In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-
6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googlea
%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2
.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

Importing libraries :

In [0]:

```
!pip install matplotlib==3.1.0
```

In [8]:

```
import matplotlib
matplotlib.__version__
```

Out[8]:

```
'3.1.0'
```

In [0]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import gc
import warnings
warnings.filterwarnings("ignore")

%matplotlib inline

from sklearn import linear_model
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV, train_test_split
from sklearn import preprocessing # KBinsDiscretizer
```

```
from sklearn import preprocessing # KBinsDiscretizer
from sklearn import metrics
from joblib import parallel_backend
import scipy.stats as st
import xgboost as xgb
import lightgbm as lgb
from prettytable import PrettyTable

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

Util Functions :

In [0]:

```
def auc_plot(c, tr, cv, param):
  plt.figure(figsize=(20, 6))
  plt.plot(c, tr, label='Train AUC')
  plt.plot(c, cv, label='CV AUC')
  plt.scatter(c, tr, label='Train AUC points')
  plt.scatter(c, cv, label='CV AUC points')
  plt.legend()
  plt.xlabel("{} : hyperparameter".format(param))
  plt.ylabel("AUC")
  plt.title("AUC plots for Train vs CV")
  plt.grid()
  plt.show()

def roc_plot(tr_fpr, tr_tpr, te_fpr, te_tpr):
  plt.plot(tr_fpr, tr_tpr, label="Train AUC = {}".format(metrics.auc(tr_fpr, tr_tpr)))
  plt.plot(te_fpr, te_tpr, label="Test AUC = {}".format(metrics.auc(te_fpr, te_tpr)))
  plt.legend()
  plt.xlabel("FPR")
  plt.ylabel("TPR")
  plt.title("ROC Curve")
  plt.grid()
  plt.show()

def plotTrainVsCV_AUC(search, subplots, figsize, idx, cols):
  # Print seaborn heatmaps in subplots: https://stackoverflow.com/a/42712772/9079093
  figure, axes = plt.subplots(*subplots, figsize=figsize)
  # Using grid search parameters for heatmap : https://stackoverflow.com/a/48792210/9079093
  # Print values in seaborn heatmap without scientific notation: https://stackoverflow.com/a/29648332/9079093
  g1 = sns.heatmap(pd.pivot_table(pd.DataFrame(search.cv_results_),values='mean_train_score', index=idx, columns=cols),\
                   annot=True, annot_kws={"size": 13},cmap='Oranges', fmt='.2g', ax=axes[0], xticklabels=True, yticklabels=True)
  g1.set_title('Train AUC Heat Map plot')
  g1.set_ylabel(idx)
  g1.set_xlabel(cols)

  g2 = sns.heatmap(pd.pivot_table(pd.DataFrame(search.cv_results_),values='mean_test_score', index=idx, columns=cols),\
                   annot=True, annot_kws={"size": 13},cmap=None, fmt='.2g', ax=axes[1], xticklabels=True, yticklabels=True)
```

```
  g2.set_title('CV AUC Heat Map plot')
  g2.set_ylabel(idx)
  g2.set_xlabel(cols)

  plt.subplots_adjust(hspace=.3)
  plt.show()

def predict(proba, threshold, fpr, tpr, dtyp):
  t = threshold[np.argmax(tpr*(1-fpr))]
  # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
  print("The maximum value of {} tpr*(1-fpr) is {} for threshold {}".format(dtyp, max(tpr*(1-fpr)), np.round(t,3)),'\n')

  predictions = []
  for i in proba:
    if i>=t:
      predictions.append(1)
    else:
      predictions.append(0)

  return predictions

def printConfusionMatrix(y_tr, y_te, y_tr_pred, y_te_pred, tr_thresh, te_thresh, tr_fpr, tr_tpr, te_fpr, te_tpr):
  # Print seaborn heatmaps in subplots: https://stackoverflow.com/a/42712772/9079093
  figure, axes = plt.subplots(1,2, figsize=(15,5))
  # Print values in seaborn heatmap without scientific notation: https://stackoverflow.com/a/29648332/9079093
  g1 = sns.heatmap(metrics.confusion_matrix(y_tr, predict(y_tr_pred, tr_thresh, tr_fpr, tr_tpr, 'Train')),\
                   annot=True, annot_kws={"size": 16}, cmap='Oranges', fmt='g', ax=axes[0])
  g1.set_title('Train confusion matrix')
  g1.set_xlabel('Predicted Value')
  g1.set_ylabel('Actual Value')

  g2 = sns.heatmap(metrics.confusion_matrix(y_te, predict(y_te_pred, te_thresh, te_fpr, te_tpr, 'Test')),\
                   annot=True, annot_kws={"size": 16}, cmap='Purples', fmt='g', ax=axes[1])
  g2.set_title('Test confusion matrix')
  g2.set_xlabel('Predicted Value')
  g2.set_ylabel('Actual Value')

  plt.subplots_adjust(top=.9, wspace=.5, hspace=.5)
  plt.show()
```

