea of the results. They both recognise most of the frauds, but the autoencoder produces a higher number of false positives.

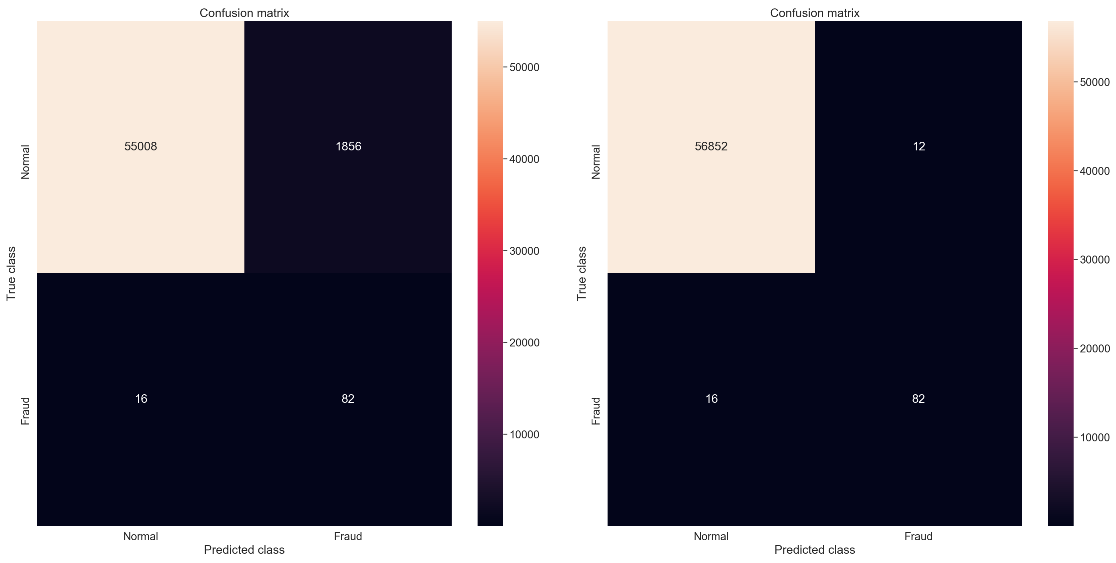


Figure 8: Confusion matrix of the autoencoder (left) and random forest (right).

In order to validate the two models and validate their robustness, we can perform a k-fold cross validation. As you can see from Table 3, the random forest perfectly adapts to new inputs, while the autoencoder has a strong standard deviation, meaning that the model is not that reliable with new and unseen data.

	Autoencoder	Ran. Forest
Mean Acc.	86.631%	99.888%
Std	(+/- 6.87%)	(+/- 0.00%)

Table 3: Mean accouracy and standard deviation after a 10-fold cross validation.

Justification

To compare the results with the benchmark values we need to normalise the confusion matrices with percentages.

	Normal	Fraud
Normal	96,620%	3,260%
Fraud	0,028%	0,144%

Table 4: Confusion Matrix of the autoencoder [6].

The autoencoder was completely outperformed by both models. The random forest instead, is more precise than model [6] . Both [5] and [6] have an higher number of real frauds that have been detected.

	Normal	Fraud
Normal	99,859%	0,021%
Fraud	0,028%	0,144%

Table 5: Confusion Matrix of the random forest [6].

Conclusion

Free-Form Visualisation

From the results above we can conclude that both models learned how to recognise frauds. Figure 8 help us to have an idea of the balance of the true-positives/true-negatives rate of autoencoder and random forest.

Reflection

Frauds detection is an important and sensitive task. It requires techniques that automatically recognise anomalies with low error rates. In fact, those systems need to correctly identify all the bad transactions, possibly with few false positives. The approach that we proposed in this project, correctly solve the problem, even though they should still be improved. As expected the autoencoders are not the best model to solve this kind of problems. The network obviously learn, but is not as accurate as other models. Random Forests instead, are a better choice for anomaly detection problems but it is important to balance the model to obtain decent results, thus it requires a bit of pre-processing steps.

Improvement

The aim of this project was to compare a ”standard” model as random forests with an alternative approach (autoencoders). I did not expect the autoencoder to outperform the most established techniques. Further improvements can be obtained using alternative techniques. Random Forest model can be even improved with some other experimentation. I do expect to see some improvement with combined approach for balancing the dataset using SMOTE+ENN.

References

[1] Hillary Hoffower, BusinessInsider - *There's a good chance you're a victim of credit card scams and you don't even know it here's what to do.*

[2] Application of Credit Card Fraud Detection: Based on Bagging Ensemble Classifier, by Masoumeh Zareapoora and PouryaShamsolmoali.

[3] Credit Card Fraud Detection: A case study, by Ayushi Agrawal, Shiv Kumar and Amit Kumar Mishra.

[4] Credit Card Fraud Detection - Kaggle. Available at: https://www.kaggle.com/mlg-ulb/creditcardfraud.

[5] Case Study: How to Implement Credit Card Fraud Detection Using Java and Apache Spark. Available at: https://www.romexsoft.com/blog/implement-credit-card-fraud-detection/

[6] Credit Card Fraud Detection via KNN and Naive Bayes classifiers. Available at: https://www.kaggle.com/yuridias/credit-card-fraud-detection-knn-naive-bayes

[7] Neural data mining for credit card fraud detection. Available at: https://ieeexplore.ieee.org/abstract/document/809773/.

[8] Credit card fraud detection with a neural-network. Available at: https://ieeexplore.ieee.org/abstract/document/323314/.