



It's a Fraud

10/12/22

Team Name - Shark

Team members

MT2022064	Niraj Lande	Niraj.Lande@iiitb.ac.in
MT2022107	Saurabh Dixit	Saurabh.Dixit@iiitb.ac.in

Overview

The aim of this competition is to predict by probability whether given payment transaction is fraudulent. In order to achieve this goal, different machine learning models will be deployed to help us understand and analyze the large dataset given. The competition is rated by rank, where rank is decided by the accuracy of the prediction submitted by the participant. The competition can then improve the efficiency of fraud detection for many people and help many businesses to increase their revenue and decrease their economic loss.

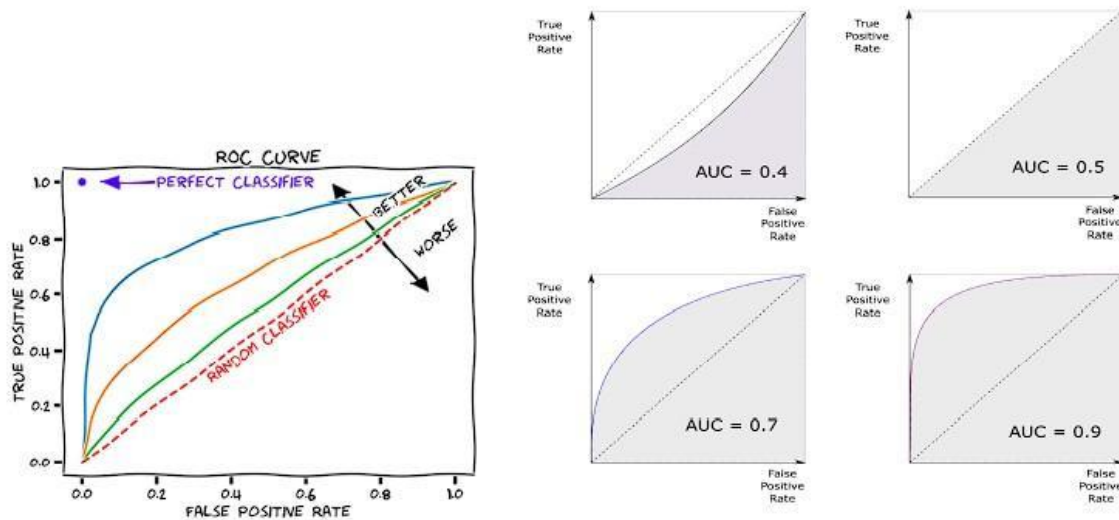
Data Description

- `TransactionDT: timedelta from a given reference datetime (not an actual timestamp)
"TransactionDT "corresponds to the number of seconds in a day.
- TransactionAMT: transaction payment amount in USD
- `ProductCD: product code, the product for each transaction
- card1 - card6: payment card information, such as card type, card category, issue bank, country, etc.
- addr: address
"both addresses are for purchaser
addr1 as billing region
addr2 as billing country"
- dist: distances between (not limited) billing address, mailing address, zip code, IP address, phone area, etc."
- P_ and (R__) emaildomain: purchaser and recipient email domain
" certain transactions don't need a recipient, so R_emaildomain is null."
- C1-C14: counting, such as how many addresses are found to be associated with the payment card, etc.
- D1-D15: timedelta, such as days between previous transactions, etc.
- M1-M9: match, such as names on card and address, etc.
- Vxxx: Vesta engineered rich features, including ranking, counting, and other entity relations.
- "id01 to id11 are numerical features for identity.
Other columns such as network connection information (IP, ISP, Proxy, etc),digital signature (UA/browser/os/version, etc) associated with transactions are also present

Evaluation Matrix

Kaggle submissions are evaluated on the Area Under the ROC curve (AUC) - ROC means receiver operating characteristic curve - it is a graph showing the performance of a classification model at all classification thresholds. AUC - provides an aggregate measure of performance across all possible classification thresholds.

Score less than 0.5 is considered worse and as it approaches 1 we will get the perfect classifier model.



