Santander Customer Transaction Prediction

Introduction and Overview :

At Santander our mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals.

Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

In this challenge, we invite Kagglers to help us identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted. The data provided for this competition has the same structure as the real data we have available to solve this problem.

Problem Statement

Build model to predict whether customer will make a particular transaction or not.

Data Description

We are provided with an anonymized dataset containing numeric feature(200) variables, the binary target column, and a string ID_code column.

The task is to predict the value of target column in the test set.

File descriptions

train.csv - the training set.

test.csv - the test set. The test set contains some rows which are not included in scoring.

sample_submission.csv - a sample submission file in the correct format

Evaluation Metrics

Submissions are evaluated on area under the ROC curve between the predicted probability and the observed target.

Submission File

For each Id in the test set, you must make a binary prediction of the target variable. The file should contain a header and have the following format:

ID_code,target

test_0,0

test_1,1

test_2,0 etc.

Solution

Read csv and filter 200 features from it

In [0]:

```
import zipfile
with zipfile.ZipFile('/content/drive/My Drive/proj_1/train.csv.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/drive/My Drive/proj_1/')
```

In [0]:

```python
import zipfile
with zipfile.ZipFile('/content/drive/My Drive/proj_1/test.csv.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/drive/My Drive/proj_1/')
```

In [21]:

```python
import pandas as pd
import numpy as np
import os
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
data = pd.read_csv('/content/drive/My Drive/proj_1/train.csv', index_col=0)
target = 'target'

features = [i for i in data.columns if i != target]

print(data.shape, len(features))
data.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
(200000, 201) 200
```

Out[21]:

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12
ID_code														
train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470	2.9252	3.1821	14.0137
train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.0851	-0.4032	8.0585	14.0239
train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525	-0.3249	-11.2648	14.1929
train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450	2.3061	2.8102	13.8463
train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784	-9.4458	-12.1419	13.8481

5 rows × 201 columns

Read test csv and keep index_col as ID_Code

In [22]:

```python
test = pd.read_csv('/content/drive/My Drive/proj_1/test.csv', index_col=0)
test.head()
```

Out[22]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_1
ID_code														
test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	8.8100	-2.0248	-4.3554	13.9696	0.345
test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	5.9739	-1.3809	-0.3310	14.1129	2.566
test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	8.3442	-4.7057	-3.0422	13.6751	3.818

ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_1
test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	7.4578	0.0095	5.0659	14.0526	13.501
test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	7.1437	5.1025	-3.2827	14.1013	8.967

5 rows × 200 columns

Exploratory Data Analysis

Drop duplicates if any

In [0]:

```
data.drop_duplicates(keep='first',inplace=True)
```

Shape of data after drop duplicates (No duplicate)

In [0]:

```
data.shape
```

Out[0]:

```
(200000, 201)
```

Observation

There is no duplicate in the train data.

In [0]:

```
data.target.value_counts()
```

Out[0]:

```
0    179902
1     20098
Name: target, dtype: int64
```

Percentage of value for each class(Imbalance data)

In [0]:

```
data.target.value_counts()/data.shape[0] *100
```

Out[0]:

```
0    89.951
1    10.049
Name: target, dtype: float64
```

Observation

1. Around 90 percent data belong to Class 0 and approx. 10 percent belong to Class 1.
2. It is an imbalance data two class classification problem.

Check for missing values rows

In [0]:

```
data.isnull().sum().sort_values()
```

Out[0]:

```
target     0
var_126    0
var_127    0
var_128    0
var_129    0
          ..
var_69     0
var_70     0
var_71     0
var_61     0
var_199    0
Length: 201, dtype: int64
```

Observation

1. There are no missing values in train data.

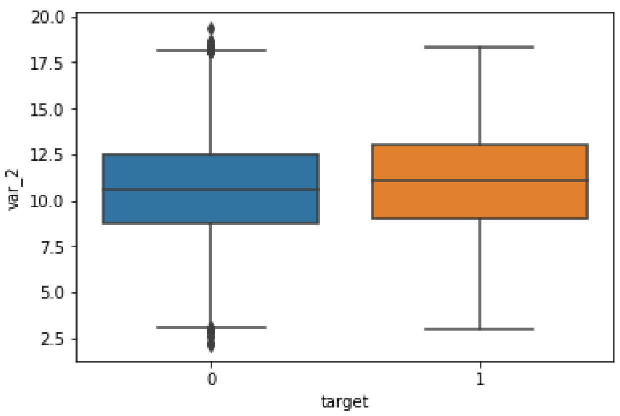
Box plot for var_2 ,var_10 and var_25

In [0]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

sns.boxplot(x="target",y="var_2",data=data)
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9804ad18d0>
```

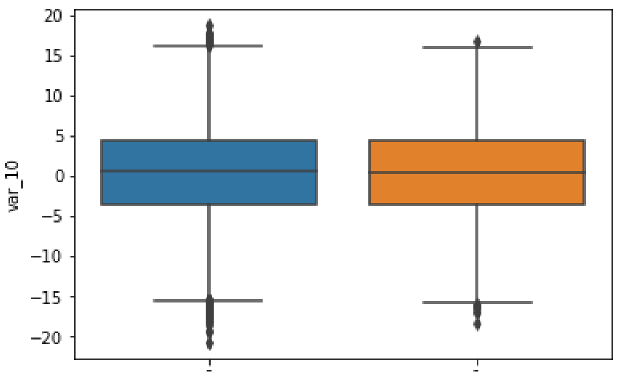


In [0]:

```
sns.boxplot(x="target",y="var_10",data=data)
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f98027b2c18>
```

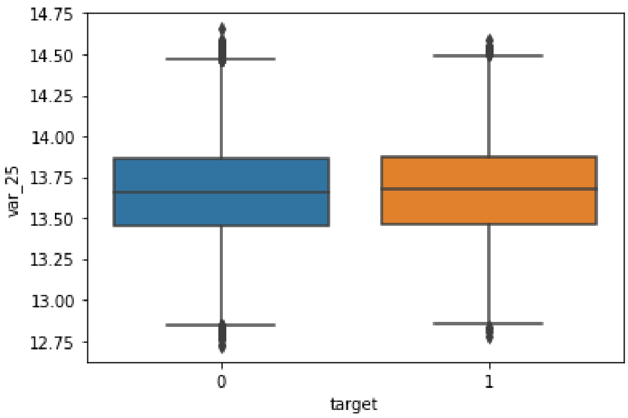


In [0]:

```
sns.boxplot(x="target",y="var_25",data=data)
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f980278fcc0>
```



Observation

1. Presence of diamond shape peaked at minima and maxima in above box plots suggest that there are outliers in the data.
2. In case of var_2, the outliers are only there for target 0 .
3. In case of var_10 and var_25 , there are outliers for both target 0 and target 1.

Conclusion

We will look at futher percentile level to identify outliers for sample feature i.e. var_10 .

Approach 1 : Percentile method for outlier removal

Finding upper bound (maxima)

In [0]:

```
for i in np.arange(0.0, 1.0, 0.1):
    var =data["var_10"].values
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(99+i,var[int(len(var)*(float(99+i)/100))]))
print("100 percentile value is ",var[-1])
```

```
99.0 percentile value is 11.9777
99.1 percentile value is 12.088
99.2 percentile value is 12.2104
99.3 percentile value is 12.347000000000001
99.4 percentile value is 12.4978
99.5 percentile value is 12.6769
99.6 percentile value is 12.8937
99.7 percentile value is 13.1359
99.8 percentile value is 13.4987
99.9 percentile value is 14.1528
100 percentile value is  18.6702
```

Finding lower bound (minima)

In [0]:

```
for i in np.arange(0.0, 1, 0.1):
    var =data["var_10"].values
```

```
    var = np.sort(var,axis = None)
    print("{} percentile value is {}".format(0+i,var[int(len(var)*(float(0+i)/100))]))
```

```
0.0 percentile value is -20.7313
0.1 percentile value is -14.7729
0.2 percentile value is -13.9897
0.30000000000000004 percentile value is -13.4133
0.4 percentile value is -13.0439
0.5 percentile value is -12.7568
0.6000000000000001 percentile value is -12.4935
0.7000000000000001 percentile value is -12.3016
0.8 percentile value is -12.0763
0.9 percentile value is -11.8839
```

In [0]:

```
data_after_outlier_removal_combined=data[((data.var_10>=-14.7729) & (data.var_10<=14.1528))]
```

In [0]:

```
data_after_outlier_removal_combined.shape
```

Out[0]:

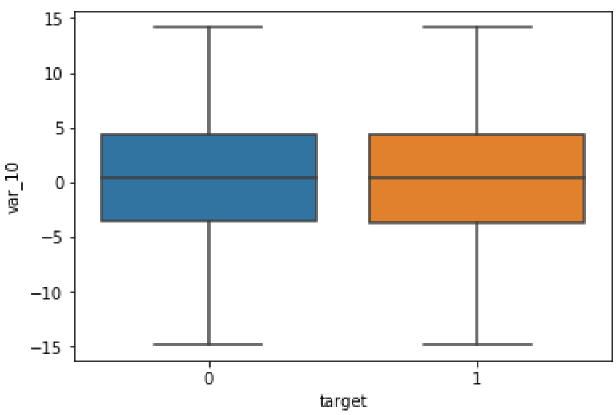
```
(199601, 201)
```

In [0]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.boxplot(x="target",y="var_10",data=data_after_outlier_removal_combined)
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9802752198>
```



Approach 2: Outlier removal using Beyond IQR Range

In [0]:

```
d_var_10=data
```

In [0]:

```
var =d_var_10['var_10'].values
var = np.sort(var,axis = None)
minima=var[int(len(var)*(float(25.0)/100))]-1.5*(var[int(len(var)*(float(75.0)/100))]-var[int(len(va
r)*(float(25.0)/100))])
maxima=var[int(len(var)*(float(75.0)/100))]+1.5*(var[int(len(var)*(float(75.0)/100))]-var[int(len(va
r)*(float(25.0)/100))])
d_var_10=d_var_10[((d_var_10['var_10']>=minima) & (d_var_10['var_10']<=maxima))]
```

In [0]:

```
d_var_10.shape
```

Out[0]:

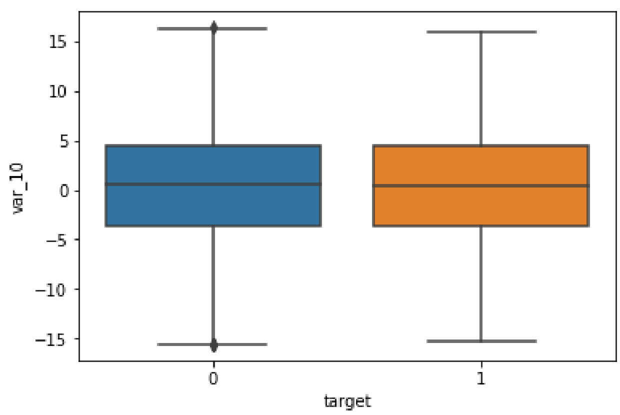
```
(199875, 201)
```

In [0]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.boxplot(x="target",y="var_10",data=d_var_10)
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f97ef616208>
```



Conclusion after var_10 outlier removal

1. Approach 1 : Outlier removal percentage data lost= 2 percent.
2. Approach 2 : Outlier removal percentage data lost= .06 percent.
3. Hence We can use the Approach 2 outlier removal strategy for all other features such as var_0 to var_199.

In [0]:

```
### Function for finding lower_bound and upper_bound

def lower_bound_fn(feature):
  """This function return lower bound i.e. data point which is lowest point in  IQR range"""
    var =data[feature].values
    var = np.sort(var,axis = None)
    # result= 25th percentile-1.5 *(75th percentile -25th percentile)
    result=var[int(len(var)*(float(25.0)/100))]-1.5*(var[int(len(var)*(float(75.0)/100))]-var[int(le
n(var)*(float(25.0)/100))])
    return result
def upper_bound_fn(feature):
  """This function return upper bound i.e.highest data  point in  IQR range"""
    var =data[feature].values
    var =data[feature].values
    var = np.sort(var,axis = None)
    # result= 75th percentile+1.5 *(75th percentile -25th percentile)
    result=var[int(len(var)*(float(75.0)/100))]+1.5*(var[int(len(var)*(float(75.0)/100))]-var[int(le
n(var)*(float(25.0)/100))])
    return result
```

In [0]:

```
from tqdm import tqdm
lower_bound=[]
upper_bound=[]
for i in tqdm(features):
    a=lower_bound_fn(i)
```

```
    b=upper_bound_fn(i)
    lower_bound.append(a)
    upper_bound.append(b)
```

```
100%|██████████| 200/200 [00:07<00:00, 26.28it/s]
```

In [0]:

```
print(len(lower_bound))
print(len(upper_bound))
```

```
200
200
```

In [0]:

```
df=data
```

In [0]:

```
for i in tqdm(range(200)):
    a=lower_bound[i]
    b=upper_bound[i]
    df=df[((df['var_'+str(i)]>=a) & (df['var_'+str(i)]<=b))]
```

```
100%|██████████| 200/200 [00:15<00:00, 13.72it/s]
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/data_cleaned.sav'
pickle.dump(df, open(filename, 'wb'))
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/data_cleaned.sav'
data_cleaned = pickle.load(open(filename, 'rb'))
```

Shape of data after outlier removal

In [0]:

```
data_cleaned.shape
```

Out[0]:

```
(175107, 201)
```

In [0]:

```
data_cleaned.target.value_counts()
```

Out[0]:

```
0    158002
1     17105
Name: target, dtype: int64
```

Check Correlation between features

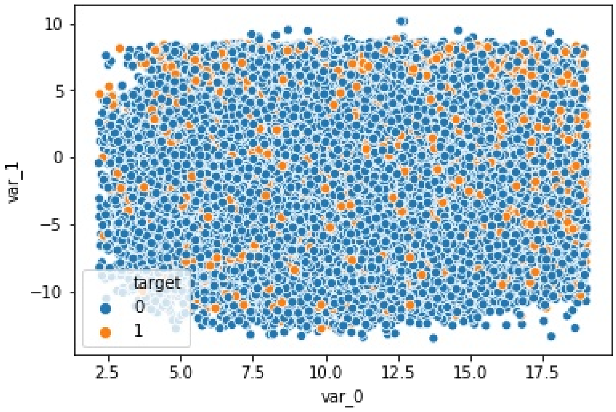
Scatter plot to check correlation between feature i.e. var_0 and var_1

In [0]:

```
sns.scatterplot(x=data_cleaned['var_0'], y=data_cleaned['var_1'], hue=data_cleaned['target'])
```

Out[0]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7f86dd8a4ba8&gt;
```

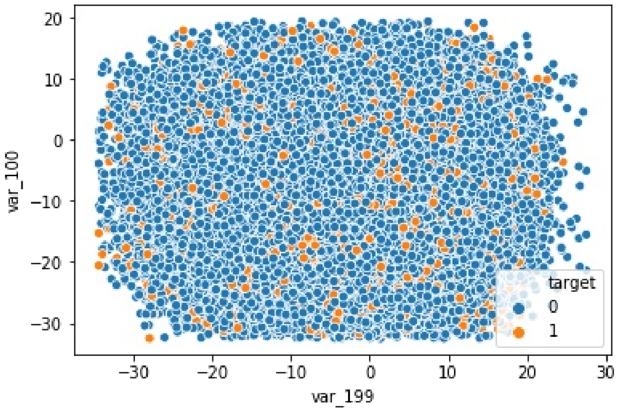


In [0]:

```
sns.scatterplot(x=data_cleaned['var_199'], y=data_cleaned['var_100'], hue=data_cleaned['target'])
```

Out[0]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7f868f997a58&gt;
```



Observation

1. There seems to be no or very less correlation between var_0 and var_1 and var_100 and var_199.

In [0]:

```
###https://www.kaggle.com/gpreda/santander-eda-and-prediction
correlations =
data_cleaned[features].corr().abs().unstack().sort_values(kind="quicksort").reset_index()
correlations = correlations[correlations['level_0'] != correlations['level_1']]
correlations.head(10)
```

Out[0]:

	level_0	level_1	0
0	var_26	var_76	1.355562e-07
1	var_76	var_26	1.355562e-07
2	var_115	var_96	1.446246e-07
3	var_96	var_115	1.446246e-07

	level_0	level_1	0
4	var_162	var_113	3.227749e-07
5	var_113	var_162	3.227749e-07
6	var_66	var_187	5.188254e-07
7	var_187	var_66	5.188254e-07
8	var_24	var_85	7.484635e-07
9	var_85	var_24	7.484635e-07

In [0]:

```
correlations.tail(10)
```

Out[0]:

	level_0	level_1	0
39790	var_123	var_12	0.009513
39791	var_12	var_123	0.009513
39792	var_166	var_12	0.009649
39793	var_12	var_166	0.009649
39794	var_183	var_189	0.009765
39795	var_189	var_183	0.009765
39796	var_139	var_26	0.009787
39797	var_26	var_139	0.009787
39798	var_127	var_162	0.010177
39799	var_162	var_127	0.010177

Observations

1. var_26 and var_76 have a lowest correlation value of 1.355562e-07 amongst all other features.
2. var_162 and var_127 have a highest correlation value of 0.010177 among all other features.
3. Hence,All features are very least correlated with each other .

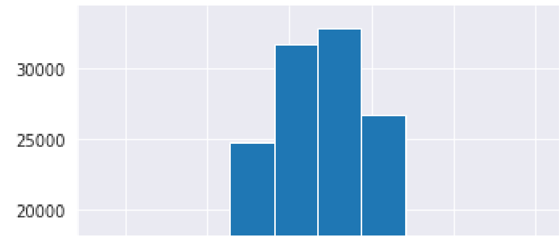
Conclusion

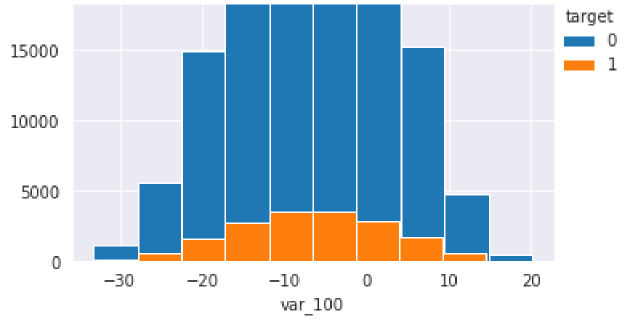
There is no Multicollinearity (as very less correlation) i.e. there will no degradation of performance of model.

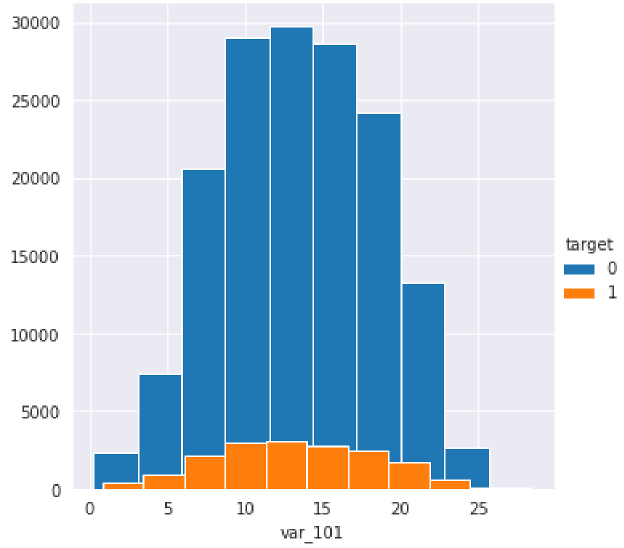
Histogram plot for some features

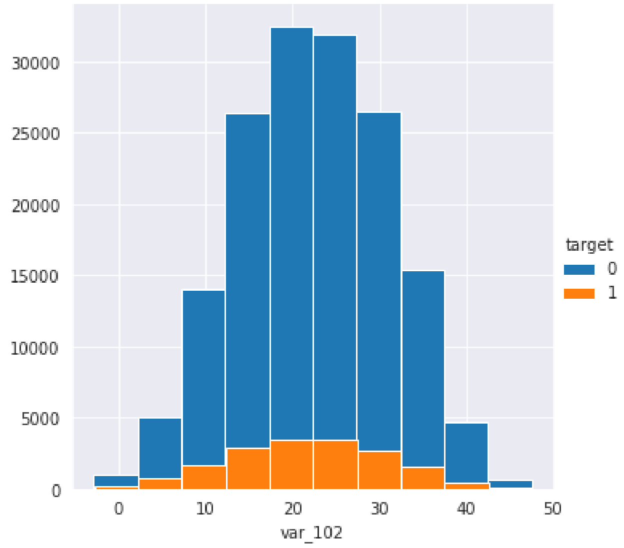
In [0]:

```
%matplotlib inline
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style("darkgrid");
for i in list(features[100:103]):
  sns.FacetGrid(data_cleaned, hue="target", height=5) \
  .map(plt.hist,i)\
  .add_legend();
```









Observation

1. Some of values have verify high frequency. Hence, we can use some featurization around duplicate value.
2. The Spread of Class 1 is more as compared to Class 0 for each of the feature.

Conclusion after EDA

1. We have cleaned data by removing outliers which may help us lesser performance impact on model.
2. All features are very less correlated i.e.Correlation imply dependency.Hence , We can use Naive Bayes as baseline model.
3. Also Naive Bayes have very less computation cost as compared to other simpler models.
4. Some of values have verify high frequency. Hence, we can use some featurization around duplicate value.

Modeling with original 200 features

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/data_cleaned.sav'
data_cleaned = pickle.load(open(filename, 'rb'))
```

In [0]:

```
data_cleaned.head()
```

Out[0]:

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12
ID_code														
train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470	2.9252	3.1821	14.0137
train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525	-0.3249	-11.2648	14.1929
train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450	2.3061	2.8102	13.8463
train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784	-9.4458	-12.1419	13.8481
train_5	0	11.4763	-2.3182	12.6080	8.6264	10.9621	3.5609	4.5322	15.2255	3.5855	5.9790	0.8010	-0.6192	13.6380

5 rows × 201 columns

In [0]:

```
def split_train_test(data):
  """This function will train and cv data point """
    from sklearn.model_selection import train_test_split
    df=data.drop(['target'],axis=1)
    y=data['target']
    X_train, X_cv, y_train, y_cv = train_test_split(df, y, test_size = 0.20, stratify=y)
    print('Train data shape : '+str(X_train.shape))
    print('CV data shape : '+str(X_cv.shape))
    return X_train,y_train,X_cv,y_cv
```

In [0]:

```
X_train,y_train,X_cv,y_cv=split_train_test(data_cleaned)
```

```
Train data shape : (140085, 200)
CV data shape : (35022, 200)
```

In [0]:

```
def grid_search(clf,params,n_folds,X_train,y_train,jobs):
    """It will do hyperparameter tuning using grid search and return best parameters"""
    from sklearn.metrics import roc_auc_score
    from sklearn.model_selection import StratifiedKFold
    from sklearn.model_selection import GridSearchCV
    cv_method = StratifiedKFold(n_splits=n_folds)
    gs = GridSearchCV(estimator=clf,
                      param_grid=params,
                      cv=cv_method,
                      verbose=1,
                      scoring='roc_auc',n_jobs=jobs)
    gs.fit(X_train, y_train)
    print('best param : '+str(gs.best_params_))
    print('best score : '+str(gs.best_score_))
    return gs.best_params_
```

Naive Bayes with 200 original features

In [0]:

```
#https://www.featureranking.com/tutorials/machine-learning-tutorials/sk-part-3-cross-validation-an
d-hyperparameter-tuning/
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV

gnb = GaussianNB(priors = [0.5,0.5])

params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
best_param=grid_search(gnb,params_NB,5,X_train,y_train,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   11.9s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   48.9s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  2.2min finished
```

```
best param : {'var_smoothing': 1.873817422860383e-08}
best score : 0.8873297741295822
```

In [0]:

```
#https://github.com/ClimbsRocks/machineJS/issues/176
def baseline_model(best_param,X_train,y_train,X_cv,y_cv,test,best_feat):
    gnb = GaussianNB(priors = [0.5,0.5],var_smoothing=best_param['var_smoothing'])
    gnb.fit(X_train, y_train)
    y_train_pred = gnb.predict(X_train)
    y_cv_pred = gnb.predict(X_cv)
    predictions=gnb.predict(test[best_feat])
    test=test.reset_index()
    submission = pd.DataFrame({"ID_code": test.ID_code.values})
    submission['target'] = predictions
    submission.to_csv("/content/drive/My Drive/proj_1/submission_nb.csv", index=False)
    print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
    print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))

    return y_train_pred,y_cv_pred
```

In [0]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
best_param={'var_smoothing':1.873817422860383e-08}
best_feat=test.columns
y_train_pred,y_cv_pred=baseline_model(best_param,X_train,y_train,X_cv,y_cv,test,best_feat)
```

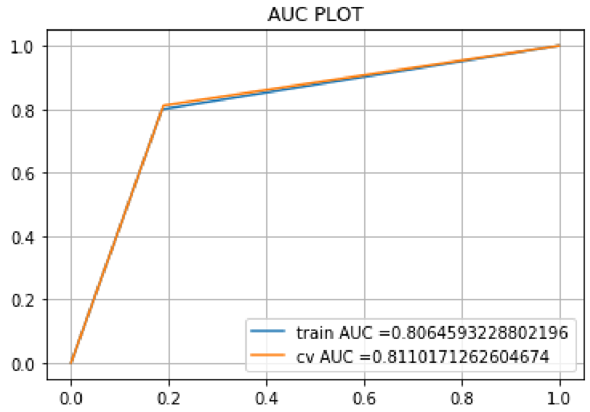
```
train auc score : 0.8064593228802196
cv auc score : 0.8110171262604674
```

In [0]:

```
%matplotlib inline
def auc_plot(y_train,y_train_pred,y_cv,y_cv_pred):
    """It will plot tpr vs fpr plot """
    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
    train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
    test_fpr, test_tpr, te_thresholds = roc_curve(y_cv, y_cv_pred)
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="cv AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.title("AUC PLOT")
    plt.grid()
    plt.show()
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



Kaggle Score

1. Private Score: 0.80495
2. Public Score : 0.80560

SVM(Support Vector Machine) with 200 original features

In [0]:

```
from sklearn.linear_model import SGDClassifier

params = {'alpha': np.logspace(4,-9, num=100)}
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',max_iter=100,tol=1e-3,class_weight = 'balanced')
best_param=grid_search(sgd,params,5,X_train,y_train,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   12.6s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:  8.1min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  9.9min finished
```

```
best param : {'alpha': 0.04132012400115335}
best score : 0.847044723236534
```

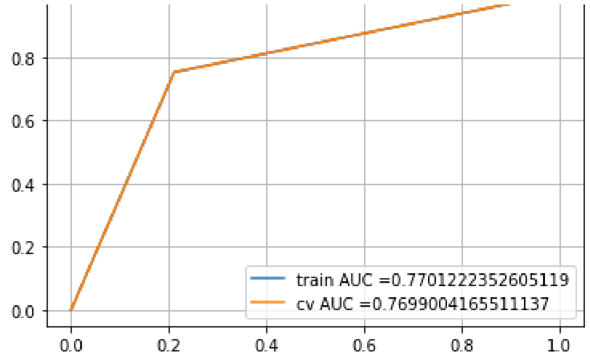
In [0]:

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=100,tol=1e-3,
alpha = 0.04132012400115335)
sgd.fit(X_train ,y_train)
y_train_pred=sgd.predict(X_train)
y_cv_pred = sgd.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7701222352605119
cv auc score : 0.7699004165511137
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=sgd.predict(test)
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_sgd.csv", index=False)
```

Kaggle Score

1. Private Score: 0.76828
2. Public Score : 0.76753

Xgboost with 200 original features

In [0]:

```
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,7,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,cv=5,scoring = 'roc_auc',n
_jobs=-1)
random_cfl.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed: 18.3min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 24.8min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 43.0min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed: 59.5min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 149.5min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 157.3min remaining: 34.5min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 175.5min remaining: 11.2min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 179.6min finished
```

Out[0]:

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'n_estimators': [100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 7,
```

```
10], 'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'colsample_bytree': [0.1, 0.3, 0.5, 1],
'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='roc_auc', verbose=10)
```

In [0]:

```
random_cfl.best_estimator_
```

Out[0]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.1, gamma=0,
       learning_rate=0.15, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=1000, n_jobs=1,
       nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=0.5, verbosity=1)
```

In [0]:

```
from xgboost import XGBClassifier
xgb = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.1, gamma=0,
       learning_rate=0.15, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=1000, n_jobs=-1,
       nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=0.5, verbosity=1)
xgb.fit(X_train, y_train)
y_train_pred = xgb.predict(X_train)
y_cv_pred=xgb.predict(X_cv)
```

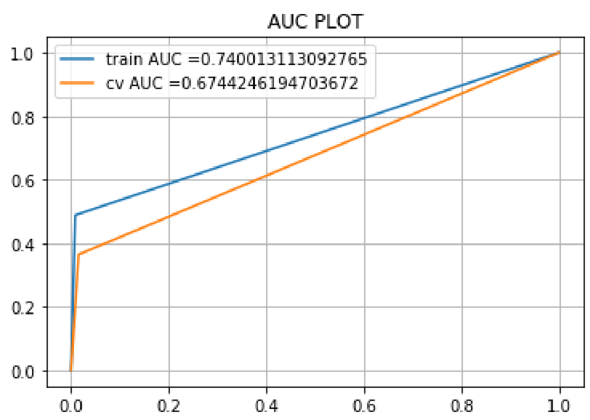
In [0]:

```
from sklearn.metrics import roc_auc_score
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.740013113092765
cv auc score : 0.6744246194703672
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=xgb.predict(test)
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_xgb.csv", index=False)
```

Kaggle Score

1. Private Score: 0.67655
2. Public Score : 0.67846

lightgbm with 200 original features

In [0]:

```
#basic tools
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
import os
import numpy as np
import pandas as pd
import warnings

#tuning hyperparameters
from bayes_opt import BayesianOptimization
from skopt  import BayesSearchCV

#graph, plots
import matplotlib.pyplot as plt
import seaborn as sns

#building models
import lightgbm as lgb
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
import time
import sys

#metrics
from sklearn.metrics import roc_auc_score, roc_curve
import shap
warnings.simplefilter(action='ignore', category=FutureWarning)

def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=3, random_seed=6, output_pro
cess=False):
    # prepare data
    train_data = lgb.Dataset(data=X, label=y, free_raw_data=False)
    # parameters
    def lgb_eval(learning_rate,num_leaves, feature_fraction, bagging_fraction, max_depth, max_bin,
min_data_in_leaf,min_sum_hessian_in_leaf,subsample,lambda_l1,lambda_l2):
        params = {'application':'binary', 'metric':'auc'}
        params['learning_rate'] = max(min(learning_rate, 1), 0)
        params["num_leaves"] = int(round(num_leaves))
        params['feature_fraction'] = max(min(feature_fraction, 1), 0)
        params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
        params['max_depth'] = int(round(max_depth))
        params['max_bin'] = int(round(max_depth))
        params['min_data_in_leaf'] = int(round(min_data_in_leaf))
        params['min_sum_hessian_in_leaf'] = min_sum_hessian_in_leaf
        params['subsample'] = max(min(subsample, 1), 0)
        params['lambda_l1']=max(min(lambda_l1, 1), 0)
        params['lambda_l2']=max(min(lambda_l1, 1), 0)
        #evaluate cv for above paramters
        cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, ve
rbose_eval =200, metrics=['auc'])

        #return max mean for multiple folds
        return max(cv_result['auc-mean'])

    lgbBO = BayesianOptimization(lgb_eval, {'learning_rate': (0.01, 1.0),
                                            'num_leaves': (24, 80),
                                            'feature_fraction': (0.1, 0.9),
                                            'bagging_fraction': (0.8, 1),
                                            'max_depth': (5, 80),
                                            'max_bin':(20,150),
                                            'lambda_l1': (0.01,1),
                                            'lambda_l2': (0.01,1),
                                            'min_data_in_leaf': (20, 80),
                                            'min_sum_hessian_in_leaf':(0,100),
                                           'subsample': (0.01, 1.0)}, random_state=200)
```

```python
    #n_iter: How many steps of bayesian optimization you want to perform. The more steps the more likely to find a good maximum you are.
    #init_points: How many steps of random exploration you want to perform. Random exploration can help by diversifying the exploration space.

    lgbBO.maximize(init_points=init_round, n_iter=opt_round)

    model_auc=[]
    for model in range(len( lgbBO.res)):
        model_auc.append(lgbBO.res[model]['target'])

    # return best parameters
    return lgbBO.res[pd.Series(model_auc).idxmax()]['target'],lgbBO.res[pd.Series(model_auc).idxmax()]['params']

opt_params = bayes_parameter_opt_lgb(X_train, y_train, init_round=5, opt_round=15, n_folds=10, random_seed=6)
```

```
|   iter    |  target   | baggin... | featur... | lambda_l1 | lambda_l2 | learni... |  max_bin  | max_depth | min_da... | min_su... | num_le... | subsample |
-------------------------------------------------------------------------------------------------------------------------------------------------------------
|  1        |  0.8312   |  0.9895   |  0.2812   |  0.5985   |  0.434    |  0.7665   |  20.37    | 31.81     |  74.58    |  45.61    |  78.98    |  0.8687   |
|  2        |  0.8719   |  0.9972   |  0.8386   |  0.3107   |  0.8476   |  0.13     |  122.1    | 23.79     |  25.76    |  94.35    |  70.26    |  0.5231   |
|  3        |  0.8612   |  0.9747   |  0.5627   |  0.4556   |  0.6834   |  0.4252   |  103.3    | 50.65     |  26.33    |  96.6     |  66.49    |  0.6828   |
|  4        |  0.8709   |  0.8659   |  0.1212   |  0.8056   |  0.9731   |  0.2901   |  104.4    | 24.92     |  31.26    |  41.9     |  61.3     |  0.5222   |
|  5        |  0.8707   |  0.9709   |  0.2368   |  0.9784   |  0.3083   |  0.2401   |  122.9    | 71.3      |  79.24    |  1.606    |  35.45    |  0.6491   |
|  6        |  0.835    |  0.9246   |  0.4454   |  0.5665   |  0.4644   |  0.02659  |  148.7    | 76.52     |  24.45    |  5.056    |  79.2     |  0.7439   |
|  7        |  0.8509   |  0.8136   |  0.2585   |  0.1111   |  0.36     |  0.08178  |  136.9    | 5.564     |  22.66    |  99.75    |  25.69    |  0.8743   |
|  8        |  0.8547   |  0.9312   |  0.6735   |  0.1587   |  0.8524   |  0.1187   |  148.4    | 6.087     |  79.43    |  4.33     |  76.36    |  0.4445   |
|  9        |  0.8503   |  0.9011   |  0.2515   |  0.06111  |  0.8813   |  0.6513   |  22.11    | 21.72     |  20.26    |  4.632    |  24.73    |  0.6119   |
|  10       |  0.8479   |  0.8567   |  0.6408   |  0.8629   |  0.5351   |  0.6544   |  145.7    | 63.79     |  78.16    |  99.44    |  73.15    |  0.4831   |
|  11       |  0.8482   |  0.8896   |  0.3597   |  0.2208   |  0.8963   |  0.8779   |  46.74    | 5.629     |  20.26    |  91.96    |  79.48    |  0.2009   |
|  12       |  0.8218   |  0.8949   |  0.7475   |  0.2667   |  0.166    |  0.9944   |  63.63    | 6.706     |  79.51    |  3.019    |  26.09    |  0.7404   |
|  13       |  0.8647   |  0.8788   |  0.8361   |  0.1604   |  0.4895   |  0.119    |  25.14    | 77.79     |  27.87    |  0.5888   |  73.86    |  0.1782   |
|  14       |  0.873    |  0.9889   |  0.1523   |  0.2916   |  0.5505   |  0.2856   |  20.9     | 79.28     |  42.58    |  40.04    |  24.63    |  0.9563   |
|  15       |  0.8686   |  0.8806   |  0.5225   |  0.6978   |  0.2634   |  0.333    |  48.61    | 79.86     |  21.41    |  1.556    |  25.48    |  0.1976   |
|  16       |  0.8697   |  0.8318   |  0.2636   |  0.5765   |  0.1076   |  0.2983   |  145.0    | 6.756     |  24.33    |  2.303    |  24.62    |  0.5951   |
|  17       |  0.8593   |  0.9115   |  0.8303   |  0.1222   |  0.1597   |  0.3882   |  147.3    | 7.365     |  20.99    |  53.04    |  76.15    |  0.208    |
|  18       |  0.8635   |  0.8286   |  0.1732   |  0.9375   |  0.1168   |  0.1015   |  20.84    | 66.25     |  78.36    |  99.43    |  24.02    |  0.848    |
|  19       |  0.8489   |  0.8037   |  0.2421   |  0.7778   |  0.5669   |  0.03034  |  24.6     | 75.46     |  79.85    |  98.86    |  78.69    |  0.9903   |
|  20       |  0.8516   |  0.862    |  0.4066   |  0.4964   |  0.3447   |  0.7459   |  147.3    | 25.1      |  76.58    |  40.75    |  24.46    |  0.1995   |
=============================================================================================================================================================
```

In [0]:

```python
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
opt_params[1]["num_leaves"] = int(round(opt_params[1]["num_leaves"]))
opt_params[1]['max_depth'] = int(round(opt_params[1]['max_depth']))
opt_params[1]['min_data_in_leaf'] = int(round(opt_params[1]['min_data_in_leaf']))
opt_params[1]['max_bin'] = int(round(opt_params[1]['max_bin']))
opt_params[1]['objective']='binary'
```

```
opt_params[1]['metric']='auc'
opt_params[1]['is_unbalance']=True
opt_params[1]['boost_from_average']=False
opt_params=opt_params[1]
opt_params
```

Out[0]:

```
{'bagging_fraction': 0.9888504387768287,
 'boost_from_average': False,
 'feature_fraction': 0.1522821422842494,
 'is_unbalance': True,
 'lambda_l1': 0.29162576402836243,
 'lambda_l2': 0.5505010295618853,
 'learning_rate': 0.2855727327499761,
 'max_bin': 21,
 'max_depth': 79,
 'metric': 'auc',
 'min_data_in_leaf': 43,
 'min_sum_hessian_in_leaf': 40.03510879288107,
 'num_leaves': 25,
 'objective': 'binary',
 'subsample': 0.9562916362414675}
```

In [0]:

```
##https://www.kaggle.com/graf10a/lightgbm-lb-0-9675
import lightgbm as lgb
d_train = lgb.Dataset(X_train, label=y_train)
clf = lgb.train(opt_params, d_train)
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/lgbm_200_model.sav'
pickle.dump(clf, open(filename, 'wb'))
```

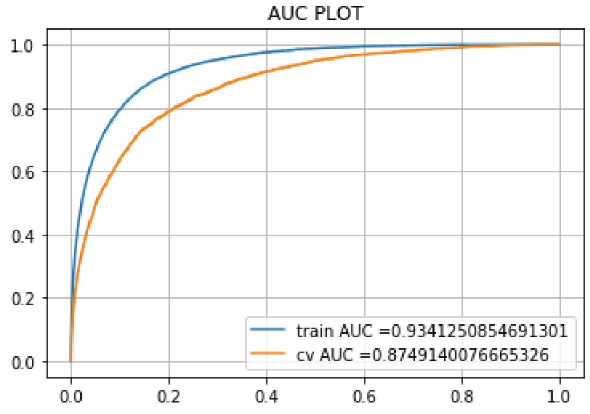
In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/lgbm_200_model.sav'
lm = pickle.load(open(filename, 'rb'))
y_train_pred = lm.predict(X_train)
y_cv_pred=lm.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.9341250854691301
cv auc score : 0.8749140076665326
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=lm.predict(test)
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_lgb.csv", index=False)
```

Kaggle Score

1. Private Score: 0.86914
2. Public Score : 0.87191

Featurization 1

Duplicate values per each feature

In [0]:

```
# https://www.kaggle.com/kakenovyernur/kakenov-yernur
unique_max_train = []
for feature in features:
    values = data_cleaned[feature].value_counts()
    unique_max_train.append([feature, values.max(), values.idxmax()])
duplicate_val_per_feat=pd.DataFrame(unique_max_train, columns=['Feature', 'Max duplicates',
'Value']).sort_values(by = 'Max duplicates', ascending=False)
```

In [0]:

```
duplicate_val_per_feat.head(5)
```

Out[0]:

	Feature	Max duplicates	Value
68	var_68	961	5.0208
126	var_126	267	11.5356
108	var_108	267	14.1999
12	var_12	182	13.5545
91	var_91	60	7.0360

In [0]:

```
duplicate_val_per_feat.tail(5)
```

Out[0]:

	Feature	Max duplicates	Value
61	var_61	6	-4.3454
136	var_136	6	16.8290
45	var_45	6	-2.8410
30	var_30	6	-0.0119
158	var_158	6	17.1384

Observation

All features have duplicate value. Hence, we can use duplicate value or not as feature.

Creating duplicate feature for each 200 feature (set to 1 if duplicate else 0)

In [0]:

```
##https://www.kaggle.com/super13579/lgbm-model-catboost?scriptVersionId=11574592
from tqdm import tqdm
for f in tqdm(features):
    data_cleaned[f+'dup'] = data_cleaned.duplicated(f,False).astype(int)
```

```
100%|██████████| 200/200 [00:04<00:00, 41.64it/s]
```

In [0]:

```
##https://www.kaggle.com/super13579/lgbm-model-catboost?scriptVersionId=11574592
from tqdm import tqdm
for f in tqdm(features):
    test[f+'dup'] = test.duplicated(f,False).astype(int)
```

```
100%|██████████| 200/200 [00:04<00:00, 40.57it/s]
```

In [0]:

```
data_cleaned.columns
```

Out[0]:

```
Index(['target', 'var_0', 'var_1', 'var_2', 'var_3', 'var_4', 'var_5', 'var_6',
       'var_7', 'var_8',
       ...
       'var_190dup', 'var_191dup', 'var_192dup', 'var_193dup', 'var_194dup',
       'var_195dup', 'var_196dup', 'var_197dup', 'var_198dup', 'var_199dup'],
      dtype='object', length=401)
```

In [0]:

```
test.columns
```

Out[0]:

```
Index(['var_0', 'var_1', 'var_2', 'var_3', 'var_4', 'var_5', 'var_6', 'var_7',
       'var_8', 'var_9',
       ...
       'var_190dup', 'var_191dup', 'var_192dup', 'var_193dup', 'var_194dup',
       'var_195dup', 'var_196dup', 'var_197dup', 'var_198dup', 'var_199dup'],
      dtype='object', length=400)
```

In [0]:

```
y=data_cleaned['target']
data_cle=data_cleaned.drop(['target'],axis=1)
```

In [0]:

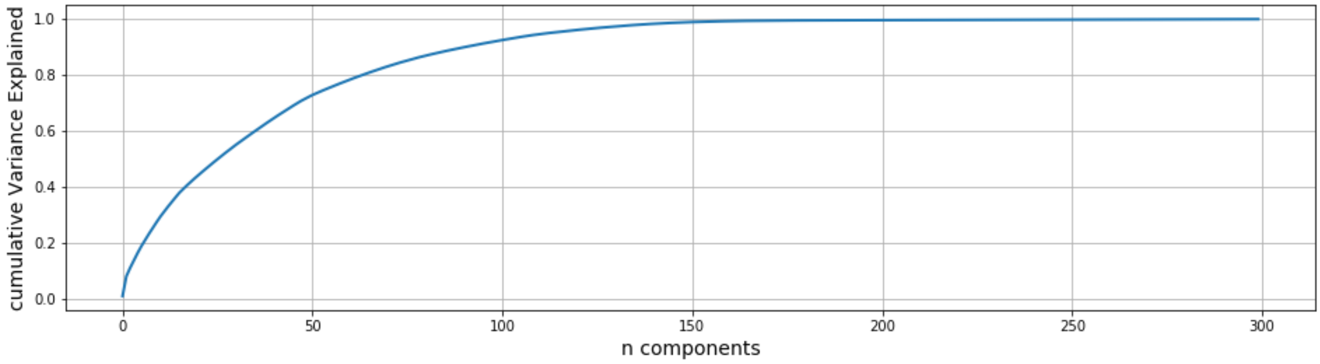
```
new_features = [i for i in data_cle.columns]
```

In [0]:

```
## https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD( n_components = 300, random_state=42 )

trsvd = svd.fit_transform( data_cle )

cumVarianceExplained = np.cumsum( svd.explained_variance_ratio_ )
```

In [0]:

```python
import matplotlib.pyplot as plt

plt.figure( figsize=(16, 4))
plt.plot( cumVarianceExplained, linewidth = 2 )
plt.grid()
plt.xlabel('n components',size=14)
plt.ylabel('cumulative Variance Explained',size=14)
plt.show()
```



In [0]:

```python
#https://stackoverflow.com/questions/44633571/how-can-i-get-the-feature-names-from-sklearn-truncatedsvd-object
best_features=[]
best_features = [new_features[i] for i in svd.components_[0].argsort()[::-1]]
```

Naive Bayes with top 125 features

In [0]:

```python
best_feat=[]
best_feat=best_features[0:125]
best_feat.append('target')
len(best_feat)
```

Out[0]:

```
126
```

In [0]:

```python
best_feat
```

Out[0]:

```
['var_120',
 'var_70',
 'var_160',
 'var_136',
 'var_102',
 'var_174',
 'var_172',
 'var_74',
 'var_73',
 'var_165',
 'var_77',
 'var_97',
 'var_109',
 'var_85',
 'var_194',
 'var_158',
 'var_107',
 'var_137',
 'var_21',
 'var_75',
 'var_150',
 'var_153',
```

```
'var_49',
'var_56',
'var_7',
'var_117',
'var_198',
'var_129',
'var_188',
'var_18',
'var_48',
'var_33',
'var_81',
'var_15',
'var_96',
'var_79',
'var_108',
'var_12',
'var_25',
'var_51',
'var_101',
'var_20',
'var_156',
'var_126',
'var_184',
'var_55',
'var_92',
'var_50',
'var_125',
'var_19',
'var_143',
'var_130',
'var_60',
'var_142',
'var_67',
'var_177',
'var_175',
'var_43',
'var_104',
'var_34',
'var_87',
'var_46',
'var_121',
'var_42',
'var_4',
'var_163',
'var_94',
'var_2',
'var_0',
'var_38',
'var_24',
'var_93',
'var_31',
'var_146',
'var_159',
'var_151',
'var_181',
'var_16',
'var_41',
'var_44',
'var_186',
'var_59',
'var_197',
'var_183',
'var_88',
'var_144',
'var_106',
'var_13',
'var_113',
'var_132',
'var_139',
'var_152',
'var_9',
'var_14',
'var_191',
'var_123',
'var_91',
'var_154',
'var_133',
'var_3',
```

```
 'var_5',
 'var_64',
 'var_111',
 'var_57',
 'var_76',
 'var_53',
 'var_37',
 'var_80',
 'var_66',
 'var_169',
 'var_161',
 'var_86',
 'var_28',
 'var_110',
 'var_6',
 'var_162',
 'var_78',
 'var_149',
 'var_29',
 'var_68',
 'var_168',
 'var_145',
 'var_22',
 'var_124',
 'var_105',
 'var_148',
 'target']
```

In [0]:

```
X_train,y_train,X_cv,y_cv=split_train_test(data_cleaned[best_feat])
```

```
Train data shape : (140085, 125)
CV data shape : (35022, 125)
```

In [0]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV

gnb = GaussianNB(priors = [0.5,0.5])
params_NB = {'var_smoothing': np.logspace(0,-11, num=100)}
best_param=grid_search(gnb,params_NB,5,X_train,y_train,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   10.9s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:   47.9s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  2.1min finished
```

```
best param : {'var_smoothing': 2.782559402207126e-10}
best score : 0.8447760890564531
```

In [0]:

```
best_param['var_smoothing']=2.782559402207126e-10
```

In [0]:

```
best_feat=best_feat[:-1]
```

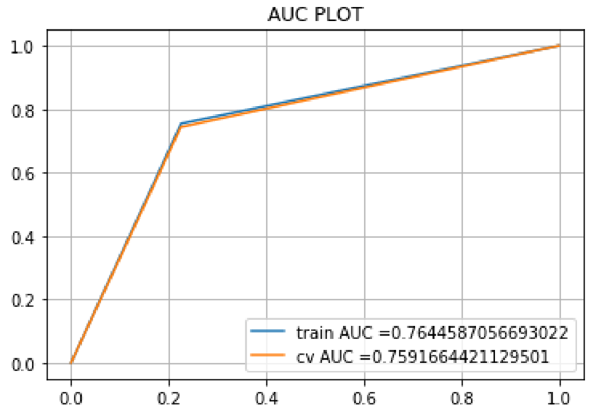
In [0]:

```
from sklearn.naive_bayes import GaussianNB
y_train_pred,y_cv_pred=baseline_model(best_param,X_train,y_train,X_cv,y_cv,test[best_feat],best_fe
at)
```

```
train auc score : 0.7644587056693022
cv auc score : 0.7591664421129501
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



Kaggle Score

1. Private Score: 0.76306
2. Public Score : 0.75937

NB with top 150 features

In [0]:

```
best_feat=[]
best_feat=best_features[0:150]
best_feat.append('target')
```

In [0]:

```
best_feat
```

Out[0]:

```
['var_120',
 'var_70',
 'var_160',
 'var_136',
 'var_102',
 'var_174',
 'var_172',
 'var_74',
 'var_73',
 'var_165',
 'var_77',
 'var_97',
 'var_109',
 'var_85',
 'var_194',
 'var_158',
 'var_107',
 'var_137',
 'var_21',
 'var_75',
 'var_150',
 'var_153',
 'var_49',
 'var_56',
 'var_7',
 'var_117',
 'var_198',
```

```
    'var_129',
    'var_188',
    'var_18',
    'var_48',
    'var_33',
    'var_81',
    'var_15',
    'var_96',
    'var_79',
    'var_108',
    'var_12',
    'var_25',
    'var_51',
    'var_101',
    'var_20',
    'var_156',
    'var_126',
    'var_184',
    'var_55',
    'var_92',
    'var_50',
    'var_125',
    'var_19',
    'var_143',
    'var_130',
    'var_60',
    'var_142',
    'var_67',
    'var_177',
    'var_175',
    'var_43',
    'var_104',
    'var_34',
    'var_87',
    'var_46',
    'var_121',
    'var_42',
    'var_4',
    'var_163',
    'var_94',
    'var_2',
    'var_0',
    'var_38',
    'var_24',
    'var_93',
    'var_31',
    'var_146',
    'var_159',
    'var_151',
    'var_181',
    'var_16',
    'var_41',
    'var_44',
    'var_186',
    'var_59',
    'var_197',
    'var_183',
    'var_88',
    'var_144',
    'var_106',
    'var_13',
    'var_113',
    'var_132',
    'var_139',
    'var_152',
    'var_9',
    'var_14',
    'var_191',
    'var_123',
    'var_91',
    'var_154',
    'var_133',
    'var_3',
    'var_64',
    'var_111',
    'var_57',
    'var_76',
```

```
 'var_53',
 'var_37',
 'var_80',
 'var_66',
 'var_169',
 'var_161',
 'var_86',
 'var_28',
 'var_110',
 'var_6',
 'var_162',
 'var_78',
 'var_149',
 'var_29',
 'var_68',
 'var_168',
 'var_145',
 'var_22',
 'var_124',
 'var_105',
 'var_148',
 'var_35',
 'var_89',
 'var_140',
 'var_193',
 'var_112',
 'var_190',
 'var_58',
 'var_114',
 'var_119',
 'var_23',
 'var_166',
 'var_141',
 'var_179',
 'var_116',
 'var_196',
 'var_115',
 'var_36',
 'var_192',
 'var_98',
 'var_103',
 'var_138',
 'var_122',
 'var_54',
 'var_83',
 'var_68dup',
 'target']
```

In [0]:

```
X_train,y_train,X_cv,y_cv=split_train_test(data_cleaned[best_feat])
```

```
Train data shape : (140085, 150)
CV data shape : (35022, 150)
```

In [0]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
gnb = GaussianNB(priors = [0.5,0.5])
params_NB = {'var_smoothing': np.logspace(0,-11, num=100)}
best_param=grid_search(gnb,params_NB,5,X_train,y_train,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   15.2s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:   60.0s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  2.2min
```

```
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  2.5min finished
```

```
best param : {'var_smoothing': 1.6681005372000592e-08}
best score : 0.8629465821720975
```

In [0]:

```
y_train_pred,y_cv_pred=baseline_model(best_param,X_train,y_train,X_cv,y_cv,test,best_feat)
```

```
train auc score : 0.7700368482269374
cv auc score : 0.771088919377401
```

Kaggle Score

1. Private Score: 0.77180
2. Public Score: 0.76913

SVM with top 150 features

In [0]:

```
from sklearn.linear_model import SGDClassifier
params = {'alpha': np.logspace(4,-9, num=100)}
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',max_iter=100,tol=1e-3,class_weight = 'balanced')
best_param=grid_search(sgd,params,5,X_train,y_train,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   23.4s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 16.4min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 19.7min finished
```

```
best param : {'alpha': 0.012328467394420634}
best score : 0.8234957330517834
```

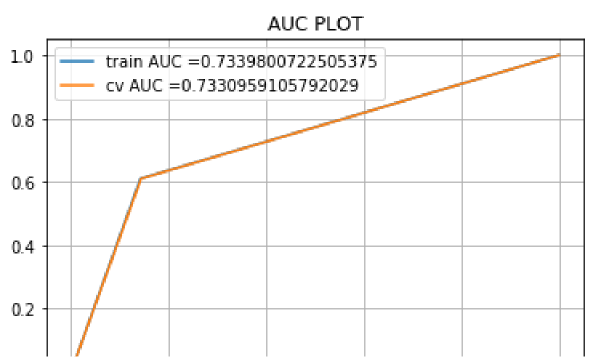
In [0]:

```
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=100,tol=1e-3,
alpha = best_param['alpha'])
sgd.fit(X_train ,y_train)
y_train_pred=sgd.predict(X_train)
y_cv_pred = sgd.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7339800722505375
cv auc score : 0.7330959105792029
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



In [0]:

```
best_feat=best_feat[:-1]
```

In [0]:

```
predictions=sgd.predict(test[best_feat])
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
test=test.reset_index()
submission = pd.DataFrame({"ID_code": test.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_svm.csv", index=False)
```

