CIS 8395 Big Data Analytics Experience
Term Project
Credit Card Fraud Detection Using GAN

Team Credit Pro:
David Lim
Priyanka Nikumbh
Bo Wang

1. **Project Description**

In the 21st century, as more and more people adopt Credit Card banking system for transactions, the system faces an increasing number of threats. The credit card system works like this: a borrower could use a limit amount of first and have to return the money by the end of the month. Also, an additional feature is when a user looks at the monthly statement and he notices an out-of-place transaction that he doesn't recollect, the Bank will remove that transaction so that he does not have to pay for it. It is the Bank's responsibility to investigate suspicious transactions and users owe nothing to them (in most cases). This provided an unparalleled amount of security for one's money as they are not using their own money to pay for stuff in the first place. However, some people started to steal money from other people, stealing a card and swiping it would mean that the robber won't be asked any questions as there is no requirement to show an ID to use a credit card. The Bank must identify such uses as soon as possible to prevent monetary loses.This is where the requirement of Data Analysis lies. The Bank's Data Analytics – or more commonly known as Fraud Analytics – division works day and night to quickly identify and notify the customer as well as bank management of suspicious transactions. A 2009 LexisNexis study showed that, in the United States alone, merchants lose about $190 billion a year, banks lose $11 billion a year and customers lose around $4.8 billion a year.

However, analyzing transaction data is very complex. Considering hundreds of variables is extremely difficult not to mention the invasion of privacy of the customer. Sometimes Banks have to approach an external source to brainstorm ideas to add to the "fraudulent transaction" classification. When this happens, to prevent disclosing confidential customer information the bank resorts to conducting pre-processing of the data and substitute the names of its recorded data points with random variables like X and Y. This essentially removes the possibility of characterizing the data points which basically means all variables are treated the same. A big flaw with this is that an approach to analyze the "random" bunch of variables is unclear. So, analysts are forced to test all known models to try to find a pattern that certain amount of transactions follow to classify them as "fraud".

For the dataset we use, there is a limited amount of fraudulent cases. We want to use GAN to create realistic and enough data to help us detect fraudulent cases. GAN is a type of neural network that is able to generate new data from scratch. It is useful to produce new data in data-limited situations. Although it is difficult, expensive and time-consuming to generate new data, the new data has to be realistic enough to guarantee the authenticity of the results. GAN is a good technology for solving this problem. We could learn more about GAN from this project.

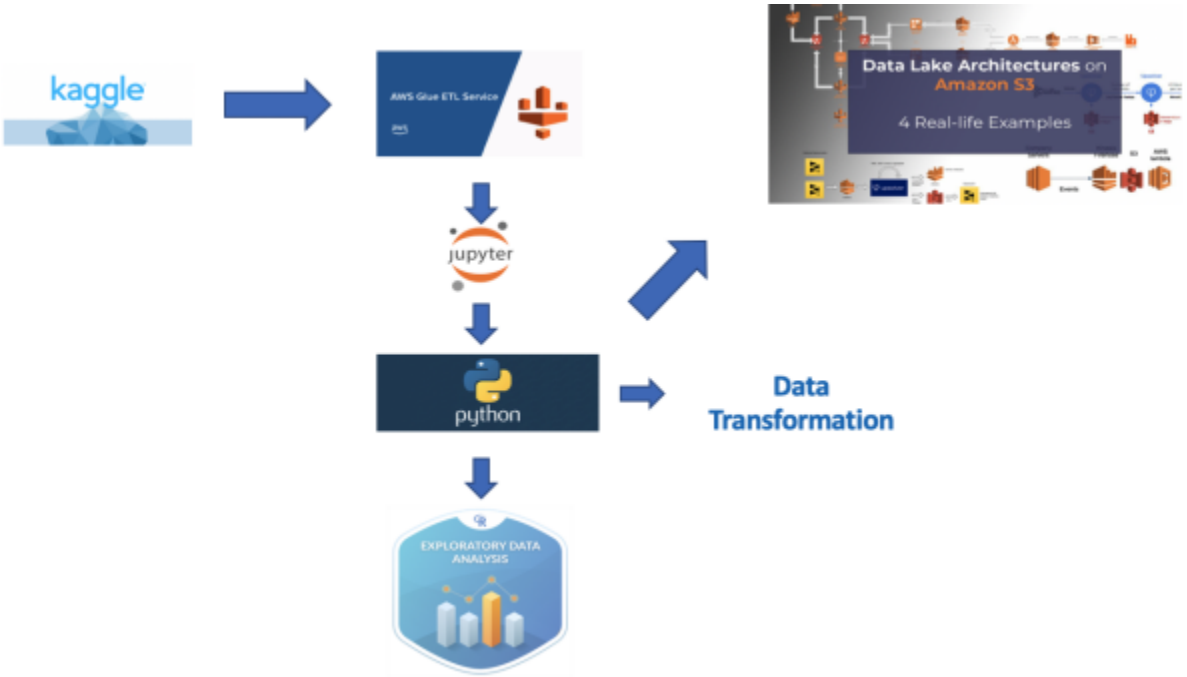
2. Paper Objective

The main objective of this paper is to describe in detail the different stages that occur within a corporate environment after a project proposal has been approved and before data analysis and visualization of results. However, due to security reasons, we will not have access to raw credit card data obtained from point of sale (pos) devices,

credit card readers or credit card transactions that occur via the internet. Therefore, we have detailed what we have actually done, given this limitation, but we have also given some details on what would be required if we did have access to raw credit card data. In this paper, the data sourcing, data collection, data transformation, data storage and data maintenance stages used for credit card fraud detection are all described in detail.

