PROJECT REPORT

CSD 361
Introduction to Machine Learning

Fraud detection in customer transactions

Akarsh Tyagi
1910110040

Anshit Bansal
1910110074

Diya Sachdev
1910110140

Acknowledgement

We deem it a pleasure to acknowledge our sense of gratitude to our project guide **Prof. Harish Karnick** under whom we have carried out our project. His inclusive and objective guidance and timely advice encouraged us with constant flow of energy to work.

My completion of the project would not have been possible without the help and support of the teaching assistants - Prakhar Rathi and Jaydeep Kishore. They provided us with a strong academic atmosphere by enforcing strict discipline to do the project work with utmost concentration and dedication.

Finally we must say that no height is ever achieved without some sacrifices made at some end and it is here where we owe our special debt to our parents and our friends for showing their generous love and care throughout the entire period of time.

Date : 01 May 2022
Place : Shiv Nadar University, Dadri

Akarsh Tyagi
1910110040

Anshit Bansal
1910110074

Diya Sachdev
1910110140

Abstract

Imagine standing at the check-out counter at the grocery store with a long line behind you and the cashier not-so-quietly announces that your card has been declined. At this moment, you probably aren't thinking about the data science that determined your fate. As you step aside and allow the cashier to tend to the next customer, you receive a text message from your bank. "Press 1 if you really tried to spend $500 on cheddar cheese."

While perhaps cumbersome (and often embarrassing) at the moment, this fraud prevention system is actually saving consumers millions of dollars per year.

Our main objective is to develop the machine learning models on a challenging large-scale dataset. The data comes from Vesta's real-world e-commerce transactions and contains a wide range of features from device type to product features. We also had the opportunity to create new features to improve our results.

If successful, we'll improve the efficacy of fraudulent transaction alerts for millions of people around the world, helping hundreds of thousands of businesses reduce their fraud loss and increase their revenue. And of course, we will save party people just like us the hassle of false positives.

Content

Introduction

Machine learning is a sub-domain of computer science which evolved from the study of pattern recognition in data, and also from the computational learning theory in artificial intelligence. It is the first-class ticket to most interesting careers in data analytics today.

The training and test set consists of a set of examples consisting of input and output vectors, and the goal of the machine learning algorithm is to infer a function that maps the input vector to the output vector with minimal error. In an optimal scenario,
a model trained on a set of examples will classify an unseen example.

A wide array of supervised machine learning algorithms are available to the machine learning enthusiast, for example Neural Networks, Decision Trees, Support Vector Machines,
Random Forest etc. There is no single algorithm that works for all cases. In this project, we try to find patterns in the dataset provided to us by Vesta Corporation which is a sample of transaction details via online e-transfers and the question to be answered is if the transaction is Fraud or not.

Vesta Corporation is the forerunner in guaranteed e-commerce payment solutions. Vesta has firmly expanded data science and machine learning capabilities across the globe and solidified its position as the leader in guaranteed ecommerce payments. Today, Vesta guarantees more than $18B in transactions annually.

Software Requirements

1. Product Perspective

The purpose of the project work is to provide an efficient algorithm for the purpose of the same. In this world of evolving wireless communications it is becoming incredibly important for us to have a secure channel for electronic cash transfer. So we aim at developing a machine learning model which can detect if the transaction is fraud or not and especially try to reduce any possibility of a fraud transaction being labeled as true.

2. User Characteristics

The users who are developing this whole application or are trying to make an evaluation of the proper working and algorithms implemented need to be proficient in the following areas:

- Python and Jupyter notebook for coding
- Supervised machine learning algorithms like Randomized forest and decision trees to fit the model.
- Undersampling vs Oversampling and how to balance the dataset.

3. Specific Requirements

The whole project was developed on a jupyter notebook but due to the large size of the dataset, the program crashed a number of times. So the specific requirements for this project work are :

- Jupyter Notebook
- System space to handle the entire dataset.
- Compatible operating system.

4. Performance Requirements

The algorithms developed are purely based on a wireless domain and will be implemented on embedded devices as an end product. So it is not our concern about the performance of the embedded devices rather the concern of the company Vesta corporation. For the purpose of development, the only performance requirements are related to the specific software requirements mentioned above and a system which is capable of smoothly running the entire program and good enough for the project members to work on the proposed algorithms.

Dataset

The data is broken down into two files: identity and transactions which are joined by Transaction ID. And we have to predict the probability that an online transaction is fraudulent, denoted by the binary target IsFraud.

Categorical Features - Transaction

- ProductCD
- card1 - card6
- addr1, addr2
- P_emaildomain
- R_emaildomain
- M1 - M9

Categorical Features - Identity

- DeviceType
- DeviceInfo
- id_12 - id_38

The TransactionDT feature is a timedelta from a given reference datetime (not an actual timestamp).

The original dataset was of the size 590540 rows × 434 columns.