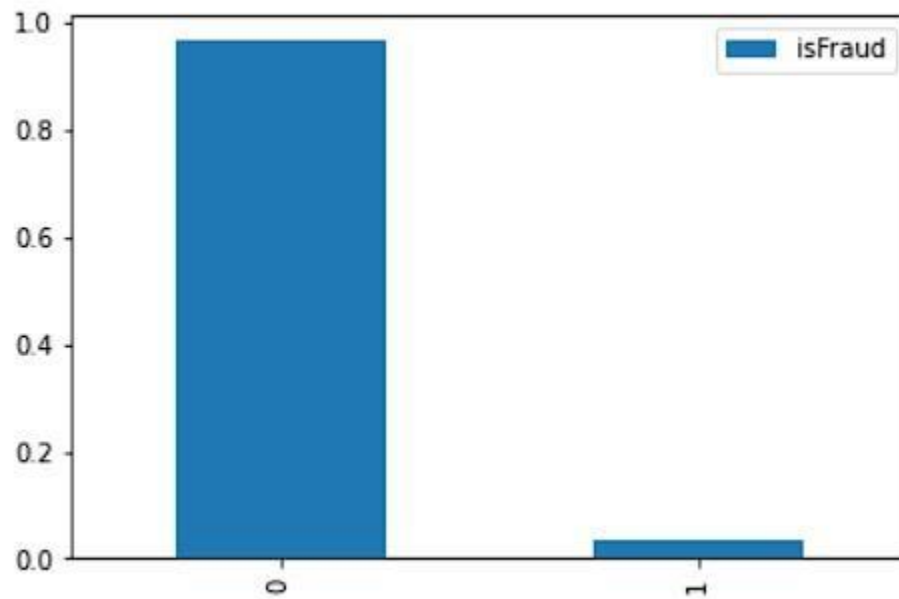
Exploratory Data Analysis

First, the variables have 3 types, the variables with type of data, high number of missing entries and highly correlated variables. And also, the variables have different types of data (numerical and categorical). And it will be explained in the following

I. Data Distribution Pattern

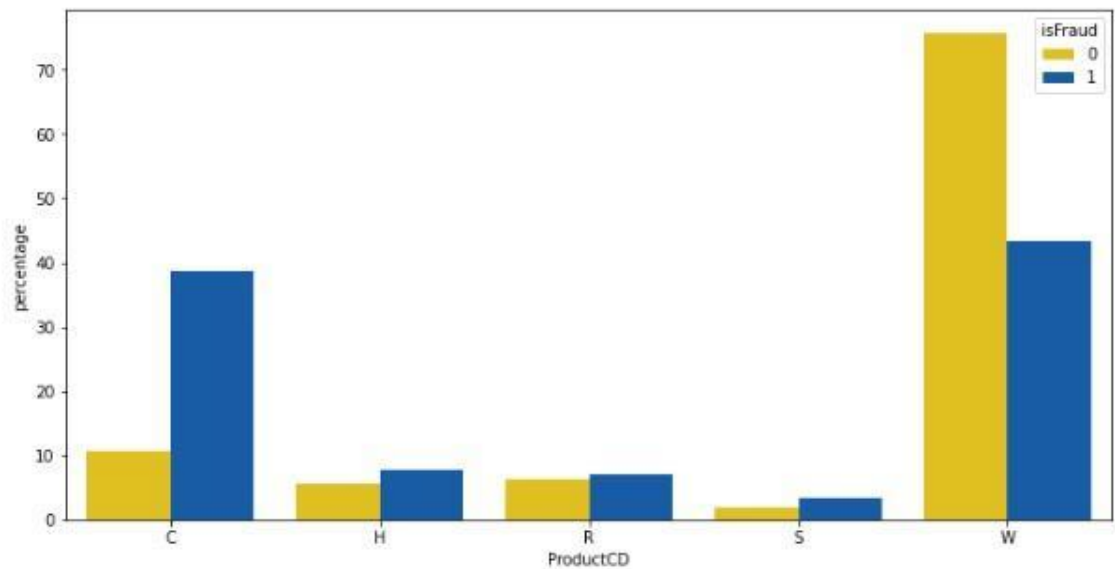
We aim to predict the probability that an online transaction is fraudulent, thus it is important to determine the distribution of fraud labels. As shown in the first map below, the shaded blue rectangle represents the non fraudulent transactions

(96.501%), and the shaded green rectangle represents fraudulent transactions (3.499%). It is obvious that the data is highly imbalanced and most of the transactions are non-fraud. If we use this data frame as the base for our predictive models, we might get a lot of errors and our algorithms will probably overfit



II. Analysis on product_cd

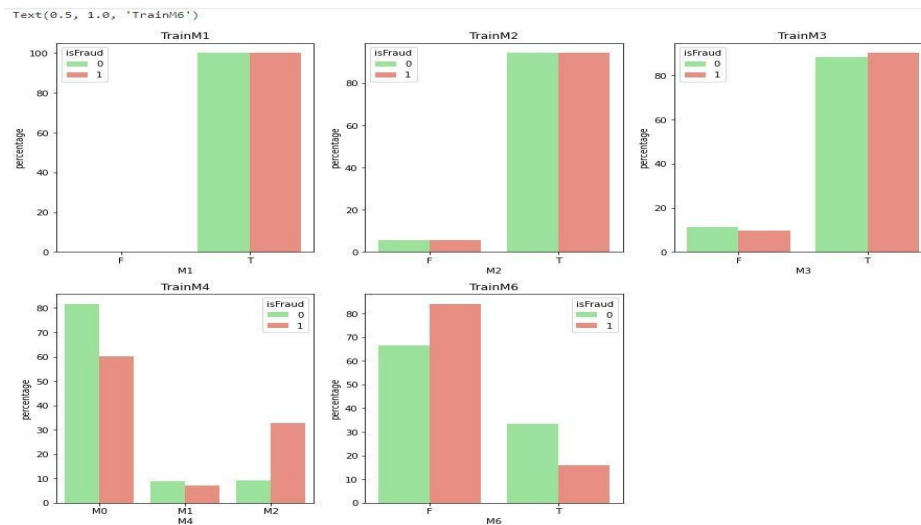
Product_cd gives the product code.



