ITS A FRAUD - Team Fraud

Hardik Khandelwal (IMT2020509)

Aakash Khot (IMT2020512)

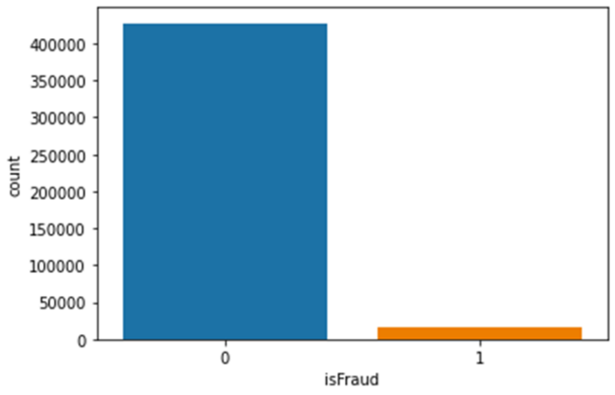
Problem Statement : Given details about a transaction determine whether the transaction is a Fraud or Not.

We are given a training dataset train.csv which will be used for training of the ML model.
Training data contains over 400 columns and 4 lakh rows. 434 columns and 4,42,905 rows precisely.
We are also given a testing dataset test.csv which will be used for testing our model.

Preprocessing -

‘isFraud’ Column -

As we can see from diagram, only 3.5% of the total transactions in the training data are fraud. From this we can say that the data set is **highly imbalanced.**



Cardinality Reduction -

If many categories have very similar target distribution, merge them into a single category. Results in faster computation/performance. We applied this technique to the following three features -

- P_emaildomain : 18 domains have 0 fraud transactions so we group them together as a single domain - 'trusted.com'
- R_emaildomain : 30 domains have 0 fraud transactions so we group them together as a single domain - 'trusted.com'
- DeviceInfo : Number of devices with 0% and 100% fraud rate respectively: 1250 , 52. Grouped them together under categories - 'trusted' and 'fraudulent' respectively.

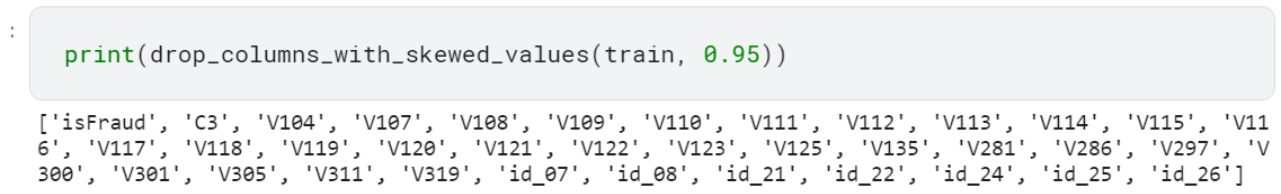
Features	Before Reduction	After Reduction
P_emaildomain	60	43
R_emaildomain	61	32
DeviceInfo	1658	358

Analyzing Columns -

Number of columns containing NAN values : 414

We obtain the list of columns which do not hold much importance for the following reasons :

- A majority of values (>95%) are null.
- Column values are highly skewed/imbalanced (>95% of the values are very close to a particular value)



```
print(drop_columns_with_skewed_values(train, 0.95))
```

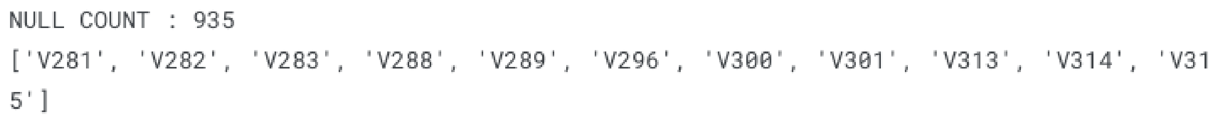
```
['isFraud', 'C3', 'V104', 'V107', 'V108', 'V109', 'V110', 'V111', 'V112', 'V113', 'V114', 'V115', 'V11
6', 'V117', 'V118', 'V119', 'V120', 'V121', 'V122', 'V123', 'V125', 'V135', 'V281', 'V286', 'V297', 'V
300', 'V301', 'V305', 'V311', 'V319', 'id_07', 'id_08', 'id_21', 'id_22', 'id_24', 'id_25', 'id_26']
```

Correlation (Heatmap)

The following has been done for V, C and D columns separately.

