2020

COMP9417 Team Project

IEEE-CIS FRAUD DETECTION

MOHIT KHANNA (Z5266543), USAMA SADIQ (Z5235652), UTTKARSH SHARMA(Z5269665), SIBO ZHANG(Z5252570)

1. *Introduction:*

In this assignment we aim to implement a 'Transaction Fraud Prevention System' leveraging machine learning models, which aims to predict whether a given financial transaction is 'Fraudulent' or 'Valid'.
As the we move towards a more digitized ecosystem, more tools will have to be developed to secure individuals against frauds and any other mishaps.

The dataset for the model was taken from the Kaggle competition: https://www.kaggle.com/c/ieee-fraud-detection and was provided via the collaboration of IEEE and Vesta Corporation.

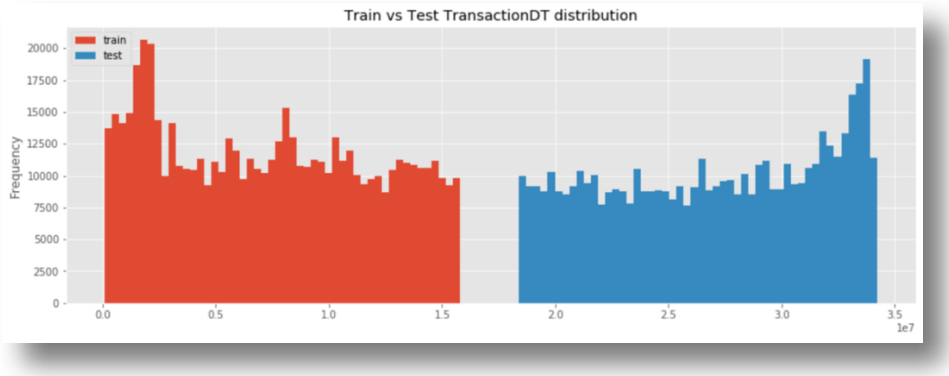
Summary:
After solving class imbalance, leveraging feature selection and Exploratory Data Analysis, we executed tested the following models for the given data:

- Decision Tree: This was our baseline model
- Bernoulli Naive Bayes
- K-Nearest Neighbour
- SVM: We could not get the conclusive answer via the SVM.
- Random Forest
- Light Gradient Boost
- Integrated Stacked Model

The final model is an LGB model with hyper parameter tuning giving the Kaggle Score of 93.

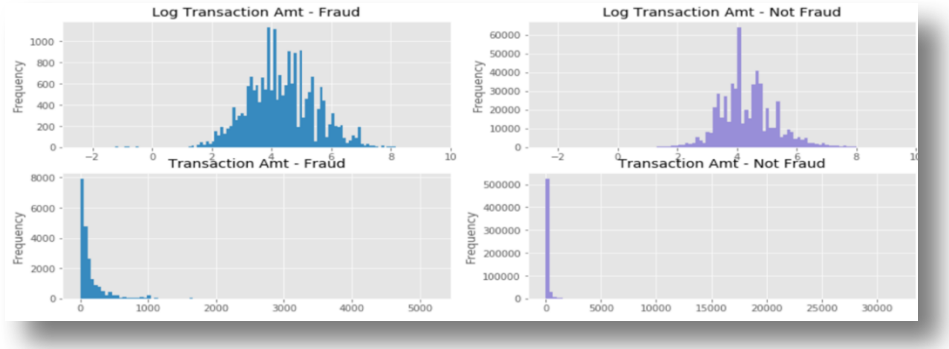
2. *Exploratory Data Analysis*

Figure 1



TransactionDT represents a time delta from a given reference DateTime.
It is not to be confused with an actual timestamp. On examining the data, it is evident that the training and test datasets are split by time.
The training dataset appears to be from an earlier period and the test dataset appears to be from a later period in time. This impacts the choice of cross-validation techniques.