- From the plot it is clear that if ProductCD is "C" then there is a 40% chance that it is fraud.
- For H,R,S it is clear that the rate of fraud is high.

III. Analysis on M1 -M2

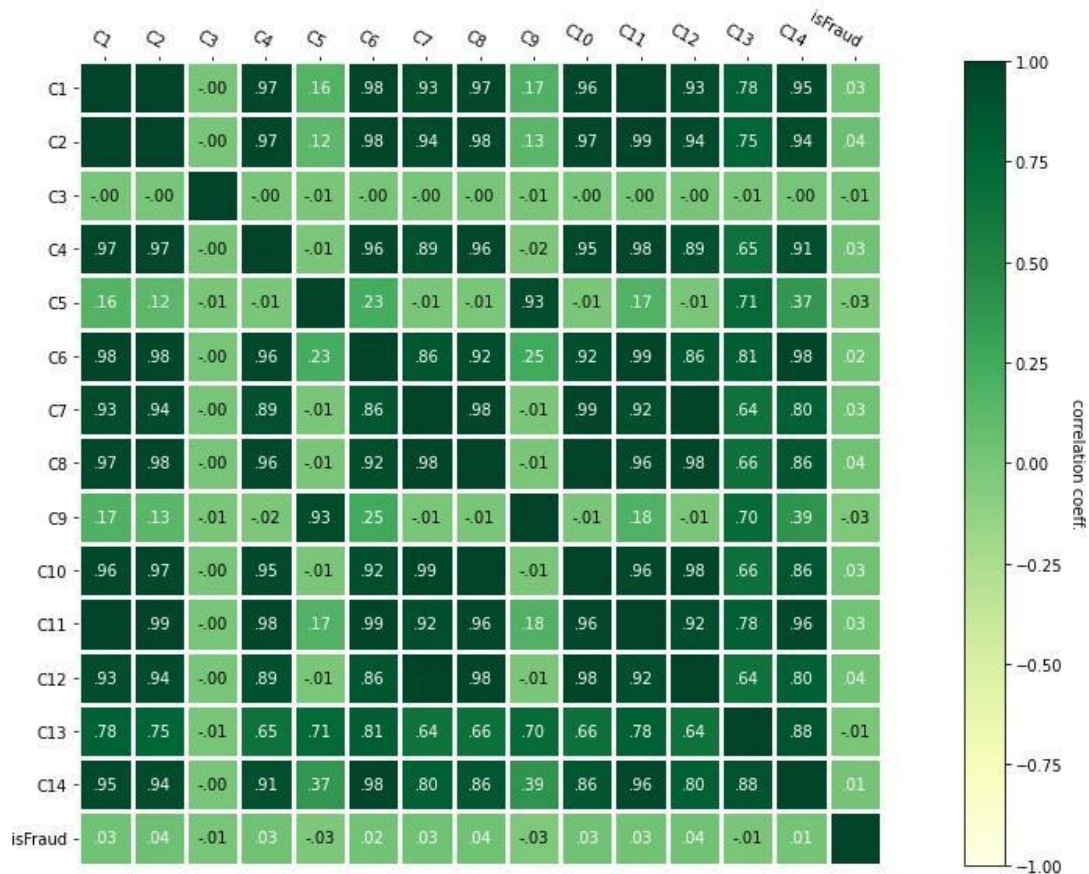
M1-M9: match, such as names on card and address, etc.



From the plot it is clear that M1 and M2 have no contribution towards finding if a transaction is fraud and legit.