Exploratory Data Analysis EDA

After seeing the database, we quickly realized that a lot of the data values were missing, that is, NaN

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	card4	card5	...	V330	V331	V332	V333	V334	V335
590520	3577520	0	15810785	93.000	W	17150	292.0	150.0	visa	226.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590521	3577521	0	15810802	48.877	C	12019	305.0	106.0	mastercard	224.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590522	3577522	0	15810823	54.500	W	3166	559.0	150.0	visa	166.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590523	3577523	0	15810833	58.950	W	13076	456.0	150.0	mastercard	117.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590524	3577524	0	15810836	75.000	W	7826	481.0	150.0	mastercard	224.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590525	3577525	0	15810866	57.950	W	11942	570.0	150.0	visa	226.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590526	3577526	1	15810876	250.000	R	1214	174.0	150.0	visa	226.0	...	0.0	0.0	0.0	0.0	0.0	0.0
590527	3577527	0	15810883	189.950	W	6453	555.0	150.0	visa	226.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590528	3577528	0	15810907	279.950	W	15066	170.0	150.0	mastercard	102.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590529	3577529	0	15810912	73.838	C	5096	555.0	185.0	mastercard	137.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590530	3577530	0	15810926	400.780	W	15066	170.0	150.0	mastercard	102.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590531	3577531	0	15810935	400.000	R	6019	583.0	150.0	visa	226.0	...	0.0	0.0	0.0	0.0	0.0	0.0
590532	3577532	0	15811007	204.970	W	12037	595.0	150.0	mastercard	224.0	...	NaN	NaN	NaN	NaN	NaN	NaN
590533	3577533	0	15811029	107.950	W	13071	321.0	150.0	visa	226.0	...	NaN	NaN	NaN	NaN	NaN	NaN

([1] - Reference picture for missing values in dataset)

Thus, a detailed analysis and preprocessing of the dataset was done as the first step which helped us in figuring out the best models / parameters for the algorithm.

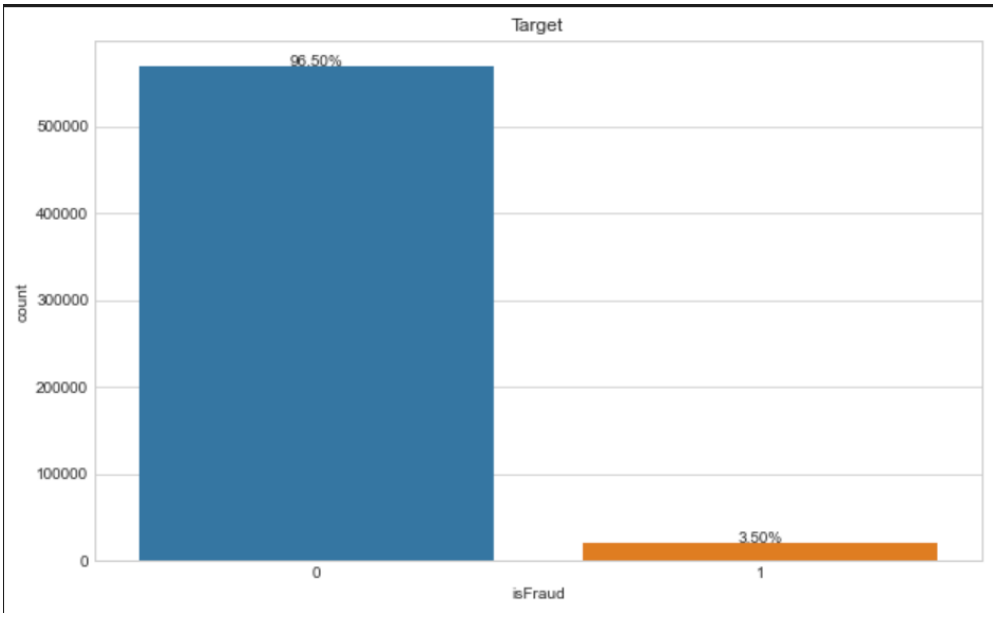
1. Data

1.1. Number of unique and missing values in the dataset

```
cat_cols = ['ProductCD','card1','card2','card3','card4','card5','card6','P_emaildomain','R_emaildomain','M1',
'M2','M3','M4','M5','M6','M7','M8','M9','id_12', 'id_13', 'id_14', 'id_15', 'id_16', 'id_17', 'id_18', 'id_19',
'id_20', 'id_21', 'id_22', 'id_23', 'id_24', 'id_25', 'id_26', 'id_27', 'id_28', 'id_29','addr1','addr2',
'id_30', 'id_31', 'id_32', 'id_33', 'id_34', 'id_35', 'id_36', 'id_37', 'id_38', 'DeviceType', 'DeviceInfo']

for col in cat_cols:
    df = pd.concat([df_train[col],df_test[col]],axis=0)
    sh = df.value_counts().shape[0]
    missing = round((df.isnull().sum()/df.shape[0])*100,2)
    print(f'No of unique values in {col} is {sh} | missing percent is {missing}%')
```

```
Output exceeds the size limit. Open the full output data in a text editor
No of unique values in ProductCD is 5 | missing percent is 0.0%
No of unique values in card1 is 17091 | missing percent is 0.0%
No of unique values in card2 is 501 | missing percent is 1.6%
No of unique values in card3 is 133 | missing percent is 0.42%
No of unique values in card4 is 4 | missing percent is 0.42%
No of unique values in card5 is 138 | missing percent is 0.8%
No of unique values in card6 is 4 | missing percent is 0.42%
No of unique values in P_emaildomain is 60 | missing percent is 14.91%
No of unique values in R_emaildomain is 60 | missing percent is 75.1%
No of unique values in M1 is 2 | missing percent is 40.81%
No of unique values in M2 is 2 | missing percent is 40.81%
No of unique values in M3 is 2 | missing percent is 40.81%
```