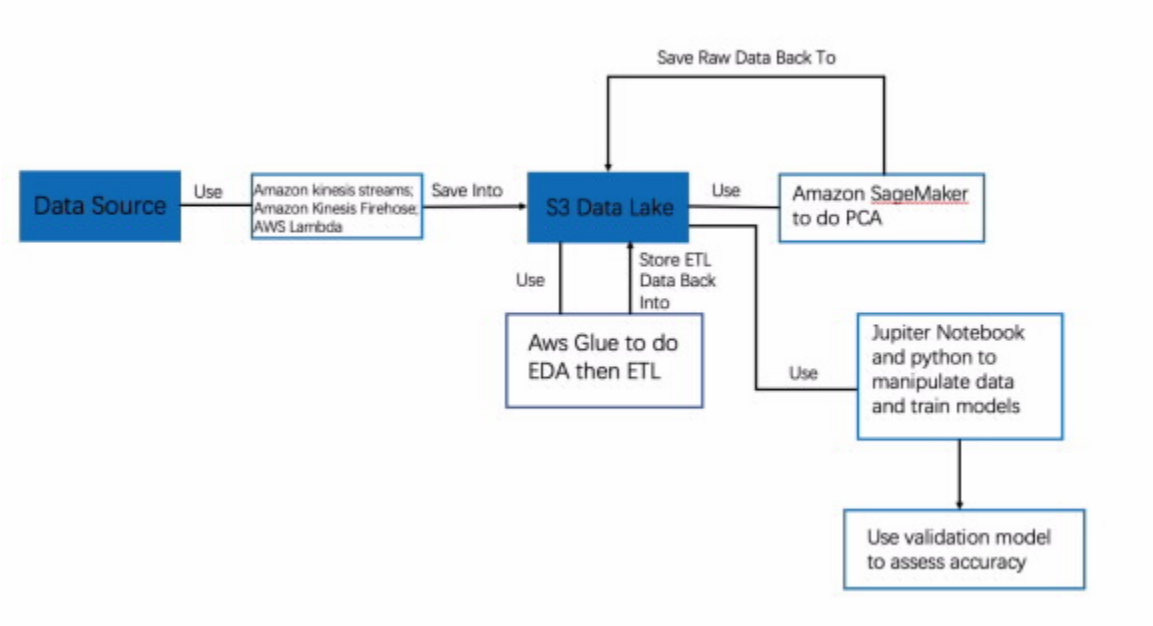
3. Strategy/Architecture

It is important to note that, the strategy/architecture used will vary greatly depending upon whether we obtain real-life, real time credit card data or if we simply use data sourced from kaggle.com. In this particular case, we use data from kaggle.com, where principal component analysis (PCA) has been performed on the original dataset for security purposes. This greatly reduces the complexity of our strategy/architecture. Simply put, we use Amazon Glue to extract, transform and load (ETL) data and the preprocessed data is stored in a s3 data lake. Below are more specific details on this process and on this architecture.



The original data is obtained from kaggle.com. Using an ETL tool, Amazon Glue, we have specified the location of the original data in the AWS Management Console in order to create a Data Catalog. Next, we specified the data source and the data target in order to generate Python code, which will extract the source data and save transformed data in a .csv format to our target (an s3 data lake). Given that principal component analysis (PCA) has already been performed on the data, the only data transformations we need to perform are related to data cleansing, which will require us to do things such as removing rows with any missing values, removing rows with any NaNs, and removing columns that will not be used for analysis. This has been achieved by editing Python code

in the Console so that first data exploration was performed, which enables us to perform the proper data transformations. Then, once the data is transformed, it will be loaded into a s3 data lake. From there, the data will be ready for any data analytics and visualization. It is expected that no more than 150 MB of S3 storage space will be needed to complete the entire credit card fraud project.



The approach described above is simply what we have done with data we have access to. If we were receiving real credit card data, there would be many more stages, and each stage would be significantly more complex. First, all credit card data stored must be in compliance with Payment Card Industry (PCI) Data Security Standards (DSS). Luckily, Quick Start makes it relatively easy to set up an AWS Cloud environment that supports an architecture that is in compliance with PCI DSS. However, there is many components and features needed to deploy such an architecture, which includes a custom AWS Identity and Access management (IAM) configuration, an Amazon Virtual Private Cloud (Amazon VPC), Amazon S3 buckets, Amazon VPC security groups for Amazon Elastic Compute Cloud (Amazon EC2) instances, a secured bastion login host for Secure Shell access to EC2 instances, an encrypted Amazon MySQL database and use of AWS CloudTrail, Amazon CloudWatch, and AWS Config rules. Outside of this secure architecture, there is the added complexity of streaming real time data using Amazon Kinesis Streams, AWS Lambda triggers and Amazon Kinesis Firehose. Also, it is likely that Apache Spark on Spark EMR would be more appropriate as it is more suited for streaming on a distributed system (since data streaming in real time), and Amazon SageMaker would also be utilized in order to perform PCA once a day on the raw data, which would then have to be stored on a S3 data lake (with the proper IAM configurations), so that analysts can run data analytics without compromising sensitive data. Finally, we would have to be careful about placing service limits on both the EC2 and S3 databases so that the cost of storing the data can be predicted. It is estimated that

~200 MB of data will be stored daily, ~100 MB of data in EC2 storage and ~100 MB of data in S3 storage, which is ~73 GB per year, which is more than financially feasible for a credit card corporation. To be clear, all of these steps will be performed prior to extract, transform and loading (ETL) procedures. The data stored as a result of the steps mentioned, will be the source of data that is used for ETL procedures, whose purpose is to provide data for analytics and visualization.

4. Delivery

The section below describes the data sources utilized; the extract, ETL procedure; and the data storage approach for this project. In particular, there is a description of where the data was obtained, the tools and approach used in ETL and a description of how and where the data will be stored.

4.1. Data Source

We have found several data sources for obtaining credit card data, but as mentioned before, for security reasons, all data sources have been previously transformed to mask the specifics of what type of data has been collected. These are the three sources we have found:

1. https://www.kaggle.com/mlg-ulb/creditcardfraud
2. http://weka.8497.n7.nabble.com/file/n23121/credit_fruad.arff
3. https://archive.ics.uci.edu/ml/datasets/Statlog+%28Australian+Credit+Approval%29

All three datasets include a combination of categorical and numerical data saved in a relational database format as a .csv file. Unfortunately, upon examination of the data, due to differing methods of transformations performed on the data sets, it was apparent that it would be impossible to combine the datasets in any meaningful manner, which would be useful for credit card fraud detection. Therefore, we simply determined that the data from kaggle.com was the largest, highest quality data set out of the three choices. Also, it appears to be the most popular dataset used for credit card fraud studies, as there are dozens of independent studies that have used this dataset in order to detect credit card fraud.

In a realistic environment, the data would be derived from a number of credit card transactions: apple pay, samsung pay, pos, online, etc. Without worrying about the complicated process of credit card transactions, we will assume that a custom application manages credit card data during transactions. Once the data has been received, it is immediately preprocessed via the steps mentioned in the strategy/architecture section of this paper. The result of the preprocessing procedure will be the source of data used for ETL.

