Machine Learning Engineer Nanodegree Capstone Project

Tarun Bhardwaj
February 4, 2019

1 Definition

1.1 Project Overview

The PwC global economic crime survey of 2016 suggests that approximately 36% of organizations experienced economic crime. Therefore, there is definitely a need to solve the problem of credit card fraud detection. The task of fraud detection often boils down to outlier detection, in which a dataset is scanned through to find potential anomalies in the data. In the past, this was done by employees which checked all transactions manually. With the rise of machine learning, artificial intelligence, deep learning and other relevant fields of information technology, it becomes feasible to automate this process and to save some of the intensive amount of labor that is put into detecting credit card fraud.
The dataset I'm have worked on can be downloaded from Kaggle.

1.2 Problem Statement

As already mentioned that, with the help of machine learning techniques labor used in identifying the fraud transactions can be reduced to a great extent. Basically the problem can be stated as a binary classification i.e. fraud transaction or genuine transaction. But the frequencies of the two classes is very unbalanced in this case, so, we don't have comparable number of observations for each classes.

As part of this project, my aim is to create few machine learning models which can identify the fraud transactions from given data of transactions.

More precisely I will be implementing 3 different machine learning algorithms to identify the fraud transactions. Those three algorithms are Logistic Regression, Random Forest Classifier and Autoencoder. Along with these algorithms, I will be using few techniques to reduce the imbalance in the dataset like dividing the training set into multiple parts and under-sampling the genuine transactions with respect to fraud transactions. And finally, I will identify which model is most appropriate for this problem.

1.3 Evaluation Metrics

As this problem is related with unbalanced data, accuracy can't serve as a good metric to evaluate the model. Instead of using accuracy as evaluation metric, let's look from the organization's perspective to identify some relevant and useful metrics.
Different aspects are important for the organizations using various techniques for identifying the frauds. The first straightforward requirement from the model is, it should correctly identify the maximum number of fraud transactions which is same as recall of the model.
Along with that, the complete revenue of such organizations depends on their customer base. The model ideally should not identify genuine transactions as fraud one because that will affect the good customers. So, the second metric I will be using is the number of genuine transactions identified as fraud which is the same as the False Positive rate of the model.
Along with using machine learning techniques, organizations use manual inspection for the transactions reported fraud by these models. And, there is some cost involved with the manual inspection of each transaction. Hence the third and final metrics I will be using is the number of transactions reported fraud by the model and sent for manual inspection which is same as True Positive + False Positive rate.

So different metrics I will be using to evaluate the model are:

- Total fraud transactions identified as fraud by the model (*Recall*).
- Total genuine transactions(good customers) reported as frauds (*False Positive Rate*).
- Total fraction of transactions marked as frauds and sent for manual inspection (*True Positive + False Positive Rate*).

2 Analysis

The dataset I'm going to use can be downloaded from Kaggle. 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, the positive class (frauds) account for 0.172% of all transactions.

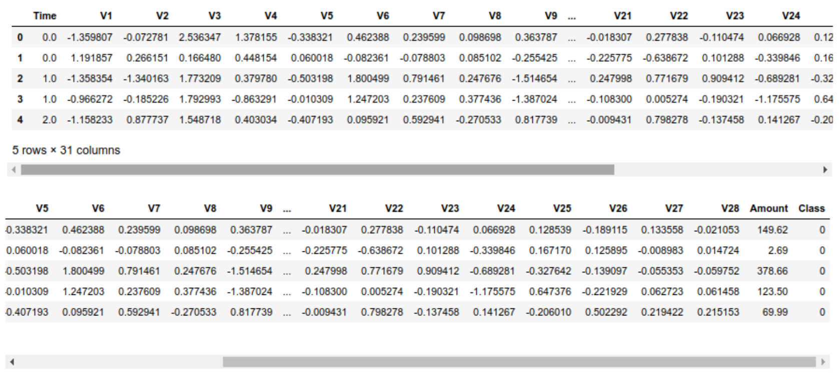
2.1 Data Exploration

The data has already been transformed using PCA transformation(s) and feature names are changed due to privacy reasons. All the features in data are numerical. It has following features :

'Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Class'

- Features V1, V2, ... V28 are the principal components obtained with PCA.
- The two features that havent been changed are 'Time' and 'Amount'.
- Time contains the seconds elapsed between each transaction and the first transaction in the dataset.
- Feature 'Class' is the target variable and it takes value 1 in case of fraud and 0 otherwise.