Kaggle Score

1. Private Score :0.73673
2. Public Score: 0.72987

In [0]:

```
!pip install bayesian-optimization
!pip install scikit-optimize
```

Lightgbm with top 150 features

In [0]:

```
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
import os
import numpy as np
import pandas as pd
import warnings

#tuning hyperparameters
from bayes_opt import BayesianOptimization
from skopt  import BayesSearchCV

#graph, plots
import matplotlib.pyplot as plt
import seaborn as sns

#building models
import lightgbm as lgb
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
import time
import sys

#metrics
from sklearn.metrics import roc_auc_score, roc_curve
warnings.simplefilter(action='ignore', category=FutureWarning)

def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=3, random_seed=6, output_pro
cess=False):
    # prepare data
    train_data = lgb.Dataset(data=X, label=y, free_raw_data=False)
    # parameters
    def lgb_eval(learning_rate,num_leaves, feature_fraction, bagging_fraction, max_depth, max_bin,
min_data_in_leaf,min_sum_hessian_in_leaf,subsample,lambda_l1,lambda_l2):
        params = {'application':'binary', 'metric':'auc'}
        params['learning_rate'] = max(min(learning_rate, 1), 0)
        params["num_leaves"] = int(round(num_leaves))
        params['feature_fraction'] = max(min(feature_fraction, 1), 0)
```

```
        params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
        params['max_depth'] = int(round(max_depth))
        params['max_bin'] = int(round(max_depth))
        params['min_data_in_leaf'] = int(round(min_data_in_leaf))
        params['min_sum_hessian_in_leaf'] = min_sum_hessian_in_leaf
        params['subsample'] = max(min(subsample, 1), 0)
        params['lambda_l1']=max(min(lambda_l1, 1), 0)
        params['lambda_l2']=max(min(lambda_l1, 1), 0)
        #evaluate cv for above paramters
        cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, ve
rbose_eval =200, metrics=['auc'])

        #return max mean for multiple folds
        return max(cv_result['auc-mean'])

    lgbBO = BayesianOptimization(lgb_eval, {'learning_rate': (0.01, 1.0),
                                            'num_leaves': (24, 80),
                                            'feature_fraction': (0.1, 0.9),
                                            'bagging_fraction': (0.8, 1),
                                            'max_depth': (5, 80),
                                            'max_bin':(20,150),
                                            'lambda_l1': (0.01,1),
                                            'lambda_l2': (0.01,1),
                                            'min_data_in_leaf': (20, 80),
                                            'min_sum_hessian_in_leaf':(0,100),
                                           'subsample': (0.01, 1.0)}, random_state=200)


    #n_iter: How many steps of bayesian optimization you want to perform. The more steps the more
likely to find a good maximum you are.
    #init_points: How many steps of random exploration you want to perform. Random exploration can
help by diversifying the exploration space.

    lgbBO.maximize(init_points=init_round, n_iter=opt_round)

    model_auc=[]
    for model in range(len( lgbBO.res)):
        model_auc.append(lgbBO.res[model]['target'])

    # return best parameters
    return lgbBO.res[pd.Series(model_auc).idxmax()]['target'],lgbBO.res[pd.Series(model_auc).idxmax
()]['params']

opt_params = bayes_parameter_opt_lgb(X_train, y_train, init_round=5, opt_round=15, n_folds=10, rand
om_seed=6)
```

```
|   iter    |  target   | baggin... | featur... | lambda_l1 | lambda_l2 | learni... |  max_bin  |
max_depth | min_da... | min_su... | num_le... | subsample |
-------------------------------------------------------------------------------------------------------
-------------------------------------------------------
|  1        |  0.8074   |  0.9895   |  0.2812   |  0.5985   |  0.434    |  0.7665   |  20.37    |
31.81     |  74.58    |  45.61    |  78.98    |  0.8687   |
|  2        |  0.8515   |  0.9972   |  0.8386   |  0.3107   |  0.8476   |  0.13     |  122.1    |
23.79     |  25.76    |  94.35    |  70.26    |  0.5231   |
|  3        |  0.8365   |  0.9747   |  0.5627   |  0.4556   |  0.6834   |  0.4252   |  103.3    |
50.65     |  26.33    |  96.6     |  66.49    |  0.6828   |
|  4        |  0.8476   |  0.8659   |  0.1212   |  0.8056   |  0.9731   |  0.2901   |  104.4    |
24.92     |  31.26    |  41.9     |  61.3     |  0.5222   |
|  5        |  0.8487   |  0.9709   |  0.2368   |  0.9784   |  0.3083   |  0.2401   |  122.9    |
71.3      |  79.24    |  1.606    |  35.45    |  0.6491   |
|  6        |  0.8483   |  0.9278   |  0.2247   |  0.3594   |  0.5246   |  0.1838   |  146.0    |
5.661     |  60.55    |  55.68    |  25.24    |  0.248    |
|  7        |  0.8378   |  0.9315   |  0.7056   |  0.8465   |  0.1393   |  0.2674   |  149.1    |
11.2      |  77.36    |  10.83    |  79.46    |  0.06753  |
|  8        |  0.8342   |  0.9233   |  0.4561   |  0.5436   |  0.1925   |  0.4222   |  148.9    |
67.49     |  20.49    |  2.499    |  36.04    |  0.1192   |
|  9        |  0.8116   |  0.9339   |  0.2826   |  0.4884   |  0.8421   |  0.8658   |  22.0     |
5.611     |  26.66    |  5.463    |  29.41    |  0.9683   |
|  10       |  0.8426   |  0.9723   |  0.1623   |  0.622    |  0.4529   |  0.03641  |  149.0    |
78.08     |  75.22    |  82.95    |  71.79    |  0.2054   |
|  11       |  0.8417   |  0.9889   |  0.4887   |  0.6475   |  0.1956   |  0.4353   |  97.39    |
7.497     |  78.19    |  96.93    |  33.22    |  0.03883  |
|  12       |  0.8333   |  0.9423   |  0.1805   |  0.7083   |  0.4165   |  0.5368   |  120.7    |
20.64     |  30.22    |  90.65    |  66.44    |  0.1767   |
|  13       |  0.8124   |  0.9265   |  0.4214   |  0.8321   |  0.2891   |  0.6592   |  20.07    |
79.91     |  22.89    |  0.5562   |  59.26    |  0.06673  |
```

```
|  14        |  0.8439    |  0.9182    |  0.4895    |  0.5189    |  0.997     |  0.431     |  22.62     |
4.03    |  79.93    |  53.38    |  25.13    |  0.996    |
|  15        |  0.8373    |  0.929     |  0.8396    |  0.7895    |  0.3234    |  0.4027    |  45.38     |
24.0      |  79.37    |  4.409    |  32.97    |  0.9975   |
|  16        |  0.8194    |  0.8414    |  0.3289    |  0.8593    |  0.9363    |  0.5016    |  115.2     |
15.54     |  21.73    |  0.5854   |  79.97    |  0.8202   |
|  17        |  0.8511    |  0.9184    |  0.1942    |  0.111     |  0.9885    |  0.3815    |  32.04     |
18.49     |  22.38    |  99.92    |  24.17    |  0.04808  |
|  18        |  0.8524    |  0.9969    |  0.4394    |  0.3309    |  0.4669    |  0.2428    |  148.4     |
65.29     |  25.87    |  92.54    |  24.68    |  0.6967   |
|  19        |  0.8082    |  0.8187    |  0.6425    |  0.1904    |  0.4984    |  0.6789    |  149.4     |
79.36     |  20.38    |  38.4     |  79.1     |  0.9087   |
|  20        |  0.7834    |  0.8085    |  0.7825    |  0.5511    |  0.6138    |  0.01678   |  20.49     |
77.28     |  20.8     |  85.03    |  24.46    |  0.2848   |
=================================================================================================
=====================================================
```

In [0]:

```
##https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
opt_params[1]["num_leaves"] = int(round(opt_params[1]["num_leaves"]))
opt_params[1]['max_depth'] = int(round(opt_params[1]['max_depth']))
opt_params[1]['min_data_in_leaf'] = int(round(opt_params[1]['min_data_in_leaf']))
opt_params[1]['max_bin'] = int(round(opt_params[1]['max_bin']))
opt_params[1]['objective']='binary'
opt_params[1]['metric']='auc'
opt_params[1]['is_unbalance']=True
opt_params[1]['boost_from_average']=False
opt_params=opt_params[1]
opt_params
```

Out[0]:

```
{'bagging_fraction': 0.996926204997424,
 'boost_from_average': False,
 'feature_fraction': 0.43942538998685643,
 'is_unbalance': True,
 'lambda_l1': 0.3309052418087291,
 'lambda_l2': 0.4669489881794176,
 'learning_rate': 0.24275827984778078,
 'max_bin': 148,
 'max_depth': 65,
 'metric': 'auc',
 'min_data_in_leaf': 26,
 'min_sum_hessian_in_leaf': 92.53585867436709,
 'num_leaves': 25,
 'objective': 'binary',
 'subsample': 0.6966790400075604}
```

In [0]:

```
import lightgbm as lgb
d_train = lgb.Dataset(X_train, label=y_train)
clf = lgb.train(opt_params, d_train)
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/lgbm_150_model.sav'
pickle.dump(clf, open(filename, 'wb'))
```

In [0]:

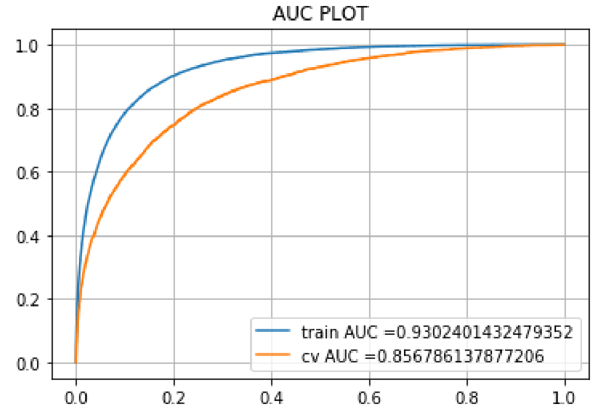
```
import pickle
filename = '/content/drive/My Drive/proj_1/lgbm_150_model.sav'
lm = pickle.load(open(filename, 'rb'))
y_train_pred = lm.predict(X_train)
y_cv_pred=lm.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.9302401432479352
cv auc score : 0.856786137877206
```

```
cv auc score : 0.856786137877206
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



In [0]:

```
predictions=lm.predict(test[best_feat])
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
test=test.reset_index()
submission = pd.DataFrame({"ID_code": test.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_lgb.csv", index=False)
```

Kaggle Score

1. Private Score: 0.84961
2. Public Score : 0.85237

Select k best features

In [0]:

```
y=data_cleaned['target']
data_cle=data_cleaned.drop(['target'],axis=1)
```

In [0]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
from sklearn.feature_selection import SelectKBest, f_classif
k_best = SelectKBest(f_classif, k=399)
k=k_best.fit(data_cle,y)
features = k.transform(data_cle)
```

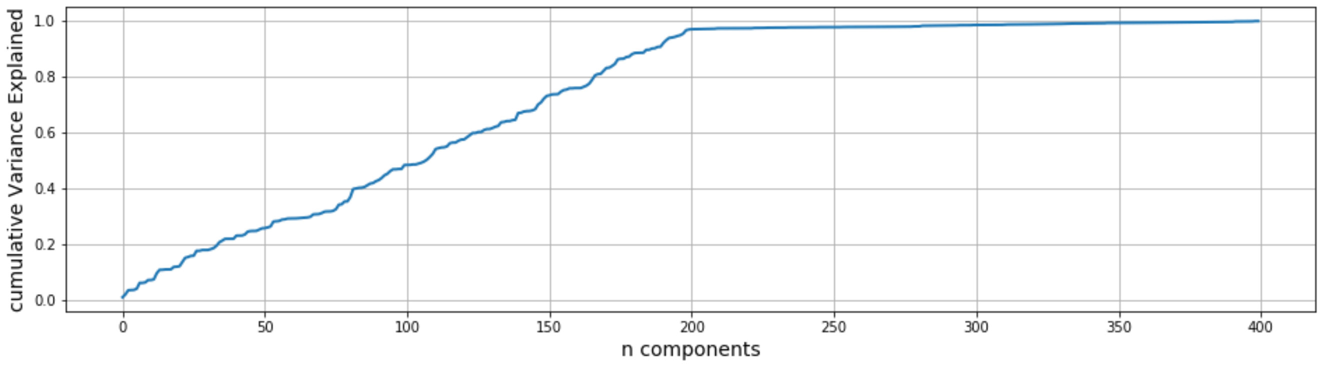
In [0]:

```
cumVarianceExplained = np.cumsum( k.scores_ )/np.sum(k.scores_)
```

In [0]:

```
import matplotlib.pyplot as plt

plt.figure( figsize=(16, 4))
plt.plot( cumVarianceExplained, linewidth = 2 )
plt.grid()
plt.xlabel('n components',size=14)
plt.ylabel('cumulative Variance Explained',size=14)
```

```
plt.show()
```



In [0]:

```
#https://stackoverflow.com/questions/39839112/the-easiest-way-for-getting-feature-names-after-running-selectkbest-in-scikit-le
selector = SelectKBest(f_classif,k=200)
selector.fit(data_cle, y)
cols = selector.get_support(indices=True)
data_kbest = data_cle.iloc[:,cols]
```

In [0]:

```
best_feat=[]
best_feat=data_kbest.columns
print(best_feat)
```

```
Index(['var_0', 'var_1', 'var_2', 'var_3', 'var_4', 'var_5', 'var_6', 'var_8',
       'var_9', 'var_11',
       ...
       'var_133dup', 'var_139dup', 'var_146dup', 'var_154dup', 'var_163dup',
       'var_177dup', 'var_179dup', 'var_191dup', 'var_195dup', 'var_198dup'],
      dtype='object', length=200)
```

NB with top 200 features

In [0]:

```
from sklearn.model_selection import train_test_split
df=data_kbest

X_train, X_cv, y_train, y_cv = train_test_split(df, y, test_size = 0.20, stratify=y)
print('Train data shape : '+str(X_train.shape))
print('CV data shape : '+str(X_cv.shape))
```

```
Train data shape : (140085, 200)
CV data shape : (35022, 200)
```

In [0]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV

gnb = GaussianNB(priors = [0.5,0.5])
params_NB = {'var_smoothing': np.logspace(0,-11, num=100)}
best_param=grid_search(gnb,params_NB,5,X_train,y_train,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   13.4s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:   52.8s
```

```
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  2.3min finished
```

```
best param : {'var_smoothing': 0.0001668100537200059}
best score : 0.877904655709755
```

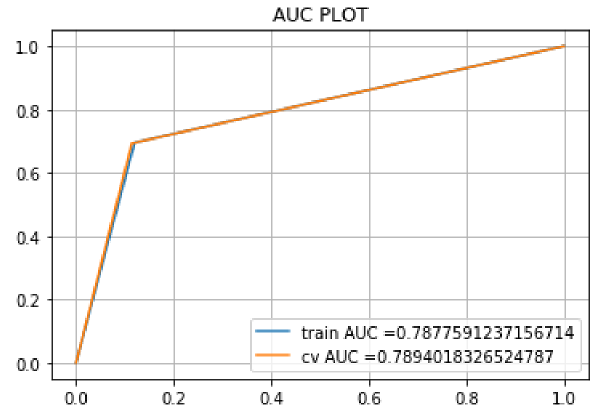
In [0]:

```
y_train_pred,y_cv_pred=baseline_model(best_param,X_train,y_train,X_cv,y_cv,test,best_feat)
```

```
train auc score : 0.7877591237156714
cv auc score : 0.7894018326524787
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



Kaggle Score

1. Private Score: 0.78190
2. Public Score : 0.78044

SVM with top 200 features

In [0]:

```
from sklearn.linear_model import SGDClassifier
params = {'alpha': np.logspace(4,-9, num=100)}
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',max_iter=100,tol=1e-3,class_weight = 'balanced')
best_param=grid_search(sgd,params,5,X_train,y_train,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   19.7s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 13.6min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 16.3min finished
```

```
best param : {'alpha': 0.012328467394420634}
best score : 0.8482980628677597
```

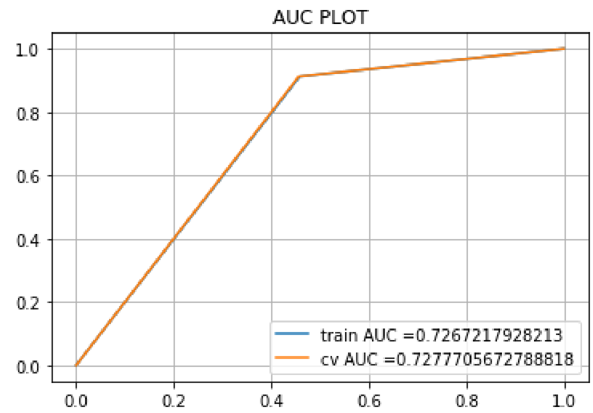
In [0]:

```
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=100,tol=1e-3,
alpha = best_param['alpha'])
sgd.fit(X_train ,y_train)
y_train_pred=sgd.predict(X_train)
y_cv_pred = sgd.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7267217928213
cv auc score : 0.7277705672788818
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



In [0]:

```
predictions=sgd.predict(test[best_feat])
```

In [0]:

```
test.head(2)
```

Out[0]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13
ID_code														
test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	8.8100	-2.0248	-4.3554	13.9696	0.3458
test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	5.9739	-1.3809	-0.3310	14.1129	2.5667

2 rows × 400 columns

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
test=test.reset_index()
submission = pd.DataFrame({"ID_code": test.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_svm.csv", index=False)
```

Kaggle Score

1. Private Score: 0.73442
2. Public Score : 0.73360

Lightgbm with top 200 features

In [0]:

```
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
import os
import numpy as np
import pandas as pd
```

```
import warnings

#tuning hyperparameters
from bayes_opt import BayesianOptimization
from skopt  import BayesSearchCV

#graph, plots
import matplotlib.pyplot as plt
import seaborn as sns

#building models
import lightgbm as lgb
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
import time
import sys

#metrics
from sklearn.metrics import roc_auc_score, roc_curve
warnings.simplefilter(action='ignore', category=FutureWarning)

def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=3, random_seed=6, output_pro
cess=False):
    # prepare data
    train_data = lgb.Dataset(data=X, label=y, free_raw_data=False)
    # parameters
    def lgb_eval(learning_rate,num_leaves, feature_fraction, bagging_fraction, max_depth, max_bin,
min_data_in_leaf,min_sum_hessian_in_leaf,subsample,lambda_l1,lambda_l2):
        params = {'application':'binary', 'metric':'auc'}
        params['learning_rate'] = max(min(learning_rate, 1), 0)
        params["num_leaves"] = int(round(num_leaves))
        params['feature_fraction'] = max(min(feature_fraction, 1), 0)
        params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
        params['max_depth'] = int(round(max_depth))
        params['max_bin'] = int(round(max_depth))
        params['min_data_in_leaf'] = int(round(min_data_in_leaf))
        params['min_sum_hessian_in_leaf'] = min_sum_hessian_in_leaf
        params['subsample'] = max(min(subsample, 1), 0)
        params['lambda_l1']=max(min(lambda_l1, 1), 0)
        params['lambda_l2']=max(min(lambda_l1, 1), 0)
        #evaluate cv for above paramters
        cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, ve
rbose_eval =200, metrics=['auc'])

        #return max mean for multiple folds
        return max(cv_result['auc-mean'])

    lgbBO = BayesianOptimization(lgb_eval, {'learning_rate': (0.01, 1.0),
                                            'num_leaves': (24, 80),
                                            'feature_fraction': (0.1, 0.9),
                                            'bagging_fraction': (0.8, 1),
                                            'max_depth': (5, 80),
                                            'max_bin':(20,150),
                                            'lambda_l1': (0.01,1),
                                            'lambda_l2': (0.01,1),
                                            'min_data_in_leaf': (20, 80),
                                            'min_sum_hessian_in_leaf':(0,100),
                                           'subsample': (0.01, 1.0)}, random_state=200)


    #n_iter: How many steps of bayesian optimization you want to perform. The more steps the more
likely to find a good maximum you are.
    #init_points: How many steps of random exploration you want to perform. Random exploration can
help by diversifying the exploration space.

    lgbBO.maximize(init_points=init_round, n_iter=opt_round)

    model_auc=[]
    for model in range(len( lgbBO.res)):
        model_auc.append(lgbBO.res[model]['target'])

    # return best parameters
    return lgbBO.res[pd.Series(model_auc).idxmax()]['target'],lgbBO.res[pd.Series(model_auc).idxmax
()]['params']

opt_params = bayes_parameter_opt_lgb(X_train, y_train, init_round=5, opt_round=15, n_folds=10, rand
om_seed=6)
```

```
um_seed=0)
```

```
|   iter    |  target   | baggin... | featur... | lambda_l1 | lambda_l2 | learni... |  max_bin  |
max_depth | min_da... | min_su... | num_le... | subsample |
-------------------------------------------------------------------------------------------------
-----------------------------------------------------
|  1        |  0.8333   |  0.9895   |  0.2812   |  0.5985   |  0.434    |  0.7665   |  20.37    |
31.81     |  74.58    |  45.61    |  78.98    |  0.8687   |
|  2        |  0.8741   |  0.9972   |  0.8386   |  0.3107   |  0.8476   |  0.13     |  122.1    |
23.79     |  25.76    |  94.35    |  70.26    |  0.5231   |
|  3        |  0.8644   |  0.9747   |  0.5627   |  0.4556   |  0.6834   |  0.4252   |  103.3    |
50.65     |  26.33    |  96.6     |  66.49    |  0.6828   |
|  4        |  0.8719   |  0.8659   |  0.1212   |  0.8056   |  0.9731   |  0.2901   |  104.4    |
24.92     |  31.26    |  41.9     |  61.3     |  0.5222   |
|  5        |  0.8712   |  0.9709   |  0.2368   |  0.9784   |  0.3083   |  0.2401   |  122.9    |
71.3      |  79.24    |  1.606    |  35.45    |  0.6491   |
|  6        |  0.8139   |  0.9936   |  0.122    |  0.2994   |  0.5044   |  0.9007   |  149.4    |
54.77     |  31.29    |  4.874    |  77.44    |  0.5757   |
|  7        |  0.8606   |  0.8591   |  0.6514   |  0.03244  |  0.9309   |  0.6671   |  23.85    |
10.26     |  27.32    |  95.29    |  24.42    |  0.4107   |
|  8        |  0.8678   |  0.8524   |  0.6708   |  0.4791   |  0.751    |  0.4387   |  137.1    |
16.0      |  77.57    |  99.44    |  32.38    |  0.5406   |
|  9        |  0.87     |  0.9656   |  0.8767   |  0.808    |  0.5999   |  0.2059   |  38.54    |
4.21     |  26.59    |  0.7953   |  27.05    |  0.3759   |
|  10       |  0.8542   |  0.8984   |  0.842    |  0.4462   |  0.9857   |  0.5203   |  64.44    |
7.42      |  62.42    |  5.012    |  24.59    |  0.3092   |
|  11       |  0.8729   |  0.9989   |  0.8048   |  0.3322   |  0.1053   |  0.2206   |  147.8    |
66.45     |  20.24    |  71.56    |  25.51    |  0.7651   |
|  12       |  0.8438   |  0.8373   |  0.5995   |  0.6737   |  0.2634   |  0.9813   |  147.1    |
68.29     |  24.69    |  70.02    |  25.27    |  0.9589   |
|  13       |  0.754    |  0.9924   |  0.9      |  0.01     |  0.03019  |  0.01     |  150.0    |
.0       |  80.0     |  0.0      |  24.0     |  0.01     |
|  14       |  0.7703   |  1.0      |  0.9      |  0.01     |  0.01     |  0.01     |  20.0     |
0.0      |  80.0     |  100.0    |  24.0     |  0.01     |
|  15       |  0.8587   |  0.8316   |  0.26     |  0.2899   |  0.5751   |  0.7256   |  147.6    |
5.553     |  26.87    |  99.84    |  24.6     |  0.776    |
|  16       |  0.8694   |  0.9055   |  0.8927   |  0.4685   |  0.1738   |  0.3514   |  46.71    |
5.505     |  79.56    |  98.57    |  63.14    |  0.06384  |
|  17       |  0.8664   |  0.9929   |  0.3799   |  0.2199   |  0.2938   |  0.116    |  46.05    |
11.34     |  20.75    |  1.696    |  76.38    |  0.6016   |
|  18       |  0.8485   |  0.9457   |  0.6857   |  0.8245   |  0.1359   |  0.4554   |  55.21    |
78.24     |  76.75    |  4.298    |  77.3     |  0.531    |
|  19       |  0.7517   |  1.0      |  0.841    |  0.01     |  0.01809  |  0.01     |  150.0    |
.0       |  80.0     |  97.78    |  80.0     |  0.02978  |
|  20       |  0.8645   |  0.9822   |  0.264    |  0.6299   |  0.674    |  0.4735   |  128.7    |
6.26     |  21.05    |  0.9262   |  25.25    |  0.8316   |
=================================================================================================
=====================================================
```

In [0]:

```python
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
opt_params[1]["num_leaves"] = int(round(opt_params[1]["num_leaves"]))
opt_params[1]['max_depth'] = int(round(opt_params[1]['max_depth']))
opt_params[1]['min_data_in_leaf'] = int(round(opt_params[1]['min_data_in_leaf']))
opt_params[1]['max_bin'] = int(round(opt_params[1]['max_bin']))
opt_params[1]['objective']='binary'
opt_params[1]['metric']='auc'
opt_params[1]['is_unbalance']=True
opt_params[1]['boost_from_average']=False
opt_params=opt_params[1]
opt_params
```

Out[0]:

```
{'bagging_fraction': 0.9972055022804187,
 'boost_from_average': False,
 'feature_fraction': 0.8386133653331375,
 'is_unbalance': True,
 'lambda_l1': 0.31065638082201175,
 'lambda_l2': 0.84764245541438,
 'learning_rate': 0.1300097487520304,
 'max_bin': 122,
 'max_depth': 24,
 'metric': 'auc',
 'min_data_in_leaf': 26,
```

```
 'min_sum_hessian_in_leaf': 94.34910369995325,
 'num_leaves': 70,
 'objective': 'binary',
 'subsample': 0.5231418824080631}
```

In [0]:

```
import lightgbm as lgb
d_train = lgb.Dataset(X_train, label=y_train)
clf = lgb.train(opt_params, d_train)
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/lgbm_200_imp_model.sav'
pickle.dump(clf, open(filename, 'wb'))
```

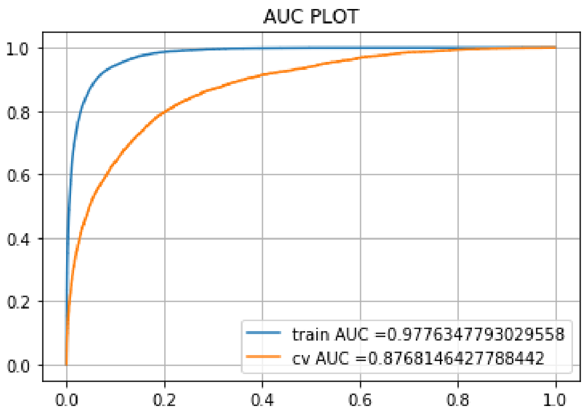
In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/lgbm_200_imp_model.sav'
lm = pickle.load(open(filename, 'rb'))
y_train_pred = lm.predict(X_train)
y_cv_pred=lm.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.9776347793029558
cv auc score : 0.8768146427788442
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=lm.predict(test[best_feat])
test=test.reset_index()
submission = pd.DataFrame({"ID_code": test.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_lgb.csv", index=False)
```

Kaggle Score

1. Private Score: 0.87081
2. Public Score : 0.87319

Naive Bayes 400 feature (200 original and 200 duplicate flag)

In [0]:

In [0]:

```
X_train,y_train,X_cv,y_cv=split_train_test(data_cleaned)
```

```
Train data shape : (140085, 400)
CV data shape : (35022, 400)
```

In [0]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV

gnb = GaussianNB(priors = [0.5,0.5])
params_NB = {'var_smoothing': np.logspace(0,-11, num=50)}
best_param=grid_search(gnb,params_NB,5,X_train,y_train,8)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

```
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:   24.9s
[Parallel(n_jobs=8)]: Done 184 tasks      | elapsed:  1.7min
[Parallel(n_jobs=8)]: Done 250 out of 250 | elapsed:  2.2min finished
```

```
best param : {'var_smoothing': 0.0002559547922699536}
best score : 0.877031186626097
```

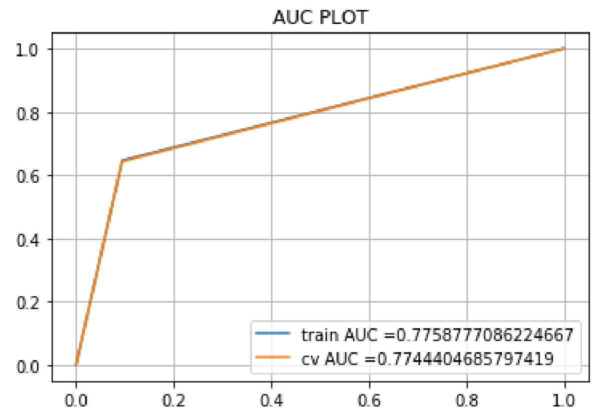
In [0]:

```
best_feat=test.columns
y_train_pred,y_cv_pred=baseline_model(best_param,X_train,y_train,X_cv,y_cv,test,best_feat)
```

```
train auc score : 0.7758777086224667
cv auc score : 0.7744404685797419
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



Kaggle Score

1. Private Score: 0.75314
2. Public Score : 0.75148

SVM with 400 feature (200 original and 200 duplicate flag)

In [0]:

```
from sklearn.linear_model import SGDClassifier
params = {'alpha': np.logspace(4,-9, num=100)}
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',max_iter=100,tol=1e-3,class_weight = 'balanced')
```

```
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',max_iter=100,tol=1e-3,class_weight = 'balanced')
best_param=grid_search(sgd,params,5,X_train,y_train,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   24.8s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed: 13.3min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 16.3min finished
```

```
best param : {'alpha': 0.030538555088334123}
best score : 0.8481811182275231
```