Figure 2

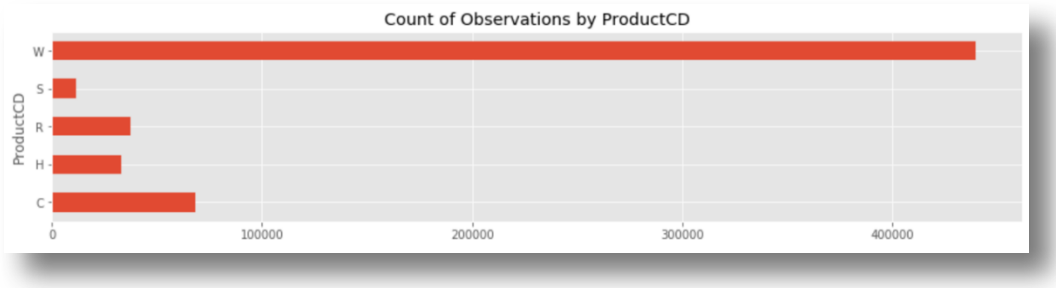


The exact meaning of ProductCD values is yet unknown.

However, we can conclude the following facts:

- 'W' has the most number of observations, 'C' has the least.
- ProductCD 'C' has the most fraud with >11%

Figure 4



'W' has the least with ~2%

Figure 5

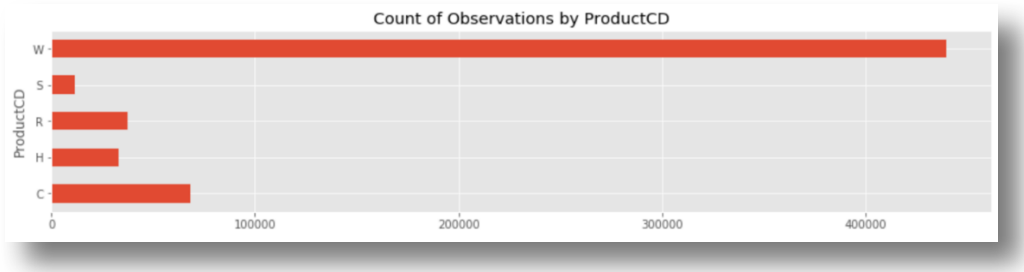
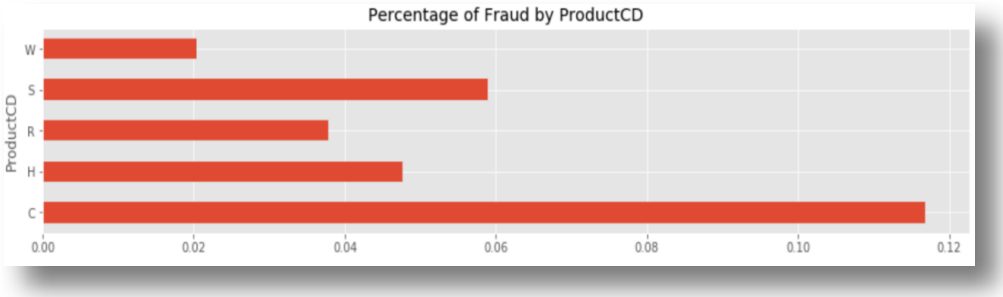