Here is a sneak-peak to the dataset.



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20

5 rows × 31 columns

V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

Figure 1: Sample of Dataset

2.2 Exploratory Visualization

First, let's plot the pie chart of the distribution of transactions among fraud and genuine categories to verify that data is highly unbalanced.

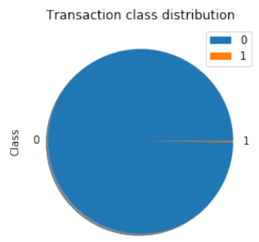


Figure 2: Distribution of Data among 2 classes

From the shown pie chart, one can easily observe that number of transaction in class 1 (fraud) are very few as compared to number of transactions in class 0 (genuine).
Exact number of transactions in class 1 are 492.
Exact number of transactions in class 0 are 284315.

Now, let's try to visualize the number of fraud and genuine transactions according to the `amount` involved in the transaction.

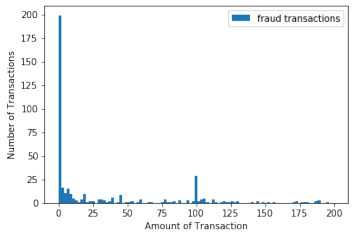


Figure 3: Distribution of Fraud transactions for different amounts of transaction

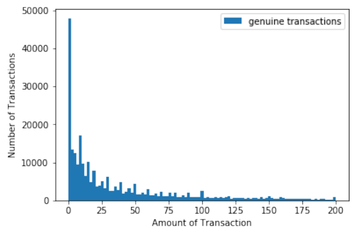


Figure 4: Distribution of Genuine transactions for different amounts of transaction

Maximum `amount` involved in Genuine transaction was much higher than 200 but more than 80% of the transactions had amount less than 200. I have plotted the transactions till amount 200 only for better visualization. It can be observed that, there is not any significant difference for distribution of both the classes.

Lastly, let's visualize the `amount` involved in the transactions according to the feature `time`. It will be helpful, if there is some pattern in the `amount` involved in the transaction and `time` of transaction.

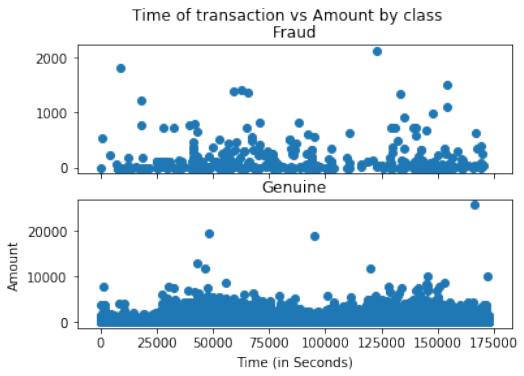


Figure 5: Distribution of transactions according to feature `time`

Clearly, there is no specific pattern in the `amount` involved in the transaction with the time at which that transaction occurred. It can be concluded that feature `time` is not important to identify if the transaction is fraud or genuine.

2.3 Algorithms and Techniques

As mentioned in the project proposal, I will be implementing 3 different models for the given problem. More details for each of the implementations are as follows:

- **Logistic Regression**: Logistic regression is a technique borrowed by machine learning from the field of statistics. It is the go-to method for binary classification problems (problems with two class values).

  Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.
  Below is an example logistic regression equation:

  $$y = e^(b0 + b1 * x)/(1 + e^(b0 + b1 * x))$$

  Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.
  The actual representation of the model that you would store in memory or in a file are the coefficients in the equation (the beta value or bs).

  As our problem is also a binary classification problem and also there isn't any public model which I can use as benchmark. So, I will be implementing Logistic Regression and will treat it as my benchmark model to evaluate other models.
  I will use Logistic Regression with a simple 70-30 train-test split and use that model as benchmark.

- **Random Forest**: As stated in sk-learn's documentation, A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

  Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.

  In Laymens term, Suppose training set is given as : [X1, X2, X3, X4] with corresponding labels as [L1, L2, L3, L4], random forest may create three decision trees taking input of subset for example,

  $$[X1, X2, X3]$$
  $$[X1, X2, X4]$$
  $$[X2, X3, X4]$$

  So finally, it predicts based on the majority of votes from each of the decision trees made.

