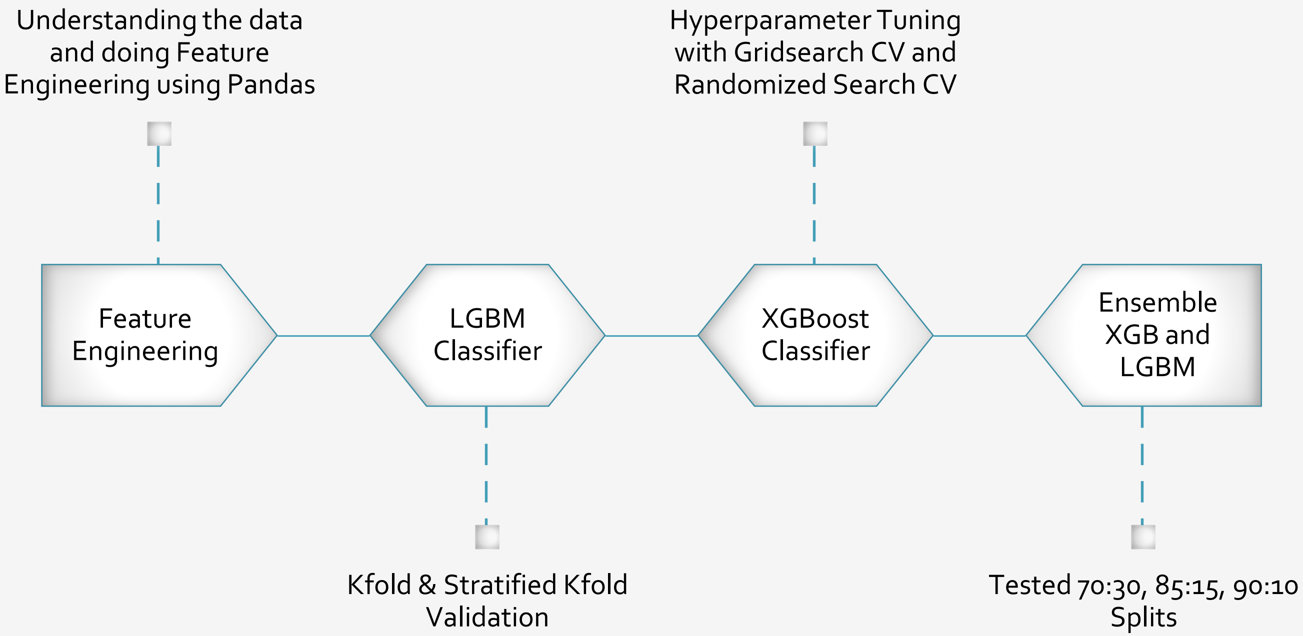
Lead Prediction using Ensemble Boosting Algorithms

Analytics Vidhya JobThon2021 Submission Using LGBM and XGBoost Classifier Ensembling

Submission by Bhooshan Gore

Approach

Started with a Feature Engineering then using Ensembled Boosted Algorithms to Predict the Customer Lead Interest



Feature Engineering

Dataset provided had two types of datatypes (object and int). For our Boosting algorithms to work needed to clean data and convert it into Integers.

```
In [4]: df.head() # to take a quick look dataframe
Out[4]:
```

	ID	Gender	Age	Region_Code	Occupation	Channel_Code	Vintage	Credit_Product	Avg_Account_Balance	Is_Active	Is_Lead
0	NNVBBKZB	Female	73	RG268	Other	X3	43	No	1045696	No	0
1	IDD62UNG	Female	30	RG277	Salaried	X1	32	No	581988	No	0
2	HD3DSEMC	Female	56	RG268	Self_Employed	X3	26	No	1484315	Yes	0
3	BF3NC7KV	Male	34	RG270	Salaried	X1	19	No	470454	No	0
4	TEASRWXV	Female	30	RG282	Salaried	X1	33	No	886787	No	0

```
In [5]: df.dtypes # checking dtypes of the dataframe
Out[5]: ID                     object
        Gender                 object
        Age                     int64
        Region_Code            object
        Occupation             object
        Channel_Code           object
        Vintage                 int64
        Credit_Product         object
        Avg_Account_Balance     int64
        Is_Active              object
        Is_Lead                 int64
        dtype: object
```