Reading the Data :

In [5]:

```
df_train = pd.read_csv('drive/My Drive/CoLab/CustomerTransactionPrediction/train.csv')
df_train.head()
```

Out[5]:

	ID_code	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	var_13	var_14	var_15	var_16	var_17	var_18	var_19	var_20	v
0	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	0.5745	8.7989	14.5691	5.7487	-7.2393	4.2840	30.7133	10.5350	1
1	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	8.4135	5.4345	13.7003	13.8275	-15.5849	7.8000	28.5708	3.4287	:
2	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	7.3124	7.5244	14.6472	7.6782	-1.7395	4.7011	20.4775	17.7559	1
3	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	11.9704	6.4569	14.8372	10.7430	-0.4299	15.9426	13.7257	20.3010	1.
4	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	7.8895	7.7894	15.0553	8.4871	-3.0680	6.5263	11.3152	21.4246	1

5 rows × 202 columns

In [0]:

```
df_test = pd.read_csv('drive/My Drive/CoLab/CustomerTransactionPrediction/test.csv')
df_test.head()
```

High level statistics:

In [0]:

```
print('Total Datapoints:',df_train.shape[0],\
      '\nTotal Features:',df_train.shape[1],'\n')

print('Some of the Features:')
for idx, col in enumerate(df_train.columns[1::20]):
    print(idx+1,':',col)
```

```
Total Datapoints: 200000
Total Features: 202

Some of the Features:
1 : target
2 : var_19
3 : var_39
4 : var_59
5 : var_79
6 : var_99
7 : var_119
8 : var_139
9 : var_159
10 : var_179
11 : var_199
```

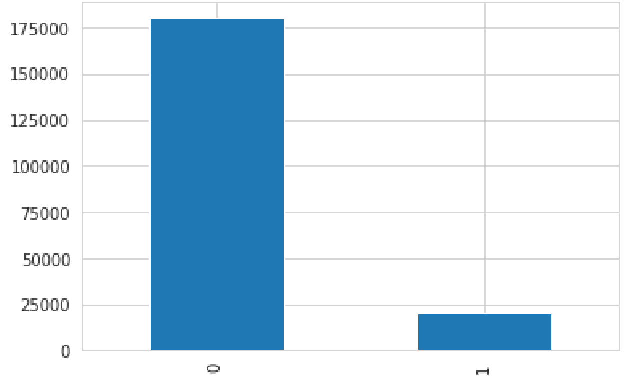
In [0]:

```
print('Number of classes:', df_train.target.unique().size,'\n')

print('DataPoints per class:')
print(df_train.target.value_counts())
```

```
Number of classes: 2

DataPoints per class:
0    179902
1     20098
Name: target, dtype: int64
```

In [0]:

```
df_train['target'].value_counts().plot.bar()
plt.show()
```



In [0]:

```
if not (df_train.isnull().sum() > 0).any():
  print('No missing Values in the Training Data.')
else:
  print('Missing value features of Training Data :')
  print(df_train.columns[ df_train.isnull().sum() > 0 ])
```

```
No missing Values in the Training Data.
```