1.2. Checking how imbalanced the entire dataset is

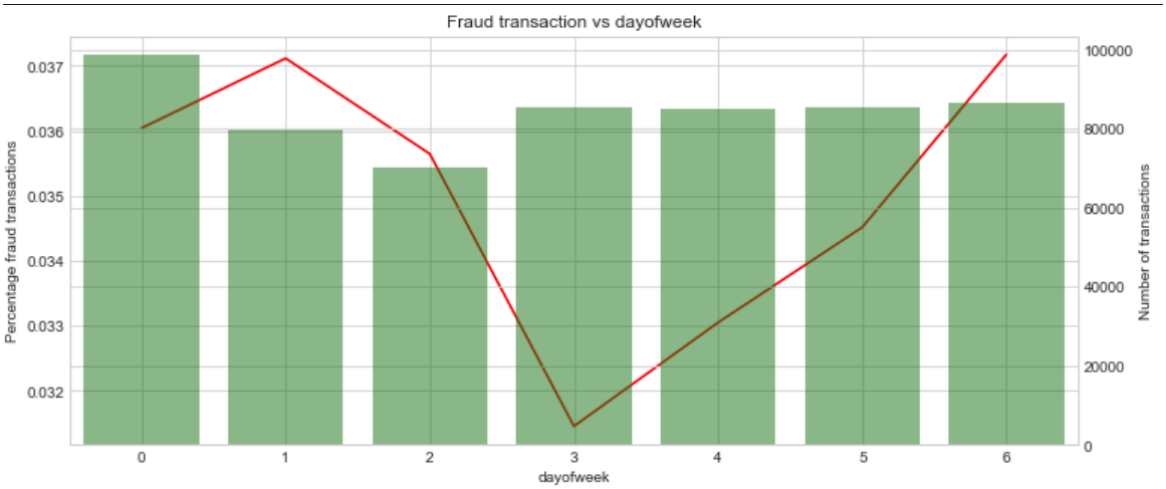


We plotted a bar graph and found that about **97%** of the transaction details in the dataset were true, while only **3% were Fraud.** This is not an ideal situation for the model to learn, since examples of Fraud transactions are really low. This is the case of highly imbalanced dataset.

Solution to the above problem, is to **undersample or oversample** the data values. Random oversampling involves randomly duplicating examples in the minority class, whereas random undersampling involves randomly deleting examples from the majority class. Undersampling does not help the model to learn, rather lose information, thus, we decided to implement oversampling.

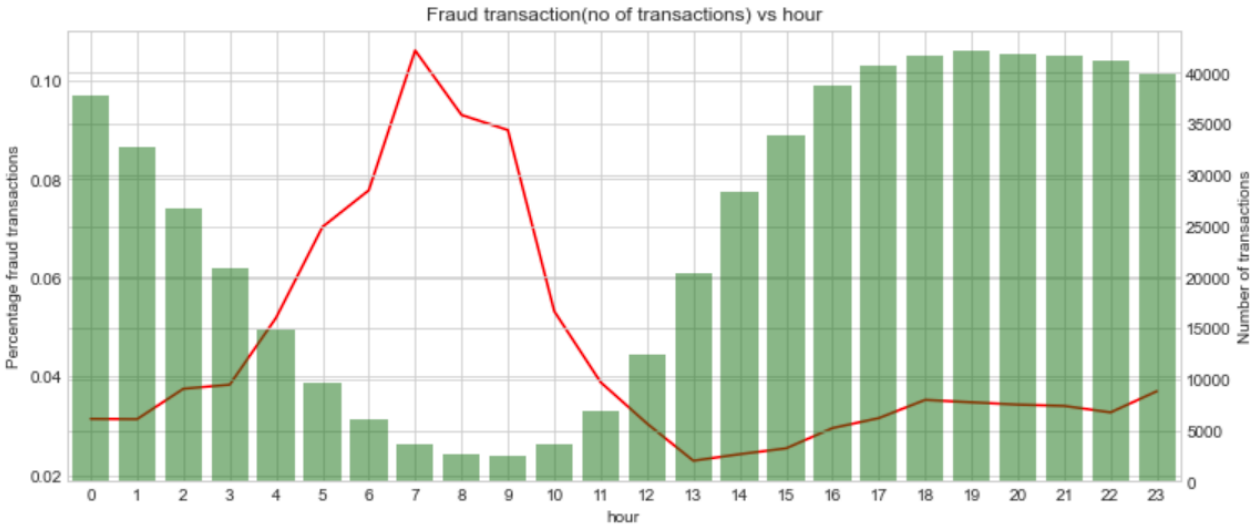
1.3. Checking for trends in Fraud Transactions