4.2. Extract, Transform and Load (ETL)

As mentioned earlier, Amazon Glue will be the primary tool used to extract, transform and load the datasets utilized for this particular project. Data will be extracted from kaggle.com then will immediately be transformed after an exploratory data analysis procedure. Then, the data resulting from ETL, will be saved in an S3 data lake. In the case of real credit card data, data will be extracted from an S3 data lake (data resulting from preprocessing), and then will be transformed and loading into a separate tier in the S3 data lake.

4.2.1. Extract

Initially, all columns and all rows will be extracted from the dataset. This will only need to be done once for this particular project. For real data, all the data will be saved in real time, but it will only be extracted for ETL on a daily basis. Only PCA data will be available to analysts. All the data will simply be a set of columns and everything is stored as .csv file.

4.2.2. Load

The data is loaded to s3. It should not take too long time to load the data since we just load a balanced dataset which has only 500 fraud and 500 non-fraud data daily. In addition, there should be no parallelization needed, we only need a subsample of data to train the models. The amount of data is not necessarily overwhelming (1000 rows and 29 columns). We think the data does not need any tuning (SQL).

4.3. Storage

For the purposes of this study, we use a simple relational database design (files stored as .csv files) and store only the transformed data into an s3 data lake using AWS Lake Formation. Since our data storage needs are very simple and straightforward (we obtain a single dataset once, transform the data once, and store only transformed data), we take advantage of the existing data lake architecture that is cheap to maintain, can be used off the shelf and can be deployed easily within one hour. Many of the components of this architecture including AWS Lambda, Amazon Elasticsearch, etc. will not be needed, but we have chosen to use an Amazon data lake because of its essentially automatic integration with Amazon S3 and AWS Glue and because of its friendly, easy to use user interface, which gives access to a console for easy data management.

If real credit card data was used, it is likely that both a data warehouse and a data lake would be utilized, since there is trade-offs depending on what you are using the data

for. in real time using amazon kinesis streams, AWS Lambda and Amazon Kinesis Firehose. Then we use Amazon SageMaker to do a PCA once a day on the raw data that is collected in real time, and store the raw data back into data lake on s3. Next, we use aws glue to do some EDA then do some ETL. We take the ETL data and store it back into aws s3. So there will be three levels of data in the s3 data lake(raw data, pca data, and ETL data) whose access will be managed by aws IAM, where people have different levels of access. Many of the components of this architecture including AWS Lambda, Amazon Elasticsearch, etc. will not be needed.

5. Operations

We have obtained all of our data used in this study from www.kaggle.com and have essentially zero operating needs, especially given the existing architecture of Amazon AWS. Instead of going into detail about what we have used, below, we have given a description of the architecture that would be required in a real corporate environment, where real credit card data is used. If real credit card data were obtained and used, all of the credit card data would not be obtained all at once. Typically, the data will be available at a continuous rate and will accumulate over time. This will require more than simply just a tool to monitor, maintain and support the data architecture. A regular schedule must be in place where databases are updated, people are able to detect any inefficiencies and/or errors and infrastructure can be modified and/or corrected to meet standardized procedures or standardized data formats.

5.1. Scheduling

As mentioned, the data doesn't need any parallelization. We should load raw data as soon as possible (in real time), put data through ETL once a day, and train models once a month. In addition, we use aws lambda to trigger ETL actions. The ETL data will be processed once a day. We should retrain models and compare models every month, anytime there is a change to the number of columns or type of data in each column, we should retrain models.

5.2. Monitoring

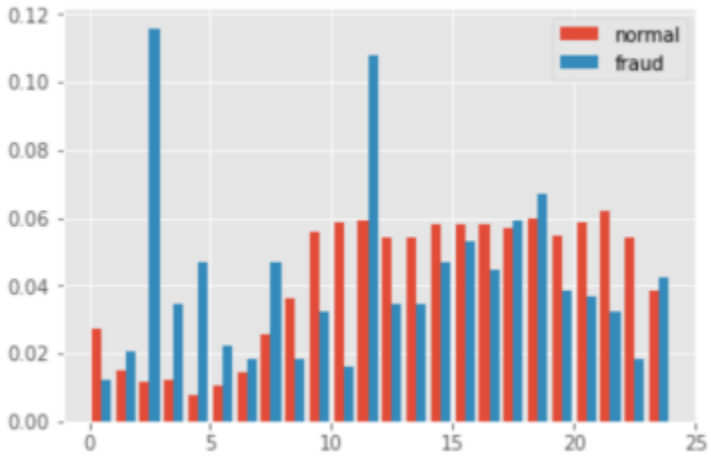
We need to monitor the procedures of data processing, find what problem causes the jobs fail, and find suitable methods to solve the problem. There are several common problems we should pay attention to: network failure, formatting errors, and the file did not transfer fully.