In [0]:

```
if not (df_test.isnull().sum() > 0).any():
```

```
  print('No missing Values in the Test Data.')
else:
  print('Missing value features of Test Data :')
  print(df_test.columns[ df_test.isnull().sum() > 0 ])
```

```
No missing Values in the Test Data.
```

In [0]:

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

Out[0]:

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

- Based on the # data points per class we can observe that it's a highly imbalanced dataset.
  - Class_1 : 10%
  - Class_0 : 90%

EDA

Objective :

To identify if a customer will make a specific transaction or not in the future, irrespective of the amount of money transacted. The data provided here has the same structure as the real data we have available to solve this problem.

In [0]:

```
# Let's consider some of the features for the EDA part as there are 200 features and each is numeric.

eda_features = df_train.columns[2::20]
eda_features
```

Out[0]:

```
Index(['var_0', 'var_20', 'var_40', 'var_60', 'var_80', 'var_100', 'var_120',
       'var_140', 'var_160', 'var_180'],
      dtype='object')
```

In [0]:

```
# Since all the required features are numeric we are excluding the non-numeric features and the target which is to be predicted.
```

```
df_train[df.columns.difference(['target'])].describe(exclude=['object'])
```

Out[0]:

	var_0	var_1	var_10	var_100	var_101	var_102	var_103	var_104	var_105	var_106	var_107	var_108	var
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.00
mean	10.679914	-1.627622	0.394340	-6.600518	13.413526	22.294908	1.568393	11.509834	4.244744	8.617657	17.796266	14.224435	18.45
std	3.040051	4.050044	5.500793	9.181683	4.950537	8.628179	0.185020	1.970520	0.855698	1.894899	7.604723	0.171091	4.35
min	0.408400	-15.043400	-20.731300	-39.179100	0.075700	-7.382900	0.979300	4.084600	0.715300	0.942400	-5.898000	13.729000	5.76
25%	8.453850	-4.740025	-3.594950	-13.198700	9.639800	16.047975	1.428900	10.097900	3.639600	7.282300	12.168075	14.098900	15.10
50%	10.524750	-1.608050	0.487300	-6.401500	13.380850	22.306850	1.566000	11.497950	4.224500	8.605150	17.573200	14.226600	18.28
75%	12.758200	1.358625	4.382925	0.132100	17.250225	28.682225	1.705400	12.902100	4.822200	9.928900	23.348600	14.361800	21.85
max	20.315000	10.376800	18.670200	25.140900	28.459400	51.326500	2.188700	19.020600	7.169200	15.307400	46.379500	14.743000	32.05

8 rows × 200 columns

In [0]:

```
df_test.describe()
```

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	va
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.00
mean	10.658737	-1.624244	10.707452	6.788214	11.076399	-5.050558	5.415164	16.529143	0.277135	7.569407	0.371335	-3.268551	14.02
std	3.036716	4.040509	2.633888	2.052724	1.616456	7.869293	0.864686	3.424482	3.333375	1.231865	5.508661	5.961443	0.19
min	0.188700	-15.043400	2.355200	-0.022400	5.484400	-27.767000	2.216400	5.713700	-9.956000	4.243300	-22.672400	-25.811800	13.42
25%	8.442975	-4.700125	8.735600	5.230500	9.891075	-11.201400	4.772600	13.933900	-2.303900	6.623800	-3.626000	-7.522000	13.89
50%	10.513800	-1.590500	10.560700	6.822350	11.099750	-4.834100	5.391600	16.422700	0.372000	7.632000	0.491850	-3.314950	14.02
75%	12.739600	1.343400	12.495025	8.327600	12.253400	0.942575	6.005800	19.094550	2.930025	8.584825	4.362400	0.832525	14.16
max	22.323400	9.385100	18.714100	13.142000	16.037100	17.253700	8.302500	28.292800	9.665500	11.003600	20.214500	16.771300	14.68