Figure 6



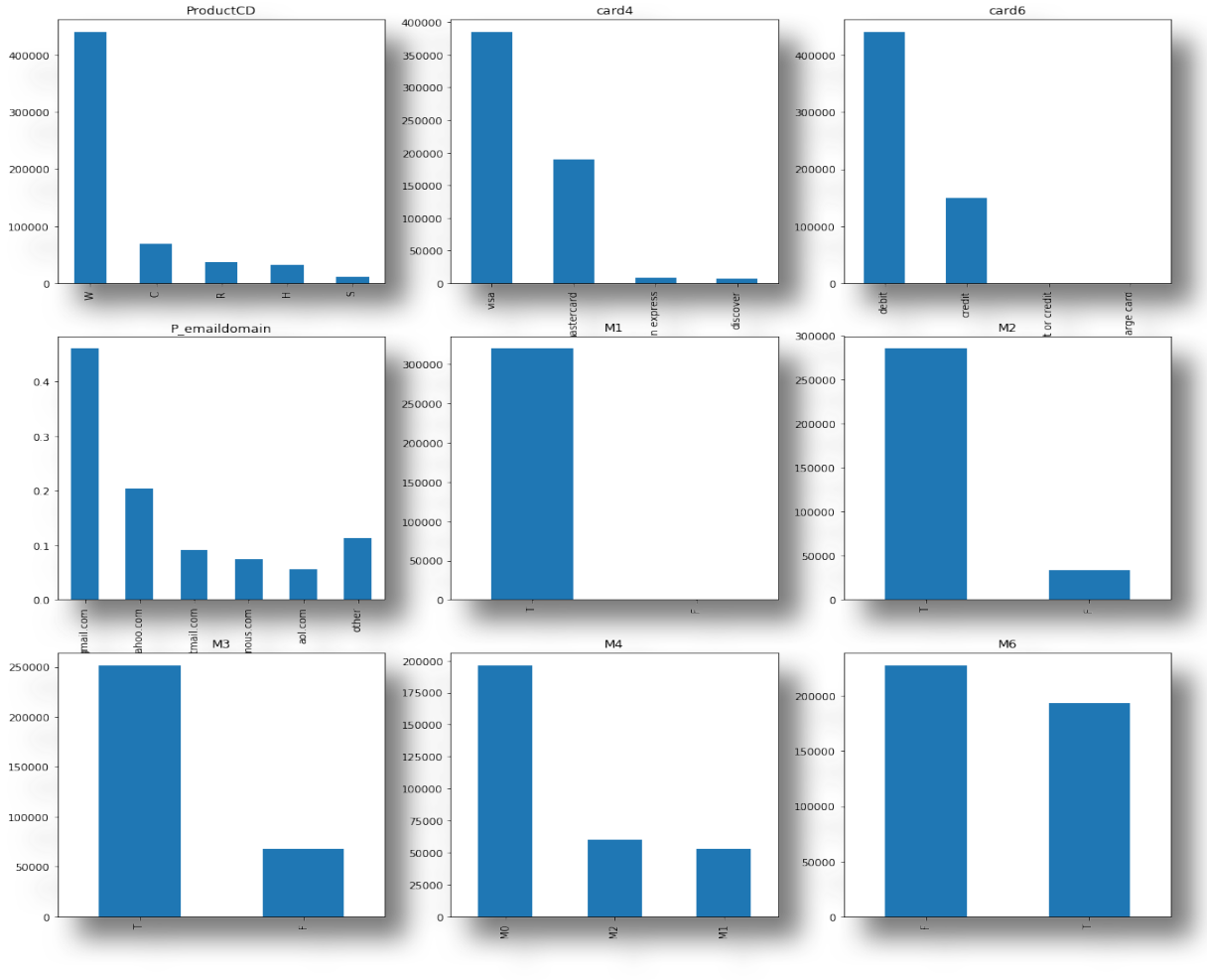
After running Decision Tree with all the features of the DataSet. Correlation matrix was calculated and only columns which had
Correlation values between 0 and 1 were used and rest were discarded as a first feature engineering step. Later by using RFECV algorithm our approach turned out to be valid.

3. *Data pre-processing*

The following observations can be made by basic EDA:

- Categorical columns with no missing values: ProductCD
- Categorical columns with few missing values: card4, card6
- Categorical columns with many missing values: P_emaildomain, M6
- Categorical columns with huge number of missing values: M1,M2,M3,M4

Figure 7



The following observations can be made from the figure:

- It is clearly seen that 'W' represents the ProductCD in over 400,000 transactions.
- It is observed that card4 type represents ‘visa’ in 350k+ transactions and ‘mastercard’ in 200k+ transactions. Other representations are rare.
- card6 type represents 'debit' in approximately. 430k transactions and 'credit' in approximately 150k transactions. Other representations are extremely rare.
- P_emaildomain type represents 'google.com' in more than 40% of the transactions and 'yahoo.com' in more than 20% of the transactions. Some values are relatively less while most values are rare.
- For M2 and M3, the 'T' category is used for most transactions.
- For M1, the 'F' category is rarely used.
- When we look at M4, 'M0' turns out to be the most occurring value.
- In M6, an equal occurrence of 'T' and 'F' is observed.‘F’ is found more suitable to fill the missing values