IV. Analysis on C1-C14

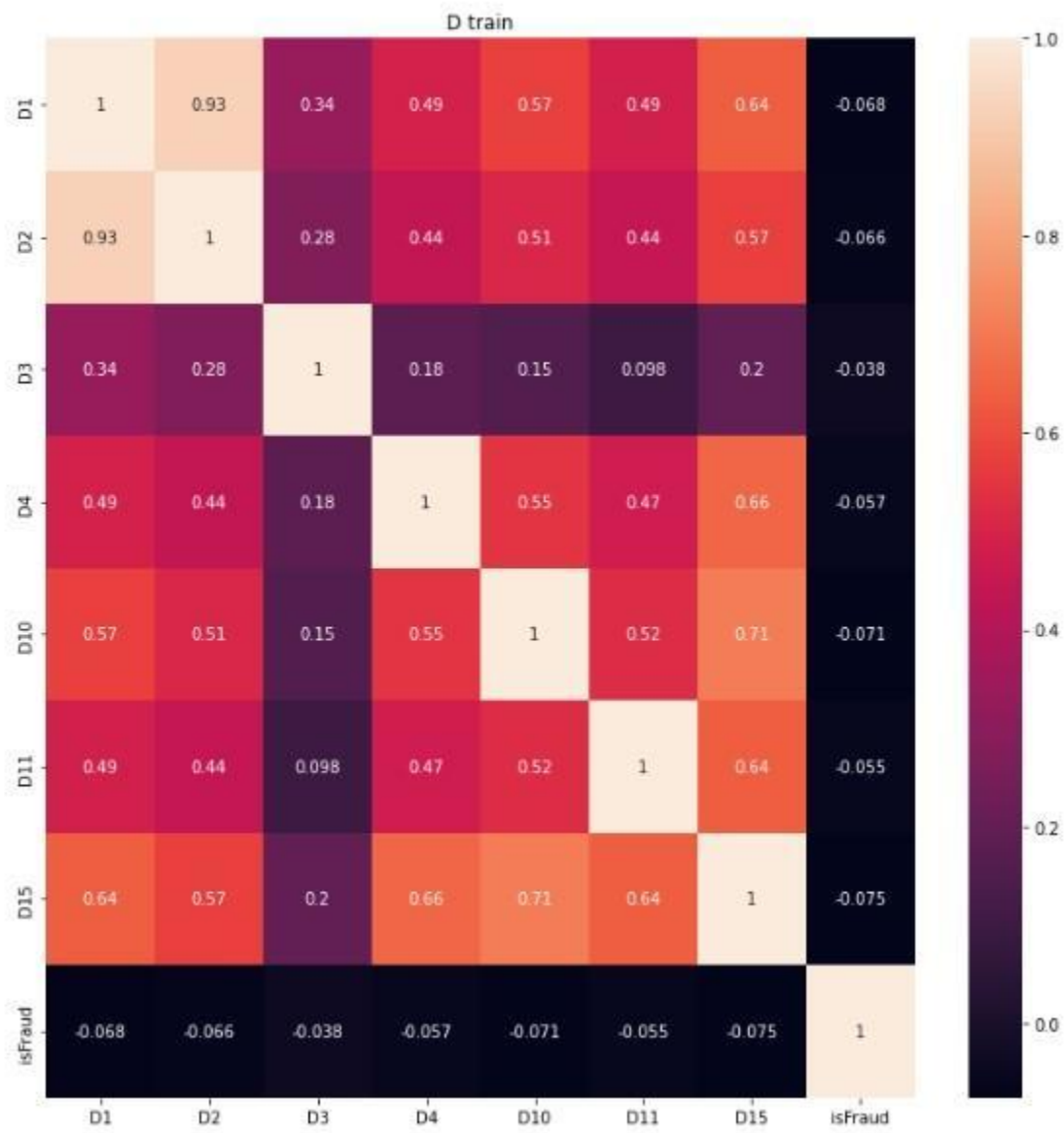
C1-C14: counting, such as how many addresses are found to be associated with the payment card, etc.



- We can see that some of the features are highly correlated with each other.
- For eg - 'C1' and 'C2', 'C1' and 'C6', 'C1' and 'C14', 'C6' and 'C14' etc..
- In the modeling part we will see removing these highly correlated features improves the acc or decreases the acc.

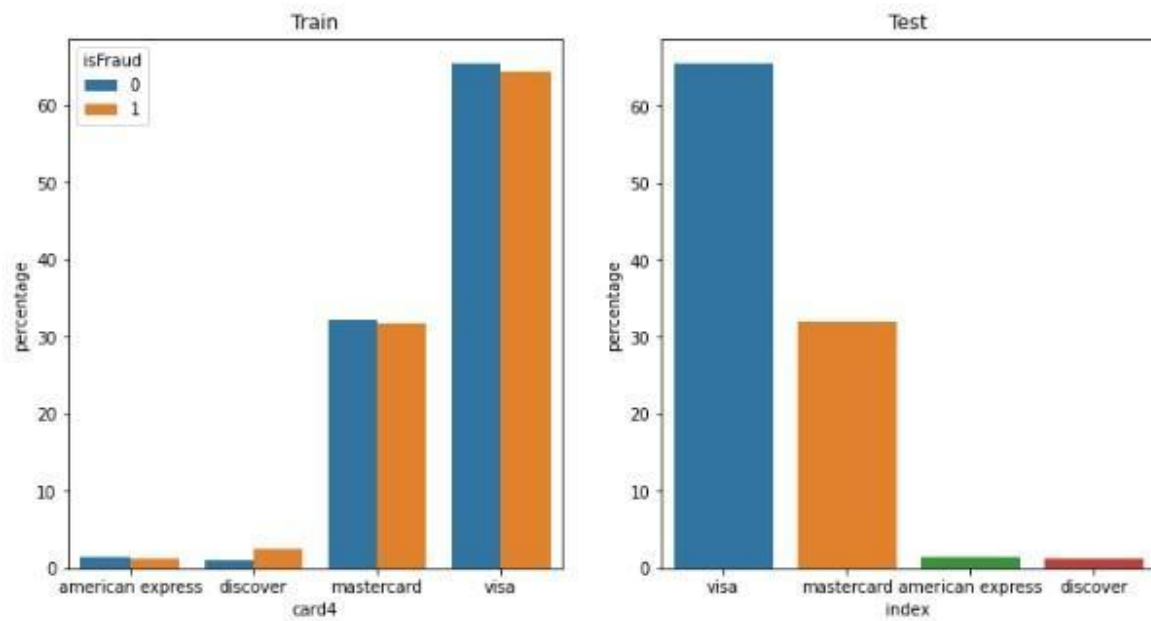
V. Analysis of Dxx

M1-M9: match, such as names on card and address, etc.