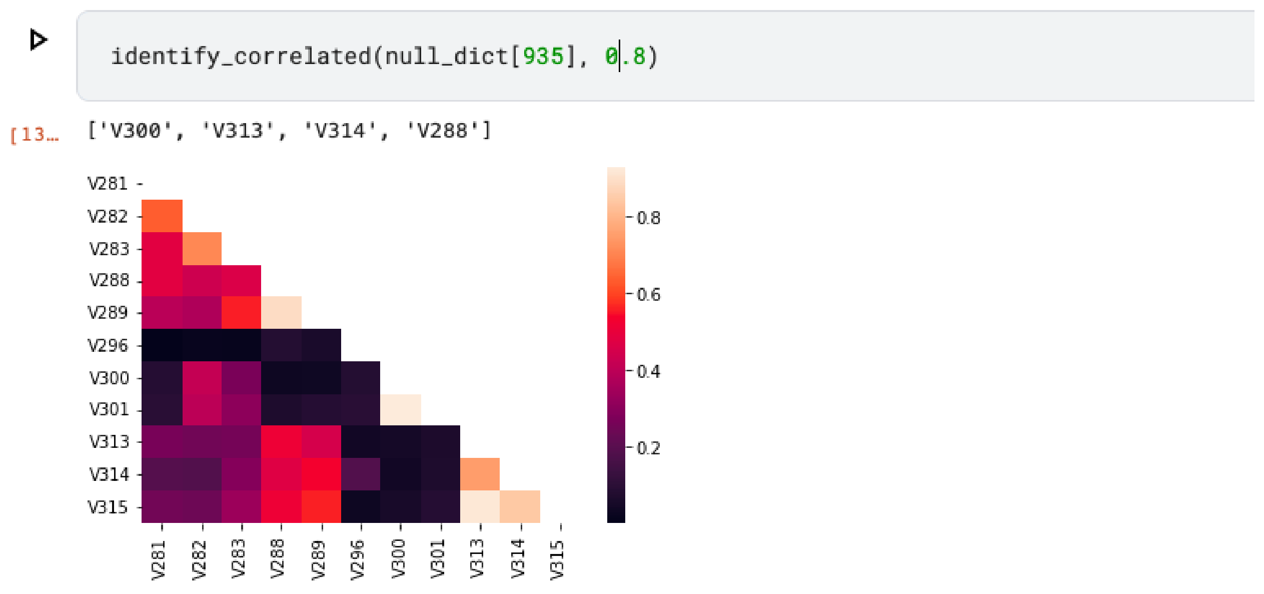
1. Group columns based on number of missing values.
2. Within each group, we further divide the features based on their correlation (> 80%) and keep a single feature from each subgroup.
3. From the heatmap, mask the upper triangular half and make groups of correlated columns programmatically.

For instance, here is the snippet of V columns having 935 missing values.



```
NULL COUNT : 935
['V281', 'V282', 'V283', 'V288', 'V289', 'V296', 'V300', 'V301', 'V313', 'V314', 'V31
5']
```

We further divide it into subgroups of features with high correlation and pick one feature from each group using the masked heatmap.



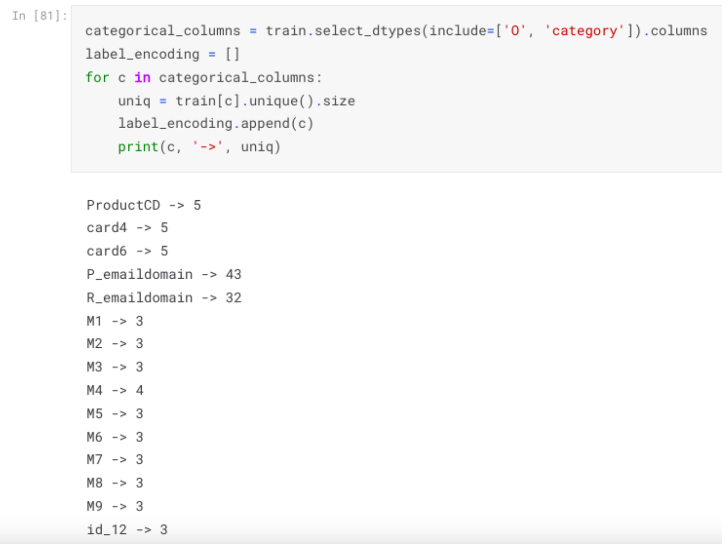
```
identify_correlated(null_dict[935], 0.8)
```

```
[13… ['V300', 'V313', 'V314', 'V288']
```

After processing V, C and D columns along with the ones with majority null values (>95%) and high skew (>95%) we were able to effectively reduce the number of features from 434 to 186.