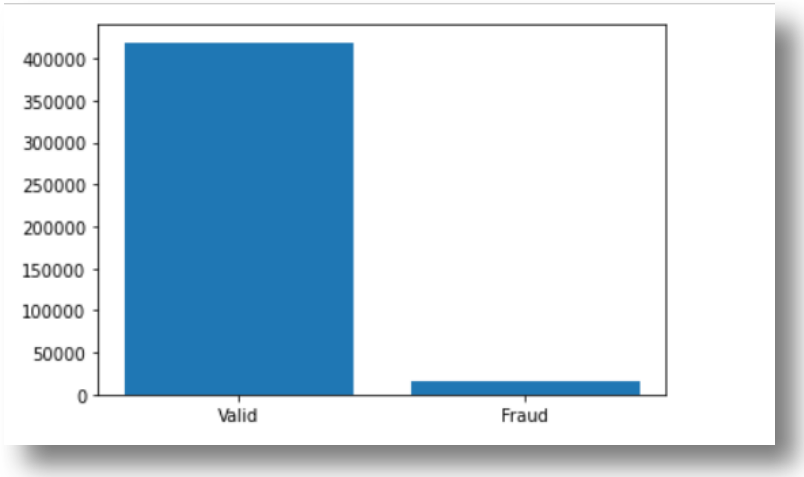
Using the insights, it is safe to fill the missing values with the mode of the object columns.

* Above insights hold true for test data too.

3.1 Solving Class Imbalance using Simulated Minority Over-sampling :

The following is the distribution between the 'Fraud' and 'Valid' transactions.

Figure 8



As can be observed, there is a significant class imbalance present, it is critical to solve this issue since it hinders in learning the new characteristics for the model. Moreover, it is imperative to solve the issue of imbalance especially for a 2-class problem.

Since it hinders in learning the new characteristics for the model. Moreover, it is imperative to solve the issue of imbalance especially for a 2-class problem.
There are two approaches which we considered

- Minority Over Sampling
- Majority Under Sampling

Majority under sampling was rejected since there could be a possibility of losing important information.
Hence, we leveraged Synthetic Minority Over Sampling (SMOTE). The details of the algorithm are present in the appendix.
The class data after the SMOTE is as follows:
after the SMOTE is as follows:

Figure 9

4. Feature Engineering:

Since the given data had over 400 features, feature engineering became one of the core aspects of this project.

Feature Selection:
Leveraging the correlation matrix combined with the graphs created in the Exploratory Data Analysis, we created a list of the most relevant features needed for this dataset.

As part of this process, we had also leveraged sklearn's RFECV [1] for a recursive feature elimination to get the most optimal set of features.

Feature Selection	***Parameters***
RFECV	BernoulliNB(), step=15, scoring='roc_auc', cv=5, verbose=1, n_jobs=3

There was an increase in the Kaggle score observed upon feature selection.

5. Machine Learning Models:

The models which were implemented are as follows:

1. Decision Trees:

Decision Trees are one of the predictive modelling approaches used in statistics, data mining and machine learning.[2]

This approach leverages classifying discrete input vectors to predict the outcome of the target variable.

The tree is built by splitting the input vectors into root and subsets which are subsequently divided further until all leaf nodes are reached.

Table 1

Scenario	***Parameters***	***Kaggle Score***
One hot encoding, Columns with 90 percent or more null values removed, Classes are imbalanced,	random_state=0, criterion='entropy', max_depth=8, splitter='best', min_samples_split=30	0.694
One hot encoding, Columns with 90 percent or more null values removed, Classes are balanced,	random_state=0 criterion='entropy', max_depth=8, splitter='best', min_samples_split=30	0.70
Label encoding, Columns with 90 percent or more null values removed, Classes are imbalanced	random_state=0, criterion='entropy', max_depth=30, splitter='best', min_samples_split=30	0.72
Label encoding, Columns with 50 percent or more null values removed, Classes are balanced	random_state=0, criterion='entropy',max_depth=30,splitter='best', min_samples_split=30	

The finals score of the decision Tree model is:

2. *Naïve Bias:*

Bernoulli Naïve Bias implements the naïve Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions. [3][4][5]

The algorithm can learn the posterior probability from the test data and follows the principle of MAP, thus allowing for a decent predictive model, for the transactions being checked for their validity.
The decision rule for the algorithm is as follows:

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

Note: From the equation above this algorithm penalizes heavily for a non-occurrence of a feature when transactions are being checked to see if they are valid or not.
The implemented model had the following accuracies based on the scenarios covered:

Table 2

Scenario	***Kaggle Score***
With class imbalance	0.50
Without Class Imbalance, no parameter tuning	0.63

2.1 Balanced Classed & Grid Search for Hyper Parameter Tuning

2.1.1 Grid Search Parameters:

In order to fine tune the hyper parameters, we utilized Grid Search and the details are as follows:
Table 4

Parameter	***Value(s)***
alpha	[0.001,0.01,0.1,1]
Fit_prior	[True]

Table 3

Scenario	***Kaggle Score***
Grid Search and Feature Selection	*0.75*

In the end this model had the Kaggle score of: 75

3. *KNN:*

A nature thought process when facing this Fraudulent problem is that: the record itself might contain some meaningful information to classify Fraudulent transactions. The first attempt is to see after our original feature selection, how well KNN is performing.

Hyperparameters	**Kaggle Score**
N_neighbors=3, metric=”minkowski” with p=2	0.500000

The result was disappointing, same as flipping a coin to decide. It is fair guess to think I might have chosen the wrong number of k. My original approach is to use grid search for performing 30 different number of k, in range(1, 31). However, the GridSearchCV from sklearn.model_selection has been running for more than 24 hours by the time this report is generated. Therefore, I give up this approach, instead manual choose 2 other k parameters for comparison.

Hyperparameters	**Kaggle Score**
N_neighbors=5, metric=”minkowski” with p=2	0.500000
N_neighbors=7, metric=”minkowski” with p=2	0.500000

Interestingly enough, the score has no change. I understand ideally I would need to test more k values then only 2 for a fair comparison. It did occur to me that after feature selection, we still have more than 200

features. Which would lead KNN to perform poorly due to "curse of dimensionality". The next question to answer is how can I reduce number of features to a reasonable amount and how.

In order to achieve this, I used sklearn.feature_selection.SelectKBest with following features

- Score_func = "f_classif"
- K = 20

I chose "f_classif" since it uses ANOVA F-value between label/feature for classification tasks. I chose K = 20 because it was mentioned during the lecture that KNN normally perform well when K is less or equal to 20. The new result is as follow. We did see some improvement. But not as ideal as other models.

Hyperparameters	**Kaggle Score**
N_neighbors=5, metric="minkowski" with p=2	0.672110

4. SVM

Support-vector machines (SVMs, additionally Support Vector Networks) are supervised learning models that are specifically required to analyse data used for regression analysis and classification.
The SVM algorithm is a widely accepted ML tool that provides a solution for both classification and regression problems. The training of SVM is usually done using a dual objective optimization method.
In this case, it uses a matrix called the Gram or Kernel Matrix that accommodates the similarities between all the examples. Therefore, for every N examples, this matrix is expected to contain N^2 elements [6].

The parameters that need to be tuned are Decision_function_shape, C, kernel, random_state, and Gamma. Decision_function_shape determines the form of the hyperplane. The use of 'ovo' gives 11 hyperplanes and the use of 'ovr' gives 55 hyperplanes.C is the regularization parameter on the optimization process. The larger the C value, the smaller strength of regularization, which implies SVM will accept smaller margins which classify the training set more correctly. Gamma determines the radius to select supporting vectors. With a larger gamma value, less supporting vectors are selected. However, it's a well-known fact that SVM algorithms are slow. This is because they are not incremental. The entire dataset is required to be in RAM at once.

Table 5

Hyperparameters	**Kaggle Score**
'kernel': 'rgb', 'gamma': 0.001	N/A
'kernel': 'rgb', 'random_state': 1, 'probability': True, 'gamma': 0.01	N/A
'kernel': 'linear', 'random_state': 5, 'probability': True, 'gamma': 1000	N/A
'kernel': 'linear', 'random_state': 1, 'probability': True, 'gamma': 100, 'random_state': 108, 'decision_function_shape': 'ovo'	N/A

5. Random Forest

Random Forest is another recognized supervised learning algorithm. As the name suggests, it works by building a 'forest'. This forest that it builds is an ensemble of decision trees.