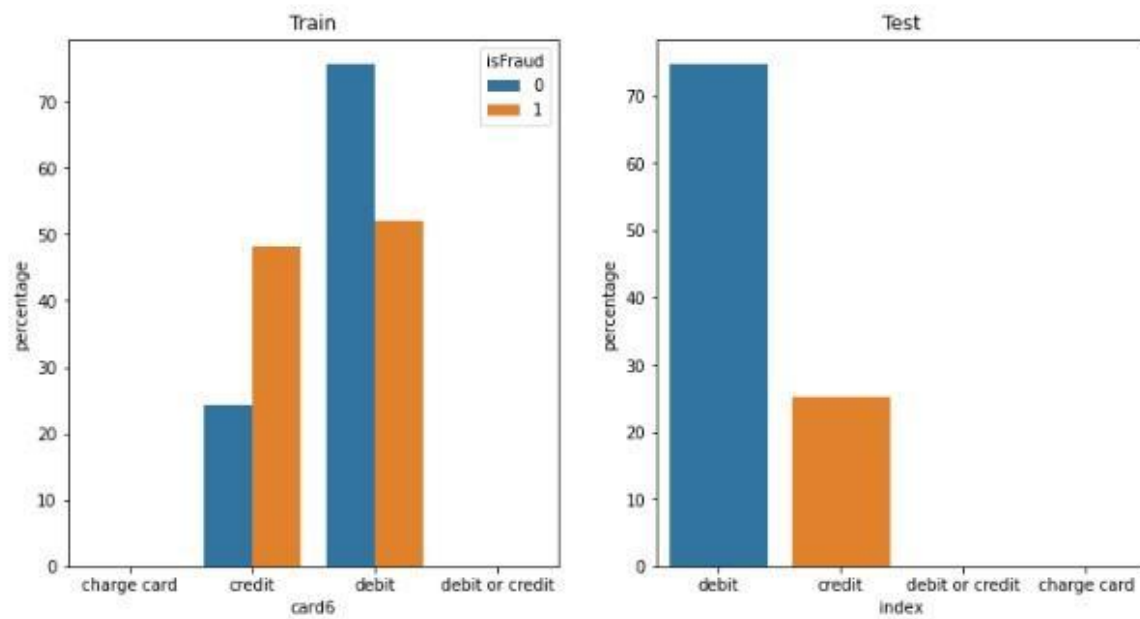
Highly correlated D features are there, like 'D1 and D2', removing one of them can increase the score.

VI. Analysis of Card 4



- Most of the transactions happened through visa card type.
- One observation is that in fraud transactions, use of the discover card is more than the legit transaction.
- Otherwise other cards are relatively of the same frequency.

VII. Analysis on Card 6



- Debit or credit and charge card are almost 0% in the dataset.
- Let's combine them into other card categories.
- Credit card holders tends to have more fraud transaction then debit card holders

Pre-Processing

I. Missing Value

The next important process to be addressed is dealing with missing values. Within certain codes, it is clear that some columns contain missing value. The figure below shows the top 20 features displaying the highest missing value percentage. From the figure, we can observe that most features containing high missing value percentages are found in the identity dataset.

M4	47.67
D2	47.53
V1	47.30
V10	47.30
D11	47.30
V2	47.30
V3	47.30
V4	47.30
V5	47.30
V6	47.30
V7	47.30
V8	47.30
V9	47.30
V11	47.30
M1	45.92
M3	45.92
M2	45.92
D3	44.50
M6	28.63
V36	28.61

- As we can see, the columns are having a high % of missing values, so we made some imputation to the missing data.
- Threshold for column removal = 50%.
- For remaining missing values - We filled numerical variables with median and categorical columns with mean.

II. Outlier Finding

We can see that the max values are very large in comparison to other values so we can conclude that there is a presence of outliers. (no need to check box plots).

mean	7.366857e+06	135.273690	14.254165	15.440282	0.005733	4.179122	5.575586	9.161105	2.894711	5.237857	4.482079	5.319520	10.355934	4.142256
std	4.617568e+06	232.955213	135.955827	157.244297	0.157655	70.003818	25.817411	72.632178	62.702975	96.925438	16.680473	96.946912	95.868153	88.075522
min	8.646900e+04	0.251000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	3.021834e+06	43.744000	1.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000
50%	7.301614e+06	68.950000	1.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	0.000000	1.000000	0.000000
75%	1.123641e+07	125.000000	3.000000	3.000000	0.000000	0.000000	1.000000	2.000000	0.000000	0.000000	2.000000	0.000000	2.000000	0.000000
max	1.581113e+07	*6085.230000	*4685.000000	*5691.000000	*26.000000	*2253.000000	*349.000000	*2253.000000	*2255.000000	*3331.000000	*210.000000	3257.000000	3188.000000	3188.000000