Encoding Categorical Columns

We were left with 29 Categorical columns.
We used label encoding for the features with a very large number of classes or for those where classes had a meaningful rank/order. Used One hot encoding for the rest.



```
In [81]:
categorical_columns = train.select_dtypes(include=['O', 'category']).columns
label_encoding = []
for c in categorical_columns:
    uniq = train[c].unique().size
    label_encoding.append(c)
    print(c, '->', uniq)
```

```
ProductCD -> 5
card4 -> 5
card6 -> 5
P_emaildomain -> 43
R_emaildomain -> 32
M1 -> 3
M2 -> 3
M3 -> 3
M4 -> 4
M5 -> 3
M6 -> 3
M7 -> 3
M8 -> 3
M9 -> 3
id_12 -> 3
```

Missing Values

- For the features where we could identify a meaningful pattern during EDA, we manually filled the NAN values.
- For other features, we observed the data distribution (gaussian, skew etc.) and used SimpleImputer to fill mean/mode accordingly.

Oversampling / Undersampling -

As we know the dataset is imbalanced. We used Random Oversampler and Random Under Sampler to deal with this problem. Random Over sampler duplicates the minority class and Random UnderSampler decreases the majority class randomly to the given ratio.

```
over = RandomOverSampler(sampling_strategy=0.2)
under = RandomUnderSampler(sampling_strategy=0.1)
X_over, y_over = over.fit_resample(X, y)
```

Recursive Feature Exclusion / Inclusion -

This is one of the important methods we used to increase the Score for XGBoost. What we did is, we recursively fit and eliminate features based on the feature importance we get after each iteration.

Our Score increased from 0.93 to 0.97 from this method with some hyperparameter tuning.

Drawback for this method is it takes a lot of time and as there are more than 400 features of this dataset, it's better to use this method after reducing a major number of columns from preprocessing.
Another Drawback of this method is overfitting as we are finding an optimal solution for the training dataset, so it may overfit and reduce the score for the test dataset.

Coming to Models-

Cross Validation -

Cross-Validation is a statistical method of evaluating and comparing learning algorithms by dividing data into two segments in which one is to learn or train and other is for validation.
As our data is imbalanced, we used Stratified K-Fold Cross validation. In this method, the data is partitioned into k groups (we chose k=5) such that the validation data has an equal number of instances of target. This ensures that one particular class is not over present in validation or training data.