  One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems.

  Below you can see how a random forest would look like with two trees:

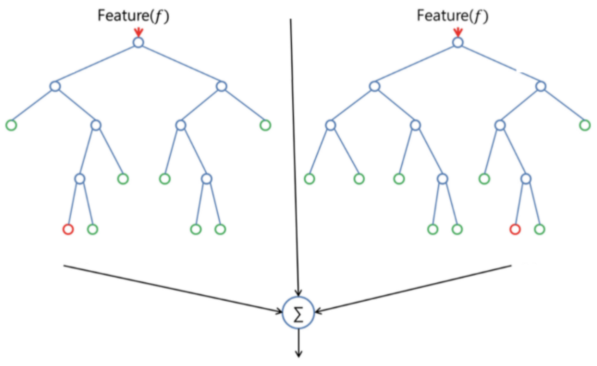


Figure 6: A simple Random Forest

I have implemented multiple variances of Random Forest model which includes:

- Using the whole data in single random forest.
- Dividing the data into 4 different batches and training on 4 different random forests with and without undersampling.

- **Autoencoder**: According to wikipedia, an autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal noise.

  Autoencoders (AE) are a family of neural networks for which the input is the same as the output. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation.

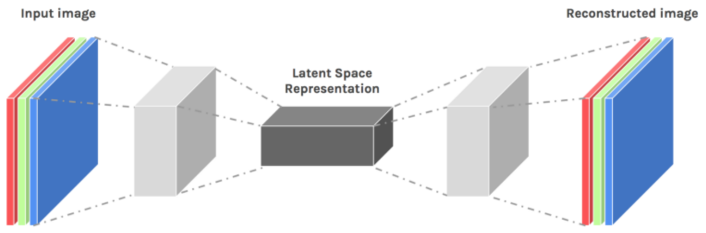


Figure 7: Autoencoder Architecture

Despite the fact, the practical applications of autoencoders were pretty rare some time back, today data denoising and dimensionality reduction for data visualization are considered as two main interesting practical applications of autoencoders. With appropriate dimensionality and sparsity constraints, autoencoders can learn data projections that are more interesting than PCA or other basic techniques.

I will remove the labels provided in dataset and use all the genuine transactions for training and calulate the reconstruction error involved in predicting the input. Fraud transactions will comparatively have higher reconstruction error then the genuine transactions. Based on the distribution of reconstruction error, I will choose a threshold above which I will mark transactions as fraud.

2.4 Benchmark

As the problem is related to one of the most sensitive areas of implementation, individuals/organizations using machine learning techniques for solving this problem which includes Banks, transaction agencies, etc don't usually share their works. Also, because the data is highly unbalanced we can't use Average Prediction as a benchmark model. And as I mentioned in the previous section, I will be using the Logistic Regression model with 70-30 train-test split as benchmark model to evaluate other model's performance.

3 Methodology

3.1 Data Preprocessing

The dataset I'm going to use can be downloaded from Kaggle. All variables in the dataset are numerical. The data has been transformed using PCA transformation(s) due to privacy reasons. Features V1, V2, ... V28 are the principal components obtained with PCA. The two features that havent been changed are Time and Amount. Time contains the seconds elapsed between each transaction and the first transaction in the dataset.
From my side, I have dropped the feature `time` as it didn't had any relationship with the distribution of transactions. Apart from that, I have also normalized the feature `amount` in between -1 and +1.

3.2 Implementation

I have implemented 3 different algorithms. Let's discuss them individually,

- **Logistic Regression**
  For logistic regression, I had a simple implementation which involved splitting the data into train and test set with ratio 70:30. Also, I have used l2 regularization as penalty attribute for the regression. Complete structure of Logistic Regression is,

  ```
  LogisticRegression(C=1.0, class_weight=None, dual=False,
  fit_intercept=True, intercept_scaling=1,
  max_iter=100, multi_class='warn', n_jobs=None,
  penalty='l2', random_state=None, solver='warn',
  tol=0.0001, verbose=0, warm_start=False)
  ```

  Implementation of Logistic Regression is pretty straight forward and this model is also used as the benchmark model.