1.3.1 According to days of week



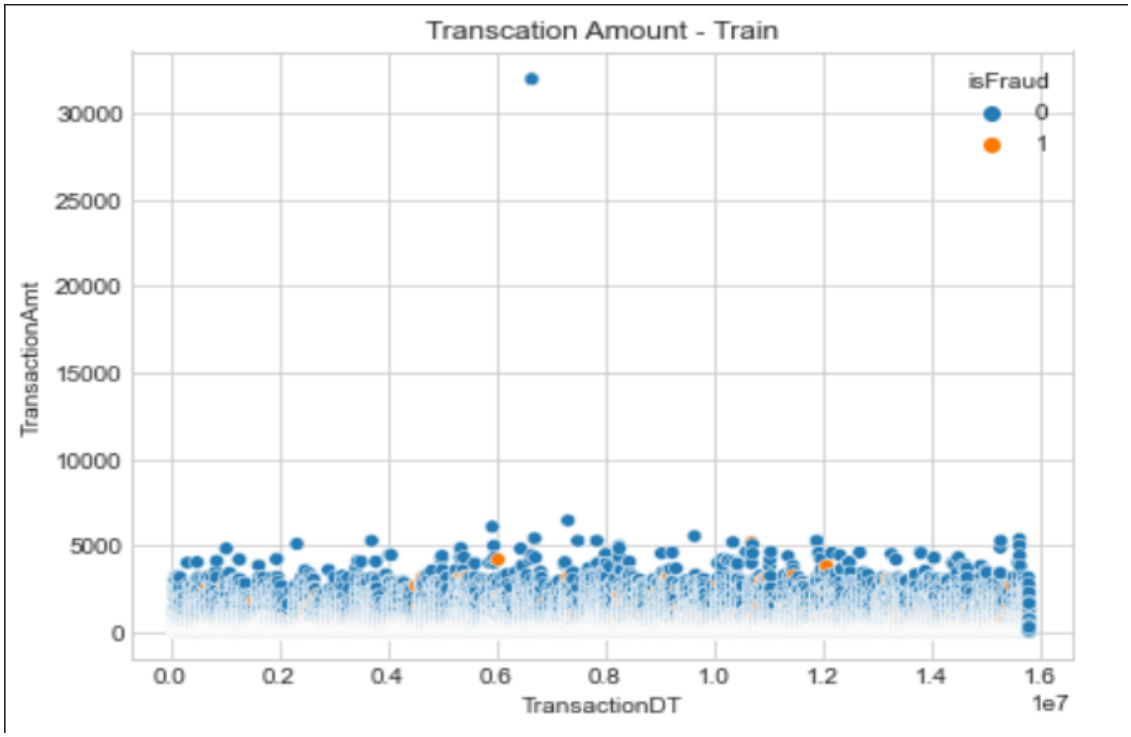
Observation - Number of fraudulent transactions reduced in the middle of week, i.e, probability of such transactions on Tuesdays, Wednesdays and Thursdays are less than the rest of the week.

1.3.2. According to the hour during a day

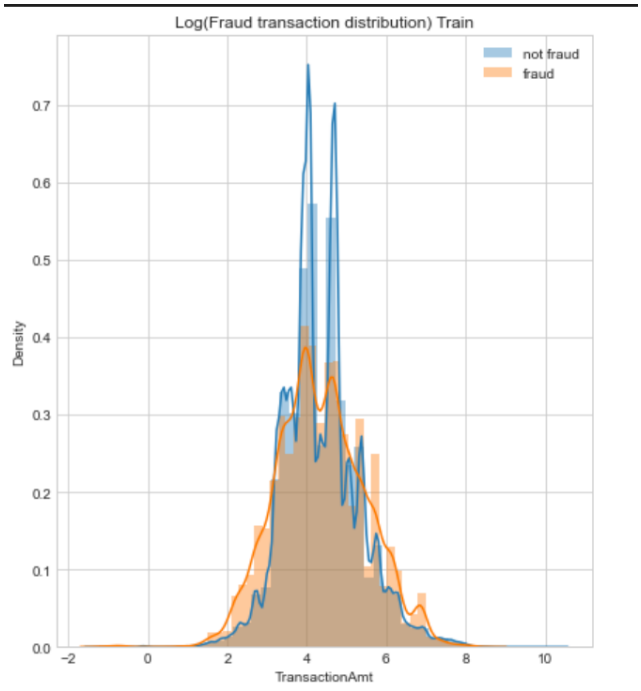


Observation - Probability of fraud transactions increases tenfold during the mornings between 4 AM to 12 noon, even though the number of transactions during that time are low.