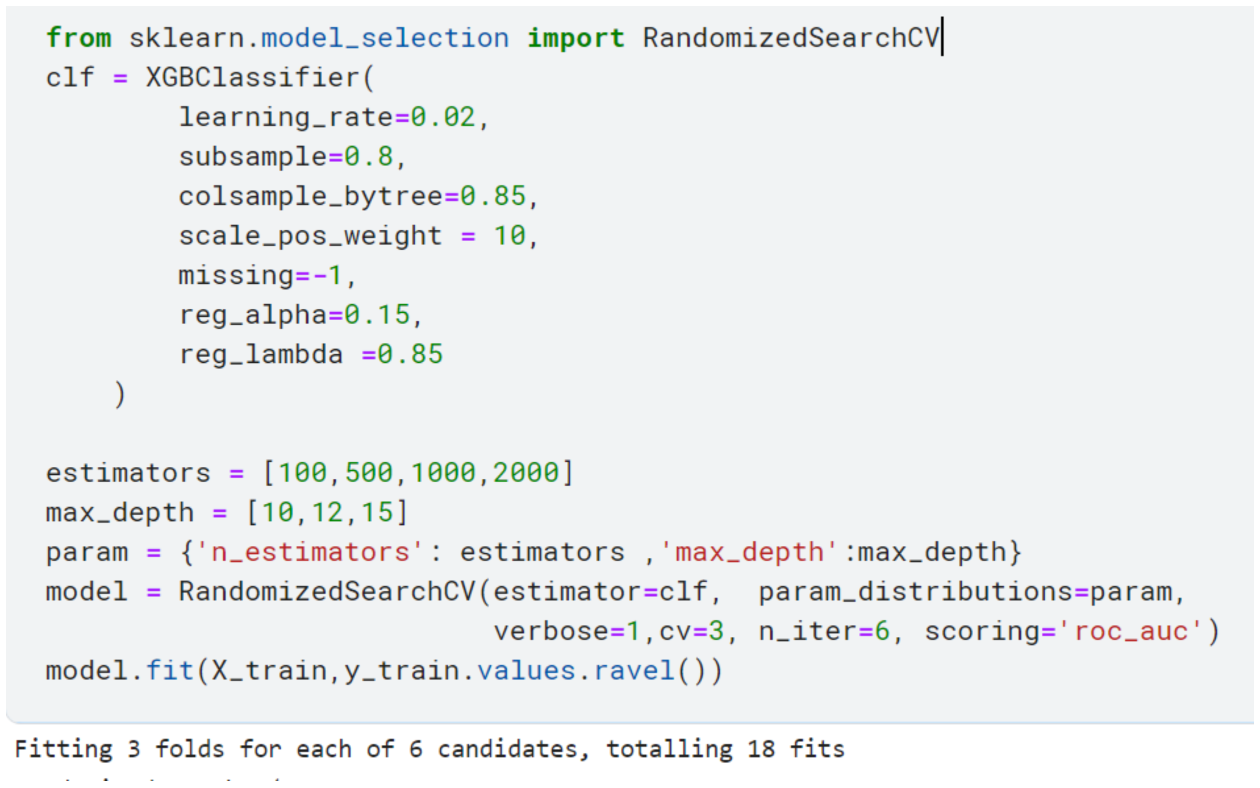
```
n_fold = 5
folds = StratifiedKFold(n_splits=n_fold, shuffle = True)
for fold_n, (train_index, valid_index) in enumerate(folds.split(X, y)):
    clf = XGBClassifier(
        n_estimators=1000,max_depth=15,
        learning_rate=0.03,subsample=0.8,
        colsample_bytree=0.85,scale_pos_weight = 10,missing=-1,
        reg_alpha=0.15,
        reg_lambda =0.85
    )

    X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
    y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]
    clf.fit(X_train,y_train)
    pred_valid[valid_index] = clf.predict_proba(X_valid)[:, 1]
    pred_test += clf.predict_proba(test)[:, 1] / folds.n_splits
    roc_auc_scores.append(roc_auc_score(y_valid, pred_valid[valid_index]))
    print('Fold %2d AUC : %.6f' % (fold_n + 1, roc_auc_score(y_valid,
                                                pred_valid[valid_index])))
```

```
Fold  1 AUC : 0.963352
Fold  2 AUC : 0.965826
Fold  3 AUC : 0.964197
Fold  4 AUC : 0.965661
Fold  5 AUC : 0.964751
Full AUC score 0.964727
```

The final score is computed by taking the mean of scores of each fold.

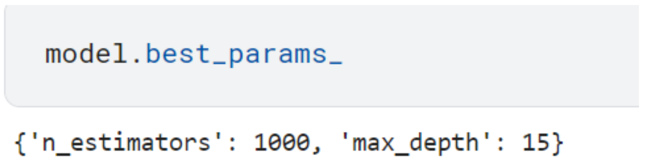
Hyperparameter Tuning -

This method is used to find the optimal values of the hyperparameter of a learning algorithm which will result in highest score.



```python
from sklearn.model_selection import RandomizedSearchCV
clf = XGBClassifier(
        learning_rate=0.02,
        subsample=0.8,
        colsample_bytree=0.85,
        scale_pos_weight = 10,
        missing=-1,
        reg_alpha=0.15,
        reg_lambda =0.85
    )

estimators = [100,500,1000,2000]
max_depth = [10,12,15]
param = {'n_estimators': estimators ,'max_depth':max_depth}
model = RandomizedSearchCV(estimator=clf,  param_distributions=param,
                           verbose=1,cv=3, n_iter=6, scoring='roc_auc')
model.fit(X_train,y_train.values.ravel())
```