- **Random Forest**
  In random forest, I have implemented several different variants. Before all the different implementations, I have divided the whole dataset into train and test with 70:30 ratio. Also, structure of RandomForestClassifier used in all the implementation is same. Structure of the classifier is,

  ```
  RandomForestClassifier(bootstrap=True, class_weight=None,
  criterion='gini', max_depth=None,
  max_features='auto', max_leaf_nodes=None,
  min_impurity_decrease=0.0, min_impurity_split=None,
  min_samples_leaf=1, min_samples_split=2,
  min_weight_fraction_leaf=0.0, n_estimators=50,
  n_jobs=None, oob_score=False, random_state=0,
  verbose=0, warm_start=False))
  ```

  In the first implementation, I have used whole training-data to train the model and in rest of the implementations, I have modified the training data in few different ways to balance out the unbalanced data. Let's look at the details of each implementation,

  1. **Using whole training data as it is and use it for training.**

2. **Dividing the training data into 4 batches**
   In this part, I have divided the training data into 4 equal batches. However, while creating the batches, I have only divided the genuine transactions of training data and included fraud transactions of the training data into all 4 batches. In this way, I decreased the number of genuine transactions to 25% of the total training data which creates little more balanced training sets as compared to original training data.

   After training 4 different batches of training data on 4 different classifiers, I have predicted the `Label` using all 4 classifiers. If at least 2 classifiers reports the transaction as a fraud, then that transaction is marked as fraud otherwise genuine. I have chosen the value 2 because each training batch doesn't have all the genuine transactions and it might easily classify genuine transaction from some other batch as a fraud.

3. **Dividing the training data into 4 batches along with under-sampling the genuine transactions to a ratio of 90:10 in comparison to fraud transactions.**
   In this part, after splitting the training data into 4 different batches in the same way as mentioned in the previous part, I have again sampled each batch such that ratio of genuine to fraud transactions remain 90:10 in each batch. In this way I have increased the proportion of fraud cases in the training data so that maximum of the fraud transactions can be identified. Testing part is same as that in the previous part.

4. **Dividing the training data into 4 batches along with under-sampling the genuine transactions to a ratio of 80:20 in comparison to fraud transactions.**
   This part is exactly similar to the third part apart from the ratio of under-sampling for each batch.

- **Autoencoder**
  Autoencoder is an unsupervised method of machine learning, I have created this situation by training the model on the genuine transactions only. Reserving the correct class on the test set will give us a way to evaluate the performance of our model. I have split the whole data into training and testing set with ratio of 80:20 and from the training set only genuine transactions are used for training the model.
  This Autoencoder uses 4 fully connected layers with 14, 7, 7 and 29 neurons respectively. The first two layers are used for encoder, the last two go for the decoder. Additionally, L1 regularization is used during training. Model was trained for 100 epochs with batch size of 32. Exact architecture of the Autoencoder model is

```
encoder = Dense(encoding_dim, activation="tanh",
             activity_regularizer=regularizers.l1(10e-5))
             (input_layer)
encoder = Dense(int(encoding_dim / 2), activation="relu")
             (encoder)

decoder = Dense(int(encoding_dim / 2), activation='tanh')
             (encoder)
decoder = Dense(input_dim, activation='relu')(decoder)

autoencoder = Model(inputs=input_layer, outputs=decoder)
```

3.3 Refinement

When I decided to split the train data into multiple batches, I wasn't sure about the number of batches to start with. I started with splitting the train data into 10 batches. But because each batch didn't had the knowledge of other 90% genuine transaction, a large number of genuine transactions were classified incorrectly. Then I reduced the number of batches by 2 each time and it improved the false positive rate by some amount each time. Finally, I decided to split the data into 4 batches as both recall and precision was quite good and trade between both metrics was acceptable.

4 Results

4.1 Model Evaluation and Validation

As I have implemented 3 different algorithms, let's look at performance of each one individually. I will be demonstrating the confusion matrix of each model and the 3 evaluation matrices that I decided to consider.