1.3.3 Checking for outliers



Observation - There was only 1 outlier, while plotting the graph between transaction amount and Transaction time delta. Due to such skewness of the transaction amount, we took log and plotted the graph again.



Observation - According to the above graph, the transactions with LogTransactionAmt' larger than 5.5 and smaller than 3.3 have a higher frequency and probability density of being fraudulent. On the other hand, the ones with 'LogTransaction -Amt ' from 3.3 to 5.5 have a higher chance of being legit.

1.3.4 Checking for mean transaction amount in Fraud transactions

Observation - The above discussion made us curious on transaction details in Fraud transfers.

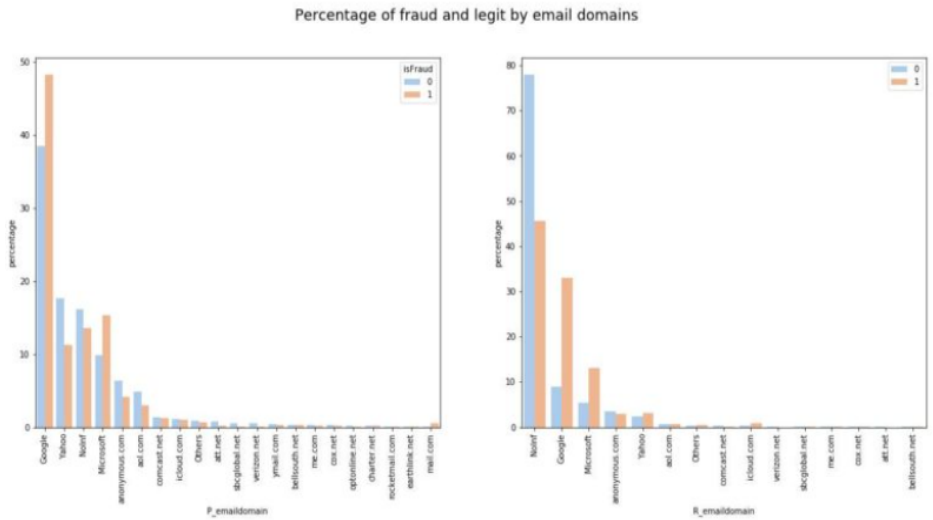
Mean transaction amt for fraud is 4.3506 ie, 77.5264 dollars

Mean transaction amt for non-fraud is 4.3643 ie, 78.5978 dollars

Median transaction amt for fraud is 4.3175 ie, 75.0000 dollars

Median transaction amt for non-fraud is 4.2268 ie, 68.500 dollars

1.3.5. Trends in emails used during transactions.



Observation - In purchaser set, most emails are gmail.com, but more fraud transactions are done using protonmail.com (90% approx)

Preprocessing Data

2. Reducing the dataset

```
features = data.columns
missing_data = pd.DataFrame(features, columns=['Features'])
null_perc = []

for feature in features:
  null_perc.append(((data[feature].isna().sum())/data.shape[0])*100)

missing_data['null_perc'] = null_perc
```

```
missing_data.sort_values(by=['null_perc'], ascending=False, inplace=True, ignore_index=True)
```

```
missing_data.head(15)
```

	Features	null_perc
0	id_24	99.196159
1	id_25	99.130965
2	id_07	99.127070
3	id_08	99.127070
4	id_21	99.126393

In the dataset, there were features with more than 90% missing values, such columns did not add any value to the algorithm and only increased the time and space complexity.

Thus, we decided to drop features with more than 90% missing values.

3. Filling missing values in the dataset

The feature space was made up of two kinds of columns - numerical and categorical.

For missing values in the numerical dataset, we calculated the median in each case / column variable.

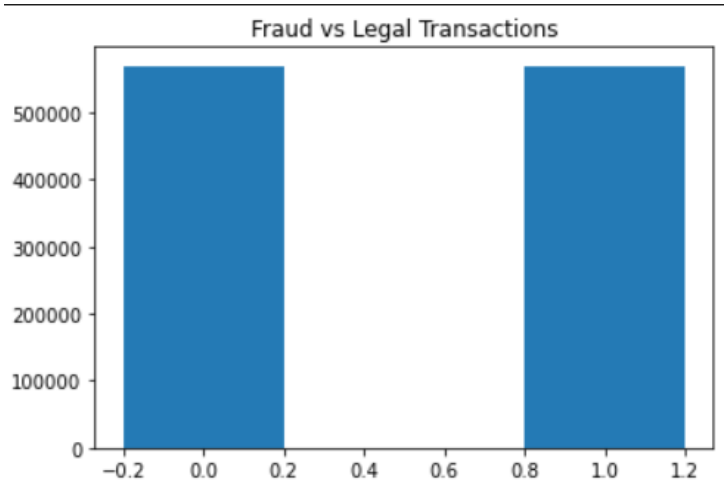
For the categorical columns, we used two approaches.