6. Analysis and Visualization of Results

6.1. Data Exploration and Cleaning

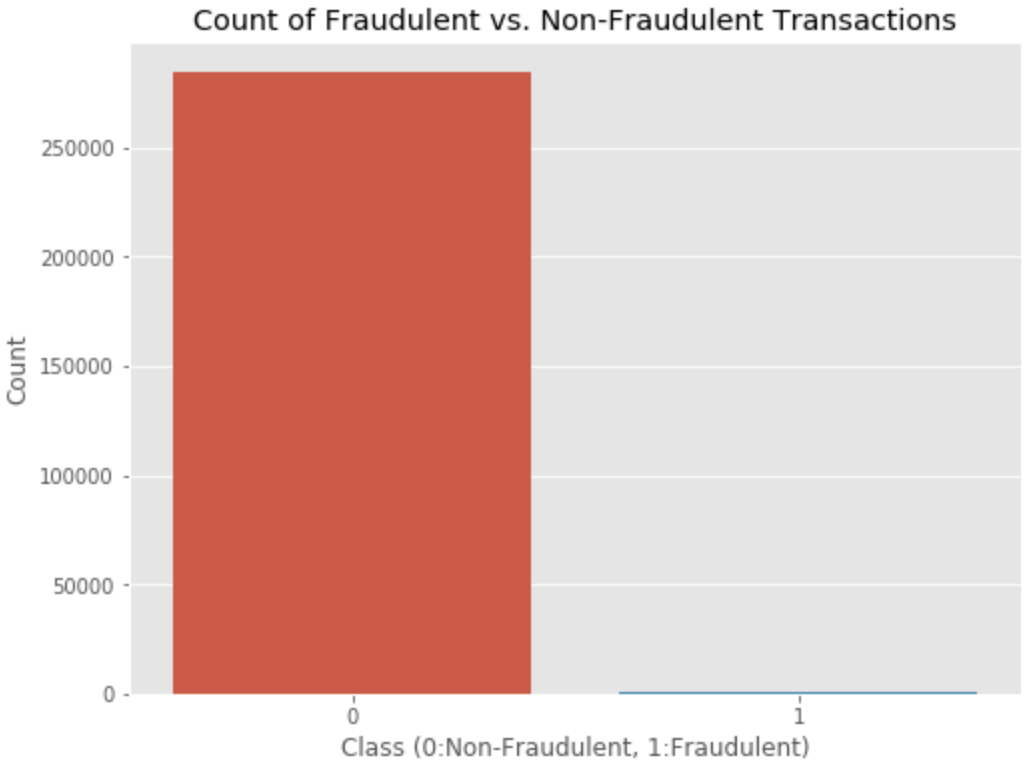
'Time' is in seconds in the dataset, we could convert it to hours. We find the dataset contains transactions in 48 hours, which is two days. From the result we could find normal transactions have a bias towards 8am to midnight, and Fraud has spikes at 2-3am and noon. We guess noon is the busiest time and 2-3 am has very few customers, people who do fraud transactions could reduce the chance to be caught.

```
Last time value: 48.00
```



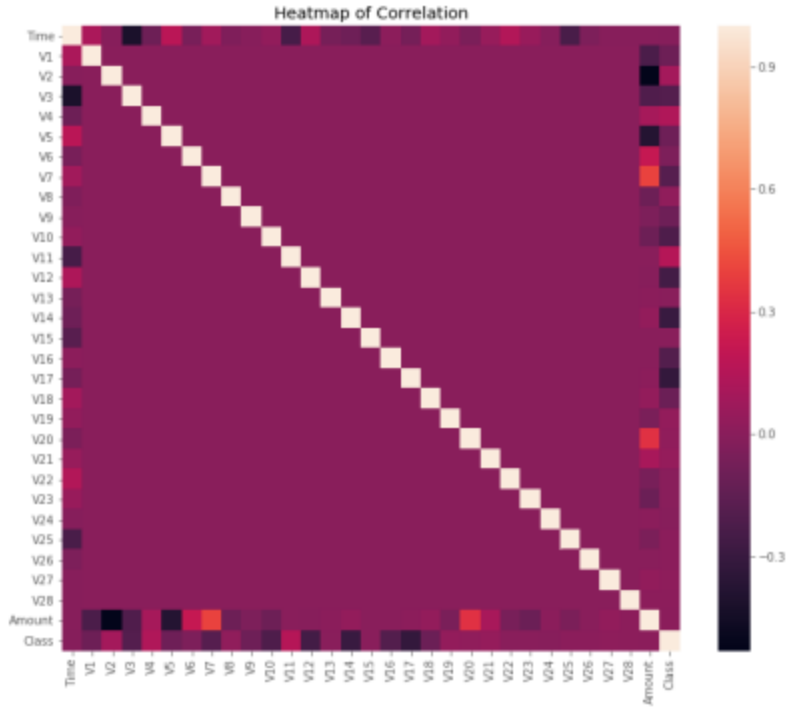
Count of fraudalent and non fraudalent transactions:

There are total 284315 non fraudalent transactions and 492 fraudalent transactions.

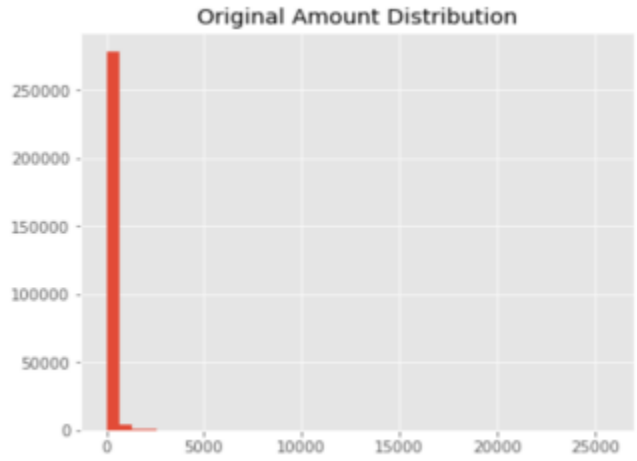


For the 28 anonymized variables (“V1” - “V28”), the number of variables is quite high. A possible solution for this is to perform a Principle Component Analysis (PCA). This is a statistical procedure that uses an orthogonal transformation to convert a set of possible correlated variables to a (smaller) set of uncorrelated variables. In lay-man terms, correlated terms are terms that respond similarly to a change in a variable and for the sake of increasing the analysis efficiency can be put under one "bracket" such that in the end, the "brackets" of variables are linearly uncorrelated. When we checked the actual

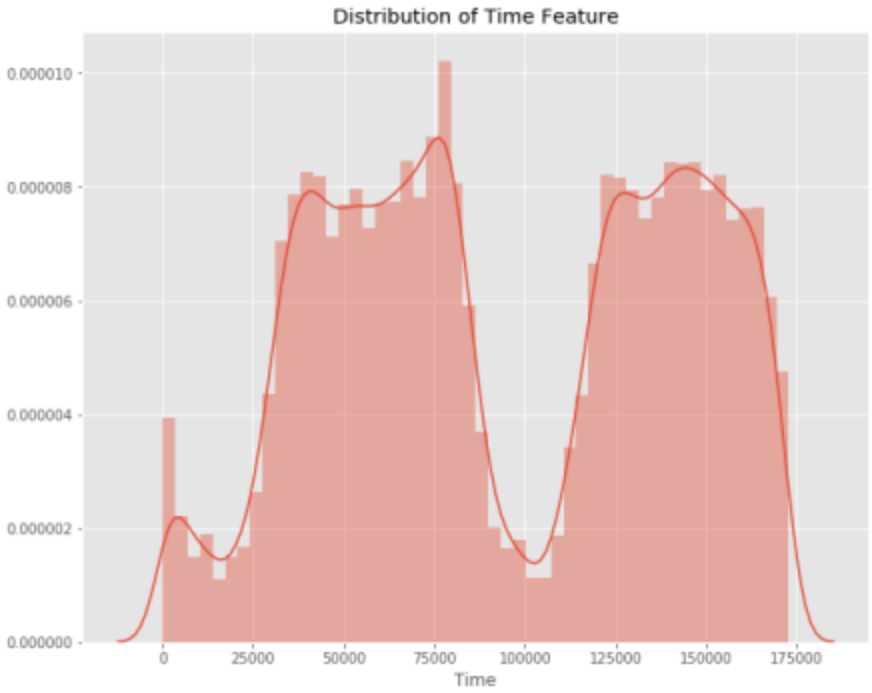
values of correlation, we found that PCA was already performed on the dataset.