- **Logistic Regression**
  Trainig Data : 70% of the transactions
  Testing Data : 30% of the transactions

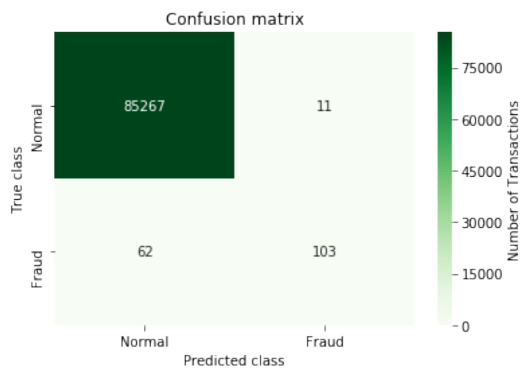


Figure 8: Confusion Matrix for Logistic Regression

  - Fraud transactions correctly identified = 103 (62.42%)
  - Genuine transactions reported as fraud = 11 (0.012%)
  - Transactions sent for manual inspection = 114 (0.13%)

- **Random Forest (1/4)**
  Trainig Data : 70% of the transactions
  Testing Data : 30% of the transactions

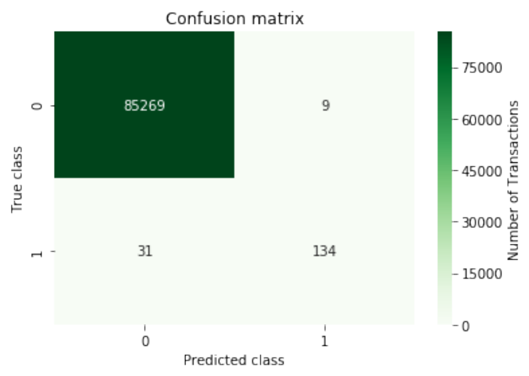


Figure 9: Confusion Matrix for simple Random Forest

  - Fraud transactions correctly identified = 134 (81.21%)
  - Genuine transactions reported as fraud = 9 (0.010%)
  - Transactions sent for manual inspection = 143 (0.16%)

- **Random Forest (2/4)**
  In this part, I divided the training data into 4 different batches keeping the fraud transactions same in each batch and trained 4 different RandomForestClassifiers using each batch.

  Trainig Data : 70% of the transactions divided into 4 different batches
  Testing Data : 30% of the transactions

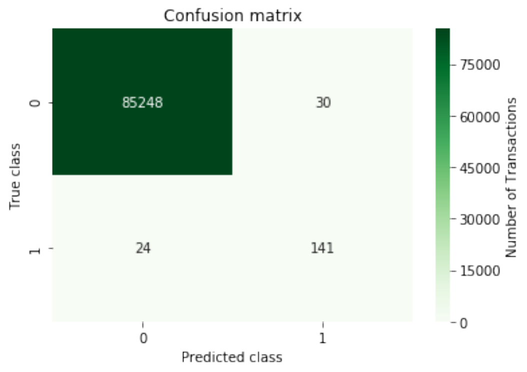


Figure 10: Confusion Matrix for Random Forest with 4 batches

  - Fraud transactions correctly identified = 141 (86.66%)
  - Genuine transactions reported as fraud = 30 (0.035%)
  - Transactions sent for manual inspection = 171 (0.20%)

- **Random Forest (3/4)**
  In this part, I divided the training data into 4 different batches keeping the fraud transactions same in each batch and and then re-sampled the each batch such that they maintain the ration of 90:10 for genuine and fraud transactions. After that trained 4 different RandomForestClassifiers using each batch.

  Trainig Data : 70% of the transactions divided into 4 different batches with 90:10 sampling
  Testing Data : 30% of the transactions

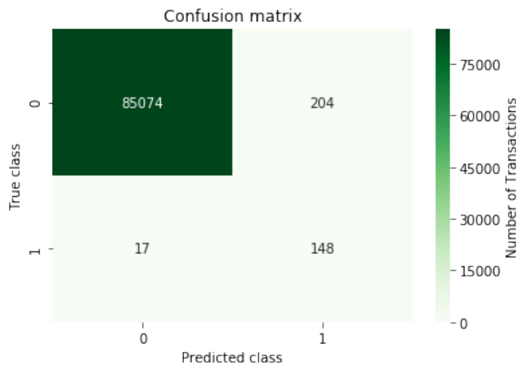


Figure 11: Confusion Matrix for Random Forest with 4 batches and 90:10 sampling

  - Fraud transactions correctly identified = 148 (89.69%)
  - Genuine transactions reported as fraud = 204 (0.239%)
  - Transactions sent for manual inspection = 352 (0.41%)

- **Random Forest (4/4)**
  In this part, I divided the training data into 4 different batches keeping the fraud transactions same in each batch and and then re-sampled the each batch such that they maintain the ration of 80:20 for genuine and fraud transactions. After that trained 4 different RandomForestClassifiers using each batch.