III. Outlier treatment using IQR

- Using the Interquartile range method we shifted the outliers which were above Upper Bound value equal to UpperBound and which were below Lower Bound equal to LowerBound.
- Now the data after shifting is outlier free and we can see it in the describe function or through box plots.

std	1.705119e+05	0.183753	4.617568e+06	73.236752	4901.960786	156.667937	11.300402	41.146151	95.976331
min	2.987002e+06	0.000000	8.646900e+04	0.251000	1001.000000	100.000000	100.000000	100.000000	100.000000
25%	3.134224e+06	0.000000	3.021834e+06	43.744000	6019.000000	215.000000	150.000000	166.000000	205.000000
50%	3.282155e+06	0.000000	7.301614e+06	68.950000	9680.000000	360.000000	150.000000	226.000000	299.000000
75%	3.429574e+06	0.000000	1.123641e+07	125.000000	14184.000000	512.000000	150.000000	226.000000	327.000000
max	3.577539e+06	1.000000	1.581113e+07	*246.884000	*18396.000000	*600.000000	*231.000000	*237.000000	*540.000000

IV. Encoding

Models

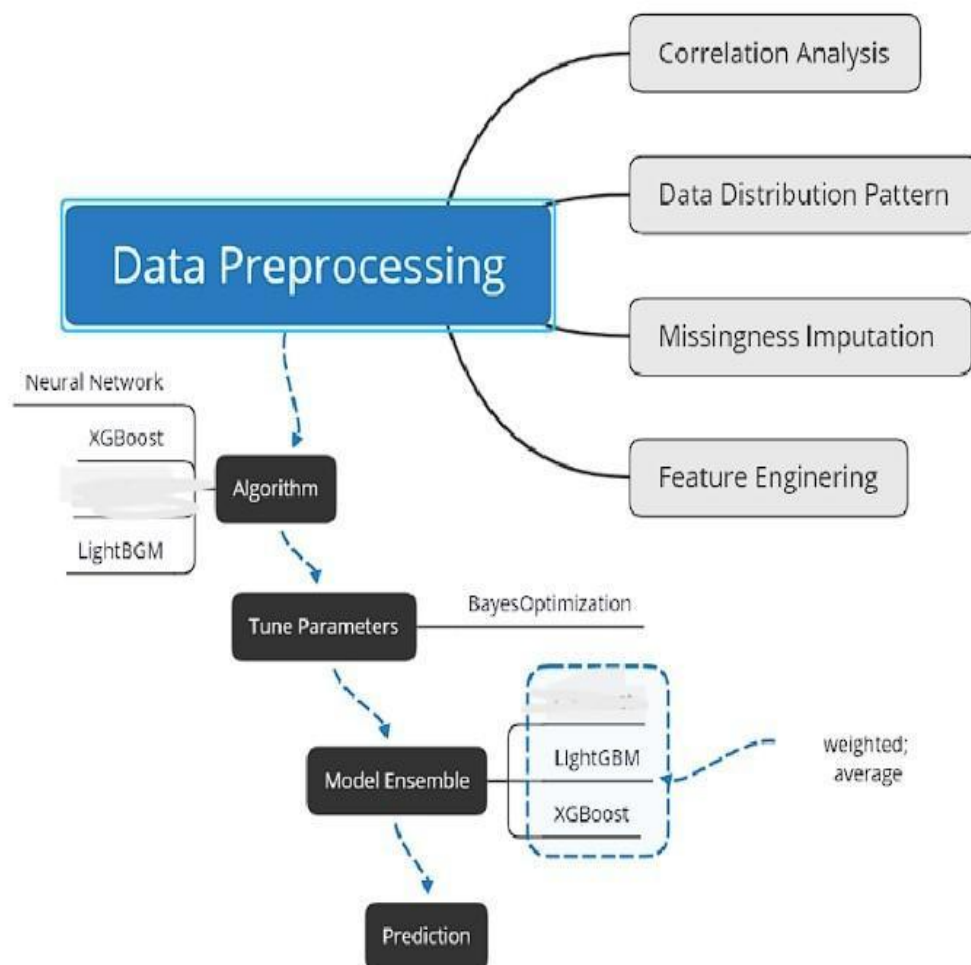


Figure 1. Workflow of the Program

I. KNN

KNN is a machine learning algorithm which is used for both classification (using KNearestClassifier) and Regression (using KNearestRegressor) problems. In KNN algorithm K is the **Hyperparameter**. Choosing the right value of K matters. A machine learning model is said to have high model complexity if the built model is having low Bias and High Variance. Here we Take the Value of K to be 5.

II. Logistic Regression

Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable(or output), y, can take only discrete values for a given set of features(or inputs), X.

Contrary to popular belief, logistic regression is a regression model. The model builds a regression model to predict the probability that a given data entry belongs to the category numbered as "1". Just like Linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid function.