- Removed ID from the features
- Gender, Credit_Product, Is_Active, Occupation, Region_Code & Channel_Code are converted into ordinal values.
- Credit_Product being the only feature where missing values were found had to be treated first before converting to Ordinal (Next Slide)
- Result dataset (Below)

```
In [18]: df.dtypes #To check dtypes all being converted to Integers
Out[18]: ID                          object
         Age                          int64
         Region_Code                  int32
         Channel_Code                 int32
         Vintage                      int64
         Avg_Account_Balance          int64
         Is_Lead                      int64
         Credit_Product_Imputed       int32
         Gender_Ordinal               int64
         Credit_Product_Ordinal       int64
         Is_Active_Ordinal            int64
         Occupation_Ordinal           int64
         dtype: object
```

Feature Engineering

Missing values in Credit_Product imputed using Mode (Most Occurred) and added a new feature to identify imputed values.

Function to impute most occurred category and add importance variable for Credit Product feature

In [4]:
```
#1. Function to replace NAN values with mode value
def impute_nan_add_variable(DataFrame,ColName):
    # add new column and replace if category is null then 1 else 0
    DataFrame[ColName+"_Imputed"] = np.where(DataFrame[ColName].isnull(),1,0)
    #Take most occurred category
    Mode_Cat = DataFrame[ColName].mode()[0]
    #Replace NAN values with most occurred category
    DataFrame[ColName].fillna(Mode_Cat,inplace=True)

#Call function to impute and add var
for Columns in ['Credit_Product']:
       impute_nan_add_variable(df,Columns)

for Columns in ['Credit_Product']:
       impute_nan_add_variable(dftest,Columns)


#Display Top 10 Rows to see Result
df[['Credit_Product','Credit_Product_Imputed']].head(10)
```

Out[4]:

	Credit_Product	Credit_Product_Imputed
0	No	0
1	No	0
2	No	0
3	No	0
4	No	0
5	No	0
6	No	1
7	No	0
8	No	0
9	Yes	0

- Added a new feature variable Credit_Product_Imputed which would identify the rows which were imputed.

Feature Engineering

Missing values in Credit_Product imputed using Mode (Most Occurred) and added a new feature to identify imputed values.

Function to impute most occurred category and add importance variable for Credit Product feature

In [4]:
```
#1. Function to replace NAN values with mode value
def impute_nan_add_variable(DataFrame,ColName):
    # add new column and replace if category is null then 1 else 0
    DataFrame[ColName+"_Imputed"] = np.where(DataFrame[ColName].isnull(),1,0)
    #Take most occurred category
    Mode_Cat = DataFrame[ColName].mode()[0]
    #Replace NAN values with most occurred category
    DataFrame[ColName].fillna(Mode_Cat,inplace=True)

#Call function to impute and add var
for Columns in ['Credit_Product']:
       impute_nan_add_variable(df,Columns)

for Columns in ['Credit_Product']:
       impute_nan_add_variable(dftest,Columns)


#Display Top 10 Rows to see Result
df[['Credit_Product','Credit_Product_Imputed']].head(10)
```