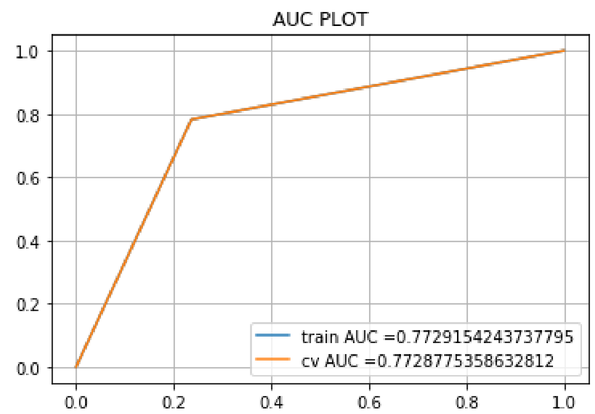
In [0]:

```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=100,tol=1e-3,
alpha = 0.030538555088334123)
sgd.fit(X_train ,y_train)
y_train_pred=sgd.predict(X_train)
y_cv_pred = sgd.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7729154243737795
cv auc score : 0.7728775358632812
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=sgd.predict(test)
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_sgd.csv", index=False)
```

Kaggle Score

1. Private Score: 0.76438
2. Public Score : 0.76639

Xgboost with 400 features (200 original and 200 duplicate flag)

In [0]:

```
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier
```

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,7,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,cv=5,scoring = 'roc_auc',n
_jobs=-1)
random_cfl.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 57.2min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 62.6min
```

In [0]:

```
random_cfl.best_estimator_
```

Out[0]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.2,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=1000, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
```

In [0]:

```
from xgboost import XGBClassifier
xgb = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.2,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=1000, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
xgb.fit(X_train, y_train)
y_train_pred = xgb.predict(X_train)
y_cv_pred=xgb.predict(X_cv)
```

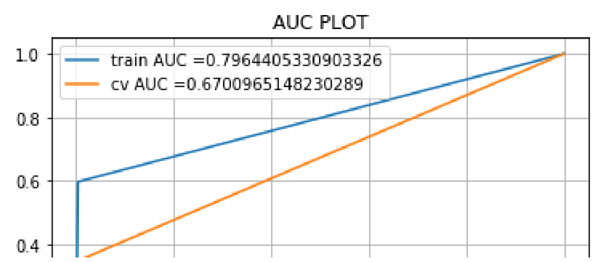
In [0]:

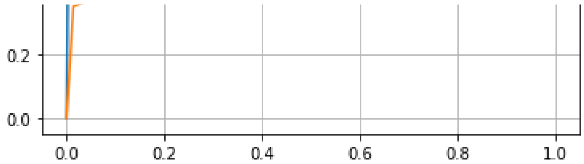
```
from sklearn.metrics import roc_auc_score
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7964405330903326
cv auc score : 0.6700965148230289
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```





In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=xgb.predict(test)
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_xgb.csv", index=False)
```

Kaggle Score

1. Private Score: 0.71413
2. Public Score : 0.70979

lightgbm with 400 features (200 original and 200 duplicate flag)

In [0]:

```
#basic tools
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
import os
import numpy as np
import pandas as pd
import warnings

#tuning hyperparameters
from bayes_opt import BayesianOptimization
from skopt  import BayesSearchCV

#graph, plots
import matplotlib.pyplot as plt
import seaborn as sns

#building models
import lightgbm as lgb
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
import time
import sys

#metrics
from sklearn.metrics import roc_auc_score, roc_curve
import shap
warnings.simplefilter(action='ignore', category=FutureWarning)

def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=3, random_seed=6, output_pro
cess=False):
    # prepare data
    train_data = lgb.Dataset(data=X, label=y, free_raw_data=False)
    # parameters
    def lgb_eval(learning_rate,num_leaves, feature_fraction, bagging_fraction, max_depth, max_bin,
min_data_in_leaf,min_sum_hessian_in_leaf,subsample,lambda_l1,lambda_l2):
        params = {'application':'binary', 'metric':'auc'}
        params['learning_rate'] = max(min(learning_rate, 1), 0)
        params["num_leaves"] = int(round(num_leaves))
        params['feature_fraction'] = max(min(feature_fraction, 1), 0)
        params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
        params['max_depth'] = int(round(max_depth))
        params['max_bin'] = int(round(max_depth))
        params['min_data_in_leaf'] = int(round(min_data_in_leaf))
        params['min_sum_hessian_in_leaf'] = min_sum_hessian_in_leaf
        params['subsample'] = max(min(subsample, 1), 0)
        params['lambda_l1']=max(min(lambda_l1, 1), 0)
        params['lambda_l2']=max(min(lambda_l1, 1), 0)
        #evaluate cv for above paramters
```

```
        cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, ve
rbose_eval =200, metrics=['auc'])

        #return max mean for multiple folds
        return max(cv_result['auc-mean'])

    lgbBO = BayesianOptimization(lgb_eval, {'learning_rate': (0.01, 1.0),
                                            'num_leaves': (24, 80),
                                            'feature_fraction': (0.1, 0.9),
                                            'bagging_fraction': (0.8, 1),
                                            'max_depth': (5, 80),
                                            'max_bin':(20,150),
                                            'lambda_l1': (0.01,1),
                                            'lambda_l2': (0.01,1),
                                            'min_data_in_leaf': (20, 80),
                                            'min_sum_hessian_in_leaf':(0,100),
                                           'subsample': (0.01, 1.0)}, random_state=200)


    #n_iter: How many steps of bayesian optimization you want to perform. The more steps the more
likely to find a good maximum you are.
    #init_points: How many steps of random exploration you want to perform. Random exploration can
help by diversifying the exploration space.

    lgbBO.maximize(init_points=init_round, n_iter=opt_round)

    model_auc=[]
    for model in range(len( lgbBO.res)):
        model_auc.append(lgbBO.res[model]['target'])

    # return best parameters
    return lgbBO.res[pd.Series(model_auc).idxmax()]['target'],lgbBO.res[pd.Series(model_auc).idxmax
()]['params']

opt_params = bayes_parameter_opt_lgb(X_train, y_train, init_round=5, opt_round=15, n_folds=10, rand
om_seed=6)
```

```
|   iter    |  target   | baggin... | featur... | lambda_l1 | lambda_l2 | learni... |  max_bin  |
max_depth | min_da... | min_su... | num_le... | subsample |
----------------------------------------------------------------------------------------------------
-------------------------------------------------------
|  1        |  0.8329   |  0.9895   |  0.2812   |  0.5985   |  0.434    |  0.7665   |  20.37    |
31.81     |  74.58    |  45.61    |  78.98    |  0.8687   |
|  2        |  0.8756   |  0.9972   |  0.8386   |  0.3107   |  0.8476   |  0.13     |  122.1    |
23.79     |  25.76    |  94.35    |  70.26    |  0.5231   |
|  3        |  0.8608   |  0.9747   |  0.5627   |  0.4556   |  0.6834   |  0.4252   |  103.3    |
50.65     |  26.33    |  96.6     |  66.49    |  0.6828   |
|  4        |  0.8714   |  0.8659   |  0.1212   |  0.8056   |  0.9731   |  0.2901   |  104.4    |
24.92     |  31.26    |  41.9     |  61.3     |  0.5222   |
|  5        |  0.8708   |  0.9709   |  0.2368   |  0.9784   |  0.3083   |  0.2401   |  122.9    |
71.3      |  79.24    |  1.606    |  35.45    |  0.6491   |
|  6        |  0.8592   |  0.9258   |  0.1191   |  0.5631   |  0.7059   |  0.7564   |  148.5    |
6.282     |  48.48    |  89.67    |  25.91    |  0.9656   |
|  7        |  0.8359   |  0.9952   |  0.6816   |  0.6452   |  0.654    |  0.5627   |  149.1    |
10.89     |  58.26    |  3.856    |  78.32    |  0.767    |
|  8        |  0.8632   |  0.8732   |  0.3013   |  0.118    |  0.5893   |  0.4221   |  30.86    |
79.42     |  22.04    |  0.3505   |  26.29    |  0.6956   |
|  9        |  0.8608   |  0.9038   |  0.4345   |  0.8126   |  0.78     |  0.6343   |  31.71    |
5.386     |  20.04    |  91.68    |  27.85    |  0.5834   |
|  10       |  0.8467   |  0.9731   |  0.181    |  0.7606   |  0.1353   |  0.04324  |  149.5    |
73.75     |  20.35    |  12.55    |  31.09    |  0.103    |
|  11       |  0.8636   |  0.8886   |  0.3217   |  0.7562   |  0.5767   |  0.2814   |  49.07    |
5.114     |  73.44    |  0.5567   |  30.42    |  0.04354  |
|  12       |  0.8615   |  0.8416   |  0.7882   |  0.1263   |  0.5645   |  0.126    |  34.55    |
7.852     |  22.37    |  1.795    |  79.35    |  0.7763   |
|  13       |  0.8664   |  0.957    |  0.3727   |  0.7303   |  0.03326  |  0.09905  |  132.4    |
9.223     |  78.98    |  95.42    |  79.05    |  0.5873   |
|  14       |  0.8586   |  0.8383   |  0.8977   |  0.1237   |  0.4598   |  0.5897   |  85.58    |
6.312     |  20.16    |  99.11    |  68.35    |  0.1195   |
|  15       |  0.8271   |  0.8662   |  0.2732   |  0.7339   |  0.561    |  0.9743   |  148.8    |
78.32     |  73.76    |  78.72    |  73.88    |  0.2395   |
|  16       |  0.8651   |  0.9671   |  0.263    |  0.8584   |  0.05981  |  0.1124   |  20.8     |
74.41     |  79.36    |  84.46    |  24.89    |  0.03264  |
|  17       |  0.8731   |  0.837    |  0.6122   |  0.9409   |  0.7334   |  0.1952   |  63.4     |
24.26     |  77.6     |  97.34    |  24.33    |  0.8901   |
|  18       |  0.8407   |  0.8057   |  0.5256   |  0.3017   |  0.4155   |  0.8215   |  44.73    |
71.29     |  79.49    |  0.8168   |  25.61    |  0.09577  |
```

```
|  19      |  0.857   |  0.9461  |  0.2258  |  0.1854  |  0.9505  |  0.5561  |  85.72   |
7.154    |  22.57   |  3.06     |  24.83   |  0.9984  |
|  20      |  0.8509  |  0.8097  |  0.1444  |  0.9551  |  0.6596  |  0.9506  |  22.41   |
76.64    |  21.97   |  97.74    |  24.01   |  0.4584  |
====================================================================================================
=======================================================
```

In [0]:

```
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
opt_params[1]["num_leaves"] = int(round(opt_params[1]["num_leaves"]))
opt_params[1]['max_depth'] = int(round(opt_params[1]['max_depth']))
opt_params[1]['min_data_in_leaf'] = int(round(opt_params[1]['min_data_in_leaf']))
opt_params[1]['max_bin'] = int(round(opt_params[1]['max_bin']))
opt_params[1]['objective']='binary'
opt_params[1]['metric']='auc'
opt_params[1]['is_unbalance']=True
opt_params[1]['boost_from_average']=False
opt_params=opt_params[1]
opt_params
```

Out[0]:

```
{'bagging_fraction': 0.9972055022804187,
 'boost_from_average': False,
 'feature_fraction': 0.8386133653331375,
 'is_unbalance': True,
 'lambda_l1': 0.31065638082201175,
 'lambda_l2': 0.84764245541438,
 'learning_rate': 0.1300097487520304,
 'max_bin': 122,
 'max_depth': 24,
 'metric': 'auc',
 'min_data_in_leaf': 26,
 'min_sum_hessian_in_leaf': 94.34910369995325,
 'num_leaves': 70,
 'objective': 'binary',
 'subsample': 0.5231418824080631}
```

In [0]:

```
import lightgbm as lgb
d_train = lgb.Dataset(X_train, label=y_train)
clf = lgb.train(opt_params, d_train)
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/lgbm_400_model.sav'
pickle.dump(clf, open(filename, 'wb'))
```

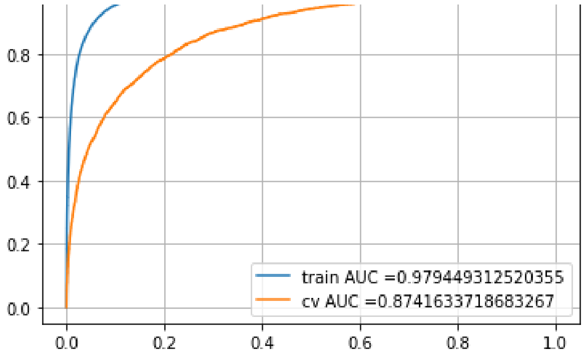
In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/lgbm_400_model.sav'
lm = pickle.load(open(filename, 'rb'))
y_train_pred = lm.predict(X_train)
y_cv_pred=lm.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.979449312520355
cv auc score : 0.8741633718683267
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=lm.predict(test)
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_lgb.csv", index=False)
```

Kaggle Score

1. Private Score: 0.86945
2. Public Score : 0.87246

Featurization 2

Augmentation

In [0]:

```
#https://www.kaggle.com/jiweiliu/lgb-2-leaves-augment
def augment(x,y,t=2):
    xs,xn = [],[]
    for i in range(t):
        mask = y>0
        x1 = x[mask].copy()
        ids = np.arange(x1.shape[0])
        for c in range(x1.shape[1]):
            np.random.shuffle(ids)
            x1[:,c] = x1[ids][:,c]
        xs.append(x1)

    for i in range(t//2):
        mask = y==0
        x1 = x[mask].copy()
        ids = np.arange(x1.shape[0])
        for c in range(x1.shape[1]):
            np.random.shuffle(ids)
            x1[:,c] = x1[ids][:,c]
        xn.append(x1)


    xs = np.vstack(xs)
    xn = np.vstack(xn)
    ys = np.ones(xs.shape[0])
    yn = np.zeros(xn.shape[0])
    x = np.vstack([x,xs,xn])
    y = np.concatenate([y,ys,yn])
    return x,y
```

Min,Max,Median,Mean,standard-deviation and kurtosis features

In [0]:

```
#https://www.kaggle.com/gpreda/santander-eda-and-prediction
```

```
%%time
idx = features = data.columns.values[2:202]
for df in [data]:
    df['sum'] = df[idx].sum(axis=1)
    df['min'] = df[idx].min(axis=1)
    df['max'] = df[idx].max(axis=1)
    df['mean'] = df[idx].mean(axis=1)
    df['std'] = df[idx].std(axis=1)
    df['skew'] = df[idx].skew(axis=1)
    df['kurt'] = df[idx].kurtosis(axis=1)
    df['med'] = df[idx].median(axis=1)
```

```
CPU times: user 3.3 s, sys: 27.9 ms, total: 3.33 s
Wall time: 3.34 s
```

In [0]:

```
#https://www.kaggle.com/gpreda/santander-eda-and-prediction
%%time
test = pd.read_csv('/content/drive/My Drive/proj_1/test.csv', index_col=0)
idx = features = test.columns.values[2:202]
for df in [test]:
    df['sum'] = df[idx].sum(axis=1)
    df['min'] = df[idx].min(axis=1)
    df['max'] = df[idx].max(axis=1)
    df['mean'] = df[idx].mean(axis=1)
    df['std'] = df[idx].std(axis=1)
    df['skew'] = df[idx].skew(axis=1)
    df['kurt'] = df[idx].kurtosis(axis=1)
    df['med'] = df[idx].median(axis=1)
```

```
CPU times: user 10.6 s, sys: 196 ms, total: 10.8 s
Wall time: 11.1 s
```

In [0]:

```
train=data
```

In [0]:

```
print('train data shape'+str(train.shape))
print('test data shape'+str(test.shape))
```

```
train data shape(200000, 209)
test data shape(200000, 208)
```

In [0]:

```
from tqdm import tqdm
col=[]
for i in tqdm(range(200)):
    m='var_'+str(i)
    col.append(m)
train=train.drop(col,axis=1)
test=test.drop(col,axis=1)
```

```
100%|██████████| 200/200 [00:00<00:00, 335812.97it/s]
```

In [0]:

```
y=train['target']
train=train.drop(['target'],axis=1)
```

In [0]:

```
print('train data shape'+str(train.shape))
print('test data shape'+str(test.shape))
```

```
train data shape(200000, 8)
test data shape(200000, 8)
```

Naive Bayes with only 8 features(mean,median etc.)

In [0]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV

gnb = GaussianNB(priors = [0.5,0.5])
params_NB = {'var_smoothing': np.logspace(0,-11, num=100)}
best_param=grid_search(gnb,params_NB,5,train,y,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    6.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  1.0min
```

```
best param : {'var_smoothing': 3.5938136638046257e-09}
best score : 0.5286861796233061
```

```
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  1.1min finished
```

In [0]:

```
from sklearn.model_selection import train_test_split
X_train, X_cv, y_train, y_cv = train_test_split(train, y, test_size = 0.20, stratify=y)
print('Train data shape : '+str(X_train.shape))
print('CV data shape : '+str(X_cv.shape))
```

```
Train data shape : (160000, 8)
CV data shape : (40000, 8)
```

In [0]:

```
best_feat=test.columns
y_train_pred,y_cv_pred=baseline_model(best_param,X_train,y_train,X_cv,y_cv,test,best_feat)
```

```
train auc score : 0.5199837754199206
cv auc score : 0.5215426480714824
```

Kaggle Score

1. Private Score: 0.50000
2. Public Score : 0.50000

lightgbm with 8 features only(mean,median,skew,min,max,std,sum and skew)

In [0]:

```
#basic tools
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
import os
import numpy as np
import pandas as pd
import warnings

#tuning hyperparameters
from bayes_opt import BayesianOptimization
from skopt  import BayesSearchCV
```

```
#graph, plots
import matplotlib.pyplot as plt
import seaborn as sns

#building models
import lightgbm as lgb
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
import time
import sys

#metrics
from sklearn.metrics import roc_auc_score, roc_curve
warnings.simplefilter(action='ignore', category=FutureWarning)

def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=3, random_seed=6, output_pro
cess=False):
    # prepare data
    train_data = lgb.Dataset(data=X, label=y, free_raw_data=False)
    # parameters
    def lgb_eval(learning_rate,num_leaves, feature_fraction, bagging_fraction, max_depth, max_bin,
min_data_in_leaf,min_sum_hessian_in_leaf,subsample,lambda_l1,lambda_l2):
        params = {'application':'binary', 'metric':'auc'}
        params['learning_rate'] = max(min(learning_rate, 1), 0)
        params["num_leaves"] = int(round(num_leaves))
        params['feature_fraction'] = max(min(feature_fraction, 1), 0)
        params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
        params['max_depth'] = int(round(max_depth))
        params['max_bin'] = int(round(max_depth))
        params['min_data_in_leaf'] = int(round(min_data_in_leaf))
        params['min_sum_hessian_in_leaf'] = min_sum_hessian_in_leaf
        params['subsample'] = max(min(subsample, 1), 0)
        params['lambda_l1']=max(min(lambda_l1, 1), 0)
        params['lambda_l2']=max(min(lambda_l1, 1), 0)
        #evaluate cv for above paramters
        cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, ve
rbose_eval =200, metrics=['auc'])

        #return max mean for multiple folds
        return max(cv_result['auc-mean'])

    lgbBO = BayesianOptimization(lgb_eval, {'learning_rate': (0.01, 1.0),
                                             'num_leaves': (24, 80),
                                             'feature_fraction': (0.1, 0.9),
                                             'bagging_fraction': (0.8, 1),
                                             'max_depth': (5, 80),
                                             'max_bin':(20,150),
                                             'lambda_l1': (0.01,1),
                                             'lambda_l2': (0.01,1),
                                             'min_data_in_leaf': (20, 80),
                                             'min_sum_hessian_in_leaf':(0,100),
                                            'subsample': (0.01, 1.0)}, random_state=200)


    #n_iter: How many steps of bayesian optimization you want to perform. The more steps the more
likely to find a good maximum you are.
    #init_points: How many steps of random exploration you want to perform. Random exploration can
help by diversifying the exploration space.

    lgbBO.maximize(init_points=init_round, n_iter=opt_round)

    model_auc=[]
    for model in range(len( lgbBO.res)):
        model_auc.append(lgbBO.res[model]['target'])

    # return best parameters
    return lgbBO.res[pd.Series(model_auc).idxmax()]['target'],lgbBO.res[pd.Series(model_auc).idxmax
()]['params']

opt_params = bayes_parameter_opt_lgb(train, y, init_round=5, opt_round=15, n_folds=11, random_seed=
6)
```

```
|   iter    |  target   | baggin... | featur... | lambda_l1 | lambda_l2 | learni... |  max_bin  |
max_depth | min_da... | min_su... | num_le... | subsample |
-------------------------------------------------------------------------------------------------
-------------------------------------------------------
|  1        |  0.5448   |  0.9895   |  0.2812   |  0.5985   |  0.434    |  0.7665   |  20.37    |
```

```
|  1        |  0.5448   |  0.9695   |  0.2812   |  0.5985   |  0.454    |  0.7665   |  20.57    |
31.81     |  74.58    |  45.61    |  78.98    |  0.8687   |
|  2        |  0.5658   |  0.9972   |  0.8386   |  0.3107   |  0.8476   |  0.13     |  122.1    |
23.79     |  25.76    |  94.35    |  70.26    |  0.5231   |
|  3        |  0.5565   |  0.9747   |  0.5627   |  0.4556   |  0.6834   |  0.4252   |  103.3    |
50.65     |  26.33    |  96.6     |  66.49    |  0.6828   |
|  4        |  0.561    |  0.8659   |  0.1212   |  0.8056   |  0.9731   |  0.2901   |  104.4    |
24.92     |  31.26    |  41.9     |  61.3     |  0.5222   |
|  5        |  0.5601   |  0.9709   |  0.2368   |  0.9784   |  0.3083   |  0.2401   |  122.9    |
71.3      |  79.24    |  1.606    |  35.45    |  0.6491   |
|  6        |  0.56     |  0.8186   |  0.3938   |  0.3417   |  0.6519   |  0.7375   |  145.4    |
21.05     |  72.03    |  99.14    |  24.62    |  0.2575   |
|  7        |  0.5593   |  0.9315   |  0.7056   |  0.8465   |  0.1393   |  0.2674   |  149.1    |
11.2      |  77.36    |  10.83    |  79.46    |  0.06753  |
|  8        |  0.5642   |  0.8806   |  0.1135   |  0.208    |  0.9826   |  0.5399   |  147.1    |
5.005     |  24.24    |  12.98    |  24.58    |  0.7746   |
|  9        |  0.5519   |  0.9517   |  0.4553   |  0.02553  |  0.8857   |  0.6264   |  149.4    |
77.81     |  24.87    |  5.627    |  64.87    |  0.2295   |
|  10       |  0.5642   |  0.9834   |  0.1785   |  0.05244  |  0.4356   |  0.5398   |  21.51    |
6.135     |  27.79    |  98.15    |  43.18    |  0.9553   |
|  11       |  0.5643   |  0.8502   |  0.6926   |  0.3709   |  0.3089   |  0.2081   |  40.72    |
15.34     |  21.6     |  0.7744   |  26.54    |  0.5496   |
|  12       |  0.5642   |  0.9423   |  0.1805   |  0.7083   |  0.4165   |  0.5368   |  120.7    |
20.64     |  30.22    |  90.65    |  66.44    |  0.1767   |
|  13       |  0.5647   |  0.9194   |  0.6984   |  0.1348   |  0.4765   |  0.3332   |  109.5    |
5.441     |  22.9     |  84.31    |  27.73    |  0.2564   |
|  14       |  0.5542   |  0.9673   |  0.6879   |  0.8459   |  0.3833   |  0.9238   |  149.2    |
10.37     |  23.11    |  99.68    |  78.93    |  0.6884   |
|  15       |  0.5572   |  0.9787   |  0.6995   |  0.8158   |  0.879    |  0.9932   |  57.12    |
5.494     |  79.62    |  97.93    |  51.05    |  0.6213   |
|  16       |  0.557    |  0.9396   |  0.4526   |  0.7973   |  0.8663   |  0.5227   |  37.19    |
6.405     |  22.91    |  13.01    |  77.4     |  0.7295   |
|  17       |  0.5595   |  0.8805   |  0.3813   |  0.3777   |  0.3615   |  0.7451   |  72.93    |
6.908     |  78.58    |  6.491    |  24.15    |  0.6035   |
|  18       |  0.5651   |  0.971    |  0.1682   |  0.7103   |  0.7343   |  0.6589   |  140.5    |
69.67     |  75.98    |  89.63    |  80.0     |  0.09719  |
|  19       |  0.5561   |  0.8422   |  0.4036   |  0.1188   |  0.4754   |  0.5675   |  36.55    |
8.88      |  27.86    |  95.37    |  77.63    |  0.04599  |
|  20       |  0.5571   |  0.9446   |  0.7721   |  0.6853   |  0.7807   |  0.8841   |  149.5    |
75.72     |  59.25    |  75.39    |  28.74    |  0.207    |
===================================================================================================
======================================================
```

In [0]:

```python
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
opt_params[1]["num_leaves"] = int(round(opt_params[1]["num_leaves"]))
opt_params[1]['max_depth'] = int(round(opt_params[1]['max_depth']))
opt_params[1]['min_data_in_leaf'] = int(round(opt_params[1]['min_data_in_leaf']))
opt_params[1]['max_bin'] = int(round(opt_params[1]['max_bin']))
opt_params[1]['objective']='binary'
opt_params[1]['metric']='auc'
opt_params[1]['is_unbalance']=True
opt_params[1]['boost_from_average']=False
opt_params=opt_params[1]
opt_params
```

Out[0]:

```
{'bagging_fraction': 0.9972055022804187,
 'boost_from_average': False,
 'feature_fraction': 0.8386133653331375,
 'is_unbalance': True,
 'lambda_l1': 0.31065638082201175,
 'lambda_l2': 0.84764245541438,
 'learning_rate': 0.1300097487520304,
 'max_bin': 122,
 'max_depth': 24,
 'metric': 'auc',
 'min_data_in_leaf': 26,
 'min_sum_hessian_in_leaf': 94.34910369995325,
 'num_leaves': 70,
 'objective': 'binary',
 'subsample': 0.5231418824080631}
```

In [0]:

```
from sklearn.model_selection import train_test_split
X_train, X_cv, y_train, y_cv = train_test_split(train, y, test_size = 0.20, stratify=y)
print('Train data shape : '+str(X_train.shape))
print('CV data shape : '+str(X_cv.shape))
```

```
Train data shape : (160000, 8)
CV data shape : (40000, 8)
```

In [0]:

```
import lightgbm as lgb
d_train = lgb.Dataset(X_train, label=y_train)
clf = lgb.train(opt_params, d_train)
```

In [0]:

```
y_train_pred = clf.predict(X_train)
y_cv_pred=clf.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7575399241623532
cv auc score : 0.5489149582825175
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=clf.predict(test)
test=test.reset_index()
submission = pd.DataFrame({"ID_code": test.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_lgb.csv", index=False)
```

Kaggle Score

1. Private Score: 0.55095
2. Public Score : 0.55518

In [0]:

```
y=train['target']
train=train.drop(['target'],axis=1)
```

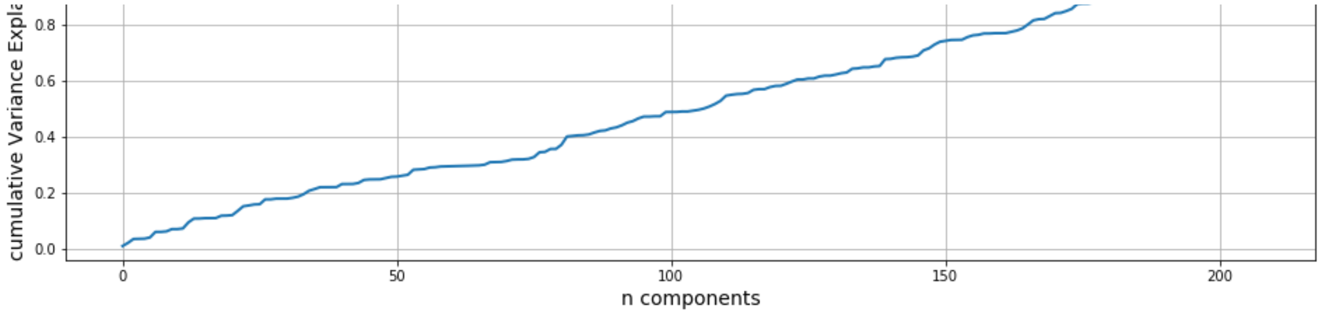
Select best k features

In [0]:

```
from sklearn.feature_selection import SelectKBest, f_classif
k_best = SelectKBest(f_classif, k=207)
k=k_best.fit(train,y)
features = k.transform(train)
```