1. If there were many unique values, we simply changed NaN to ‘missing’ string.
2. If there were only 2 - 3 types of values, we calculated the mode and converted all the missing values to the mode.

4. Oversampling the dataset.

Random Oversampling involves supplementing the training data with multiple copies of some of the minority classes. Oversampling can be done more than once (2x, 3x, 5x, 10x, etc.)

There are a number of methods available to oversample a dataset used in a typical classification problem (using a classification algorithm to classify a set of images, given a labeled training set of images). The most common technique which we have also used is known as **SMOTE: Synthetic Minority Over-sampling Technique.**



In this technique, we create a feature space and oversampling is done by taking any random two points, we get the third point by taking the middle of the two available points in the subspace.

After oversampling, the number of true and fraudulent transactions in the dataset were made equal, with about 6 lakhs of each kind of transaction. This helps in modeling the algorithm and getting better results.

Now, that our dataset was complete, we encoded all the categorical features into numerical columns and got our X - features and Y - IsFraud column to fit our model

Machine Learning Algorithms and Models

For the modeling, we implemented two algorithms - Decision tree and random forest. Below we will discuss the performance of both these algorithms on test set, using ***precision***, ***confusion matrix*** and ***AUC ROC*** curve. As discussed earlier, we won't consider ***accuracy*** as the performance metrics for the model as the data is highly imbalance, thus we can't get an estimate of how good our model are trained.

In the fraudulent problem, we need to reduce ***false positives***, since it is okay to have a warning for a legitimate transaction to be classified as fraud, but it is bad to have a legitimate transaction classified as fraud.

1. Decision trees

On fitting the model on the **unbalanced dataset** we noticed that we were getting a good overall accuracy but the accuracy for the individual labels was not balanced. We can see that the model is mostly predicting each transaction to be non fraudulent and still giving a higher accuracy.

```
pred1 = dt_clf.predict(X_test)
confusion_matrix(y_test, pred1)
```

```
array([[111980,   1957],
       [  1660,   2511]])
```

```
print(classification_report(y_test, pred1))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98    113937
           1       0.56      0.60      0.58      4171

    accuracy                           0.97    118108
   macro avg       0.77      0.79      0.78    118108
weighted avg       0.97      0.97      0.97    118108
```

In the above classification report, we can see that we are getting a precision score of 0.56 for fraud transactions, which signifies that out of all the transactions classified as fraud only 50% of the transactions were fraudulent.

As told to us during the presentation, now we have used the actual test set which was given to us without any up-scaling.

On fitting the model on the balanced dataset we noticed that the model is predicting both the categories better but there is still scope of improvement.

```
pred3 = dt_clf2.predict(X_test)
confusion_matrix(y_test, pred3)
```

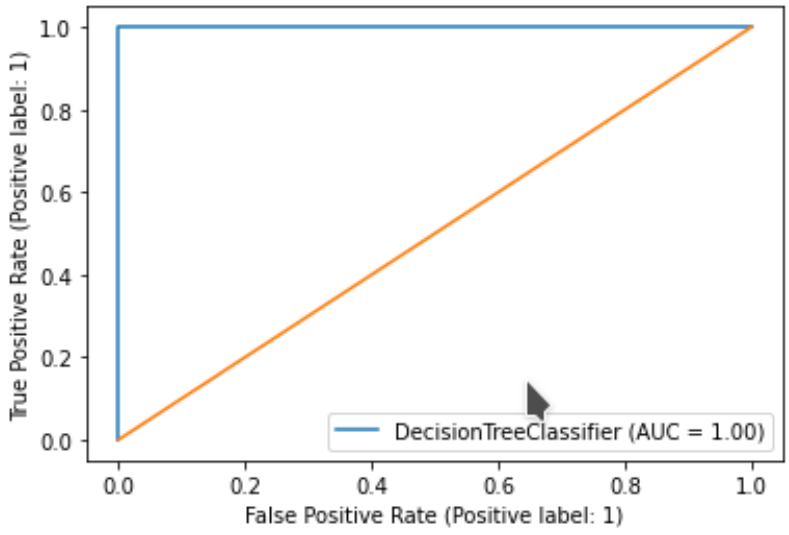
```
array([[113359,    578],
       [   397,   3774]])
```

```
print(classification_report(y_test, pred3))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    113937
           1       0.87      0.90      0.89      4171

    accuracy                           0.99    118108
   macro avg       0.93      0.95      0.94    118108
weighted avg       0.99      0.99      0.99    118108
```