8 rows × 200 columns

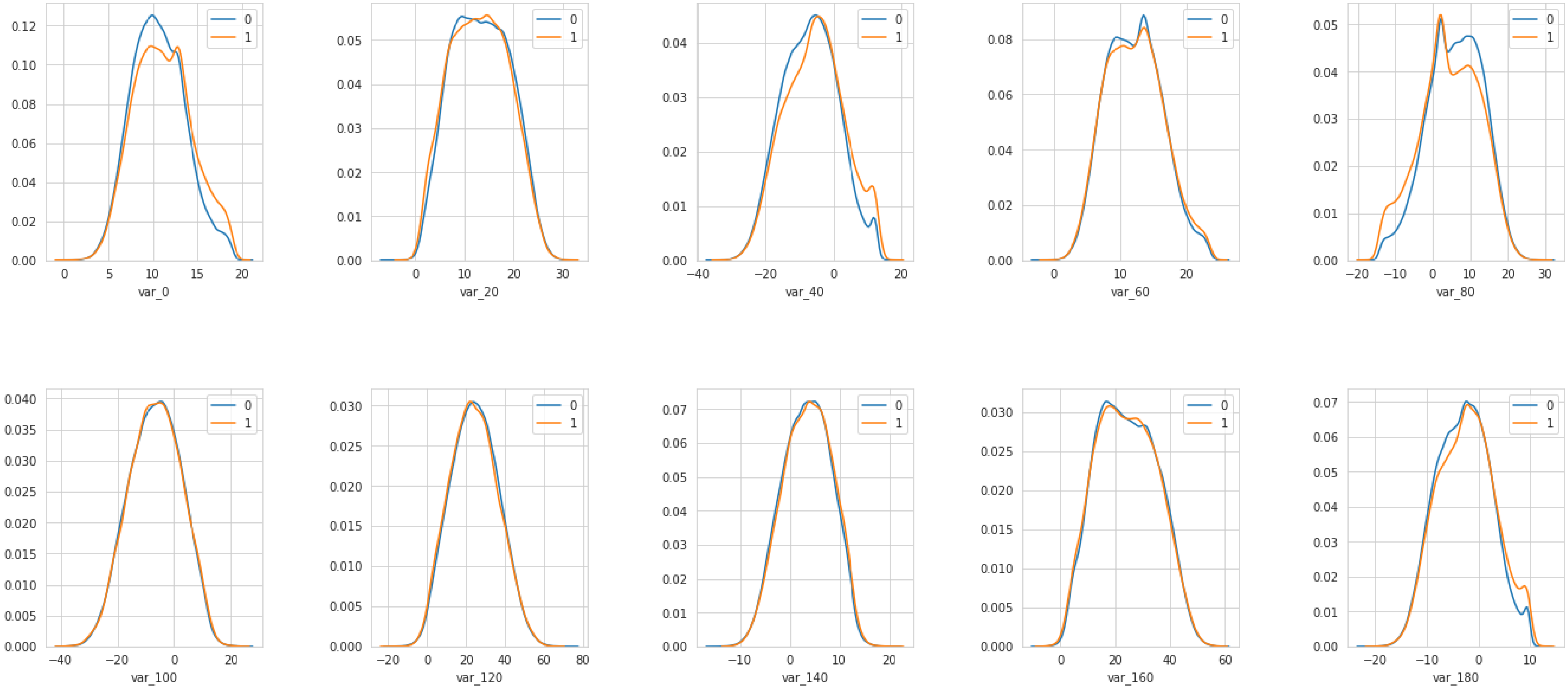
- The Mean and Spread for some of the features differ a lot between the train and test data.