In [0]:

```
import matplotlib.pyplot as plt
cumVarianceExplained = np.cumsum( k.scores_ )/np.sum(k.scores_)
plt.figure( figsize=(16, 4))
plt.plot( cumVarianceExplained, linewidth = 2 )
plt.grid()
plt.xlabel('n components',size=14)
plt.ylabel('cumulative Variance Explained',size=14)
plt.show()
```



In [0]:

```
#https://stackoverflow.com/questions/39839112/the-easiest-way-for-getting-feature-names-after-runn
ing-selectkbest-in-scikit-le
selector = SelectKBest(f_classif,k=190)
selector.fit(train, y)
cols = selector.get_support(indices=True)
data_kbest = train.iloc[:,cols]

best_feat=[]
best_feat=data_kbest.columns
print(best_feat)
```

```
Index(['var_0', 'var_1', 'var_2', 'var_3', 'var_4', 'var_5', 'var_6', 'var_8',
       'var_9', 'var_11',
       ...
       'var_197', 'var_198', 'var_199', 'sum', 'max', 'mean', 'std', 'skew',
       'kurt', 'med'],
      dtype='object', length=190)
```

Naive Bayes with top 190 features

In [0]:

```
from sklearn.model_selection import train_test_split
X_train, X_cv, y_train, y_cv = train_test_split(train[best_feat], y, test_size = 0.20, stratify=y)
print('Train data shape : '+str(X_train.shape))
print('CV data shape : '+str(X_cv.shape))
```

```
Train data shape : (160000, 190)
CV data shape : (40000, 190)
```

In [0]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV

gnb = GaussianNB(priors = [0.5,0.5])
params_NB = {'var_smoothing': np.logspace(0,-11, num=100)}
best_param=grid_search(gnb,params_NB,5,X_train[best_feat],y_train,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   20.2s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  3.2min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  3.7min finished
```

```
best param : {'var_smoothing': 1.2915496650148853e-07}
best score : 0.8825734087547039
```

In [0]:

```
y_train_pred,y_cv_pred=baseline_model(best_param,X_train,y_train,X_cv,y_cv,test[best_feat],best_fe
```

```
y_train_pred,y_cv_pred=baseline_model(best_param,x_train,y_train,x_cv,y_cv,best[best_feat],best_fe
at)
```

```
train auc score : 0.8023868759342128
cv auc score : 0.8042870002405981
```

Kaggle Score

1. Private Score: 0.50000
2. Public Score : 0.50000

lightgbm with top 190 features

In [0]:

```
#basic tools
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
import os
import numpy as np
import pandas as pd
import warnings

#tuning hyperparameters
from bayes_opt import BayesianOptimization
from skopt  import BayesSearchCV

#graph, plots
import matplotlib.pyplot as plt
import seaborn as sns

#building models
import lightgbm as lgb
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
import time
import sys

#metrics
from sklearn.metrics import roc_auc_score, roc_curve
warnings.simplefilter(action='ignore', category=FutureWarning)

def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=3, random_seed=6, output_pro
cess=False):
    # prepare data
    train_data = lgb.Dataset(data=X, label=y, free_raw_data=False)
    # parameters
    def lgb_eval(learning_rate,num_leaves, feature_fraction, bagging_fraction, max_depth, max_bin,
min_data_in_leaf,min_sum_hessian_in_leaf,subsample,lambda_l1,lambda_l2):
        params = {'application':'binary', 'metric':'auc'}
        params['learning_rate'] = max(min(learning_rate, 1), 0)
        params["num_leaves"] = int(round(num_leaves))
        params['feature_fraction'] = max(min(feature_fraction, 1), 0)
        params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
        params['max_depth'] = int(round(max_depth))
        params['max_bin'] = int(round(max_depth))
        params['min_data_in_leaf'] = int(round(min_data_in_leaf))
        params['min_sum_hessian_in_leaf'] = min_sum_hessian_in_leaf
        params['subsample'] = max(min(subsample, 1), 0)
        params['lambda_l1']=max(min(lambda_l1, 1), 0)
        params['lambda_l2']=max(min(lambda_l1, 1), 0)
        #evaluate cv for above paramters
        cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, ve
rbose_eval =200, metrics=['auc'])

        #return max mean for multiple folds
        return max(cv_result['auc-mean'])

    lgbBO = BayesianOptimization(lgb_eval, {'learning_rate': (0.01, 1.0),
                                            'num_leaves': (24, 80),
                                            'feature_fraction': (0.1, 0.9),
                                            'bagging_fraction': (0.8, 1),
                                            'max_depth': (5, 80),
                                            'max_bin':(20,150),
                                            'lambda_l1': (0.01, 1),
```

```
                                            'lambda_l1': (0.01,1),
                                            'lambda_l2': (0.01,1),
                                            'min_data_in_leaf': (20, 80),
                                            'min_sum_hessian_in_leaf':(0,100),
                                           'subsample': (0.01, 1.0)}, random_state=200)


    #n_iter: How many steps of bayesian optimization you want to perform. The more steps the more
likely to find a good maximum you are.
    #init_points: How many steps of random exploration you want to perform. Random exploration can
help by diversifying the exploration space.

    lgbBO.maximize(init_points=init_round, n_iter=opt_round)

    model_auc=[]
    for model in range(len( lgbBO.res)):
        model_auc.append(lgbBO.res[model]['target'])

    # return best parameters
    return lgbBO.res[pd.Series(model_auc).idxmax()]['target'],lgbBO.res[pd.Series(model_auc).idxmax
()]['params']

opt_params = bayes_parameter_opt_lgb(train[best_feat], y, init_round=5, opt_round=15, n_folds=11, r
andom_seed=6)
```

```
|   iter    |  target   | baggin... | featur... | lambda_l1 | lambda_l2 | learni... |  max_bin  |
max_depth | min_da... | min_su... | num_le... | subsample |
-------------------------------------------------------------------------------------------------
-------------------------------------------------------
|  1        |  0.8419   |  0.9895   |  0.2812   |  0.5985   |  0.434    |  0.7665   |  20.37    |
31.81     |  74.58    |  45.61    |  78.98    |  0.8687   |
|  2        |  0.8747   |  0.9972   |  0.8386   |  0.3107   |  0.8476   |  0.13     |  122.1    |
23.79     |  25.76    |  94.35    |  70.26    |  0.5231   |
|  3        |  0.866    |  0.9747   |  0.5627   |  0.4556   |  0.6834   |  0.4252   |  103.3    |
50.65     |  26.33    |  96.6     |  66.49    |  0.6828   |
|  4        |  0.8769   |  0.8659   |  0.1212   |  0.8056   |  0.9731   |  0.2901   |  104.4    |
24.92     |  31.26    |  41.9     |  61.3     |  0.5222   |
|  5        |  0.8765   |  0.9709   |  0.2368   |  0.9784   |  0.3083   |  0.2401   |  122.9    |
71.3      |  79.24    |  1.606    |  35.45    |  0.6491   |
|  6        |  0.8316   |  0.9936   |  0.122    |  0.2994   |  0.5044   |  0.9007   |  149.4    |
54.77     |  31.29    |  4.874    |  77.44    |  0.5757   |
|  7        |  0.8637   |  0.8591   |  0.6514   |  0.03244  |  0.9309   |  0.6671   |  23.85    |
10.26     |  27.32    |  95.29    |  24.42    |  0.4107   |
|  8        |  0.8722   |  0.8524   |  0.6708   |  0.4791   |  0.751    |  0.4387   |  137.1    |
16.0      |  77.57    |  99.44    |  32.38    |  0.5406   |
|  9        |  0.8731   |  0.9656   |  0.8767   |  0.808    |  0.5999   |  0.2059   |  38.54    |
74.21     |  26.59    |  0.7953   |  27.05    |  0.3759   |
|  10       |  0.863    |  0.8984   |  0.842    |  0.4462   |  0.9857   |  0.5203   |  64.44    |
.42     |  62.42    |  5.012    |  24.59    |  0.3092   |
|  11       |  0.8746   |  0.9989   |  0.8048   |  0.3322   |  0.1053   |  0.2206   |  147.8    |
66.45     |  20.24    |  71.56    |  25.51    |  0.7651   |
|  12       |  0.8526   |  0.8373   |  0.5995   |  0.6737   |  0.2634   |  0.9813   |  147.1    |
68.29     |  24.69    |  70.02    |  25.27    |  0.9589   |
|  13       |  0.8583   |  0.9531   |  0.8919   |  0.3739   |  0.7312   |  0.5817   |  149.8    |
6.939     |  48.02    |  4.996    |  27.21    |  0.1171   |
|  14       |  0.8695   |  0.8803   |  0.1421   |  0.205    |  0.3694   |  0.3046   |  36.89    |
8.58      |  24.43    |  0.5931   |  71.24    |  0.1819   |
|  15       |  0.8648   |  0.8316   |  0.26     |  0.2899   |  0.5751   |  0.7256   |  147.6    |
5.553     |  26.87    |  99.84    |  24.6     |  0.776    |
|  16       |  0.7697   |  1.0      |  0.9      |  0.01     |  0.01     |  0.01     |  20.0     |
0.0     |  80.0     |  100.0    |  24.0     |  0.01     |
|  17       |  0.8496   |  0.8133   |  0.7626   |  0.9315   |  0.1624   |  0.4393   |  91.72    |
6.63      |  77.87    |  7.503    |  76.92    |  0.5493   |
|  18       |  0.8585   |  0.9336   |  0.8787   |  0.2985   |  0.5772   |  0.6407   |  134.4    |
72.99     |  22.24    |  0.9899   |  25.92    |  0.2546   |
|  19       |  0.8775   |  0.9497   |  0.812    |  0.5865   |  0.7572   |  0.192    |  22.7     |
9.93    |  24.23    |  94.6     |  75.26    |  0.5526   |
|  20       |  0.8686   |  0.8065   |  0.7148   |  0.1853   |  0.1708   |  0.1026   |  21.63    |
71.63     |  26.01    |  2.811    |  75.74    |  0.01513  |
=================================================================================================
=======================================================
```

In [0]:

```
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
opt_params[1]["num_leaves"] = int(round(opt_params[1]["num_leaves"]))
```

```
opt_params[1]['max_depth'] = int(round(opt_params[1]['max_depth']))
opt_params[1]['min_data_in_leaf'] = int(round(opt_params[1]['min_data_in_leaf']))
opt_params[1]['max_bin'] = int(round(opt_params[1]['max_bin']))
opt_params[1]['objective']='binary'
opt_params[1]['metric']='auc'
opt_params[1]['is_unbalance']=True
opt_params[1]['boost_from_average']=False
opt_params=opt_params[1]
opt_params
```

Out[0]:

```
{'bagging_fraction': 0.9496636248581771,
 'boost_from_average': False,
 'feature_fraction': 0.8119744871330826,
 'is_unbalance': True,
 'lambda_l1': 0.586515432153725,
 'lambda_l2': 0.7571598185215849,
 'learning_rate': 0.19204304295071015,
 'max_bin': 23,
 'max_depth': 70,
 'metric': 'auc',
 'min_data_in_leaf': 24,
 'min_sum_hessian_in_leaf': 94.5976532519037,
 'num_leaves': 75,
 'objective': 'binary',
 'subsample': 0.5526438213080649}
```

In [0]:

```
from sklearn.model_selection import train_test_split
X_train, X_cv, y_train, y_cv = train_test_split(train[best_feat], y, test_size = 0.20, stratify=y)
print('Train data shape : '+str(X_train.shape))
print('CV data shape : '+str(X_cv.shape))
```

```
Train data shape : (160000, 190)
CV data shape : (40000, 190)
```

In [0]:

```
import lightgbm as lgb
d_train = lgb.Dataset(X_train, label=y_train)
clf = lgb.train(opt_params, d_train)
```

In [0]:

```
y_train_pred = clf.predict(X_train)
y_cv_pred=clf.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.9798753502883472
cv auc score : 0.8739105196640478
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=clf.predict(test[best_feat])
test=test.reset_index()
submission = pd.DataFrame({"ID_code": test.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_lgb.csv", index=False)
```

Kaggle Score

1. Private Score: 0.87044
2. Public Score : 0.87327

All 208 features

In [0]:

```
#https://www.kaggle.com/gpreda/santander-eda-and-prediction
%%time
idx = data.columns.values[2:201]
for df in [data]:
    df['sum'] = df[idx].sum(axis=1)
    df['min'] = df[idx].min(axis=1)
    df['max'] = df[idx].max(axis=1)
    df['mean'] = df[idx].mean(axis=1)
    df['std'] = df[idx].std(axis=1)
    df['skew'] = df[idx].skew(axis=1)
    df['kurt'] = df[idx].kurtosis(axis=1)
    df['med'] = df[idx].median(axis=1)
```

```
CPU times: user 4.62 s, sys: 41.9 ms, total: 4.67 s
Wall time: 4.68 s
```

In [0]:

```
#https://www.kaggle.com/gpreda/santander-eda-and-prediction
%%time
test = pd.read_csv('/content/drive/My Drive/proj_1/test.csv', index_col=0)
idx = test.columns.values[1:200]
for df in [test]:
    df['sum'] = df[idx].sum(axis=1)
    df['min'] = df[idx].min(axis=1)
    df['max'] = df[idx].max(axis=1)
    df['mean'] = df[idx].mean(axis=1)
    df['std'] = df[idx].std(axis=1)
    df['skew'] = df[idx].skew(axis=1)
    df['kurt'] = df[idx].kurtosis(axis=1)
    df['med'] = df[idx].median(axis=1)
```

```
CPU times: user 10.8 s, sys: 183 ms, total: 10.9 s
Wall time: 11.2 s
```

In [0]:

```
X_train,y_train,X_cv,y_cv=split_train_test(data)
```

```
Train data shape : (160000, 208)
CV data shape : (40000, 208)
```

NB with 208 features

In [0]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV

gnb = GaussianNB(priors = [0.5,0.5])
params_NB = {'var_smoothing': np.logspace(2,-18, num=100)}
best_param=grid_search(gnb,params_NB,5,X_train,y_train,-1)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   17.1s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  2.9min finished
```

```
best param : {'var_smoothing': 7.05480231071866e-14}
best score : 0.8828428927428436
```

In [0]:

```
best_feat=test.columns
y_train_pred,y_cv_pred=baseline_model(best_param,X_train,y_train,X_cv,y_cv,test,best_feat)
```

```
train auc score : 0.805123507496776
cv auc score : 0.7986143490441069
```

Kaggle Score

1. Private Score: 0.79956
2. Public Score : 0.79951

SVM with 208 features

In [0]:

```
from sklearn.linear_model import SGDClassifier
params = {'alpha': np.logspace(4,-9, num=100)}
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',max_iter=400,tol=1e-3,class_weight = 'balanced')
best_param=grid_search(sgd,params,5,X_train,y_train,16)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=16)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=16)]: Done  18 tasks      | elapsed:   18.3s
[Parallel(n_jobs=16)]: Done 168 tasks      | elapsed:  5.0min
[Parallel(n_jobs=16)]: Done 418 tasks      | elapsed: 47.9min
[Parallel(n_jobs=16)]: Done 500 out of 500 | elapsed: 61.5min finished
```

```
best param : {'alpha': 6.428073117284319 4e-06}
best score : 0.8543793226071406
```

In [0]:

```
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=400,tol=1e-3,
alpha = best_param['alpha'])
sgd.fit(X_train ,y_train)
y_train_pred=sgd.predict(X_train)
y_cv_pred = sgd.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7757864176608676
cv auc score : 0.7678074330957774
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=sgd.predict(test)
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_svm.csv", index=False)
```

Kaggle Score

1. Private Score: 0.76889
2. Public Score : 0.77222

Decision Tree with 208 features

In [0]:

```
X_train,y_train,X_cv,y_cv=split_train_test(data)
```

```
Train data shape : (160000, 208)
CV data shape : (40000, 208)
```

In [0]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(class_weight = 'balanced')

params={
       'max_depth' :[1, 5, 10, 50, 100,500,1000,2000,5000,10000,20000,30000],
       'min_samples_split':[10, 100, 500,1000,2000,5000,10000]
}
best_param=grid_search(dtc,params,5,X_train,y_train,16)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
```

```
[Parallel(n_jobs=16)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=16)]: Done  15 out of  15 | elapsed:  7.2min finished
```

```
best param : {'max_depth': 20000, 'min_samples_split': 2000}
best score : 0.6774159847151625
```

In [0]:

```
best_param={'max_depth': 20000, 'min_samples_split': 2000}
```

In [0]:

```
from sklearn.metrics import roc_auc_score
dtc =
DecisionTreeClassifier(max_depth=best_param['max_depth'],min_samples_split=best_param['min_samples_
split'],class_weight = 'balanced')
dtc.fit(X_train, y_train)
y_train_pred = dtc.predict(X_train)
y_cv_pred=dtc.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.6741248804554278
cv auc score : 0.6343949374859997
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=dtc.predict(test)
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_dtc.csv", index=False)
```

Kaggle Score

1. Private Score: 0.62694
2. Public Score : 0.62779

Voting Classifier with 208 features

In [0]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
import numpy as np
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
```

```
gnb = GaussianNB(priors = [0.5,0.5],var_smoothing=7.05480231071866e-14)
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=1000,tol=1e-3
,alpha = 6.4280731172843194e-06)
dtc =
DecisionTreeClassifier(max_depth=best_param['max_depth'],min_samples_split=best_param['min_samples_
split'],class_weight = 'balanced')

vcf = VotingClassifier(estimators=[('gnb', gnb), ('sgd', sgd), ('dtc', dtc)], voting='hard')
vcf = vcf.fit(X_train, y_train)
```

In [0]:

```
y_train_pred=vcf.predict(X_train)
y_cv_pred = vcf.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7931603794960332
cv auc score : 0.7743007447476349
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=vcf.predict(test)
test=test.reset_index()
submission = pd.DataFrame({"ID_code": test.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_vcf.csv", index=False)
```

Kaggle Score

1. Private Score: 0.77758
2. Public Score : 0.77498

Stacking classifier with 208 features

In [0]:

```
#http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/
from sklearn.tree import DecisionTreeClassifier
from mlxtend.classifier import StackingClassifier
gnb = GaussianNB(priors = [0.5,0.5],var_smoothing=7.05480231071866e-14)
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=1000,tol=1e-3
,alpha = 6.4280731172843194e-06)
dtc =
DecisionTreeClassifier(max_depth=best_param['max_depth'],min_samples_split=best_param['min_samples_
split'],class_weight = 'balanced')

clf = StackingClassifier(classifiers=[sgd,dtc], meta_classifier=gnb)
```

In [0]:

```
clf.fit(X_train,y_train)
y_train_pred=clf.predict(X_train)
y_cv_pred = clf.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.771904963158689
cv auc score : 0.768198128313408
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=clf.predict(test)
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
```

```
submission.to_csv("/content/drive/My Drive/proj_1/submission_stc.csv", index=False)
```

Kaggle Score

1. Private Score: 0.76782
2. Public Score : 0.76824

lightgbm 208 featuers without augmentation

In [0]:

```
train=data
y=train['target']
train=train.drop(['target'],axis=1)
```

In [0]:

```
#basic tools
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
import os
import numpy as np
import pandas as pd
import warnings

#tuning hyperparameters
from bayes_opt import BayesianOptimization
from skopt  import BayesSearchCV

#graph, plots
import matplotlib.pyplot as plt
import seaborn as sns

#building models
import lightgbm as lgb
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
import time
import sys

#metrics
from sklearn.metrics import roc_auc_score, roc_curve
warnings.simplefilter(action='ignore', category=FutureWarning)

def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=3, random_seed=6, output_pro
cess=False):
    # prepare data
    train_data = lgb.Dataset(data=X, label=y, free_raw_data=False)
    # parameters
    def
lgb_eval(learning_rate,bagging_freq,bagging_seed,reg_alpha,reg_lambda,min_gain_to_split,min_child_w
eight,num_leaves, feature_fraction, bagging_fraction, max_depth,
max_bin,min_data_in_leaf,min_sum_hessian_in_leaf,subsample):
        params = {'application':'binary', 'metric':'auc','tree_learner': 'serial','verbosity': -1,'
boost': 'gbdt'}
        params['learning_rate'] = max(min(learning_rate, 1), 0)
        params['bagging_freq']=int(bagging_freq)
        params['bagging_seed']=int(bagging_seed)
        params['reg_alpha']=reg_alpha
        params['reg_lambda']=reg_lambda
        params['min_gain_to_split']=min_gain_to_split
        params['min_child_weight']=min_child_weight
        params['num_leaves'] = int(round(num_leaves))
        params['feature_fraction'] = max(min(feature_fraction, 1), 0)
        params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
        params['max_depth'] = int(round(max_depth))
        params['max_bin'] = int(round(max_depth))
        params['min_data_in_leaf'] = int(round(min_data_in_leaf))
        params['min_sum_hessian_in_leaf'] = min_sum_hessian_in_leaf
        params['subsample'] = max(min(subsample, 1), 0)
        #evaluate cv for above paramters
        cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, ve
rbose_eval =200, metrics=['auc'])
```

```
        #return max mean for multiple folds
        return max(cv_result['auc-mean'])

    lgbBO = BayesianOptimization(lgb_eval, {'learning_rate': (0.01, 1.0),
                                            'bagging_freq' :(4,10),
                                            'bagging_seed':(5,10),
                                            'reg_alpha' :(0,4),
                                            'reg_lambda':(0,10),
                                            'min_gain_to_split':(.01,0.9),
                                            'min_child_weight':(5,20),
                                            'num_leaves': (10, 80),
                                            'feature_fraction': (0.01, 0.9),
                                            'bagging_fraction': (0.2, 1),
                                            'max_depth': (5, 80),
                                            'max_bin':(20,150),
                                            'min_data_in_leaf': (20, 80),
                                            'min_sum_hessian_in_leaf':(0,100),
                                           'subsample': (0.01, 1.0)}, random_state=200)


    #n_iter: How many steps of bayesian optimization you want to perform. The more steps the more
likely to find a good maximum you are.
    #init_points: How many steps of random exploration you want to perform. Random exploration can
help by diversifying the exploration space.

    lgbBO.maximize(init_points=init_round, n_iter=opt_round)

    model_auc=[]
    for model in range(len( lgbBO.res)):
        model_auc.append(lgbBO.res[model]['target'])

    # return best parameters
    return lgbBO.res[pd.Series(model_auc).idxmax()]['target'],lgbBO.res[pd.Series(model_auc).idxmax
()]['params']

opt_params = bayes_parameter_opt_lgb(train, y, init_round=5, opt_round=20, n_folds=11, random_seed=
6)
```

```
|   iter    |  target   | baggin... | baggin... | baggin... | featur... | learni... |  max_bin  |
max_depth | min_ch... | min_da... | min_ga... | min_su... | num_le... | reg_alpha | reg_la... | su
bsample |
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
|  1        |  0.8496   |  0.9581   |  5.359    |  7.972    |  0.3912   |  0.7665   |  20.37    |
1.81      |  18.65    |  47.36    |  0.8838   |  86.74    |  79.02    |  3.693    |  3.037    |  0.84
6    |
|  2        |  0.8335   |  0.297    |  8.713    |  6.253    |  0.09547  |  0.9441   |  127.4    |
3.87      |  18.1     |  54.7     |  0.4106   |  68.02    |  39.36    |  2.562    |  6.087    |  0.11
4    |
|  3        |  0.8408   |  0.9728   |  8.552    |  8.398    |  0.3031   |  0.03619  |  124.5    |
7.96      |  9.244    |  58.94    |  0.2464   |  18.77    |  39.33    |  2.664    |  5.174    |  0.85
9    |
|  4        |  0.843    |  0.3368   |  9.869    |  6.506    |  0.2169   |  0.7937   |  134.9    |
9.05      |  5.241    |  32.27    |  0.5845   |  51.07    |  24.72    |  1.236    |  0.1761   |  0.64
4    |
|  5        |  0.8269   |  0.6229   |  7.668    |  6.496    |  0.802    |  0.7224   |  143.9    |
4.78      |  19.83    |  47.15    |  0.658    |  34.41    |  74.13    |  0.2577   |  0.09294  |  0.48
5    |
|  6        |  0.8746   |  0.9662   |  5.477    |  9.296    |  0.7787   |  0.2285   |  20.8     |
4.35      |  6.659    |  23.3     |  0.2284   |  7.376    |  18.27    |  2.126    |  4.081    |  0.11
2    |
|  7        |  0.8743   |  0.4547   |  5.796    |  8.926    |  0.7923   |  0.3608   |  23.96    |
4.6       |  19.09    |  20.4     |  0.8895   |  41.53    |  10.44    |  3.131    |  8.745    |  0.66
|
|  8        |  0.8752   |  0.8258   |  7.103    |  7.268    |  0.2133   |  0.5289   |  25.35    |
6.16      |  19.16    |  24.76    |  0.3533   |  4.429    |  15.54    |  2.226    |  2.758    |  0.88
9    |
|  9        |  0.8754   |  0.4625   |  9.876    |  8.016    |  0.04898  |  0.4397   |  20.2     |
2.62      |  6.934    |  27.96    |  0.4741   |  7.934    |  14.25    |  3.783    |  2.926    |  0.88
1    |
|  10       |  0.8784   |  0.7455   |  9.856    |  8.001    |  0.1255   |  0.5      |  25.38    |
4.77      |  17.72    |  24.31    |  0.6751   |  3.244    |  10.96    |  2.64     |  2.376    |  0.99
9    |
|  11       |  0.8746   |  0.7638   |  6.332    |  9.306    |  0.3405   |  0.6091   |  21.44    |
6.91      |  12.39    |  49.27    |  0.7206   |  9.503    |  10.25    |  0.9728   |  0.9476   |
0.5925    |
```

```
|  12        |  0.8732   |  0.9273   |  5.565    |  8.71     |  0.3495   |  0.2304   |  21.81    |
0.97     |  17.24    |  21.22    |  0.02125  |  7.058    |  11.96    |  3.827    |  8.365    |  0.33
4    |
|  13        |  0.8307   |  0.3648   |  9.378    |  6.521    |  0.8773   |  0.9195   |  22.14    |
7.66     |  7.254    |  24.61    |  0.7973   |  2.063    |  22.97    |  0.7995   |  5.176    |  0.97
3    |
|  14        |  0.8731   |  0.4952   |  5.481    |  8.292    |  0.1598   |  0.4433   |  141.3    |
.026     |  17.94    |  42.38    |  0.6478   |  7.367    |  11.92    |  2.173    |  3.899    |  0.78
7    |
|  15        |  0.845    |  0.209    |  7.525    |  7.155    |  0.7049   |  0.7682   |  22.7     |
7.36     |  15.82    |  36.98    |  0.471    |  99.15    |  12.32    |  1.387    |  2.244    |  0.29
4    |
|  16        |  0.8706   |  0.33     |  8.424    |  7.541    |  0.2505   |  0.1461   |  21.3     |
2.08     |  11.14    |  33.89    |  0.7316   |  9.04     |  17.68    |  2.783    |  5.375    |  0.71
1    |
|  17        |  0.8726   |  0.7453   |  5.687    |  9.3      |  0.09804  |  0.5525   |  77.85    |
6.2      |  18.75    |  22.72    |  0.4876   |  24.21    |  24.87    |  3.924    |  1.189    |  0.27
2    |
|  18        |  0.8765   |  0.8466   |  4.988    |  7.559    |  0.7119   |  0.356    |  138.9    |
2.43     |  15.75    |  79.21    |  0.4928   |  36.46    |  10.11    |  3.882    |  0.0185   |  0.03
83   |
|  19        |  0.8687   |  0.7875   |  9.196    |  8.455    |  0.1783   |  0.5509   |  110.9    |
.361     |  6.342    |  77.14    |  0.3667   |  1.961    |  20.2     |  0.1863   |  1.067    |  0.83
4    |
|  20        |  0.8732   |  0.3887   |  4.042    |  9.969    |  0.1665   |  0.175    |  35.38    |
8.61     |  7.649    |  75.2     |  0.6926   |  99.25    |  15.57    |  0.8752   |  0.2295   |
0.09957  |
|  21        |  0.8464   |  0.484    |  8.847    |  9.896    |  0.722    |  0.7452   |  22.17    |
0.01     |  17.62    |  35.33    |  0.3148   |  8.241    |  15.67    |  0.06284  |  0.2371   |
0.8187   |
|  22        |  0.8156   |  0.4182   |  5.083    |  5.748    |  0.6646   |  0.6495   |  27.88    |
.689     |  17.24    |  76.8     |  0.376    |  15.58    |  76.93    |  2.623    |  0.7495   |  0.97
8    |
|  23        |  0.8599   |  0.6324   |  9.915    |  7.323    |  0.8936   |  0.7598   |  104.2    |
9.31     |  5.972    |  25.1     |  0.1053   |  2.199    |  14.63    |  3.867    |  0.7099   |  0.84
8    |
|  24        |  0.8631   |  0.2739   |  4.05     |  8.624    |  0.4559   |  0.5484   |  149.8    |
.806     |  5.655    |  67.08    |  0.2483   |  95.5     |  15.4     |  2.175    |  8.052    |  0.21
8    |
|  25        |  0.8189   |  0.4676   |  5.207    |  8.09     |  0.5833   |  0.03702  |  148.2    |
.307     |  7.73     |  70.3     |  0.3288   |  89.7     |  78.59    |  1.62     |  9.024    |  0.72
4    |
=========================================================================================================
=========================================================================================================
```