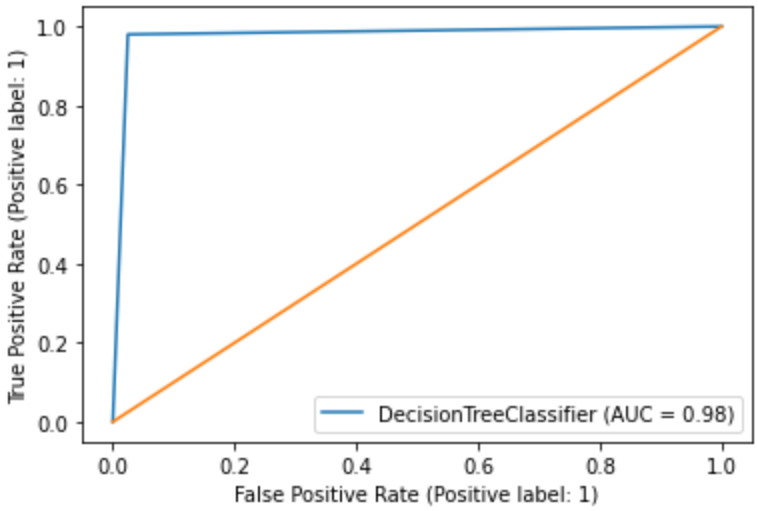
AUC ROC

```
[<matplotlib.lines.Line2D at 0x285bf63d0>]
```



Above given is the ROC curve for training data over the decision tree classifier, where we can see that the area under the curve is coming out to be 1, which is signifies that the model os highly fitting the training dataset.

```
[<matplotlib.lines.Line2D at 0x2d2f21280>]
```



While area under the curve for ROC curve is also almost close to 1 for the testing data set, but still there are chances this might be the case of **over fitting** of the data as although model is having low precision but it is highly fitting the training data set.

2. Random Forest

Results on unbalanced dataset

On fitting the imbalanced dataset over the Random Forest classifier, we were able to get a good precision score but the score for recall and f1 score is very poor.

```
pred2 = rf_clf.predict(X_test)
confusion_matrix(y_test, pred2)
```

```
array([[113851,     86],
       [  3093,   1078]])
```

```
print(classification_report(y_test, pred2))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99    113937
           1       0.93      0.26      0.40      4171

    accuracy                           0.97    118108
   macro avg       0.95      0.63      0.70    118108
weighted avg       0.97      0.97      0.97    118108
```

Results on balanced dataset

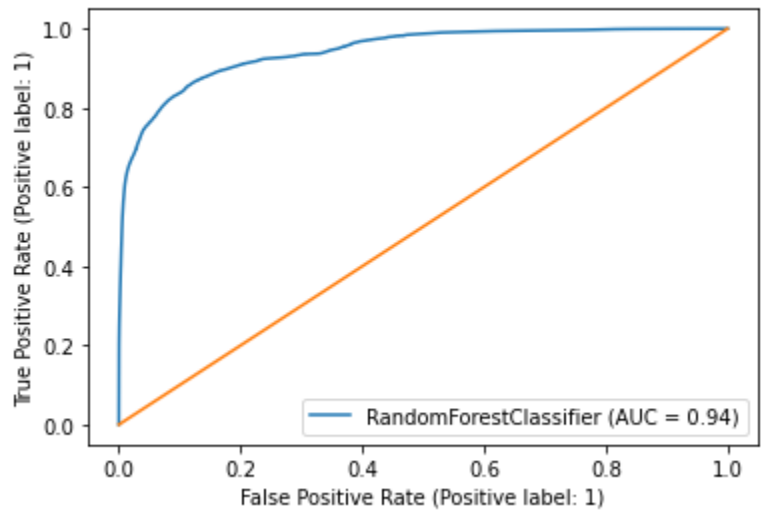
```
: pred4 = rf_clf2.predict(X_testOS)
  confusion_matrix(y_testOS, pred4)

: array([[100802,  13136],
         [ 16330,  97683]])

: print(classification_report(y_testOS, pred4))b
```

	precision	recall	f1-score	support
0	0.86	0.88	0.87	113938
1	0.88	0.86	0.87	114013
accuracy			0.87	227951
macro avg	0.87	0.87	0.87	227951
weighted avg	0.87	0.87	0.87	227951

Now, on fitting the random forest over the upscaled dataset(balanced), we are getting a quite good score for all i.e precision, recall and f1 score, but still there is a scope of improvement.

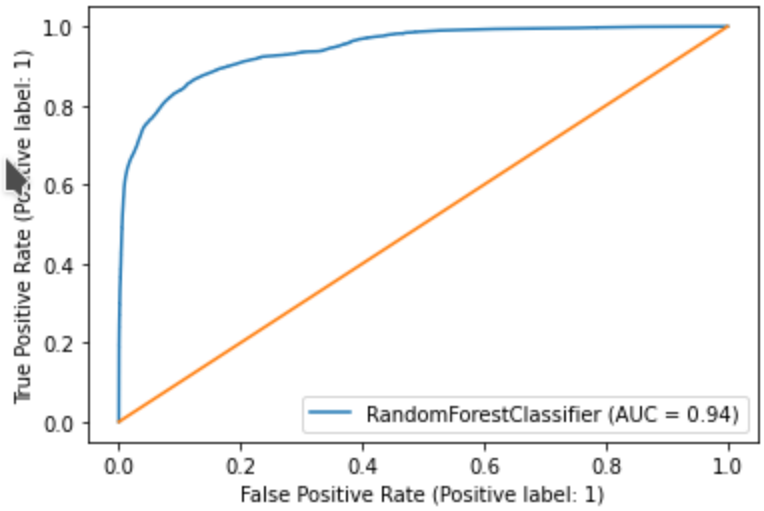
AUC ROC

```
[<matplotlib.lines.Line2D at 0x2d2f59a90>]
```