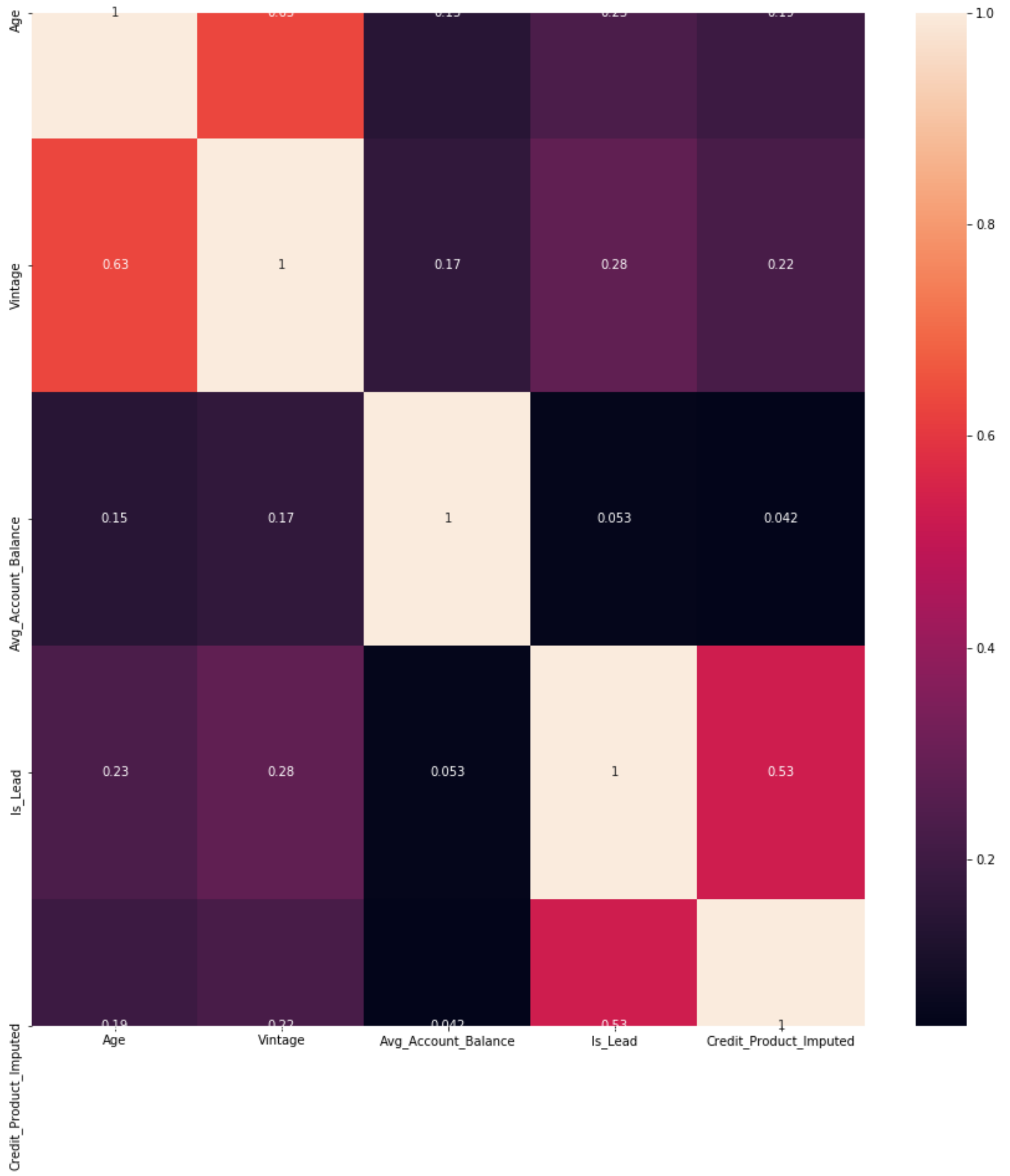
Out[4]:

	Credit_Product	Credit_Product_Imputed
0	No	0
1	No	0
2	No	0
3	No	0
4	No	0
5	No	0
6	No	1
7	No	0
8	No	0
9	Yes	0

- Added a new feature variable Credit_Product_Imputed which would identify the rows which were imputed.