In [0]:

```
sns.set_style('whitegrid')
```

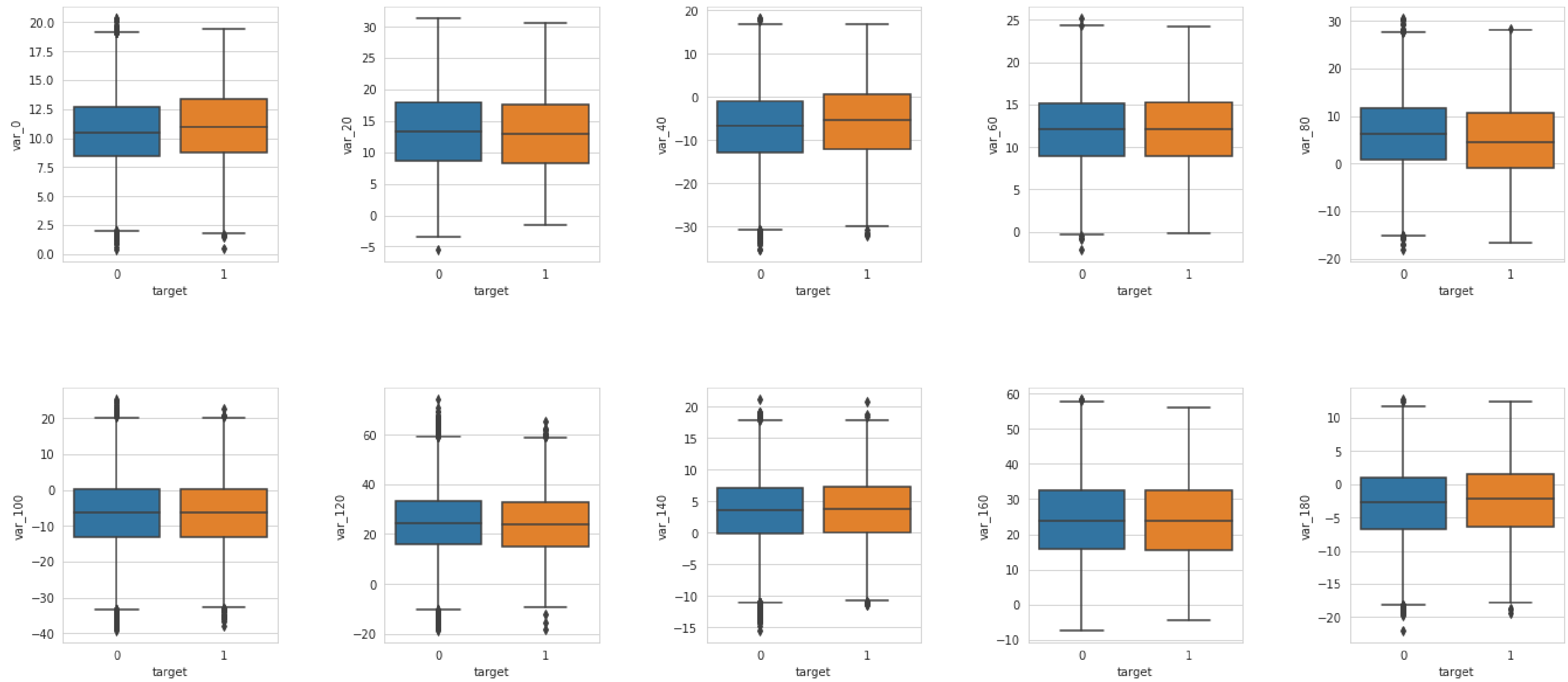
In [0]:

```
figure, axes = plt.subplots(2, 5, figsize=(23,10))
axes = axes.flatten()

df_train_0 = df_train[df_train.target == 0]
df_train_1 = df_train[df_train.target == 1]

for idx, feat in enumerate(eda_features):
  sns.kdeplot(df_train_0[feat], ax=axes[idx], label='0')
  sns.kdeplot(df_train_1[feat], ax=axes[idx], label='1').set_xlabel(feat)

plt.subplots_adjust(hspace=.5, wspace=.5)
plt.show()
plt.close()
```



In [0]:

```
#Box Plots and whiskers
figure, axes = plt.subplots(2, 5, figsize=(23,10))
axes = axes.flatten()

for idx, col in enumerate(eda_features):
    sns.boxplot(x='target', y=col, data=df_train, ax=axes[idx])

plt.subplots_adjust(wspace=.5, hspace=.5)
plt.show()
plt.close()
```