  Trainig Data : 70% of the transactions divided into 4 different batches with 80:20 sampling
  Testing Data : 30% of the transactions

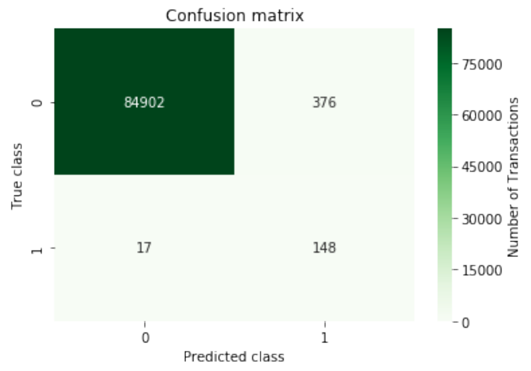


Figure 12: Confusion Matrix for Random Forest with 4 batches and 80:20 sampling

  - Fraud transactions correctly identified = 148 (89.69%)
  - Genuine transactions reported as fraud = 376 (0.44%)
  - Transactions sent for manual inspection = 524 (0.61%)

- **Autoencoder**
  Training the Autoencoder is a bit different from previous models. We have a dataset containing a lot of non fraudulent transactions at hand and we want to detect any anomaly on new transactions. I have created this situation by training our model on the genuine transactions, only. Reserving the correct class on the test set will give us a way to evaluate the performance of our model. I have reserved 20% of the data for testing:

  Trainig Data : Genuine transactions from 80% of the data.
  Testing Data : 20% of the transactions

  First let's look at the precision-recall graph of the trained Autoencoder.

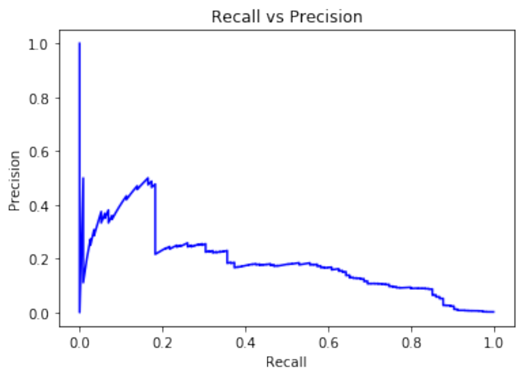


Figure 13: Precision Recall graph for Autoencoder

As shown in the plot, relationship between precision and recall is not following a particular trend. I had to make a decision for the exact point for Precision and Recall, which I made from their relationship with re-construction error of the model. Following plots depict the relationship of Precision and Recall with the threshold or reconstruction error.

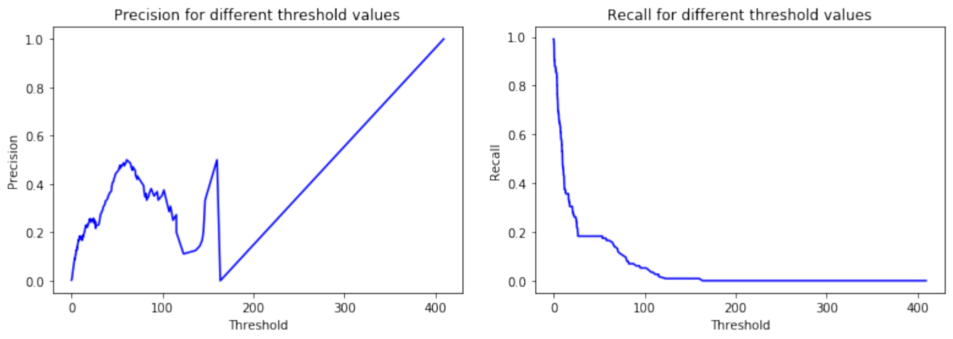


Figure 14: Precision and Recall plot against threshold

Finally let's look at the confusion matrix of the autoencoder model.