Above shown is th ROC curve for training dataset over random forest classifier, which is having 0.94 as the area under the ROC curve.

```
[<matplotlib.lines.Line2D at 0x2d2eeda30>]
```



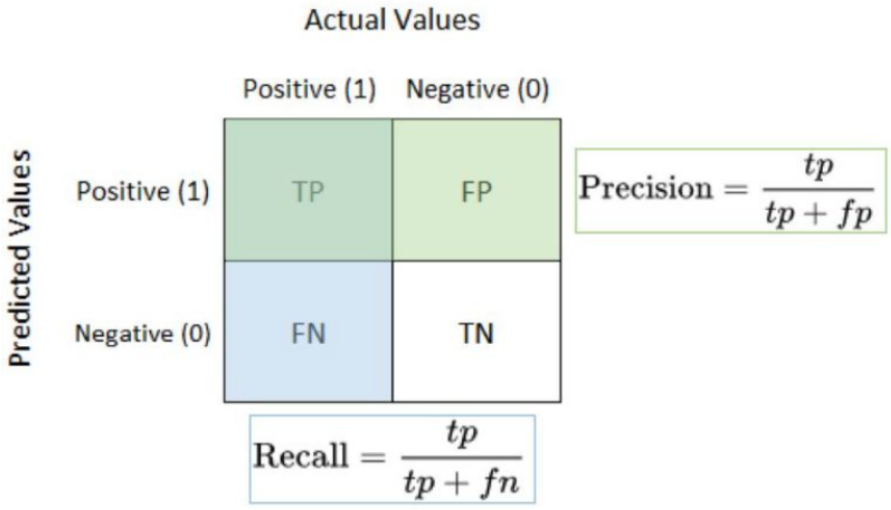
Now above is the ROC curve for testing dataset over the random forest classifier, which is having an area of around 0.94 just same as the training dataset, which shows that the random forest classifier is having a consistent performance over both testing and training dataset, thus reducing the chances of over fitting of the model.

Observations and Results

1. Observations and Parameters for result

Since our dataset was highly imbalanced at first, and we had done oversampling and pre-processing, overall accuracy is not the right parameter to check the efficiency of our models .

Instead, we look at precision and recall of the two models. **Precision** answers the question - What proportion of positive identifications were actually correct ? While **Recall** finds the proportion of actual positives identified correctly



In our case of finding fraudulent transactions, it is much more important to not have false positives, i.e, if we have a fraud transaction it should not be labeled as a true transaction.

Thus, we keep **precision** as our most important parameter for deciding which model to choose.

2. Best parameters using RandomizedSearchCV

RandomizedSearchCV is very useful when we have many parameters to try and the training time is very long like in this case.

```
from sklearn.model_selection import RandomizedSearchCV



parameters = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2,5,10, 'None'],
    'min_samples_leaf': [2,4,8,10,20]
}



RS_cv = RandomizedSearchCV(dt_clf2, parameters)



score = RS_cv.fit(X_trainOS, y_trainOS)
```

The best parameters we got on running the cell were :-

- Min_sample_leaf - 2
- Max_depth - 10
- Criterion - gini

And the best score we got was - 0.94840 . That is, 94.84%

Conclusions

In this project we were able to see how the dataset we received was unbalanced and we were able to see how the dataset is useless for prediction since even if it predicts wrong in all Fraudulent cases, the accuracy will come to be 97%.

Thus, preprocessing and cleaning the dataset are the most important and the time-consuming part of Machine learning. Implementation of models is much easier.
We also explored the method known as SMOTE for oversampling the data which helped us to balance the entire dataset and predict better results.

Upon analyzing the parameters and models, we concluded that it is not necessary to always implement complex models. Comaplex does not translate to better results. Like in this case, simple Decision trees gave us much better precision than the Random forest algorithm. Decision trees were both easier to train and gave better results as well.

```
    score.best_params_
```

```
{'min_samples_leaf': 2, 'max_depth': 10, 'criterion': 'gini'}
```

```
    score.best_score_
```

```
0.9484077148946224
```

References

1. https://www.kaggle.com/competitions/ieee-fraud-detection/
2. https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203
3. https://towardsdatascience.com/7-over-sampling-techniques-to-handle-imbalanced-data-ec51c8db349f
4. https://medium.com/analytics-vidhya/undersampling-and-oversampling-an-old-and-a-new-approach-4f984a0e8392
5. https://towardsdatascience.com/machine-learning-gridsearchcv-randomizedsearchcv-d36b89231b10