The dataset contains 284,807 transactions. The mean value of all transactions is $88.35 and the largest transaction amount is $25,691.16. The distribution of the monetary value of all transactions is heavily right-skewed. The vast majority of transactions are relatively small and only a tiny fraction of transactions comes even close to the maximum.

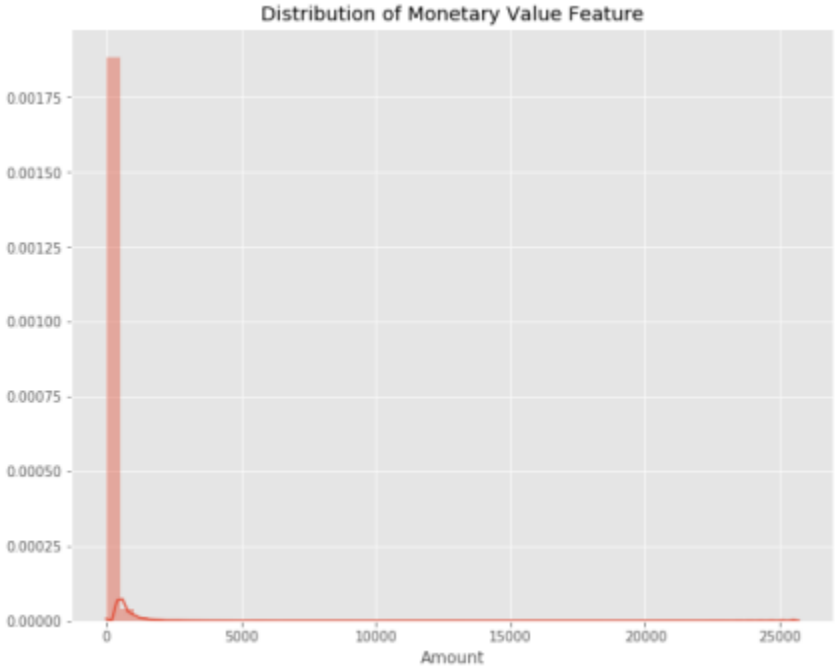


Visualization of Time and Amount variable can be seen as below:

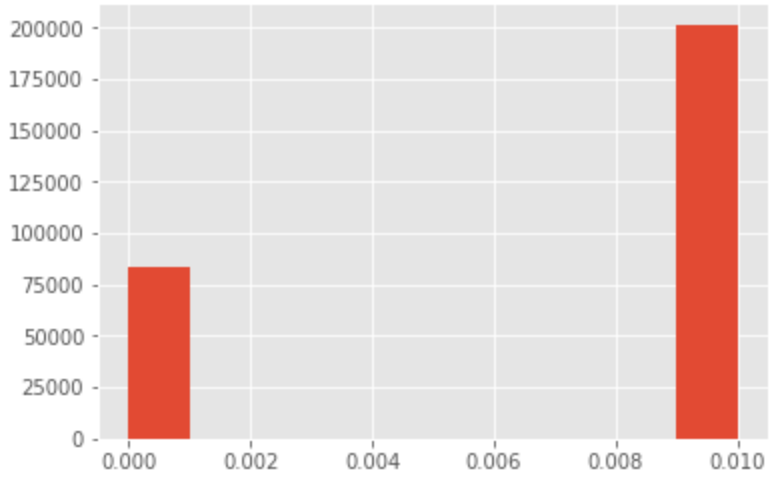


Distribution of Monetary Value is explained below:

From the dataset, we can see that the biggest transaction has a monetary value of $25,691. The mean of credit card transactions is around $88.

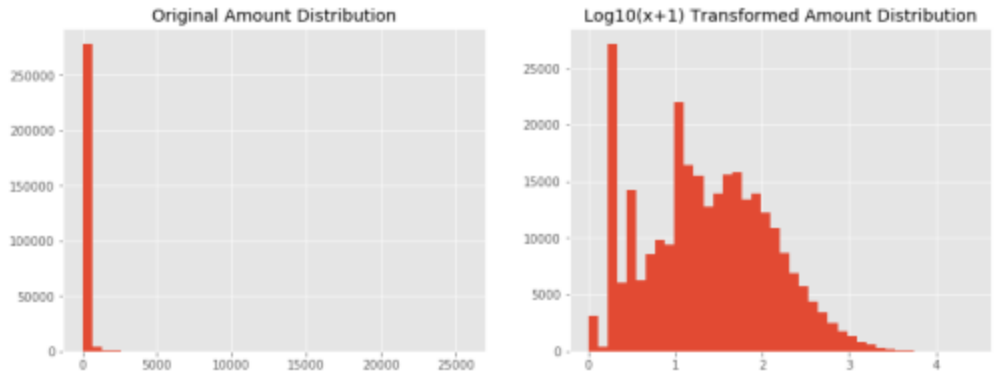


Amount values are rounded to hundredth place which can be seen from below graph.



For data cleaning part, we find the dataset has no missing values and it is already in pretty good shape. We just do a little more cleaning, adjusting the means of all the features to zero and the standard deviations to one. The result is shown below.

We have transformed “amount” values to log transform to give more normal distribution.



TSNE:

We have used TSNE to map higher dimensional data into two dimensions. From below graphs, we can see the fraud clusters with TSNE projections.