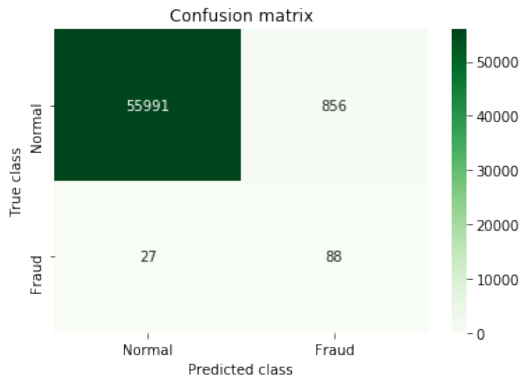


Figure 15: Confusion matrix for Autoencoder

- Fraud transactions correctly identified = 88 (76.52%)
- Genuine transactions reported as fraud = 856 (1.50%)
- Transactions sent for manual inspection = 944 (1.65%)

4.2 Justification

Random Forest Classifier has performed pretty well because Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model. And clearly it improved both precision and recall as compared to Logistic Regression.
To improve the recall, I tried to increase the fraction of fraud transaction. Hence, 4 Random Forest Classifiers used together resulted in better recall but also it had negative effect on precision because as I try to divide the genuine transactions into 4 batches, each Random Forest doesn't have the information about genuine transactions of other 3 batches and thus model classifies some genuine transactions as fraud.
Similarly, when I used undersampling on those 4 batches, recall improves while precision lowers down because model doesn't use all the genuine transactions. And as a result model doesn't have

all the patterns of genuine transactions and misclassifies them.
In case of Encoder, it's simply comes down to your choice between Precision and Recall, and you have to select the threshold as per your requirements.

5 Conclusion

5.1 Free-Form Visualization

None of the model outperform others in all the evaluation matrices. Let's visualize the precision and recall of all 6 models are try to identify which one(s) can be potentially used.

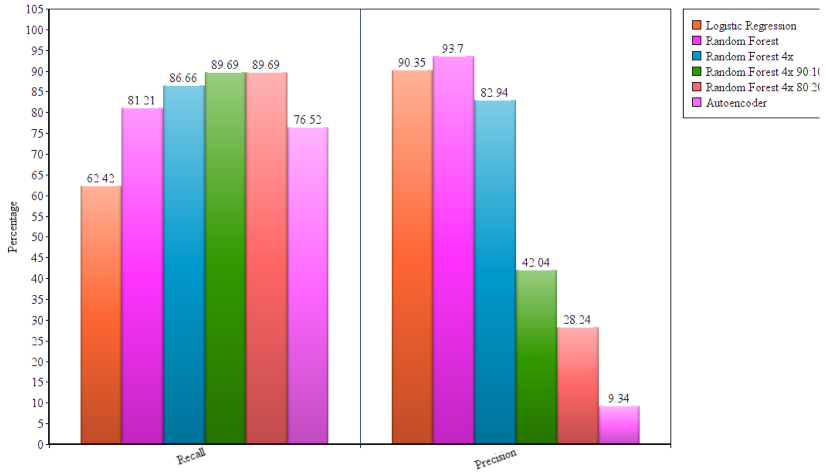


Figure 16: Comparison of different models

However, in comparison with other models, RandomForest and RandomForest with 4 batches perform better. They have pretty good values for all the evaluation matrices and it depends on the organization using machine learning for fraud detection, which metric is more important for them. For example amount lost in a fraud transaction might be much higher than loosing a genuine customer so, in this case organization would try to identify maximum of the frauds. In other case, it might be possible that cost of manual inspection is very high so, sending so many transactions for the manual inspection will again cost the organization.
Hence, RandomForest can be the go to algorithm for this problem and further variations can be applied to it.

5.2 Reflection

It was a nice experience in exploring this area. The thing which motivated me was the impact it can have when used. It can reduce lots of labor required to identify the fraud transactions. Even if we use manual inspection after these techniques, amount of transactions sent for manual inspection is reduced to a very small quantity. You only need to inspect less than 1% of the transactions and it can save lots of resources.
Dataset for this project was easily available and was already preprocessed for the use. I just needed to implement the models which can produce significant results and tried to balance out the difference between number of genuine and fraud transaction in different ways.
Random Forest Classifier (1st and 2nd) produced good results and can be used for the real world deployment of this project.

5.3 Improvement

When we think about the deployment of such models in real life implementation, it can be noted the model will be supposed to detect the current fraud transactions based on the past data it has. So, it will be better if we build the model in such way that it can easily classify the future transactions based on past data.
Also, we can improve the model by using stacking method in which we can implement several models into an ensemble.