In [0]:

```
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
opt_params[1]['num_leaves'] = int(round(opt_params[1]['num_leaves']))
opt_params[1]['max_depth'] = int(round(opt_params[1]['max_depth']))
opt_params[1]['bagging_freq'] = int(round(opt_params[1]['bagging_freq']))
opt_params[1]['min_data_in_leaf'] = int(round(opt_params[1]['min_data_in_leaf']))
opt_params[1]['max_bin'] = int(round(opt_params[1]['max_bin']))
opt_params[1]['bagging_freq']=int(round(opt_params[1]['bagging_freq']))
opt_params[1]['bagging_seed']=int(round(opt_params[1]['bagging_seed']))
opt_params[1]['objective']='binary'
opt_params[1]['metric']='auc'
opt_params[1]['is_unbalance']=True
opt_params[1]['boost_from_average']=False
opt_params1=opt_params[1]
opt_params1
```

Out[0]:

```
(0.8784269789462116,
 {'bagging_fraction': 0.7455101736702525,
  'bagging_freq': 10,
  'bagging_seed': 8,
  'boost_from_average': False,
  'feature_fraction': 0.1255437885720564,
  'is_unbalance': True,
  'learning_rate': 0.4999726215646777,
  'max_bin': 25,
  'max_depth': 65,
  'metric': 'auc',
```

```
 'min_child_weight': 17.72397148006606,
 'min_data_in_leaf': 24,
 'min_gain_to_split': 0.6751333641193706,
 'min_sum_hessian_in_leaf': 3.2438005357282917,
 'num_leaves': 11,
 'objective': 'binary',
 'reg_alpha': 2.639519243780288,
 'reg_lambda': 2.3764917298916863,
 'subsample': 0.9969364107570741})
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
from sklearn.model_selection import StratifiedKFold, KFold
import lightgbm as lgb
num_folds = 11
features = [c for c in train.columns if c not in ['ID_code', 'target']]

folds = KFold(n_splits=num_folds, random_state=44000)
oof = np.zeros(len(train))
getVal = np.zeros(len(train))
predictions = np.zeros(len(y))
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
%%time
import lightgbm as lgb
import numpy as np
print('Light GBM Model')
for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values, y.values)):

    X_train, y_train = train.iloc[trn_idx][features], y.iloc[trn_idx]
    X_valid, y_valid = train.iloc[val_idx][features], y.iloc[val_idx]


    #X_tr, y_tr = augment(X_train.values, y_train.values)
    #X_tr = pd.DataFrame(X_tr)


    print("Fold idx:{}".format(fold_ + 1))
    trn_data = lgb.Dataset(X_train, label=y_train)
    val_data = lgb.Dataset(X_valid, label=y_valid)


    clf = lgb.train(opt_params1, trn_data, 1000000, valid_sets = [trn_data, val_data],
verbose_eval=1000,
                    early_stopping_rounds = 3000)
    oof[val_idx] = clf.predict(train.iloc[val_idx], num_iteration=clf.best_iteration)
    getVal[val_idx]+= clf.predict(train.iloc[val_idx], num_iteration=clf.best_iteration) /
folds.n_splits
    predictions += clf.predict(test, num_iteration=clf.best_iteration) / folds.n_splits
```

```
Light GBM Model
Fold idx:1
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988188 valid_1's auc: 0.859839
[2000] training's auc: 0.99908 valid_1's auc: 0.855831
[3000] training's auc: 0.999916 valid_1's auc: 0.855939
Early stopping, best iteration is:
[123] training's auc: 0.912017 valid_1's auc: 0.878477
Fold idx:2
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988191 valid_1's auc: 0.856322
[2000] training's auc: 0.999037 valid_1's auc: 0.849981
[3000] training's auc: 0.999921 valid_1's auc: 0.850475
Early stopping, best iteration is:
[219] training's auc: 0.929794 valid_1's auc: 0.873563
Fold idx:3
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988557 valid_1's auc: 0.843051
[2000] training's auc: 0.999183 valid_1's auc: 0.842231
[3000] training's auc: 0.999925 valid_1's auc: 0.845646
Early stopping, best iteration is:
```

```
[171] training's auc: 0.922585 valid_1's auc: 0.869885
Fold idx:4
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988426 valid_1's auc: 0.856592
[2000] training's auc: 0.999127 valid_1's auc: 0.851889
[3000] training's auc: 0.999924 valid_1's auc: 0.85259
Early stopping, best iteration is:
[162] training's auc: 0.920232 valid_1's auc: 0.879136
Fold idx:5
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988493 valid_1's auc: 0.852109
[2000] training's auc: 0.999189 valid_1's auc: 0.84903
[3000] training's auc: 0.999924 valid_1's auc: 0.850148
Early stopping, best iteration is:
[141] training's auc: 0.915567 valid_1's auc: 0.874723
Fold idx:6
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988201 valid_1's auc: 0.85804
[2000] training's auc: 0.999233 valid_1's auc: 0.854149
[3000] training's auc: 0.999935 valid_1's auc: 0.857299
Early stopping, best iteration is:
[148] training's auc: 0.917715 valid_1's auc: 0.879645
Fold idx:7
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988013 valid_1's auc: 0.863059
[2000] training's auc: 0.999115 valid_1's auc: 0.859111
[3000] training's auc: 0.999905 valid_1's auc: 0.85968
Early stopping, best iteration is:
[125] training's auc: 0.911895 valid_1's auc: 0.88201
Fold idx:8
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988324 valid_1's auc: 0.853678
[2000] training's auc: 0.999101 valid_1's auc: 0.847545
[3000] training's auc: 0.999893 valid_1's auc: 0.849365
Early stopping, best iteration is:
[115] training's auc: 0.910127 valid_1's auc: 0.877177
Fold idx:9
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988446 valid_1's auc: 0.860864
[2000] training's auc: 0.999149 valid_1's auc: 0.856828
[3000] training's auc: 0.999925 valid_1's auc: 0.858214
Early stopping, best iteration is:
[161] training's auc: 0.919399 valid_1's auc: 0.880525
Fold idx:10
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988512 valid_1's auc: 0.864184
[2000] training's auc: 0.999095 valid_1's auc: 0.858972
[3000] training's auc: 0.999923 valid_1's auc: 0.859027
Early stopping, best iteration is:
[128] training's auc: 0.912784 valid_1's auc: 0.885245
Fold idx:11
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.988428 valid_1's auc: 0.857736
[2000] training's auc: 0.99922 valid_1's auc: 0.851354
[3000] training's auc: 0.999943 valid_1's auc: 0.85273
Early stopping, best iteration is:
[167] training's auc: 0.92102 valid_1's auc: 0.881276
CPU times: user 57min 31s, sys: 7.31 s, total: 57min 38s
Wall time: 14min 54s
```

In [0]:

```
print("\n >> CV score: {:<8.5f}".format(roc_auc_score(y, oof)))
```

```
 >> CV score: 0.87799
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
test=test.reset_index()
submission = pd.DataFrame({"ID_code": test.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_lgb.csv", index=False)
```

Kaggle Score

1. Private Score: 0.88797
2. Public Score : 0.89110

lightgbm with original 200 features+ Min,Max,Median,Mean,standard-deviation and kurtosis features

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
from sklearn.model_selection import StratifiedKFold, KFold
import lightgbm as lgb
num_folds = 11
features = [c for c in train.columns if c not in ['ID_code', 'target']]

folds = KFold(n_splits=num_folds, shuffle=True,random_state=44000)
oof = np.zeros(len(train))
getVal = np.zeros(len(train))
predictions = np.zeros(len(y))
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
%%time
import lightgbm as lgb
import numpy as np
print('Light GBM Model')
for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values, y.values)):

    X_train, y_train = train.iloc[trn_idx][features], y.iloc[trn_idx]
    X_valid, y_valid = train.iloc[val_idx][features], y.iloc[val_idx]


    X_tr, y_tr = augment(X_train.values, y_train.values)
    X_tr = pd.DataFrame(X_tr)


    print("Fold idx:{}".format(fold_ + 1))
    trn_data = lgb.Dataset(X_tr, label=y_tr)
    val_data = lgb.Dataset(X_valid, label=y_valid)


    clf = lgb.train(opt_params1, trn_data, 1000000, valid_sets = [trn_data, val_data],
verbose_eval=1000,
                    early_stopping_rounds = 3000)
    oof[val_idx] = clf.predict(train.iloc[val_idx], num_iteration=clf.best_iteration)
    getVal[val_idx]+= clf.predict(train.iloc[val_idx], num_iteration=clf.best_iteration) /
folds.n_splits
    predictions += clf.predict(test, num_iteration=clf.best_iteration) / folds.n_splits
```

```
Light GBM Model
Fold idx:1
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.955398 valid_1's auc: 0.869093
[2000] training's auc: 0.981898 valid_1's auc: 0.86222
[3000] training's auc: 0.993167 valid_1's auc: 0.857889
Early stopping, best iteration is:
[218] training's auc: 0.911226 valid_1's auc: 0.885317
Fold idx:2
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.955716 valid_1's auc: 0.867344
[2000] training's auc: 0.982003 valid_1's auc: 0.86195
[3000] training's auc: 0.993178 valid_1's auc: 0.855871
Early stopping, best iteration is:
[254] training's auc: 0.914806 valid_1's auc: 0.877526
Fold idx:3
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.954781 valid_1's auc: 0.876759
[2000] training's auc: 0.98162 valid_1's auc: 0.864124
[3000] training's auc: 0.992998 valid_1's auc: 0.861157
Early stopping, best iteration is:
[228] training's auc: 0.910951 valid_1's auc: 0.887344
```

```
Fold idx:4
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.955599 valid_1's auc: 0.870863
[2000] training's auc: 0.981927 valid_1's auc: 0.862968
[3000] training's auc: 0.993157 valid_1's auc: 0.860327
Early stopping, best iteration is:
[222] training's auc: 0.911926 valid_1's auc: 0.884605
Fold idx:5
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.955685 valid_1's auc: 0.870419
[2000] training's auc: 0.981821 valid_1's auc: 0.858058
[3000] training's auc: 0.993149 valid_1's auc: 0.850629
Early stopping, best iteration is:
[214] training's auc: 0.911141 valid_1's auc: 0.88473
Fold idx:6
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.955407 valid_1's auc: 0.868771
[2000] training's auc: 0.981868 valid_1's auc: 0.861048
[3000] training's auc: 0.993049 valid_1's auc: 0.855649
Early stopping, best iteration is:
[200] training's auc: 0.909978 valid_1's auc: 0.88771
Fold idx:7
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.955089 valid_1's auc: 0.874552
[2000] training's auc: 0.981754 valid_1's auc: 0.867732
[3000] training's auc: 0.993095 valid_1's auc: 0.863688
Early stopping, best iteration is:
[224] training's auc: 0.911298 valid_1's auc: 0.887152
Fold idx:8
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.955708 valid_1's auc: 0.872736
[2000] training's auc: 0.98202 valid_1's auc: 0.861682
[3000] training's auc: 0.993197 valid_1's auc: 0.855873
Early stopping, best iteration is:
[199] training's auc: 0.909945 valid_1's auc: 0.88547
Fold idx:9
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.955463 valid_1's auc: 0.873096
[2000] training's auc: 0.981827 valid_1's auc: 0.86366
[3000] training's auc: 0.993117 valid_1's auc: 0.859437
Early stopping, best iteration is:
[214] training's auc: 0.910965 valid_1's auc: 0.887131
Fold idx:10
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.95547 valid_1's auc: 0.873212
[2000] training's auc: 0.98184 valid_1's auc: 0.861038
[3000] training's auc: 0.993053 valid_1's auc: 0.857826
Early stopping, best iteration is:
[214] training's auc: 0.91105 valid_1's auc: 0.888769
Fold idx:11
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.955336 valid_1's auc: 0.871735
[2000] training's auc: 0.981694 valid_1's auc: 0.863308
[3000] training's auc: 0.993203 valid_1's auc: 0.860556
Early stopping, best iteration is:
[214] training's auc: 0.910927 valid_1's auc: 0.887258
CPU times: user 1h 52min 9s, sys: 17.3 s, total: 1h 52min 26s
Wall time: 59min 33s
```

In [0]:

```
from sklearn.metrics import roc_auc_score
print("\n >> CV score: {:<8.5f}".format(roc_auc_score(y, oof)))
```

```
 >> CV score: 0.88568
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission.csv", index=False)
```

Kaggle Score

1. Private Score: 0.89211
2. Public Score : 0.89417

Separating the Real/Synthetic Test Data

In [0]:

```
train_test = pd.concat([data,test], axis = 0,sort=False)
```

In [24]:

```
#https://www.kaggle.com/super13579/split-test-dataset
from tqdm import tqdm
for f in tqdm(features):
    train_test[f+'dup'] = train_test.duplicated(f,False).astype(int)
```

```
100%|██████████| 200/200 [00:09<00:00, 20.46it/s]
```

In [0]:

```
#https://www.kaggle.com/super13579/split-test-dataset
train = train_test.loc[train_test['target'].isnull()==False,:]
test = train_test.loc[train_test['target'].isnull()==True,:]
```

In [0]:

```
#https://www.kaggle.com/super13579/split-test-dataset
import warnings
warnings.filterwarnings("ignore")
test['has_dup']=test[test.columns[201:401]].sum(axis=1)
fake_te = test.loc[test['has_dup']==200,:]
real_te = test.loc[test['has_dup']!=200,:]
```

In [0]:

```
print('Shape of real test : '+str(real_te.shape))
print('Shape of fake test : '+str(fake_te.shape))
```

```
Shape of real test : (100000, 402)
Shape of fake test : (100000, 402)
```

In [0]:

```
train_test_real = pd.concat([train,real_te], axis = 0)
```

In [0]:

```
train_test_real.shape
```

Out[0]:

```
(300000, 402)
```

In [0]:

```
#https://www.kaggle.com/super13579/split-test-dataset
train_test_real=train_test_real.drop(['has_dup'],axis=1)
test=test.drop(['has_dup'],axis=1)
```

In [0]:

```
col=[]
for i in tqdm(range(200)):
```

```
    m='var_'+str(i)+'dup'
    col.append(m)
train_test_real=train_test_real.drop(col,axis=1)
```

```
100%|██████████| 200/200 [00:00<00:00, 191477.01it/s]
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/train_test_real.sav'
pickle.dump(train_test_real, open(filename, 'wb'))
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/train_test_real.sav'
train_test_real = pickle.load(open(filename, 'rb'))
```

Featurization 3

Duplicate Count featurization on train and test data

In [0]:

```
##https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/88974
from tqdm import tqdm
for f in tqdm(features):
    count=train_test_real[f].value_counts(dropna=True)
    train_test_real[f+'dup_count'] = train_test_real[f].map(count).map(lambda x:min(10,x)).astype(n
p.uint8)
    train_test_real[f + '_dup_value_2'] = train_test_real[f]* (train_test_real[f + 'dup_count'].map
(lambda x:int(x>2))).astype(np.float32)
    train_test_real[f + '_dup_value_4'] = train_test_real[f]* (train_test_real[f + 'dup_count'].map
(lambda x:int(x>4))).astype(np.float32)
    test[f+'dup_count'] =  test[f].map(count).map(lambda x:min(10,x)).astype(np.uint8)
    test[f + '_dup_value_2'] = test[f]* (test[f + 'dup_count'].map(lambda x:int(x>2))).astype(np.flo
at32)
    test[f + '_dup_value_4'] = test[f]* (test[f + 'dup_count'].map(lambda x:int(x>4))).astype(np.flo
at32)
```

```
100%|██████████| 200/200 [02:23<00:00,  1.08it/s]
```

In [0]:

```
##https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/88974
train_real = train_test_real.loc[train_test_real['target'].isnull()==False,:]
test_real = train_test_real.loc[train_test_real['target'].isnull()==True,:]
print('Shape of train data after featurization : '+str(train_real.shape))
print('Shape of test data featurization : '+str(test_real.shape))
```

```
Shape of train data after featurization : (200000, 801)
Shape of test data featurization : (100000, 801)
```

In [0]:

```
##https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/88974
for f in tqdm(features):
    train_real[f+'distance_of_mean'] = train_real[f]-train_test_real[f].mean()
    train_real[f+'distance_of_mean'] = (train_real[f+'distance_of_mean']* train_real[f+'dup_count']
.map(lambda x:int(x>1))).astype(np.float32)
    test[f+'distance_of_mean'] = test[f]-train_test_real[f].mean()
    test[f+'distance_of_mean'] = (test[f+'distance_of_mean']* test[f + 'dup_count'].map(lambda x:in
t(x>1))).astype(np.float32)
```

```
100%|██████████| 200/200 [00:38<00:00,  4.40it/s]
```

In [0]:

```
test=test.drop(['target'],axis=1)
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/test_real.sav'
pickle.dump(test, open(filename, 'wb'))
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/train_real.sav'
pickle.dump(train_real, open(filename, 'wb'))
```

In [0]:

```
del train_test_real
del real_te
del fake_te
del train
del test
import gc
gc.collect()
```

Out[0]:

0

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/train_real.sav'
train_real = pickle.load(open(filename, 'rb'))
```

In [0]:

```
import pickle
filename = '/content/drive/My Drive/proj_1/test_real.sav'
test = pickle.load(open(filename, 'rb'))
```

In [0]:

```
train_real.head()
```

Out[0]:

	target	var_0	var_1	var_10	var_100	var_101	var_102	var_103	var_104	var_105	var_106	var_107	var_108	v
ID_code														
train_0	0.0	8.9255	-6.7863	2.9252	9.4763	13.3102	26.5376	1.4403	14.7100	6.0454	9.5426	17.1554	14.1104	2
train_1	0.0	11.5006	-4.1473	-0.4032	-13.6950	8.4068	35.4734	1.7093	15.1866	2.6227	7.3412	32.0888	13.9550	1
train_2	0.0	8.6093	-2.7457	-0.3249	-0.3939	12.6317	14.8863	1.3854	15.0284	3.9995	5.3683	8.6273	14.1963	2
train_3	0.0	11.0604	-2.1518	2.3061	-19.8592	22.5316	18.6129	1.3512	9.3291	4.2835	10.3907	7.0874	14.3256	1
train_4	0.0	9.8369	-1.4834	-9.4458	-22.9264	12.3562	17.3410	1.6940	7.1179	5.1934	8.8230	10.6617	14.0837	2

5 rows × 1001 columns

In [0]:

```
from tqdm import tqdm
col=[]
for i in tqdm(range(200)):
    m='var_'+str(i)+'dup'
    col.append(m)
test=test.drop(col,axis=1)
```

```
100%|██████████| 200/200 [00:00<00:00, 557753.19it/s]
```

In [0]:

```
target=train_real['target']
train=train_real.drop(['target'],axis=1)
```

In [0]:

```
X_train,y_train,X_cv,y_cv=split_train_test(train_real)
```

```
Train data shape : (160000, 1000)
CV data shape : (40000, 1000)
```

Naive Bayes with 1000 features (200 original,600 duplicate count,200 distance of mean)

In [0]:

```
#https://www.featureranking.com/tutorials/machine-learning-tutorials/sk-part-3-cross-validation-an
d-hyperparameter-tuning/
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
import numpy as np

gnb = GaussianNB(priors = [0.5,0.5])
params_NB = {'var_smoothing': np.logspace(1,-5, num=100)}
best_param=grid_search(gnb,params_NB,5,train,target,3)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=3)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=3)]: Done  44 tasks      | elapsed:  1.5min
[Parallel(n_jobs=3)]: Done 194 tasks      | elapsed:  6.1min
[Parallel(n_jobs=3)]: Done 444 tasks      | elapsed: 13.8min
[Parallel(n_jobs=3)]: Done 500 out of 500 | elapsed: 15.5min finished
```

```
best param : {'var_smoothing': 0.014174741629268049}
best score : 0.8576351462258034
```

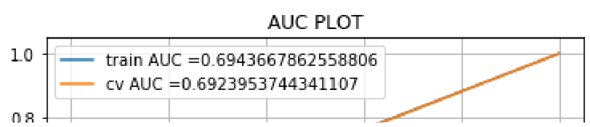
In [0]:

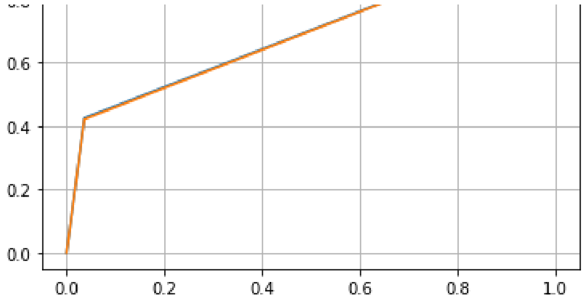
```
best_feat=test.columns
y_train_pred,y_cv_pred=baseline_model(best_param,X_train,y_train,X_cv,y_cv,test,best_feat)
```

```
train auc score : 0.6987652573603904
cv auc score : 0.6969209677017911
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```





Kaggle Score

1. Private Score: 0.50000
2. Public Score : 0.50000

SVM with 1000 features (200 original,600 duplicate count,200 distance of mean)

In [0]:

```
from sklearn.linear_model import SGDClassifier
params = {'alpha': np.logspace(4,-9, num=100)}
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',max_iter=100,tol=1e-3,class_weight = 'balanced')
best_param=grid_search(sgd,params,5,X_train,y_train,6)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.
[Parallel(n_jobs=6)]: Done  38 tasks      | elapsed:  2.2min
[Parallel(n_jobs=6)]: Done 188 tasks      | elapsed: 15.5min
[Parallel(n_jobs=6)]: Done 438 tasks      | elapsed: 48.6min
[Parallel(n_jobs=6)]: Done 500 out of 500 | elapsed: 56.5min finished
```

```
best param : {'alpha': 9.770099572992247e-05}
best score : 0.8665770304658803
```

```
/home/roydeepak2406/.local/lib/python3.5/site-
packages/sklearn/linear_model/stochastic_gradient.py:603: ConvergenceWarning: Maximum number of it
eration reached before convergence. Consider increasing max_iter to improve the fit.
  ConvergenceWarning)
```

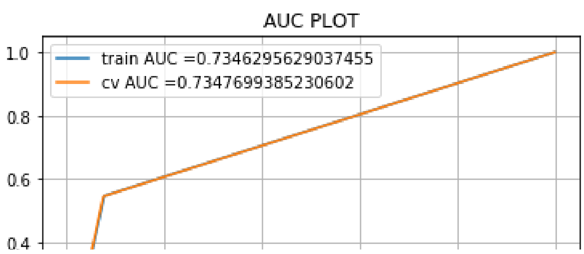
In [0]:

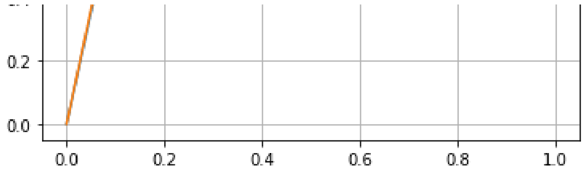
```
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=400,tol=1e-3,
alpha = best_param['alpha'])
sgd.fit(X_train ,y_train)
y_train_pred=sgd.predict(X_train)
y_cv_pred = sgd.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7346295629037455
cv auc score : 0.7347699385230602
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```





In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=sgd.predict(test)
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_sgd.csv", index=False)
```

Kaggle Score

1. Private Score: 0.52990
2. Public Score : 0.53118

select k best features from 1000 features

In [0]:

```
train.shape[1]
```

Out[0]:

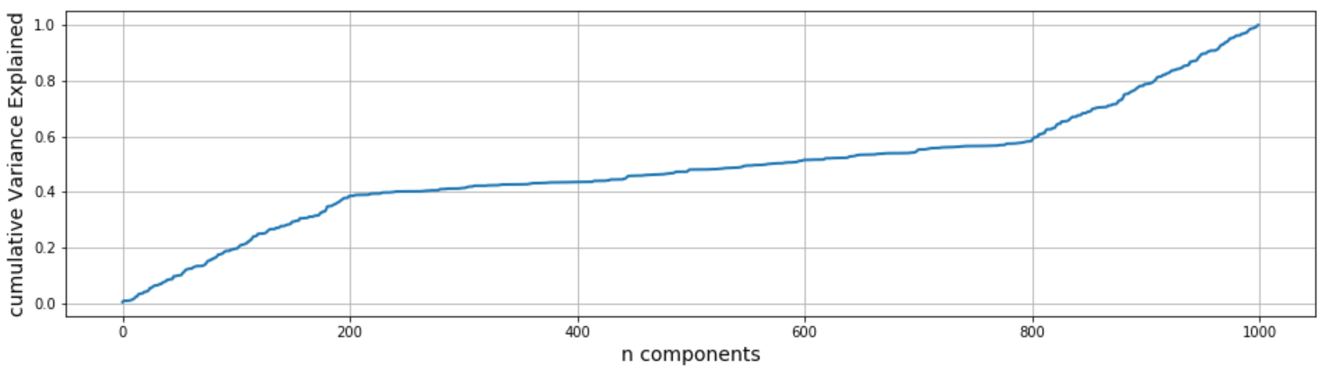
```
1000
```

In [0]:

```
#https://stackoverflow.com/questions/39839112/the-easiest-way-for-getting-feature-names-after-running-selectkbest-in-scikit-le
from sklearn.feature_selection import SelectKBest, f_classif
k_best = SelectKBest(f_classif, k=(train.shape[1])-1)
k=k_best.fit(train,target)
features = k.transform(train)
```

In [0]:

```
import matplotlib.pyplot as plt
cumVarianceExplained = np.cumsum( k.scores_ )/np.sum(k.scores_)
plt.figure( figsize=(16, 4))
plt.plot( cumVarianceExplained, linewidth = 2 )
plt.grid()
plt.xlabel('n components',size=14)
plt.ylabel('cumulative Variance Explained',size=14)
plt.show()
```



In [0]:

```
from sklearn.feature_selection import SelectKBest, f_classif
```

```
selector = SelectKBest(f_classif,k=600)
selector.fit(train,target)
cols = selector.get_support(indices=True)
data_kbest = train.iloc[:,cols]
```

In [0]:

```
best_feat=[]
best_feat=data_kbest.columns
print(best_feat)
```

```
Index(['var_0', 'var_1', 'var_101', 'var_102', 'var_104', 'var_105', 'var_106',
       'var_107', 'var_108', 'var_109',
       ...
       'var_190distance_of_mean', 'var_191distance_of_mean',
       'var_192distance_of_mean', 'var_193distance_of_mean',
       'var_194distance_of_mean', 'var_195distance_of_mean',
       'var_196distance_of_mean', 'var_197distance_of_mean',
       'var_198distance_of_mean', 'var_199distance_of_mean'],
      dtype='object', length=600)
```

NB with Top 600 features

In [0]:

```
#https://www.featureranking.com/tutorials/machine-learning-tutorials/sk-part-3-cross-validation-an
d-hyperparameter-tuning/
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
import numpy as np

gnb = GaussianNB(priors = [0.5,0.5])
params_NB = {'var_smoothing': np.logspace(1,-5, num=100)}
best_param=grid_search(gnb,params_NB,5,train[best_feat],target,3)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=3)]: Using backend LokyBackend with 3 concurrent workers.
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_executor.py:706: UserWarning:
A worker stopped while some jobs were given to the executor. This can be caused by a too short wor
ker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=3)]: Done  44 tasks      | elapsed:  1.5min
[Parallel(n_jobs=3)]: Done 194 tasks      | elapsed:  6.0min
[Parallel(n_jobs=3)]: Done 444 tasks      | elapsed: 13.4min
[Parallel(n_jobs=3)]: Done 500 out of 500 | elapsed: 15.0min finished
```

```
best param : {'var_smoothing': 0.014174741629268049}
best score : 0.8587853222000852
```

In [0]:

```
best_param['var_smoothing']=0.014174741629268049
```

In [0]:

```
from sklearn.model_selection import train_test_split
X_train, X_cv, y_train, y_cv = train_test_split(train[best_feat], target, test_size = 0.20, stratif
y=target)
print('Train data shape : '+str(X_train.shape))
print('CV data shape : '+str(X_cv.shape))
```

```
Train data shape : (160000, 600)
CV data shape : (40000, 600)
```

In [0]:

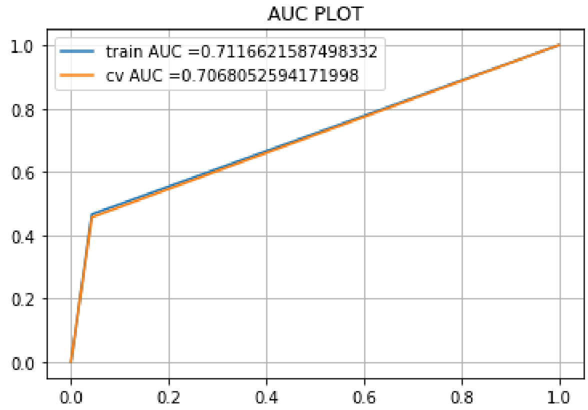
In [0]:

```
y_train_pred,y_cv_pred=baseline_model(best_param,X_train,y_train,X_cv,y_cv,test,best_feat)
```

```
train auc score : 0.7116621587498332
cv auc score : 0.7068052594171998
```

In [0]:

```
auc_plot(y_train,y_train_pred,y_cv,y_cv_pred)
```



Kaggle Score

1. Private Score: 0.71324
2. Public Score : 0.7111

SVM with Top 600 features

In [0]:

```
from sklearn.linear_model import SGDClassifier
params = {'alpha': np.logspace(2,-6, num=50)}
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',max_iter=100,tol=1e-3,class_weight = 'balanced')
best_param=grid_search(sgd,params,5,X_train,y_train,3)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

```
[Parallel(n_jobs=3)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=3)]: Done  44 tasks      | elapsed:  2.1min
[Parallel(n_jobs=3)]: Done 194 tasks      | elapsed: 24.5min
[Parallel(n_jobs=3)]: Done 250 out of 250 | elapsed: 35.4min finished
```

```
best param : {'alpha': 0.05428675439323865}
best score : 0.8731166692114062
```

In [0]:

```
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=400,tol=1e-3,
alpha = best_param['alpha'])
sgd.fit(X_train ,y_train)
y_train_pred=sgd.predict(X_train)
y_cv_pred = sgd.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7968034203140598
cv auc score : 0.7887713323322243
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=sgd.predict(test[best_feat])
```

```
predictions=sgd.predict(test[best_feat])
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_lgb.csv", index=False)
```

Kaggle Score

1. Private Score: 0.79074
2. Public Score : 0.79048

Decision Tree with Top 600 features

In [0]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(class_weight = 'balanced')

params={
        'max_depth' :[10,100,500,2000,5000,10000,20000],
        'min_samples_split':[10,1000,2000,5000]
}
best_param=grid_search(dtc,params,5,X_train,y_train,8)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

```
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_executor.py:706: UserWarning:
A worker stopped while some jobs were given to the executor. This can be caused by a too short wor
ker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed: 34.4min
[Parallel(n_jobs=8)]: Done 140 out of 140 | elapsed: 176.6min finished
```

```
best param : {'max_depth': 10000, 'min_samples_split': 2000}
best score : 0.6986127658283461
```

In [0]:

```
best_param={'max_depth': 10000, 'min_samples_split': 2000}
```

In [0]:

```
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
dtc =
DecisionTreeClassifier(max_depth=best_param['max_depth'],min_samples_split=best_param['min_samples_
split'],class_weight = 'balanced')
dtc.fit(X_train, y_train)
y_train_pred = dtc.predict(X_train)
y_cv_pred=dtc.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.689479729243881
cv auc score : 0.6457789568002124
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=dtc.predict(test[best_feat])
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_dtc.csv", index=False)
```

Kaggle Score

1. Private Score: 0.64147
2. Public Score : 0.64029

Voting Classifier with top 600 features

In [0]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
import numpy as np
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
gnb = GaussianNB(priors = [0.5,0.5],var_smoothing=0.014174741629268049)
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=1000,tol=1e-3
,alpha = 0.05428675439323865)
dtc =
DecisionTreeClassifier(max_depth=best_param['max_depth'],min_samples_split=best_param['min_samples_
split'],class_weight = 'balanced')
vcf = VotingClassifier(estimators=[('gnb', gnb), ('sgd', sgd), ('dtc', dtc)], voting='hard')
vcf = vcf.fit(X_train, y_train)

y_train_pred=vcf.predict(X_train)
y_cv_pred = vcf.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7875334515508833
cv auc score : 0.76589108376959
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=vcf.predict(test[best_feat])
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_vcf.csv", index=False)
```

Kaggle Score

1. Private Score: 0.77147
2. Public Score : 0.77101

Stacking Classifier with Top 600 features

In [0]:

```
#http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/
import warnings
warnings.filterwarnings("ignore")
from sklearn.tree import DecisionTreeClassifier
from mlxtend.classifier import StackingClassifier
gnb = GaussianNB(priors = [0.5,0.5],var_smoothing=0.014174741629268049)
sgd = SGDClassifier(loss = 'hinge', penalty = 'l2',class_weight = 'balanced',max_iter=1000,tol=1e-3
,alpha = 0.05428675439323865)
dtc =
DecisionTreeClassifier(max_depth=best_param['max_depth'],min_samples_split=best_param['min_samples_
split'],class_weight = 'balanced')

clf = StackingClassifier(classifiers=[sgd,dtc], meta_classifier=gnb)

clf.fit(X_train,y_train)
y_train_pred=clf.predict(X_train)
y_cv_pred = clf.predict(X_cv)
print('train auc score : '+str(roc_auc_score(y_train,y_train_pred)))
print('cv auc score : '+str(roc_auc_score(y_cv,y_cv_pred)))
```

```
train auc score : 0.7962259843797057
```

```
cv auc score : 0.7892530814521057
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
predictions=clf.predict(test[best_feat])
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_stc.csv", index=False)
```

Kaggle Score

1. Private Score: 0.79332
2. Public Score : 0.79205

lightgbm with no augmentation and 600 features

In [0]:

```
#basic tools
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
import os
import numpy as np
import pandas as pd
import warnings

#tuning hyperparameters
from bayes_opt import BayesianOptimization
from skopt  import BayesSearchCV

#graph, plots
import matplotlib.pyplot as plt
import seaborn as sns

#building models
import lightgbm as lgb
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
import time
import sys

#metrics
from sklearn.metrics import roc_auc_score, roc_curve
warnings.simplefilter(action='ignore', category=FutureWarning)

def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=3, random_seed=6, output_process=False):
    # prepare data
    train_data = lgb.Dataset(data=X, label=y, free_raw_data=False)
    # parameters
    def lgb_eval(learning_rate,bagging_freq,bagging_seed,lambda_l1,lambda_l2,min_gain_to_split,min_child_weight,num_leaves, feature_fraction, bagging_fraction, max_depth, max_bin,min_data_in_leaf,min_sum_hessian_in_leaf,subsample):
        params = {'application':'binary', 'metric':'auc','tree_learner': 'serial','verbosity': -1,'boost': 'gbdt'}
        params['learning_rate'] = max(min(learning_rate, 1), 0)
        params['bagging_freq']=int(bagging_freq)
        params['bagging_seed']=int(bagging_seed)
        params['lambda_l1']=max(min(lambda_l1, 1), 0)
        params['lambda_l2']=max(min(lambda_l1, 1), 0)
        params['min_gain_to_split']=min_gain_to_split
        params['min_child_weight']=min_child_weight
        params['num_leaves'] = int(round(num_leaves))
        params['feature_fraction'] = max(min(feature_fraction, 1), 0)
        params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
        params['max_depth'] = int(round(max_depth))
        params['max_bin'] = int(round(max_depth))
        params['min_data_in_leaf'] = int(round(min_data_in_leaf))
        params['min_sum_hessian_in_leaf'] = min_sum_hessian_in_leaf
        params['subsample'] = max(min(subsample, 1), 0)
```

```
        #evaluate cv for above paramters
        cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, ve
rbose_eval =200, metrics=['auc'])

        #return max mean for multiple folds
        return max(cv_result['auc-mean'])

    lgbBO = BayesianOptimization(lgb_eval, {'learning_rate': (0.01, 1.0),
                                            'bagging_freq' :(4,10),
                                            'bagging_seed':(5,10),
                                            'lambda_l1': (0.01,1),
                                            'lambda_l2': (0.01,1),
                                            'min_gain_to_split':(.01,0.9),
                                            'min_child_weight':(5,20),
                                            'num_leaves': (10, 80),
                                            'feature_fraction': (0.01, 0.9),
                                            'bagging_fraction': (0.2, 1),
                                            'max_depth': (5, 80),
                                            'max_bin':(20,150),
                                            'min_data_in_leaf': (20, 80),
                                            'min_sum_hessian_in_leaf':(0,100),
                                           'subsample': (0.01, 1.0)}, random_state=200)


    #n_iter: How many steps of bayesian optimization you want to perform. The more steps the more
likely to find a good maximum you are.
    #init_points: How many steps of random exploration you want to perform. Random exploration can
help by diversifying the exploration space.

    lgbBO.maximize(init_points=init_round, n_iter=opt_round)

    model_auc=[]
    for model in range(len( lgbBO.res)):
        model_auc.append(lgbBO.res[model]['target'])

    # return best parameters
    return lgbBO.res[pd.Series(model_auc).idxmax()]['target'],lgbBO.res[pd.Series(model_auc).idxmax
()]['params']

opt_params = bayes_parameter_opt_lgb(train[best_feat], target, init_round=5, opt_round=20, n_folds=
5, random_seed=6)
```

```
|   iter    |  target   | baggin... | baggin... | baggin... | featur... | lambda_l1 | lambda_l2 |
learni... |  max_bin  | max_depth | min_ch... | min_da... | min_ga... | min_su... | num_le... | su
bsample |
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
|  1        |  0.8987   |  0.9581   |  5.359    |  7.972    |  0.3912   |  0.7665   |  0.01283  |
0.3638   |  138.3    |  39.21    |  19.73    |  72.04    |  0.8876   |  92.33    |  31.26    |  0.8
76   |
|  2        |  0.8749   |  0.297    |  8.713    |  6.253    |  0.09547  |  0.9441   |  0.8278   |
.5231   |  133.6    |  48.38    |  11.75    |  60.81    |  0.3833   |  64.04    |  52.61    |  0.11
4   |
|  3        |  0.8667   |  0.9728   |  8.552    |  8.398    |  0.3031   |  0.03619  |  0.8056   |
0.9731   |  56.78    |  53.68    |  8.984    |  31.26    |  0.383    |  66.6     |  46.22    |  0.8
59   |
|  4        |  0.8704   |  0.3368   |  9.869    |  6.506    |  0.2169   |  0.7937   |  0.8851   |
0.9874   |  22.09    |  20.34    |  14.68    |  50.64    |  0.1972   |  30.89    |  11.23    |  0.6
84   |
|  5        |  0.8974   |  0.6229   |  7.668    |  6.496    |  0.802    |  0.7224   |  0.9534   |
.4031   |  148.5    |  38.94    |  15.92    |  40.65    |  0.8254   |  6.442    |  10.65    |  0.48
5   |
|  6        |  0.8937   |  0.7865   |  9.677    |  8.216    |  0.3855   |  0.466    |  0.3184   |
.3855   |  133.7    |  6.277    |  8.99     |  21.76    |  0.6467   |  99.48    |  22.9     |  0.57
9   |
|  7        |  0.861    |  0.2222   |  4.746    |  8.168    |  0.7697   |  0.0464   |  0.6603   |
.06009   |  105.0    |  79.43    |  6.57     |  78.83    |  0.3883   |  96.21    |  11.46    |  0.89
9   |
|  8        |  0.8948   |  0.3242   |  7.453    |  5.576    |  0.6542   |  0.3531   |  0.6822   |
0.4271   |  145.0    |  62.89    |  15.16    |  22.73    |  0.13     |  97.06    |  11.71    |  0.4
37   |
|  9        |  0.8876   |  0.3135   |  4.467    |  6.492    |  0.6108   |  0.8964   |  0.767    |
.2672   |  141.2    |  8.63     |  19.97    |  40.86    |  0.08926  |  4.022    |  28.58    |  0.65
|
|  10       |  0.8697   |  0.847    |  9.212    |  5.163    |  0.6668   |  0.6759   |  0.8964   |
.8291   |  25.53    |  15.94    |  18.53    |  64.15    |  0.565    |  94.42    |  76.3     |  0.22
4   |
```

```
4    |
|  11       |  0.8681   |  0.3782   |  4.593    |  9.089    |  0.3471   |  0.2259   |  0.7976   |
0.9962   |  20.48    |  20.85    |  19.97    |  26.99    |  0.784    |  96.04    |  13.75    |  0.1
96   |
|  12       |  0.8943   |  0.8845   |  4.159    |  7.833    |  0.4657   |  0.3483   |  0.4703   |
0.149    |  135.6    |  11.29    |  10.75    |  24.88    |  0.5751   |  98.15    |  26.83    |  0.3
31   |
|  13       |  0.8396   |  0.7332   |  7.725    |  8.292    |  0.5318   |  0.8732   |  0.4971   |
0.9449   |  20.05    |  78.2     |  19.51    |  65.04    |  0.7722   |  0.2048   |  75.72    |  0.2
42   |
|  14       |  0.8714   |  0.6956   |  5.252    |  7.322    |  0.8282   |  0.6611   |  0.3656   |
0.07599  |  147.7    |  21.74    |  19.75    |  68.35    |  0.316    |  66.85    |  12.64    |  0.5
07   |
|  15       |  0.874    |  0.2027   |  5.3      |  9.991    |  0.4537   |  0.6133   |  0.9602   |
.196    |  21.47    |  23.47    |  15.58    |  21.01    |  0.8206   |  1.068    |  72.69    |  0.54
9    |
|  16       |  0.8431   |  0.7175   |  5.081    |  8.807    |  0.5906   |  0.7642   |  0.6914   |
0.02612  |  132.4    |  13.92    |  15.88    |  23.22    |  0.6831   |  98.06    |  22.69    |  0.0
437  |
|  17       |  0.8786   |  0.31     |  8.467    |  8.236    |  0.6109   |  0.4845   |  0.3144   |
.5526   |  20.07    |  63.08    |  5.856    |  33.75    |  0.3334   |  99.87    |  74.25    |  0.43
9    |
|  18       |  0.8659   |  1.0      |  4.0      |  8.139    |  0.01     |  0.01     |  0.01     |
.2654   |  150.0    |  28.56    |  5.0      |  20.0     |  0.9      |  0.0      |  80.0     |  1.0
|
|  19       |  0.8722   |  0.9849   |  10.0     |  5.0      |  0.07194  |  0.01328  |  0.01     |
.9955   |  150.0    |  10.91    |  5.0      |  21.62    |  0.04826  |  100.0    |  80.0     |  1.0
|
|  20       |  0.8566   |  1.0      |  4.0      |  5.0      |  0.889    |  0.9977   |  0.01621  |
.0      |  95.08    |  5.0      |  5.0      |  80.0     |  0.325    |  0.0      |  36.74    |  1.0
|
|  21       |  0.8848   |  0.7917   |  4.43     |  9.164    |  0.7902   |  0.627    |  0.3139   |
.7845   |  23.37    |  66.32    |  5.9      |  20.2     |  0.4564   |  16.88    |  14.95    |  0.25
9    |
|  22       |  0.8745   |  0.3818   |  5.595    |  6.008    |  0.478    |  0.8357   |  0.4149   |
.6787   |  146.0    |  75.3     |  9.064    |  76.44    |  0.3677   |  97.14    |  49.62    |  0.43
8    |
|  23       |  0.8745   |  0.6744   |  4.778    |  6.47     |  0.5969   |  0.3356   |  0.553    |
.8908   |  142.8    |  6.887    |  5.772    |  71.94    |  0.6902   |  89.02    |  75.81    |  0.58
7    |
|  24       |  0.8797   |  0.3445   |  9.804    |  8.687    |  0.7134   |  0.908    |  0.4951   |
.7693   |  24.76    |  44.22    |  9.146    |  75.6     |  0.1705   |  95.82    |  11.69    |  0.73
5    |
|  25       |  0.8899   |  0.2425   |  6.036    |  5.394    |  0.8281   |  0.4522   |  0.01296  |
0.2957   |  139.5    |  19.69    |  6.766    |  61.39    |  0.5611   |  99.49    |  19.36    |  0.8
5    |
===================================================================================================
===================================================================================================
```

In [0]:

```
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
opt_params[1]['num_leaves'] = int(round(opt_params[1]['num_leaves']))
opt_params[1]['max_depth'] = int(round(opt_params[1]['max_depth']))
opt_params[1]['bagging_freq'] = int(round(opt_params[1]['bagging_freq']))
opt_params[1]['min_data_in_leaf'] = int(round(opt_params[1]['min_data_in_leaf']))
opt_params[1]['max_bin'] = int(round(opt_params[1]['max_bin']))
opt_params[1]['bagging_freq']=int(round(opt_params[1]['bagging_freq']))
opt_params[1]['bagging_seed']=int(round(opt_params[1]['bagging_seed']))
opt_params[1]['objective']='binary'
opt_params[1]['metric']='auc'
opt_params[1]['is_unbalance']=True
opt_params[1]['boost_from_average']=False
opt_params1=opt_params[1]
opt_params1
```

Out[0]:

```
{'bagging_fraction': 0.9581058054813363,
 'bagging_freq': 5,
 'bagging_seed': 8,
 'boost_from_average': False,
 'feature_fraction': 0.39119472958742835,
 'is_unbalance': True,
 'lambda_l1': 0.7664992796883313,
```

```
 'lambda_l2': 0.012831985764133342,
 'learning_rate': 0.3638494444744881,
 'max_bin': 138,
 'max_depth': 39,
 'metric': 'auc',
 'min_child_weight': 19.727040637374927,
 'min_data_in_leaf': 72,
 'min_gain_to_split': 0.8875644851478631,
 'min_sum_hessian_in_leaf': 92.32667066664217,
 'num_leaves': 31,
 'objective': 'binary',
 'subsample': 0.84764245541438}
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
num_folds=5
folds = StratifiedKFold(n_splits=num_folds,shuffle=True, random_state=2357)
oof = np.zeros(len(train))
getVal = np.zeros(len(train))
predictions = np.zeros(len(target))
new_features= [i for i in train[best_feat].columns]
```

In [0]:

```
len(new_features)
```

Out[0]:

```
600
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
%%time
import lightgbm as lgb
print('Light GBM Model')
for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values, target.values)):

    X_train, y_train = train.iloc[trn_idx][new_features], target.iloc[trn_idx]
    X_valid, y_valid = train.iloc[val_idx][new_features], target.iloc[val_idx]

    print("Fold idx:{}".format(fold_ + 1))
    trn_data = lgb.Dataset(X_train, label=y_train)
    val_data = lgb.Dataset(X_valid, label=y_valid)


    clf1 = lgb.train(opt_params1, trn_data, 100000, valid_sets = [trn_data, val_data],
verbose_eval=1000,
                    early_stopping_rounds = 3000)
    oof[val_idx] = clf1.predict(train.iloc[val_idx][new_features],
num_iteration=clf1.best_iteration)
    getVal[val_idx]+= clf1.predict(train.iloc[val_idx][new_features],
num_iteration=clf1.best_iteration) / folds.n_splits
    predictions += clf1.predict(test[new_features], num_iteration=clf1.best_iteration) /
folds.n_splits
    clf1.save_model('/content/drive/My Drive/model_600_best_iteration_{}.sav'.format(fold_), num_i
teration=clf1.best_iteration)
```

```
Light GBM Model
Fold idx:1
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.999975 valid_1's auc: 0.90146
[2000] training's auc: 0.999975 valid_1's auc: 0.90146
[3000] training's auc: 0.999975 valid_1's auc: 0.90146
Early stopping, best iteration is:
[749] training's auc: 0.999975 valid_1's auc: 0.901465
Fold idx:2
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.99999 valid_1's auc: 0.90117
[2000] training's auc: 0.99999 valid_1's auc: 0.90117
[3000] training's auc: 0.99999 valid_1's auc: 0.90117
Early stopping, best iteration is:
```

```
[718] training's auc: 0.999989 valid_1's auc: 0.901194
Fold idx:3
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.999989 valid_1's auc: 0.903398
[2000] training's auc: 0.999989 valid_1's auc: 0.903398
[3000] training's auc: 0.999989 valid_1's auc: 0.903398
Early stopping, best iteration is:
[793] training's auc: 0.999989 valid_1's auc: 0.903403
Fold idx:4
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.999984 valid_1's auc: 0.89895
[2000] training's auc: 0.999984 valid_1's auc: 0.89895
[3000] training's auc: 0.999984 valid_1's auc: 0.89895
Early stopping, best iteration is:
[696] training's auc: 0.999983 valid_1's auc: 0.898982
Fold idx:5
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.999983 valid_1's auc: 0.90266
[2000] training's auc: 0.999983 valid_1's auc: 0.90266
[3000] training's auc: 0.999983 valid_1's auc: 0.90266
Early stopping, best iteration is:
[733] training's auc: 0.999981 valid_1's auc: 0.90268
CPU times: user 1h 17min 19s, sys: 8.23 s, total: 1h 17min 27s
Wall time: 20min 13s
```

In [0]:

```
from sklearn.metrics import roc_auc_score, roc_curve
print("\n >> CV score: {:<8.5f}".format(roc_auc_score(target, oof)))
```

```
 >> CV score: 0.90153
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
test1=test1.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/submission_600.csv", index=False)
```

Kaggle Score

1. Private Score: 0.91352
2. Public Score : 0.91662

The above private leaderboard scores is within top 5 percent.

lightgbm 600 features with augmentation

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
num_folds=5
folds = StratifiedKFold(n_splits=num_folds,shuffle=True, random_state=2357)
oof = np.zeros(len(train))
getVal = np.zeros(len(train))
predictions = np.zeros(len(target))
new_features= [i for i in train[best_feat].columns]
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
%%time
import lightgbm as lgb
print('Light GBM Model')
for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values, target.values)):

    X_train, y_train = train.iloc[trn_idx][new_features], target.iloc[trn_idx]
```

```
    X_valid, y_valid = train.iloc[val_idx][new_features], target.iloc[val_idx]


    X_tr, y_tr = augment(X_train.values, y_train.values)
    X_tr = pd.DataFrame(X_tr)

    print("Fold idx:{}".format(fold_ + 1))
    trn_data = lgb.Dataset(X_tr, label=y_tr)
    val_data = lgb.Dataset(X_valid, label=y_valid)


    clf1 = lgb.train(opt_params1, trn_data, 100000, valid_sets = [trn_data, val_data],
verbose_eval=1000,
                    early_stopping_rounds = 3000)
    oof[val_idx] = clf1.predict(train.iloc[val_idx][new_features],
num_iteration=clf1.best_iteration)
    getVal[val_idx]+= clf1.predict(train.iloc[val_idx][new_features],
num_iteration=clf1.best_iteration) / folds.n_splits
    predictions += clf1.predict(test[new_features], num_iteration=clf1.best_iteration) /
folds.n_splits
    clf1.save_model('/content/drive/My Drive/model_600_best_iteration_{}.sav'.format(fold_), num_i
teration=clf1.best_iteration)
```

```
Light GBM Model
Fold idx:1
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.99995 valid_1's auc: 0.908006
[2000] training's auc: 0.999981 valid_1's auc: 0.908467
[3000] training's auc: 0.999981 valid_1's auc: 0.908467
Early stopping, best iteration is:
[260] training's auc: 0.989363 valid_1's auc: 0.910225
Fold idx:2
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.999947 valid_1's auc: 0.906214
[2000] training's auc: 0.999982 valid_1's auc: 0.907211
[3000] training's auc: 0.999982 valid_1's auc: 0.907211
Early stopping, best iteration is:
[230] training's auc: 0.98743 valid_1's auc: 0.907493
Fold idx:3
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.999933 valid_1's auc: 0.910988
[2000] training's auc: 0.999968 valid_1's auc: 0.911782
[3000] training's auc: 0.999968 valid_1's auc: 0.911782
Early stopping, best iteration is:
[240] training's auc: 0.987877 valid_1's auc: 0.912042
Fold idx:4
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.999955 valid_1's auc: 0.907444
[2000] training's auc: 0.999982 valid_1's auc: 0.908593
[3000] training's auc: 0.999982 valid_1's auc: 0.908593
[4000] training's auc: 0.999982 valid_1's auc: 0.908593
Early stopping, best iteration is:
[1308] training's auc: 0.999982 valid_1's auc: 0.908619
Fold idx:5
Training until validation scores don't improve for 3000 rounds.
[1000] training's auc: 0.999938 valid_1's auc: 0.90903
[2000] training's auc: 0.999975 valid_1's auc: 0.909777
[3000] training's auc: 0.999975 valid_1's auc: 0.909777
Early stopping, best iteration is:
[238] training's auc: 0.98792 valid_1's auc: 0.910785
CPU times: user 2h 55min 5s, sys: 24.9 s, total: 2h 55min 30s
Wall time: 53min 43s
```

In [0]:

```
from sklearn.metrics import roc_auc_score, roc_curve
print("\n >> CV score: {:<8.5f}".format(roc_auc_score(target, oof)))
```

```
 >> CV score: 0.90359
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
```

```
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_600_1.csv", index=False)
```

Kaggle Score

1. Private Score: 0.91485
2. Public Score : 0.91802

The above private leaderboard scores is within top 5 percent.