III. Random Forest

Random forest algorithms have three main hyperparameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled. From there, the random forest classifier can be used to solve for regression or classification problems

Key Challenges

- Time Consuming Process.
- Require More Resources.
- More Complex.

IV. Decision Tree

A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

A tree can be “learned” by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions.

V. Naive Bayes

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes’ Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

VI. Ridge ClassifierCV

Ridge classification works by adding a penalty term to the cost function that discourages complexity. The penalty term is typically the sum of the squared coefficients of the features in the model. This forces the coefficients to remain small, which prevents overfitting. The amount of regularization can be controlled by changing the penalty term. A larger penalty results in more regularization and smaller coefficient values. This can be beneficial when there is little training data available. However, if the penalty term is too large, it can result in underfitting.

VII. XG Boost

XGBoost is an implementation of Gradient Boosted decision trees. XGBoost models majorly dominate in many Kaggle Competitions.

In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

Hyperparameter Tuning

```
[22]:
params={
    "learning_rate" : [0.01, 0.03 , 0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ,0.40 ,0.50 ] ,
    "max_depth" : [ 3, 4, 5, 6, 8, 10, 12, 15],
    "min_child_weight" : [ 1, 3, 5, 7 ],
    "gamma" : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
    "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
}

from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
random_search=RandomizedSearchCV(classifier,param_distributions=params,n_iter=5,scoring='roc_auc',n_jobs=-1,cv=5,verbose=3)
classifier=xg.XGBClassifier()
```

```
def timer(start_time=None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.now() - start_time).total_seconds(), 3600)
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin, round(tsec, 2)))
```

+ Code

+ Markdown

```
[31]:
from datetime import datetime
# Here we go
start_time = timer(None) # timing starts from this point for "start_time" variable
random_search.fit(xtrain,ytrain)
timer(start_time) # timing ends here for "start_time" variable
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV 2/5] END colsample_bytree=0.3, gamma=0.2, learning_rate=0.25, max_depth=15, min_child_weight=1; score=1.000 total time=31.0min
[CV 1/5] END colsample_bytree=0.5, gamma=0.4, learning_rate=0.2, max_depth=10, min_child_weight=5; score=0.980 total time=31.8min
[CV 1/5] END colsample_bytree=0.3, gamma=0.1, learning_rate=0.03, max_depth=8, min_child_weight=5; score=0.976 total time=17.4min
[CV 4/5] END colsample_bytree=0.3, gamma=0.1, learning_rate=0.03, max_depth=8, min_child_weight=5; score=0.999 total time=16.9min
```



```
start_time = timer(None) # timing starts from this point for start_time variable
random_search.fit(xtrain,ytrain)
timer(start_time) # timing ends here for "start_time" variable
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV 2/5] END colsample_bytree=0.3, gamma=0.2, learning_rate=0.25, max_depth=15, min_child_weight=1, score=1.000 total time=31.0min
[CV 1/5] END colsample_bytree=0.5, gamma=0.4, learning_rate=0.2, max_depth=10, min_child_weight=5, score=0.980 total time=31.8min
[CV 1/5] END colsample_bytree=0.3, gamma=0.1, learning_rate=0.03, max_depth=8, min_child_weight=5, score=0.976 total time=17.4min
[CV 4/5] END colsample_bytree=0.3, gamma=0.1, learning_rate=0.03, max_depth=8, min_child_weight=5, score=0.999 total time=16.9min
[CV 3/5] END colsample_bytree=0.4, gamma=0.1, learning_rate=0.3, max_depth=4, min_child_weight=3, score=1.000 total time=11.2min
[CV 2/5] END colsample_bytree=0.5, gamma=0.3, learning_rate=0.05, max_depth=10, min_child_weight=1, score=1.000 total time=31.4min
[CV 4/5] END colsample_bytree=0.3, gamma=0.2, learning_rate=0.25, max_depth=15, min_child_weight=1, score=1.000 total time=31.0min
[CV 2/5] END colsample_bytree=0.5, gamma=0.4, learning_rate=0.2, max_depth=10, min_child_weight=5, score=1.000 total time=30.7min
[CV 5/5] END colsample_bytree=0.5, gamma=0.4, learning_rate=0.2, max_depth=10, min_child_weight=5, score=1.000 total time=30.4min
[CV 1/5] END colsample_bytree=0.4, gamma=0.1, learning_rate=0.3, max_depth=4, min_child_weight=3, score=0.976 total time=11.0min
[CV 4/5] END colsample_bytree=0.4, gamma=0.1, learning_rate=0.3, max_depth=4, min_child_weight=3, score=1.000 total time=11.2min
[CV 3/5] END colsample_bytree=0.5, gamma=0.3, learning_rate=0.05, max_depth=10, min_child_weight=1, score=1.000 total time=29.9min
[CV 1/5] END colsample_bytree=0.3, gamma=0.2, learning_rate=0.25, max_depth=15, min_child_weight=1, score=0.981 total time=29.0min
[CV 5/5] END colsample_bytree=0.3, gamma=0.2, learning_rate=0.25, max_depth=15, min_child_weight=1, score=1.000 total time=29.3min
[CV 4/5] END colsample_bytree=0.5, gamma=0.4, learning_rate=0.2, max_depth=10, min_child_weight=5, score=1.000 total time=31.3min
[CV 5/5] END colsample_bytree=0.3, gamma=0.1, learning_rate=0.03, max_depth=8, min_child_weight=5, score=0.999 total time=16.6min
[CV 5/5] END colsample_bytree=0.4, gamma=0.1, learning_rate=0.3, max_depth=4, min_child_weight=3, score=1.000 total time=11.4min
[CV 4/5] END colsample_bytree=0.5, gamma=0.3, learning_rate=0.05, max_depth=10, min_child_weight=1, score=1.000 total time=29.8min
[CV 3/5] END colsample_bytree=0.3, gamma=0.2, learning_rate=0.25, max_depth=15, min_child_weight=1, score=1.000 total time=31.4min
[CV 3/5] END colsample_bytree=0.5, gamma=0.4, learning_rate=0.2, max_depth=10, min_child_weight=5, score=1.000 total time=31.7min
[CV 2/5] END colsample_bytree=0.3, gamma=0.1, learning_rate=0.03, max_depth=8, min_child_weight=5, score=0.999 total time=16.9min
[CV 3/5] END colsample_bytree=0.3, gamma=0.1, learning_rate=0.03, max_depth=8, min_child_weight=5, score=0.999 total time=16.5min
[CV 2/5] END colsample_bytree=0.4, gamma=0.1, learning_rate=0.3, max_depth=4, min_child_weight=3, score=1.000 total time=10.8min
[CV 1/5] END colsample_bytree=0.5, gamma=0.3, learning_rate=0.05, max_depth=10, min_child_weight=1, score=0.979 total time=30.0min
[CV 5/5] END colsample_bytree=0.5, gamma=0.3, learning_rate=0.05, max_depth=10, min_child_weight=1, score=1.000 total time=22.5min

Time taken: 2 hours 48 minutes and 38.42 seconds.
```

```
random_search.best_estimator_
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.3,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0.2, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.25, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=15, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)
```