6.2. ML Models

1. **Logistic Regression**:

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 83289    18
         1  2009   127

               Accuracy : 0.9763
                 95% CI : (0.9752, 0.9773)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1085

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97645
            Specificity : 0.87586
         Pos Pred Value : 0.99978
         Neg Pred Value : 0.05946
             Prevalence : 0.99830
         Detection Rate : 0.97479
   Detection Prevalence : 0.97500
      Balanced Accuracy : 0.92615

       'Positive' Class : 0
```

From the result of the Logistic Regression model, we could find the accuracy and sensitivity are very high. It means that the majority predictions made by the Logistic Regression model are correct, and the high sensitivity percentages indicate that the true positive rate is very high. We could say Logistic Regression model is useful for predicting fraud and non-fraud transaction.

2. **Naïve Bayes**:

```
Confusion Matrix and Statistics

          Reference
Prediction    X0    X1
        X0 82640  2658
        X1    26   119

               Accuracy : 0.9686
                 95% CI : (0.9674, 0.9697)
    No Information Rate : 0.9675
    P-Value [Acc > NIR] : 0.03663

                  Kappa : 0.0785

 Mcnemar's Test P-Value : < 2e-16

            Sensitivity : 0.99969
            Specificity : 0.04285
         Pos Pred Value : 0.96884
         Neg Pred Value : 0.82069
             Prevalence : 0.96750
         Detection Rate : 0.96719
   Detection Prevalence : 0.99830
      Balanced Accuracy : 0.52127

       'Positive' Class : X0
```

From the result of the Naïve Bayes model, we also find the accuracy and sensitivity are very high. Similarly, we could say Naïve Bayes model is useful for predicting fraud and non-fraud transaction.

3. **Random Forest**:

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 82891    18
         1  2407   127

               Accuracy : 0.9716
                 95% CI : (0.9705, 0.9727)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0919

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97178
            Specificity : 0.87586
         Pos Pred Value : 0.99978
         Neg Pred Value : 0.05012
             Prevalence : 0.99830
         Detection Rate : 0.97013
   Detection Prevalence : 0.97034
      Balanced Accuracy : 0.92382

       'Positive' Class : 0
```

From the result of the Random Forest model, the accuracy and sensitivity are high too. We could say Random Forest model is useful for predicting fraud and non-fraud transaction.

4. **Xgboost**

We set up the train set as 70% of the whole dataset, the test set as 30% of the whole dataset. The train set contains 199364 data and the test set contains 85443 data.

```
classes 0, 1:  284315 492
train, test:  199364 85443
```

Then we run the xgboost algorithm until it doesn't improve recall to maximize recall on the test set. We got result as below.

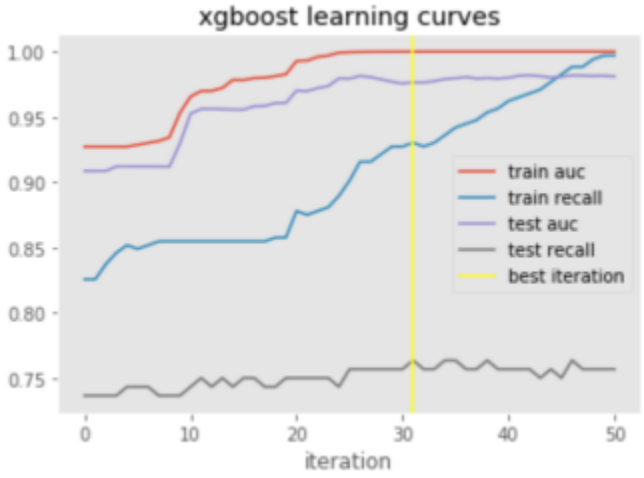
```
best iteration:  31
('recall', 0.76351351351351349)
('precision', 0.94166666666666665)
('roc_auc', 0.97648978188576063)
Confusion Matrix
```

	Pred 0	Pred 1
True 0	85288	7
True 1	35	113

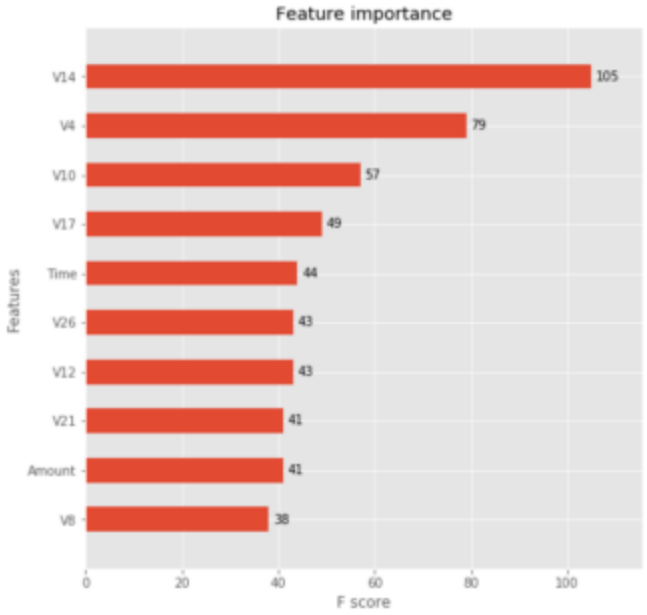
```
Accuracy : 0.9995084442259752
```

From the result of the Xgboost model, we could find we got a pretty high accuracy (0.9995). Compared to the results we got from Logistic Regression model, Naïve Bayes model and Random Forest model, we could say Xgboost mode is highly recommended to be used for predicting fraud and non-fraud transaction, which is the same as we expected.