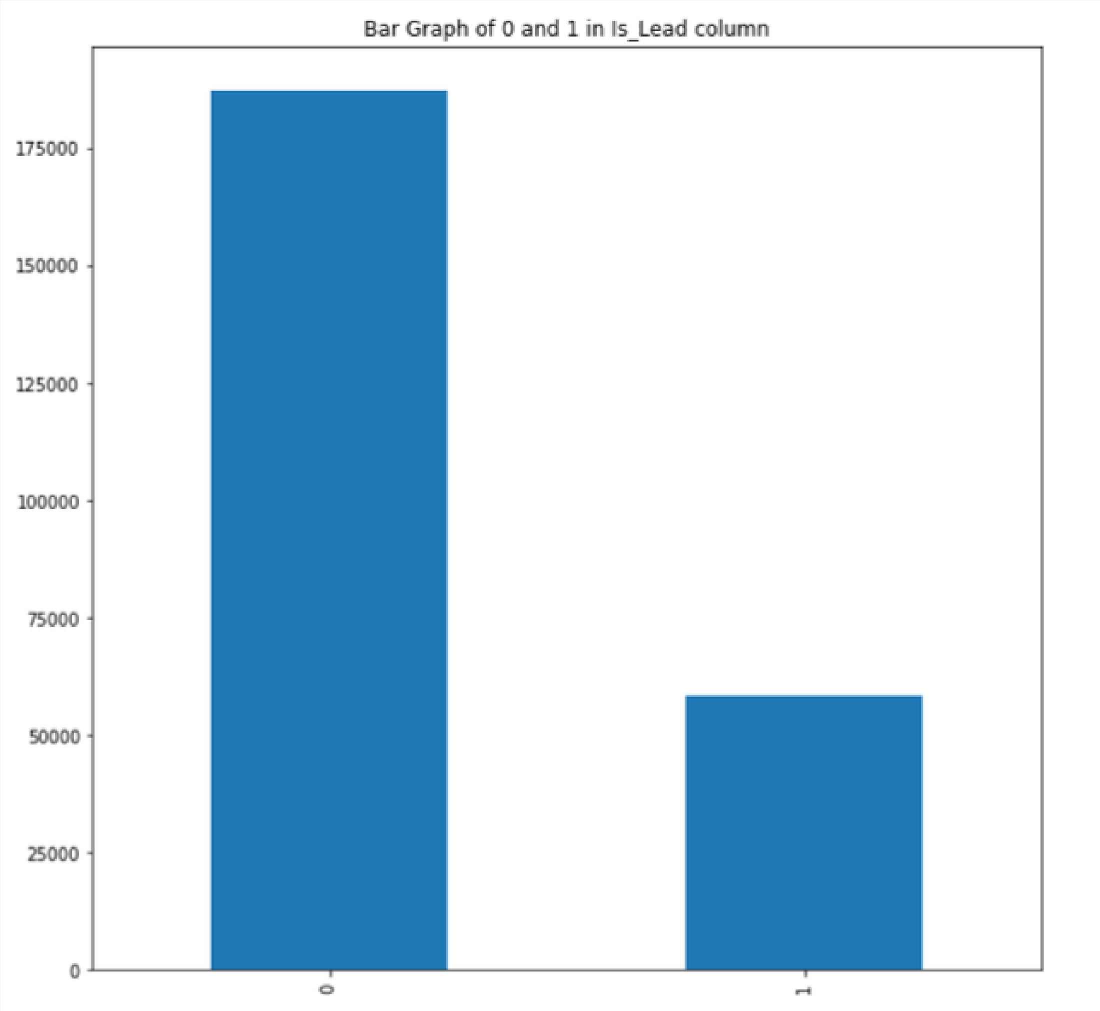
Correlation Matrix

Dataset being too limited in features decided not to drop any variables.



Data Imbalance

Dataset being Imbalanced approached the problem using boosting algorithms which inherently fixed this problem via tuning hyperparameters ie (XGB & LGBM)



XGBoost

Hyperparameter Tuning through GridSearch CV & Randomized Search CV

1. *Randomized Search CV Tuned Results*

```
In [43]: print('Classification report on whole X set\n',classification_report(y,y_pred))

Classification report on whole X set
               precision    recall  f1-score   support

           0       0.90      0.97      0.93    187437
           1       0.87      0.66      0.75     58288

    accuracy                           0.89    245725
   macro avg       0.88      0.81      0.84    245725
weighted avg       0.89      0.89      0.89    245725
```

2. *Grid Search CV Tuned Results*

```
: print('Classification report on whole X set',classification_report(y,y_pred))

Classification report on whole X set               precision    recall  f1-score   support

           0       0.89      0.96      0.93    187437
           1       0.85      0.63      0.72     58288

    accuracy                           0.89    245725
   macro avg       0.87      0.80      0.83    245725
weighted avg       0.88      0.89      0.88    245725
```

RandomizedSearchCV always gives different results. I ran this 4-5 times with different params and 0.87 was the best score I got on the test set.

LGBM

Through Kfold & StratifiedKFold being the fastest one on the lot. This boosting method gave the best result in lowest time.

1. *LGBM Params used*

```
In [9]: lgb_params= {'learning_rate': 0.045,
                     'n_estimators': 20000,
                     'max_bin': 94,
                     'num_leaves': 10,
                     'max_depth': 27,
                     'reg_alpha': 8.457,
                     'reg_lambda': 6.853,
                     'subsample': 0.749}
```

Ran this bit a couple of times with different params and CV folds. 9 folds seemed to have the best score on Test data. With learning rate of 0.045. With highest ROC_AUC score of 87.8%

Ensemble scores from XGB & LGBM

LGBM showing better accuracy has higher weightage than XGBoost.

1. *Split having best result on Test Data*

```
In [42]: lgbnp = 0.90*y_predlgb #Scores generated by LGBM
         xgbrand = 0.10*y_pred2 #Scores generated by XGB Randomized search CV

         submission = lgbnp + xgbrand
```

Ran this bit a couple of times with different splits. 90:10 being the best among the lot. Also tested 70:30, 85:15, 95:5 Splits for the same.

Thank You

Bhooshan Gore

MBA Business Analytics

Business Analyst – Anarock Property Consultants Pvt Ltd

https://bhooshangore.github.io/