The general idea behind the 'bagging method' that is used to train Random Forest models is that there is an increase in the overall result when a combination of learning models is used.
Random Forest classifier solves a major machine learning problem called overfitting by simply having enough trees in the forest so that classifier doesn't overfit the model [7].
One of the hyperparameters that are used here is max_features, which represents the maximum number of features random forest considers splitting a node. Another important hyperparameter used is called min_sample_leaf. This determines the minimum number of leaves required to split an internal node.
The random_state hyperparameter makes the model's output replicable. This ensures that with the same hyperparameters and the same training data along with a definite value of random_state, the model always produces the same results.

However, it’s a well-known fact that when the number of trees grows, the algorithm tends to become too slow and ineffective for real-time predictions.

Table 6

Hyperparameters	**Kaggle Score**
'n_estimators': 100	0.8512
'n_estimators': 500, 'random_state': 10,'max_depth': 20	0.8150
'n_estimators': 1000,'random_state': 200, 'bootstrap': False, 'max_depth': 5	0.86370
'n_estimators': 1000, 'random_state': 121, 'min_samples_split': 2, 'bootstrap': False, 'max_depth': 5	**0.875206**

6. *Light GBM*

Light GBM (Light Gradient Boosting Machine) is a fast, distributed, high-yielding gradient boosting framework based on a decision tree algorithm. It is used for ranking, classification, and several other machine learning tasks.

One advantage that it has over other well-known boosting algorithms is that Light GBM performs a leaf wise tree split (while others perform a depth-wise split) which reduces loss even when the same leaf grows in it. Accuracy is improved. Also, as suggested by the term 'light', it is speedy [8].

There are several hyperparameters used for tuning the Light GBM model.

- num_leaves (default = 31; type = int) represents the number of leaves in one tree.
- max_depth is the hyperparameter used to specify the maximum depth to which a tree can grow.

It also used to deal with the overfitting problem. n_estimators hyperparameter is used to specify the number of trees the algorithm builds before taking the maximum voting or the averages of predictions.

Table 7

Hyperparameters	**Kaggle Score**
'objective': 'binary', 'n_estimators': 300, 'learning_rate': 0.1, 'subsample': 0.8	0.8427
'objective': 'binary', 'n_estimators': 200, 'learning_rate': 0.1	0.8306
'objective': 'binary', 'n_estimators': 500, 'learning_rate': 0.1	0.8666
'objective': 'binary', 'n_estimators': 500, 'learning_rate': 0.1, 'num_leaves': 50, 'max_depth': 7, 'subsample': 0.9, 'colsample_bytree': 0.9	0.8911
'objective': 'binary', 'n_estimators': 600, 'learning_rate': 0.1, 'num_leaves': 50, 'max_depth': 7, 'subsample': 0.9, 'colsample_bytree': 0.9	0.90109
'objective': 'binary', 'n_estimators': 700, 'learning_rate': 0.1, 'num_leaves': 50, 'max_depth': 7, 'subsample': 0.9, 'colsample_bytree': 0.9, 'random_state': 108	**0.920670**

7. *Integrated Stacked Model*

When using standard models as sub-models, it may be desirable to use a standard model as a meta-learner. Specifically, the sub-networks can be embedded in a larger multi-headed network that then learns how to best combine the predictions from each input sub-model. It allows the stacking ensemble to be treated as a single large model[9].