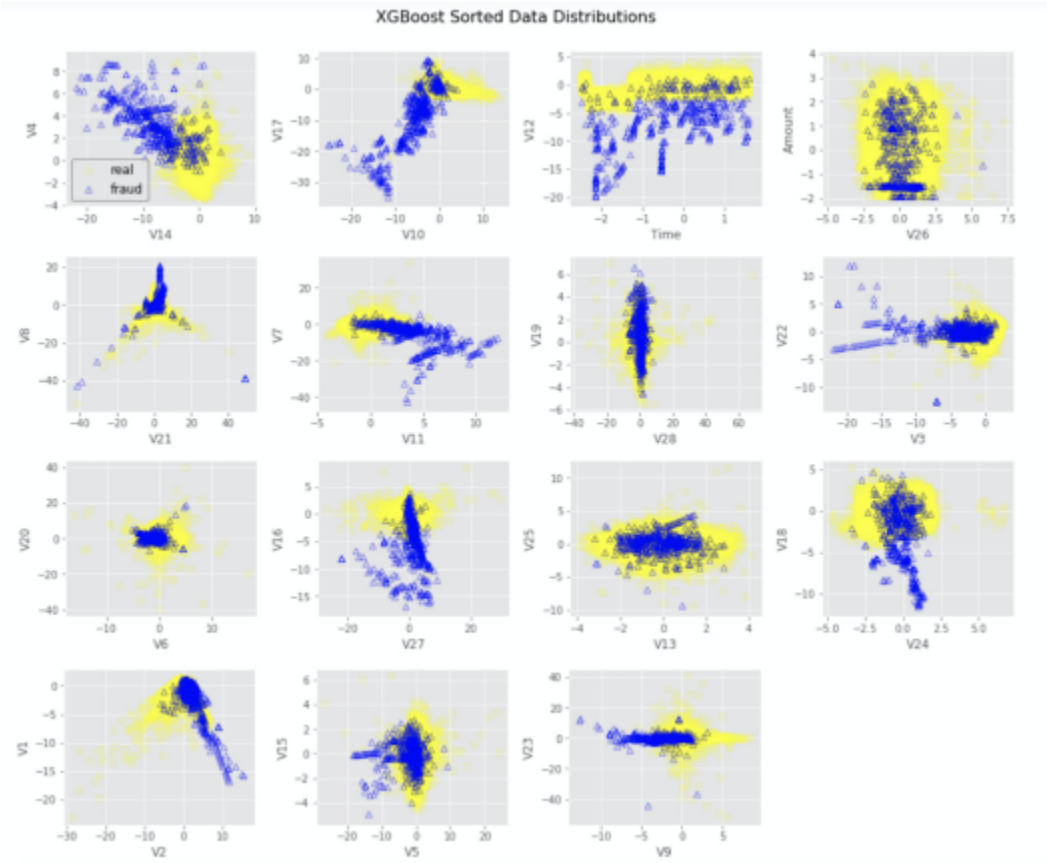
When we look at the how the metrics changed on the train and test sets as more trees were added , we could find the best iteration for this model is 31. Train accuracy, train recall and test accuracy all significantly improves as the iteration increase before it reaches the best iteration. Test recall doesn't have significant improvement as the iteration increases.



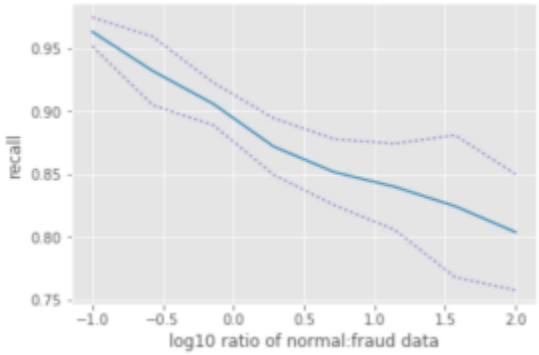
We also get a list of features sorted by their importance in detecting fraud. We can use the most important features to help visualize our results late.



The result is shown below if we plot all of the training data with paired features sorted by importance.



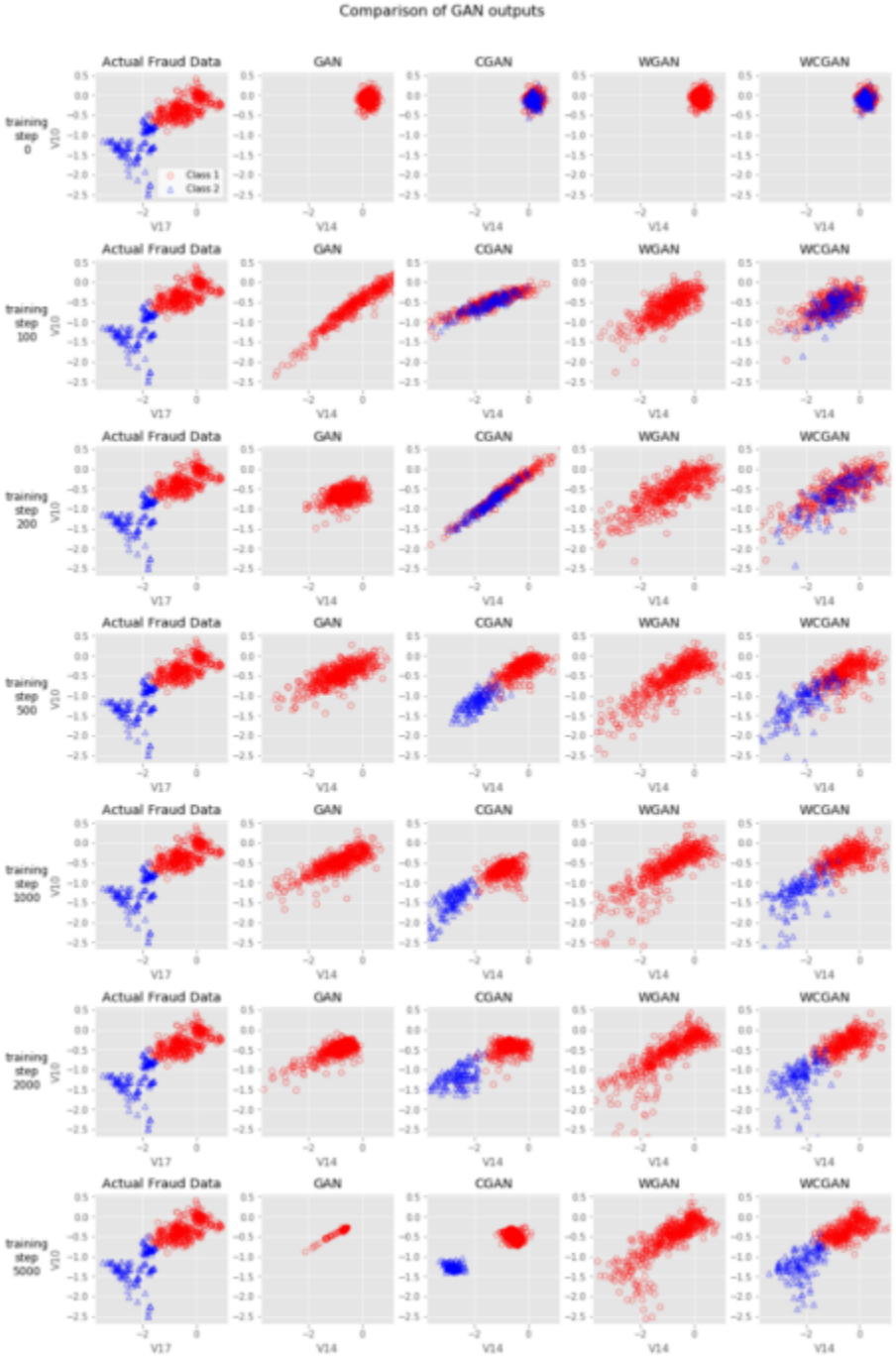
We find that recall will decrease if more normal data is added. So if we had more fraud data, we might be able to achieve a higher recall. For this reason, We will try to generate new, realistic fraud data using GANs to help us detect actual fraud.