[+ Code](#)
[+ Markdown](#)

VIII. Neural Network(MLP Classifier)

A multilayer perceptron (MLP) is a feedforward artificial neural network that generates a set of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input and output layers. MLP uses backpropagation for training the network. MLP is a deep learning method.

When one thinks of Deep Learning, the well-known libraries such as [Keras](#), [PyTorch](#) or [TensorFlow](#) immediately come to mind. Most of us may not know that the very popular machine learning library [Scikit-Learn](#) is also capable of basic deep learning modeling.

Hyperparameter Tuning

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.neural_network import MLPClassifier
def hyperparameter_tune(clf, parameters, iterations, X, y):
    randomSearch = RandomizedSearchCV(clf, param_distributions=parameters, scoring='roc_auc', n_jobs=-1, n_iter=iterations, cv=2)
    randomSearch.fit(X,y)
    params = randomSearch.best_params_
    score = randomSearch.best_score_
    return params, score


parameters = {
    'solver': ['sgd', 'adam', 'lbfgs'],
    'activation': ['relu', 'tanh']
}
clf = MLPClassifier(batch_size=256, verbose=True, early_stopping=True)
parameters_after_tuning, score_after_tuning = hyperparameter_tune(clf, parameters, 20, X_train, y_train.values.ravel());
print(parameters_after_tuning)
```

Summary of all Models

Mode Tuned	Hyper Parameters	Value	Model_score(without pca)	Model_score(with pca)
KNN	weights	default	0.66738	0.58347
	n_neighbours	5		
Neural Network(MLP)	solver	adam	0.76154	-
	activation	tanh		
	alpha	1.00E-02		
	hidden_layer_sizes	(15,12)		
	random_state	1		
Random Forest	n_estimators	1000	0.76192	-
	random_state	0	0.50066(scaled)	
Decision tree	max_depth	None	0.7376	-
	min_samples_leaf	2		
	criterion	gini		
XGBoost	learning rate	0.25	0.85555	-
	max_depth	15		
	n_estimators	10000		
	gamma	0.2		
	min_child_weight	1		
	col_sample_bytree	0.3		
Gradient Boosting Classifier	n_estimators	100	0.74344	-
	learning_rate	1		
	max_depth	1		
	random_state	0		
Naive Bayes	default	-	0.58124	0.48625
RidgeClassifierCV	default	-	0.48014	-
Logistic Regression	default	-	0.52029	0.47866
LGBMClassifier(Non-probabilistic)	reg_alpha	0.01	0.81583	-
	num_leaves	100		
	min_child_samples	12		
	max_depth	-1		
	learning_rate	0.1		

Results:

After cleaning the data with multiple preprocessing techniques and training a variety of models, we reached the best one that gave auc score of 0.8555 on unseen data.

	submitXGB10.csv	0.85555	0.85435	<input type="checkbox"/>
---	-----------------	---------	---------	--------------------------