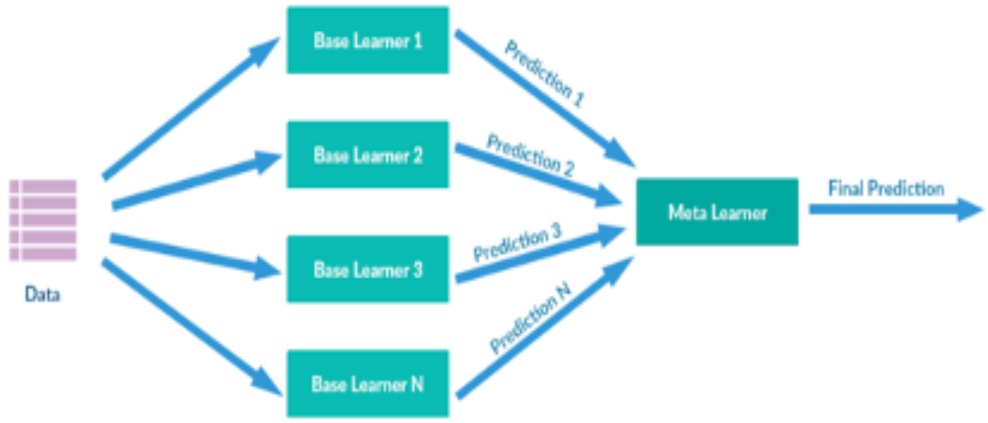


Image from http://supunsetunga.blogspot.com/

For the scope of this project we stacked all the models namely Decision Tree, Random Forest, Bernoulli Naive Bayes, KNN and Light GBM. Now, with the help of soft voting picked out the maximum value using 'argmax' function.
The following table clearly shows that there no increase in the Kaggle score instead we see a decline hence discarding this integrated stacked model.

DT + KNN + LGBM + RF + BNB	**0.775699**

6. Conclusion

The results of the various models are as shown below:

Model	***Parameter***	***Kaggle Score***
DT	random_state=0, criterion='entropy', max_depth=30,splitter='best', min_samples_split=30	0.70
NB	Alpha=0.01,prior_class = True	0.75
KNN		0.67
RF	'n_estimators': 1000, 'random_state': 121, 'min_samples_split': 2, 'bootstrap': False, 'max_depth':5	0.87
LGBM	***'objective': 'binary', 'n_estimators': 700, 'learning_rate': 0.1, 'num_leaves': 50, 'max_depth': 7, 'subsample': 0.9, 'colsample_bytree': 0.9, 'random_state': 108***	***0.92***
Integrated Stacked Model	DT + NB + KNN + RF + LGBM	0.77

Thus, we chose the model LGBM as the final model, for predicting the validity of the transaction and got the score as 0.92.

7. Contribution

Name	**Contribution**
Mohit Khanna	EDA, Data pre-processing, RF, SVM, LGBM, Integrated Stacked Model
Usama Sadiq	EDA, Data pre-processing, Feature Engineering, RF, Decision Trees, Naïve Bias, KNN, SVM, Random Forest, Light GBM, Integrated Stacked Model
Uttkarsh Sharma	EDA, Data pre-processing, Feature Engineering, RF, Decision Trees, Naïve Bias, KNN, SVM, Random Forest, Light GBM, Integrated Stacked Model
Sibo Zhang	EDA, Data pre-processing, Feature Engineering, RF, Decision Trees, Naïve Bias, KNN, SVM, Random Forest, Light GBM, Integrated Stacked Model

8. References

1. Guyon, I., Weston, J., Barnhill, S., & Vapnik, V., "Gene selection for cancer classification using support vector machines", Mach. Learn., 46(1-3), 389–422, 2002.
2. SK learn documentation :: https://scikit-learn.org/stable/modules/tree.html#decision-trees

3. C.D. Manning, P. Raghavan and H. Schütze (2008). Introduction to Information Retrieval. Cambridge University Press, pp. 234-265.
4. A. McCallum and K. Nigam (1998). A comparison of event models for Naive Bayes text classification. Proc. AAAI/ICML-98 Workshop on Learning for Text Categorization, pp. 41-48.
5. V. Metsis, I. Androutsopoulos and G. Paliouras (2006). Spam filtering with Naive Bayes – Which Naive Bayes? 3rd Conf. on Email and Anti-Spam (CEAS).
6. Ben-Hur, Asa; Horn, David; Siegelmann, Hava; Vapnik, Vladimir N. ""Support vector clustering" (2001);". *Journal of Machine Learning Research*. **2**: 125–137.
7. Ho, Tin Kam (1995). *Random Decision Forests* (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Archived from the original (PDF) on 17 April 2016. Retrieved 5 June 2016.
8. Friedman, J. H. (February 1999). "Greedy Function Approximation: A Gradient Boosting Machine" (PDF)
9. Wolpert (1992). "Stacked Generalization". *Neural Networks*. **5**(2): 241–259.
10. Jason Brownlee. "Feature Selection for Machine Learning in python"