6.3. GAN

For the GAN training, we train each GAN for 5000 rounds and examine the results. The result is shown below, we could see the actual fraud data and the generated fraud data from different GAN architectures as training progresses. The actual fraud data is divided into 2 KMeans classes. The two GANs that do not have class information, GAN and WGAN, have their generated output all in one class. The two GANs that have class

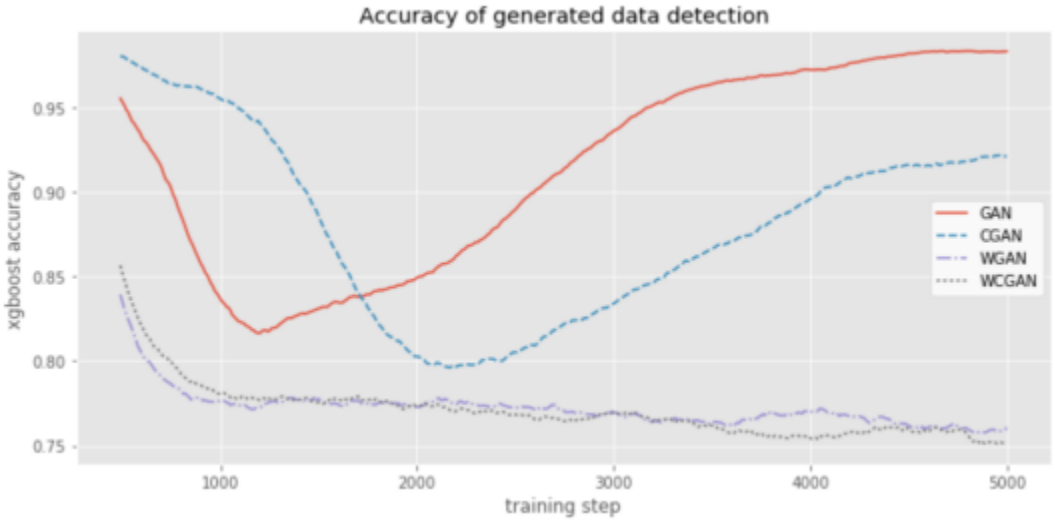
information, CGAN and WCGAN, show their generated data by classes.



We could see the original GAN architecture starts to learn the shape and range of the actual data first, but then it collapses towards a small distribution later. The CGAN architecture does a little better, it spreads out and approaches the distributions of each class of fraud data first, but it collapses at step 5000. The WGAN does not have the problem of collapse that GAN and CGAN have. It assumes the non-normal distribution

of the actual fraud data without the class label. The WCGAN works similarly to WGAN and it is able to generate the separate classes of data.

We train the xgboost classifier using half the actual fraud data (246 samples) and an equal number of GAN-generated examples. Then we test the xgboost classifier using the other half of the actual fraud data and a different set of 246 GAN generated examples. If we use perfectly realistic generated data, the xgboost algorithm should achieve an accuracy of 50%, which is not better than guessing.



For GAN generated data, we could find it decreases at first, and then increase after training step 1000 as mode collapse happens. For CGAN generated data achieves realistic data first, but collapses too after 2000. For WGAN and WCGAN generated data, both of them achieve more realistic data faster. The WCGAN does not appear to have big differences between WGAN. We could find the created classes may not be useful for the Wasserstein GAN architectures. Based on the results we got, we think WCGAN is the most suitable architecture for generating fraud transaction data.

We could test whether we generate new fraud data realistic enough to help to detect actual fraud data. We could choose the trained generator with the lowest accuracy and use it to generate data. For the training set, we use 70% of the non-fraud data (199,020 cases) and 20% of the fraud data (100 cases). Then we add different amounts of real or generated fraud data to the training set, up to 70% of the fraud data (344 cases). For the test set, we use the other 30% of the non-fraud cases (85,295 cases) and fraud cases (148 cases). We could add generated data from an untrained GAN and from the best trained GAN to test whether the generated data is better than random noise or not. From our tests, it appears that our best architecture was the WCGAN at training step 4800, where it achieved an xgboost accuracy of 70% (accuracy ideally would be 50%). So we use WCGAN to generate new fraud data.

From the result we get, we could find the recall significantly increases when actual fraud data is used to supplement the training set. The highest xgboost accuracy is about 82%, which is higher than the result we got using WCGAN generated data. We think the performance of generating new realistic data using GAN is not as good as we expected.

7. Result

The best architecture we got, WCGAN, achieves highest xgboost accuracy 70%, which is higher than 50% ideal accuracy. We could say generating data using GAN has positive effects on detecting fraud transactions based on limited amount dataset. However, it doesn't achieve as good performance as we expected. It appears that the data generated by GAN is not realistic enough to help us detect actual fraud.

There are some methods to improve the performance we could try. First, we could train longer, with larger networks and different parameters for the architectures we tried. The trends in xgboost accuracy and discriminator loss suggest more training will help the WGAN and WCGAN architectures.

Secondly, we could revisit the data cleaning process. We could create some new variables and how we address skewness in features. Perhaps different classification schemes of the fraud data would help.

Thirdly, we could try other GAN architectures. The DRAGAN has theoretical and experimental evidence shows that it trains faster and more stably than the Wasserstein GANs. It may help us in generating realistic enough data to detect fraud transactions with limited training sets.

8. Lesson Learnt

From this project, we learn a new machine learning algorithm, xgboost, to detect fraud transactions. We also have a better understanding of different types of GAN. We learnt more about their characteristics, how to use them to generate new data, and their weaknesses for generating new data. In addition, we have more experience of using Amazon Web Services (AWS). We have a better understanding of different functions of AWS and how to store and analysis real data in AWS too. The knowledge we learnt is very interesting and useful for our study and career. We will do some further study on it in our following study.