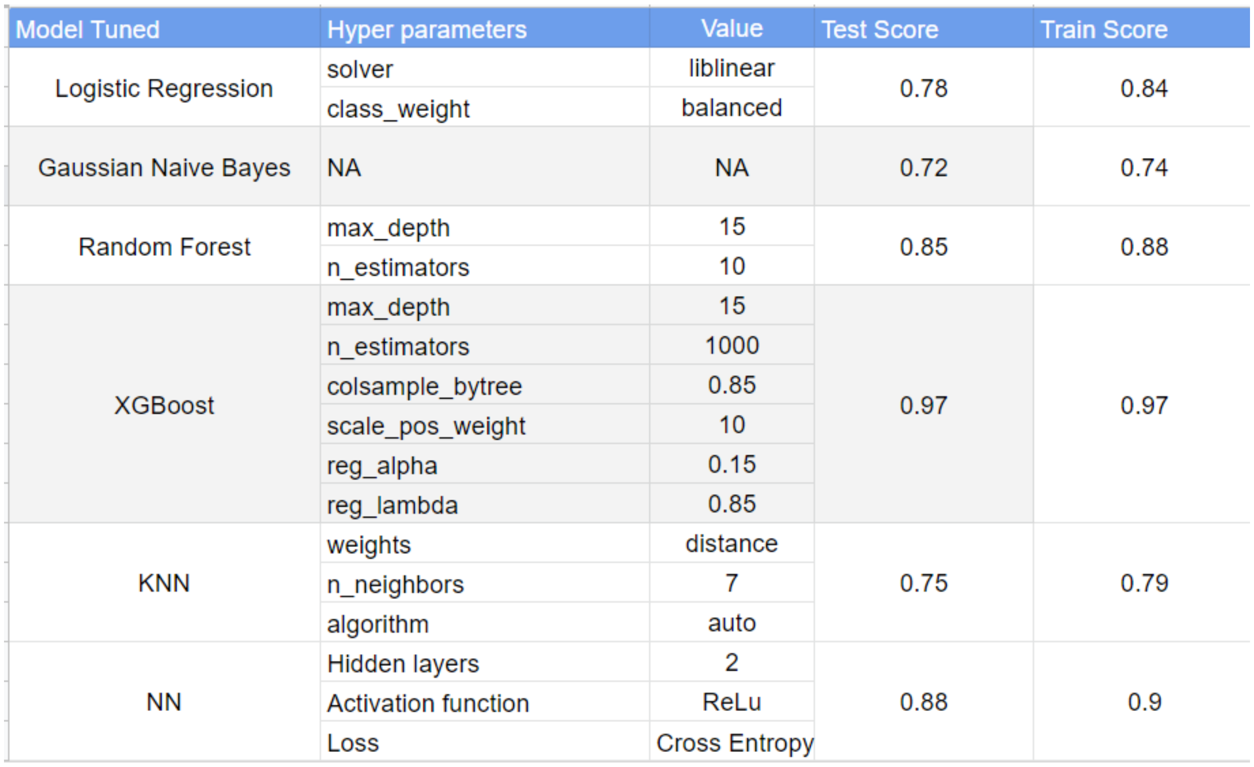
```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
```



```python
model.best_params_
```

```
{'n_estimators': 1000, 'max_depth': 15}
```

Models and their Scores -



Model Tuned	Hyper parameters	Value	Test Score	Train Score
Logistic Regression	solver	liblinear	0.78	0.84
	class_weight	balanced		
Gaussian Naive Bayes	NA	NA	0.72	0.74
Random Forest	max_depth	15	0.85	0.88
	n_estimators	10		
XGBoost	max_depth	15	0.97	0.97
	n_estimators	1000		
	colsample_bytree	0.85		
	scale_pos_weight	10		
	reg_alpha	0.15		
	reg_lambda	0.85		
KNN	weights	distance	0.75	0.79
	n_neighbors	7		
	algorithm	auto		
NN	Hidden layers	2	0.88	0.9
	Activation function	ReLu		
	Loss	Cross Entropy		

Best Model -

We got the best score from XGBoost which is a popular gradient boosted trees algorithm which takes decision trees as base. It is an ensemble of decision trees algorithm where new trees fix errors of those trees that are already part of the model.

Why it is better than others?

- Parallelized tree building capability
- Efficient handling of missing data
- Built-in cross-validation capability
- Regularization through both LASSO (L1) and Ridge (L2) for avoiding overfitting

Final Public Score - 0.97349

Final Private Score - 0.97187

Acess to the Leaderboard -
https://www.kaggle.com/competitions/its-a-fraud/leaderboard

Conclusion-

In this ML project, we learned many things like:-

1) Preprocessing techniques like Cardinality Reduction, Feature Importance/Selection etc.
2) We also came across how we deal with imbalanced dataset- Under / OverSampling , Cross Validation etc.
3) We also came across how we should deal with different types of categorical features.
4) We also came to know about the importance of features in different aspects and how it can change the score with a good margin.

Acknowledgement -

The project has taught us many things like how we should analyze a dataset and come up with a particular machine learning algorithm to solve the problem. We thank Debmalya Sen for the smooth evaluations of this ML Project.

References-

- https://scikit-learn.org/stable/
- https://www.geeksforgeeks.org/xgboost/
- https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b0269f10
- https://machinelearningmastery.com/data-sampling-methods-for-imbalanced-classification/
- https://www.kaggle.com/code/willkoehrsen/intro-to-model-tuning-grid-and-random-search/notebook