Light gbm model with 1000 features (200 original,600 duplicate count,200 distance of mean)

In [0]:

```
#basic tools
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
import os
import numpy as np
import pandas as pd
import warnings

#tuning hyperparameters
from bayes_opt import BayesianOptimization
from skopt  import BayesSearchCV

#graph, plots
import matplotlib.pyplot as plt
import seaborn as sns

#building models
import lightgbm as lgb
import xgboost as xgb
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
import time
import sys

#metrics
from sklearn.metrics import roc_auc_score, roc_curve
warnings.simplefilter(action='ignore', category=FutureWarning)

def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=3, random_seed=6, output_pro
cess=False):
    # prepare data
    train_data = lgb.Dataset(data=X, label=y, free_raw_data=False)
    # parameters
    def
lgb_eval(learning_rate,bagging_freq,bagging_seed,lambda_l1,lambda_l2,min_gain_to_split,min_child_we
ight,num_leaves, feature_fraction, bagging_fraction, max_depth,
max_bin,min_data_in_leaf,min_sum_hessian_in_leaf,subsample):
        params = {'application':'binary', 'metric':'auc','tree_learner': 'serial','verbosity': -1,'
boost': 'gbdt'}
        params['learning_rate'] = max(min(learning_rate, 1), 0)
        params['bagging_freq']=int(bagging_freq)
        params['bagging_seed']=int(bagging_seed)
        params['lambda_l1']=max(min(lambda_l1, 1), 0)
        params['lambda_l2']=max(min(lambda_l1, 1), 0)
        params['min_gain_to_split']=min_gain_to_split
        params['min_child_weight']=min_child_weight
        params['num_leaves'] = int(round(num_leaves))
        params['feature_fraction'] = max(min(feature_fraction, 1), 0)
        params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
        params['max_depth'] = int(round(max_depth))
        params['max_bin'] = int(round(max_depth))
        params['min_data_in_leaf'] = int(round(min_data_in_leaf))
        params['min_sum_hessian_in_leaf'] = min_sum_hessian_in_leaf
        params['subsample'] = max(min(subsample, 1), 0)
        #evaluate cv for above paramters
        cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, ve
rbose_eval =200, metrics=['auc'])

        #return max mean for multiple folds
        return max(cv_result['auc-mean'])
```

```
    lgbBO = BayesianOptimization(lgb_eval, {'learning_rate': (0.01, 1.0),
                                             'bagging_freq' :(4,10),
                                             'bagging_seed':(5,10),
                                             'lambda_l1': (0.01,1),
                                             'lambda_l2': (0.01,1),
                                             'min_gain_to_split':(.01,0.9),
                                             'min_child_weight':(5,20),
                                             'num_leaves': (10, 80),
                                             'feature_fraction': (0.01, 0.9),
                                             'bagging_fraction': (0.2, 1),
                                             'max_depth': (5, 80),
                                             'max_bin':(20,150),
                                             'min_data_in_leaf': (20, 80),
                                             'min_sum_hessian_in_leaf':(0,100),
                                            'subsample': (0.01, 1.0)}, random_state=200)


    #n_iter: How many steps of bayesian optimization you want to perform. The more steps the more
likely to find a good maximum you are.
    #init_points: How many steps of random exploration you want to perform. Random exploration can
help by diversifying the exploration space.

    lgbBO.maximize(init_points=init_round, n_iter=opt_round)

    model_auc=[]
    for model in range(len( lgbBO.res)):
        model_auc.append(lgbBO.res[model]['target'])

    # return best parameters
    return lgbBO.res[pd.Series(model_auc).idxmax()]['target'],lgbBO.res[pd.Series(model_auc).idxmax
()]['params']

opt_params = bayes_parameter_opt_lgb(train, target, init_round=5, opt_round=20, n_folds=5, random_s
eed=6)
```

```
|   iter    |  target   | baggin... | baggin... | baggin... | featur... | lambda_l1 | lambda_l2 |
learni... |  max_bin  | max_depth | min_ch... | min_da... | min_ga... | min_su... | num_le... | su
bsample |
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
|  1        |  0.8989   |  0.9581   |  5.359    |  7.972    |  0.3912   |  0.7665   |  0.01283  |
0.3638    |  138.3    |  39.21    |  19.73    |  72.04    |  0.8876   |  92.33    |  31.26    |  0.8
76    |
|  2        |  0.8687   |  0.297    |  8.713    |  6.253    |  0.09547  |  0.9441   |  0.8278   |
.5231    |  133.6    |  48.38    |  11.75    |  60.81    |  0.3833   |  64.04    |  52.61    |  0.11
4    |
|  3        |  0.8631   |  0.9728   |  8.552    |  8.398    |  0.3031   |  0.03619  |  0.8056   |
0.9731    |  56.78    |  53.68    |  8.984    |  31.26    |  0.383    |  66.6     |  46.22    |  0.8
59    |
|  4        |  0.8648   |  0.3368   |  9.869    |  6.506    |  0.2169   |  0.7937   |  0.8851   |
0.9874    |  22.09    |  20.34    |  14.68    |  50.64    |  0.1972   |  30.89    |  11.23    |  0.6
84    |
|  5        |  0.8958   |  0.6229   |  7.668    |  6.496    |  0.802    |  0.7224   |  0.9534   |
.4031    |  148.5    |  38.94    |  15.92    |  40.65    |  0.8254   |  6.442    |  10.65    |  0.48
5    |
|  6        |  0.8932   |  0.7865   |  9.677    |  8.216    |  0.3855   |  0.466    |  0.3184   |
.3855    |  133.7    |  6.277    |  8.99     |  21.76    |  0.6467   |  99.48    |  22.9     |  0.57
9    |
|  7        |  0.8595   |  0.2222   |  4.746    |  8.168    |  0.7697   |  0.0464   |  0.6603   |
0.06009   |  105.0    |  79.43    |  6.57     |  78.83    |  0.3883   |  96.21    |  11.46    |  0.8
29    |
|  8        |  0.8913   |  0.3242   |  7.453    |  5.576    |  0.6542   |  0.3531   |  0.6822   |
0.4271    |  145.0    |  62.89    |  15.16    |  22.73    |  0.13     |  97.06    |  11.71    |  0.4
37    |
|  9        |  0.8886   |  0.5361   |  7.914    |  8.09     |  0.8586   |  0.3846   |  0.9105   |
0.5961    |  80.79    |  6.494    |  18.98    |  20.34    |  0.06157  |  93.59    |  13.35    |  0.3
93    |
|  10       |  0.8925   |  0.979    |  6.961    |  8.16     |  0.1672   |  0.8784   |  0.3605   |
.1113    |  21.52    |  14.48    |  13.71    |  21.31    |  0.4981   |  5.012    |  78.74    |  0.23
6    |
|  11       |  0.8079   |  0.2669   |  6.9      |  9.999    |  0.46     |  0.6328   |  0.1037   |
.732     |  147.8    |  10.42    |  19.49    |  22.1     |  0.8764   |  2.701    |  45.3     |  0.34
9    |
|  12       |  0.8945   |  0.8845   |  4.159    |  7.833    |  0.4657   |  0.3483   |  0.4703   |
0.149     |  135.6    |  11.29    |  10.75    |  24.88    |  0.5751   |  98.15    |  26.83    |  0.3
31    |
```

```
|  13       |  0.8361   |  0.7332   |  7.725    |  8.292    |  0.5318   |  0.8732   |  0.4971   |
0.9449   |  20.05    |  78.2     |  19.51    |  65.04    |  0.7722   |  0.2048   |  75.72    |  0.2
42   |
|  14       |  0.8617   |  0.3027   |  4.948    |  9.122    |  0.08418  |  0.952    |  0.2028   |
.8893   |  39.22    |  5.157    |  7.894    |  72.5     |  0.6303   |  90.62    |  76.58    |  0.69
6   |
|  15       |  0.8965   |  0.6946   |  8.916    |  6.56     |  0.511    |  0.8982   |  0.0352   |
.3619   |  149.6    |  7.129    |  10.21    |  73.77    |  0.5781   |  96.87    |  13.92    |  0.15
4   |
|  16       |  0.8435   |  0.7175   |  5.081    |  8.807    |  0.5906   |  0.7642   |  0.6914   |
0.02612  |  132.4    |  13.92    |  15.88    |  23.22    |  0.6831   |  98.06    |  22.69    |  0.0
437  |
|  17       |  0.8813   |  1.0      |  4.0      |  5.0      |  0.9      |  0.01     |  0.01     |
.0      |  20.0     |  80.0     |  5.0      |  20.0     |  0.9      |  0.0      |  10.0     |  1.0
|
|  18       |  0.8674   |  1.0      |  4.0      |  5.0      |  0.0114   |  0.01     |  0.01     |
.01611  |  150.0    |  80.0     |  5.0      |  20.0     |  0.2937   |  100.0    |  80.0     |  1.0
|
|  19       |  0.8756   |  1.0      |  4.0      |  5.0      |  0.8984   |  0.01     |  0.01     |
.0      |  20.0     |  5.0      |  5.0      |  20.0     |  0.01     |  100.0    |  22.35    |  1.0
|
|  20       |  0.8817   |  1.0      |  4.0      |  5.0      |  0.02664  |  0.01     |  0.01     |
.0      |  89.12    |  5.0      |  5.0      |  80.0     |  0.04535  |  0.0      |  10.0     |  1.0
|
|  21       |  0.7587   |  0.2      |  6.007    |  5.0      |  0.9      |  0.01     |  0.01     |
.872    |  101.6    |  80.0     |  5.0      |  20.0     |  0.1412   |  0.0      |  42.07    |  1.0
|
|  22       |  0.8928   |  0.7617   |  8.446    |  5.175    |  0.7397   |  0.04204  |  0.8229   |
0.3322   |  144.7    |  16.27    |  5.405    |  70.82    |  0.5168   |  91.8     |  74.41    |  0.0
554  |
|  23       |  0.8779   |  1.0      |  4.0      |  5.0      |  0.01     |  0.01     |  0.01     |
.0      |  150.0    |  80.0     |  19.31    |  80.0     |  0.01129  |  0.0      |  10.0     |  1.0
|
|  24       |  0.8549   |  0.9851   |  4.0      |  5.0      |  0.01     |  0.01     |  0.01     |
.0      |  150.0    |  73.49    |  5.0      |  74.98    |  0.04982  |  99.08    |  46.61    |  1.0
|
|  25       |  0.8581   |  1.0      |  10.0     |  5.0      |  0.01     |  0.01     |  0.01     |
.0      |  45.74    |  5.0      |  20.0     |  80.0     |  0.9      |  0.0      |  80.0     |  1.0
|
==================================================================================================
==================================================================================================
```

In [0]:

```python
#https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes-lgbm
opt_params[1]['num_leaves'] = int(round(opt_params[1]['num_leaves']))
opt_params[1]['max_depth'] = int(round(opt_params[1]['max_depth']))
opt_params[1]['bagging_freq'] = int(round(opt_params[1]['bagging_freq']))
opt_params[1]['min_data_in_leaf'] = int(round(opt_params[1]['min_data_in_leaf']))
opt_params[1]['max_bin'] = int(round(opt_params[1]['max_bin']))
opt_params[1]['bagging_freq']=int(round(opt_params[1]['bagging_freq']))
opt_params[1]['bagging_seed']=int(round(opt_params[1]['bagging_seed']))
opt_params[1]['objective']='binary'
opt_params[1]['metric']='auc'
opt_params[1]['is_unbalance']=True
opt_params[1]['boost_from_average']=False
opt_params1=opt_params[1]
opt_params1
```

Out[0]:

```
{'bagging_fraction': 0.9581058054813363,
 'bagging_freq': 5,
 'bagging_seed': 8,
 'boost_from_average': False,
 'feature_fraction': 0.39119472958742835,
 'is_unbalance': True,
 'lambda_l1': 0.7664992796883313,
 'lambda_l2': 0.012831985764133342,
 'learning_rate': 0.3638494444744881,
 'max_bin': 138,
 'max_depth': 39,
 'metric': 'auc',
 'min_child_weight': 19.727040637374927,
 'min_data_in_leaf': 72,
```

```
 'min_gain_to_split': 0.8875644851478631,
 'min_sum_hessian_in_leaf': 92.32667066664217,
 'num_leaves': 31,
 'objective': 'binary',
 'subsample': 0.84764245541438}
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
num_folds=5
folds = StratifiedKFold(n_splits=num_folds,shuffle=True, random_state=2357)
oof = np.zeros(len(train))
getVal = np.zeros(len(train))
predictions = np.zeros(len(target))
new_features= [i for i in train.columns]
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
%%time
import lightgbm as lgb
print('Light GBM Model')
for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values, target.values)):

    X_train, y_train = train.iloc[trn_idx][new_features], target.iloc[trn_idx]
    X_valid, y_valid = train.iloc[val_idx][new_features], target.iloc[val_idx]


    X_tr, y_tr = augment(X_train.values, y_train.values)
    X_tr = pd.DataFrame(X_tr)

    print("Fold idx:{}".format(fold_ + 1))
    trn_data = lgb.Dataset(X_tr, label=y_tr)
    val_data = lgb.Dataset(X_valid, label=y_valid)


    clf1 = lgb.train(opt_params1, trn_data, 100000, valid_sets = [trn_data, val_data],
verbose_eval=1000,
                    early_stopping_rounds = 1000)
    oof[val_idx] = clf1.predict(train.iloc[val_idx][new_features],
num_iteration=clf1.best_iteration)
    getVal[val_idx]+= clf1.predict(train.iloc[val_idx][new_features],
num_iteration=clf1.best_iteration) / folds.n_splits
    predictions += clf1.predict(test[new_features], num_iteration=clf1.best_iteration) /
folds.n_splits
    clf1.save_model('/content/drive/My Drive/proj_1/model_1000_iteration_{}.sav'.format(fold_), nu
m_iteration=clf1.best_iteration)
```

```
Light GBM Model
Fold idx:1
Training until validation scores don't improve for 1000 rounds.
[1000] training's auc: 0.999974 valid_1's auc: 0.909328
[2000] training's auc: 0.999989 valid_1's auc: 0.910062
Early stopping, best iteration is:
[1246] training's auc: 0.999989 valid_1's auc: 0.910127
Fold idx:2
Training until validation scores don't improve for 1000 rounds.
[1000] training's auc: 0.999971 valid_1's auc: 0.906677
Early stopping, best iteration is:
[263] training's auc: 0.990709 valid_1's auc: 0.907424
Fold idx:3
Training until validation scores don't improve for 1000 rounds.
[1000] training's auc: 0.999961 valid_1's auc: 0.912099
[2000] training's auc: 0.999984 valid_1's auc: 0.91288
Early stopping, best iteration is:
[1273] training's auc: 0.999984 valid_1's auc: 0.912944
Fold idx:4
Training until validation scores don't improve for 1000 rounds.
[1000] training's auc: 0.999978 valid_1's auc: 0.905293
[2000] training's auc: 0.999994 valid_1's auc: 0.906328
Early stopping, best iteration is:
[1265] training's auc: 0.999994 valid_1's auc: 0.906281
Fold idx:5
```

```
Training until validation scores don't improve for 1000 rounds.
[1000] training's auc: 0.999966 valid_1's auc: 0.908903
Early stopping, best iteration is:
[247] training's auc: 0.989386 valid_1's auc: 0.909421
CPU times: user 3h 50min 31s, sys: 43.4 s, total: 3h 51min 15s
Wall time: 1h 31min 9s
```

In [0]:

```
from sklearn.metrics import roc_auc_score, roc_curve
print("\n >> CV score: {:<8.5f}".format(roc_auc_score(target, oof)))
```

```
 >> CV score: 0.90015
```

In [0]:

```
#https://www.kaggle.com/super13579/lgbm-with-duplicate-flag-value-0-923?scriptVersionId=12330297
test1=test.reset_index()
submission = pd.DataFrame({"ID_code": test1.ID_code.values})
submission['target'] = predictions
submission.to_csv("/content/drive/My Drive/proj_1/submission_1000.csv", index=False)
```

Kaggle Score

1. Private Score: 0.91553
2. Public Score : 0.91881

The above private leaderboard scores is within top 5 percent.

Conclusion

In [0]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Feature Model","Featurization","Kaggle Private LB AUC"]
x.add_row(["Gaussian Naive Bayes ","200 original features",0.80495])
x.add_row(["SVM(Suport Vector Machines) ","200 original features", 0.76828])
x.add_row(["XGBOOST ","200 original features", 0.67655])
x.add_row(["Lightgbm ","200 original features",0.86914])

x.add_row([" Gaussian Naive Bayes + TruncatedSVD for top 125 features ","Duplicate value flag ", 0
.76306])
x.add_row([" Gaussian Naive Bayes + TruncatedSVD for top 150 features ","Duplicate value flag ", 0
.77180])
x.add_row(["SVM(Suport Vector Machines)+ TruncatedSVD for top 150 features","Duplicate value flag
", 0.73673])
x.add_row(["Lightgbm + TruncatedSVD for top 150 features","Duplicate value flag ", 0.84961])

x.add_row([" Gaussian Naive Bayes + selectkbest for top 200 features","Duplicate value flag ", 0.7
8190])
x.add_row(["SVM(Suport Vector Machines) + selectkbest for top 200 features","Duplicate value flag
", 0.73442])
x.add_row(["Lightgbm + selectkbest for top 200 features","Duplicate value flag ", 0.87081])

x.add_row([" Gaussian Naive Bayes + all 400 features(200 original + 200 duplicate flag) ","Duplica
te value flag ", 0.75314])
x.add_row(["SVM(Suport Vector Machines) + all 400 features(200 original + 200 duplicate flag)","Du
plicate value flag ", 0.76438])
x.add_row(["XGBOOST + all 400 features(200 original + 200 duplicate flag)","Duplicate value flag "
, 0.71413])
x.add_row(["Lightgbm + all 400 features(200 original + 200 duplicate flag) ","Duplicate value flag
", 0.86945])

x.add_row(["Gaussian Naive Bayes with only 8 features","Sum,Mean,Median,std,skew,Min,Max and kurto
sis",0.50000])
x.add_row(["Lightgbm with only 8 features","Sum,Mean,Median,std,skew,Min,Max and kurtosis",0.55095
])

x.add_row(["Gaussian Naive Bayes with top 190 features ","200 original
+Sum,Mean,Median,std,skew,Min,Max and kurtosis",0.50000])
```

```
x.add_row(["Lightgbm with top 190 features","200 original + Sum,Mean,Median,std,skew,Min,Max and k
urtosis",0.87044])

x.add_row(["Gaussian Naive Bayes with all 208 features ","200 original
+Sum,Mean,Median,std,skew,Min,Max and kurtosis",0.79956])
x.add_row(["SVM(Suport Vector Machines) with all 208 features","200 original +
Sum,Mean,Median,std,skew,Min,Max and kurtosis",0.76889])
x.add_row(["Decision Tree with all 208 features ","200 original +Sum,Mean,Median,std,skew,Min,Max
and kurtosis",0.62694])
x.add_row(["Voting Classifier with all 208 features","200 original +
Sum,Mean,Median,std,skew,Min,Max and kurtosis",0.77758])
x.add_row(["Stacking classifier with all 208 features","200 original +
Sum,Mean,Median,std,skew,Min,Max and kurtosis",0.76782])
x.add_row(["lightgbm with all 208 features ","200 original +Sum,Mean,Median,std,skew,Min,Max and k
urtosis",0.88797])
x.add_row(["lightgbm with all 208 features and augmentation ","200 original
+Sum,Mean,Median,std,skew,Min,Max and kurtosis",0.89211])


x.add_row(["Gaussian Naive Bayes with all 1000 features ","1000 features (original+duplicate count
+distance of mean)", 0.5000])
x.add_row(["SVM with all 1000 features","1000 features (original+duplicate count+distance of mean)
", 0.52990])

x.add_row(["Gaussian Naive Bayes with top 600 features ","1000 features (original+duplicate count+
distance of mean)",0.71324])
x.add_row(["SVM with top 600 features ","1000 features (original+duplicate count+distance of mean)
", 0.79074])
x.add_row(["Decision Tree with top 600 features ","1000 features (original+duplicate count+distanc
e of mean)", 0.64147])
x.add_row(["Voting Classifier with top 600 features ","1000 features (original+duplicate count+dis
tance of mean)",  0.77147])
x.add_row(["Stacking Classifier with top 600 features ","1000 features (original+duplicate count+d
istance of mean)", 0.79332])
x.add_row(["lightgbm with top 600 features ","1000 features (original+duplicate count+distance of
mean)",0.91352])
x.add_row(["lightgbm with top 600 features and augmentation ","1000 features (original+duplicate c
ount+distance of mean)",0.91485])

x.add_row(["Lightgbm with all 1000 features and augmentation","1000 features (original+duplicate c
ount+distance of mean)",0.91553])

print(x)
```

```
+--------------------------------------------------------------------------------+--------------
---------------------------------------------+-----------------------+
|                                     Feature Model                              |
Featurization                                  | Kaggle Private LB AUC |
+--------------------------------------------------------------------------------+--------------
---------------------------------------------+-----------------------+
|                                Gaussian Naive Bayes                            |
200 original features                          |        0.80495        |
|                             SVM(Suport Vector Machines)                        |
200 original features                          |        0.76828        |
|                                      XGBOOST                                   |
200 original features                          |        0.67655        |
|                                      Lightgbm                                  |
200 original features                          |        0.86914        |
|               Gaussian Naive Bayes + TruncatedSVD for top 125 features         |
Duplicate value flag                           |        0.76306        |
|               Gaussian Naive Bayes + TruncatedSVD for top 125 features         |
Duplicate value flag                           |         0.7718        |
|            SVM(Suport Vector Machines)+ TruncatedSVD for top 125 features      |
Duplicate value flag                           |        0.73673        |
|                     Lightgbm + TruncatedSVD for top 150 features               |
Duplicate value flag                           |        0.84961        |
|                Gaussian Naive Bayes + selectkbest for top 200 features         |
Duplicate value flag                           |         0.7819        |
|            SVM(Suport Vector Machines) + selectkbest for top 200 features      |
Duplicate value flag                           |        0.73442        |
|                     Lightgbm + selectkbest for top 200 features                |
Duplicate value flag                           |        0.87081        |
|      Gaussian Naive Bayes + all 400 features(200 original + 200 duplicate flag)     |
Duplicate value flag                           |        0.75314        |
| SVM(Suport Vector Machines) + all 400 features(200 original + 200 duplicate flag) |
Duplicate value flag                           |        0.76438        |
|            XGBOOST + all 400 features(200 original + 200 duplicate flag)       |
```

```
|          XGBOOST + all 400 features(200 original + 200 duplicate flag)          |
Duplicate value flag                      |        0.71413        |
|          Lightgbm + all 400 features(200 original + 200 duplicate flag)          |
Duplicate value flag                      |        0.86945        |
|                     Gaussian Naive Bayes with only 8 features                    |
Sum,Mean,Median,std,skew,Min,Max and kurtosis         |          0.5          |
|                           Lightgbm with only 8 features                          |        Sum,Me
,Median,std,skew,Min,Max and kurtosis         |        0.55095        |
|                    Gaussian Naive Bayes with top 190 features                    | 200 original
Sum,Mean,Median,std,skew,Min,Max and kurtosis  |          0.5          |
|                           Lightgbm with top 190 features                         | 200 original
Sum,Mean,Median,std,skew,Min,Max and kurtosis |        0.87044        |
|                    Gaussian Naive Bayes with all 208 features                    | 200 original
Sum,Mean,Median,std,skew,Min,Max and kurtosis  |        0.79956        |
|                 SVM(Suport Vector Machines) with all 208 features                | 200 original
+ Sum,Mean,Median,std,skew,Min,Max and kurtosis |        0.76889        |
|                         Decision Tree with all 208 features                      | 200 original
Sum,Mean,Median,std,skew,Min,Max and kurtosis  |        0.62694        |
|                      Voting Classifier with all 208 features                     | 200 original
Sum,Mean,Median,std,skew,Min,Max and kurtosis |        0.77758        |
|                     Stacking classifier with all 208 features                    | 200 original
Sum,Mean,Median,std,skew,Min,Max and kurtosis |        0.76782        |
|                          lightgbm with all 208 features                          | 200 original
Sum,Mean,Median,std,skew,Min,Max and kurtosis  |        0.88797        |
|                  lightgbm with all 208 features and augmentation                 | 200 original
+Sum,Mean,Median,std,skew,Min,Max and kurtosis  |        0.89211        |
|                    Gaussian Naive Bayes with all 1000 features                   |  1000 feature
(original+duplicate count+distance of mean)   |          0.5          |
|                              SVM with all 1000 features                          |  1000 feature
(original+duplicate count+distance of mean)   |         0.5299        |
|                    Gaussian Naive Bayes with top 600 features                    |  1000 feature
(original+duplicate count+distance of mean)   |        0.71324        |
|                              SVM with top 600 features                           |  1000 feature
(original+duplicate count+distance of mean)   |        0.79074        |
|                         Decision Tree with top 600 features                      |  1000 feature
(original+duplicate count+distance of mean)   |        0.64147        |
|                      Voting Classifier with top 600 features                     |  1000 feature
(original+duplicate count+distance of mean)   |        0.77147        |
|                     Stacking Classifier with top 600 features                    |  1000 feature
(original+duplicate count+distance of mean)   |        0.79332        |
|                          lightgbm with top 600 features                          |  1000 feature
(original+duplicate count+distance of mean)   |        0.91352        |
|                  lightgbm with top 600 features and augmentation                 |  1000 feature
(original+duplicate count+distance of mean)   |        0.91485        |
|                  Lightgbm with all 1000 features and augmentation                |  1000 feature
(original+duplicate count+distance of mean)   |        0.91553        |
+----------------------------------------------------------------------------------+--------------
-----------------------------------------------+-----------------------+
```

Summary

I have done the following EDA:

1. Dropped if any Duplicates data are there and kept first record.There are no duplicate data.
2. Checked Value count for two Classes.It seems to be Imbalance data.
3. Checked for Missing values . There are no missing values.
4. Plotted box plot for some of features. There are some outliers in the data.
5. Then I have removed the outliers from by data using IQR range formula.
6. Checked for correlation between features.There are very less correlation between all features.
7. Plotted Histogram plot and observed that many values are repeated(high frequency) for each feature.

Then I have chosen Gaussian Naive Bayes as my baseline model as all the features have very less correlation.

I have used the below models without any featurization:

1. Gaussian Naive Bayes
2. Support Vector Machine
3. Xgboost
4. Lightgbm

The Lightgbm have performed better of all models with AUC equal to 0.86914.

Featurization 1

I have used duplicate value flag featurization for each of 200 original features. It have resulted into total 400 features(200 original + 200 duplicate).

TRUNCATEDSVD to select top 150 features

Then I have selected the top 150 features by using TRUNCATEDSVD and plotting cummulative variance. Thereafter I used the below models with top 150 features:

1. Gaussian Naive Bayes
2. Support Vector Machine
3. Lightgbm

The Lightgbm have performed better of all models with AUC equal to 0.84961.

selectkbest to select top 200 features

Then I have selected the top 200 features by using selectkbest and plotting cummulative variance. Thereafter I used the below models with top 200 features:

1. Gaussian Naive Bayes
2. Support Vector Machine
3. Lightgbm

The Lightgbm have performed better of all models with AUC equal to 0.87081.

All 400 features

I have used the below models :

1. Gaussian Naive Bayes
2. Support Vector Machine
3. Xgboost
4. Lightgbm

The Lightgbm have performed better of all models with AUC equal to 0.86945.

Featurization 2

I have used Sum,Mean,Median,std,skew,Min,Max and kurtosis featurization for each of 200 columns. It have resulted into total 208 features(200 original + 8 features).

Only 8 features (Sum,Mean,Median,std,skew,Min,Max and kurtosis)

I have used the below models:

1. Gaussian Naive Bayes
2. Lightgbm

The Lightgbm have performed better of two model with AUC equal to 0.55095. The AUC have reduced drastically from 0.87 to 0.55095.

selectkbest to select top 190 features

Then I have selected the top 190 features by using selectkbest and plotting cummulative variance. Thereafter I used the below models with top 190 features:

1. Gaussian Naive Bayes
2. Lightgbm

The Lightgbm have performed better of all models with AUC equal to 0.87044.

All 208 features

I have used the below models :

1. Gaussian Naive Bayes
2. Support Vector Machine
3. Decision Tree
4. Voting Classifier
5. Stacking classifier
6. Lightgbm with no augmentation
7. Lightgbm with augmentation

The Lightgbm with augmentation have performed better of all models with AUC equal to 0.89211.

Featurization 3

First, I have Combined train data and only test data which doesn't have all features with duplicate value. Thereafter I have used combined (train and test real) data for below featurization:

1. Duplicate Count: Take minimum of 10 and value count for that particular value.
2. Duplicate Value Count >2 : Multiply actual value of that feature with duplicate count (if only duplicate count greater than 2)
3. Duplicate Value Count >4 : Multiply actual value of that feature with duplicate count (if only duplicate count greater than 4)
4. Distance of mean : Calculate difference between current value and mean of that particular feature .Then mutiply it with duplicate count feature.

Hence , I have created 800 new features (total=1000 features(200 original+800 new)).

selectkbest to select top 600 features

Then I have selected the top 600 features by using selectkbest and plotting cummulative variance. Thereafter I used the below models with top 600 features:

1. Gaussian Naive Bayes
2. SVM
3. Decision Tree
4. Voting Classifier
5. Stacking Classifier
6. Lightgbm with no augmentation
7. Lightgbm with augmentation

The Lightgbm with augmentation have performed better of all models with AUC equal to 0.91485. This AUC score falls within top 5 percent on Kaggle private Leaderboard.

All 1000 features

I have used the below models :

1. Gaussian Naive Bayes
2. Support Vector Machine
3. Lightgbm with augmentation

The Lightgbm with augmentation have performed better of all models with AUC equal to 0.91553. This AUC score falls within top 5 percent on Kaggle private Leaderboard.