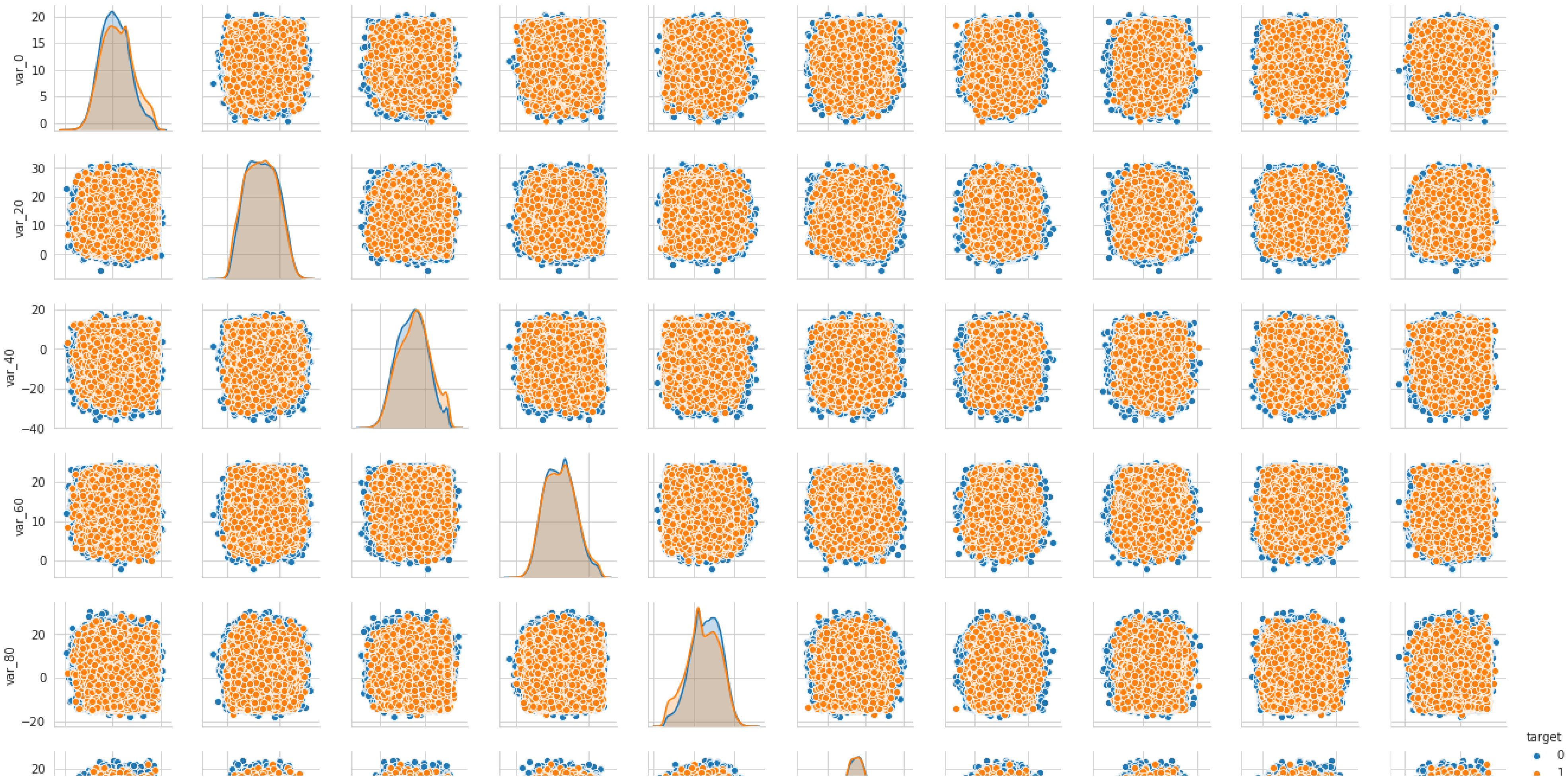
In [0]:

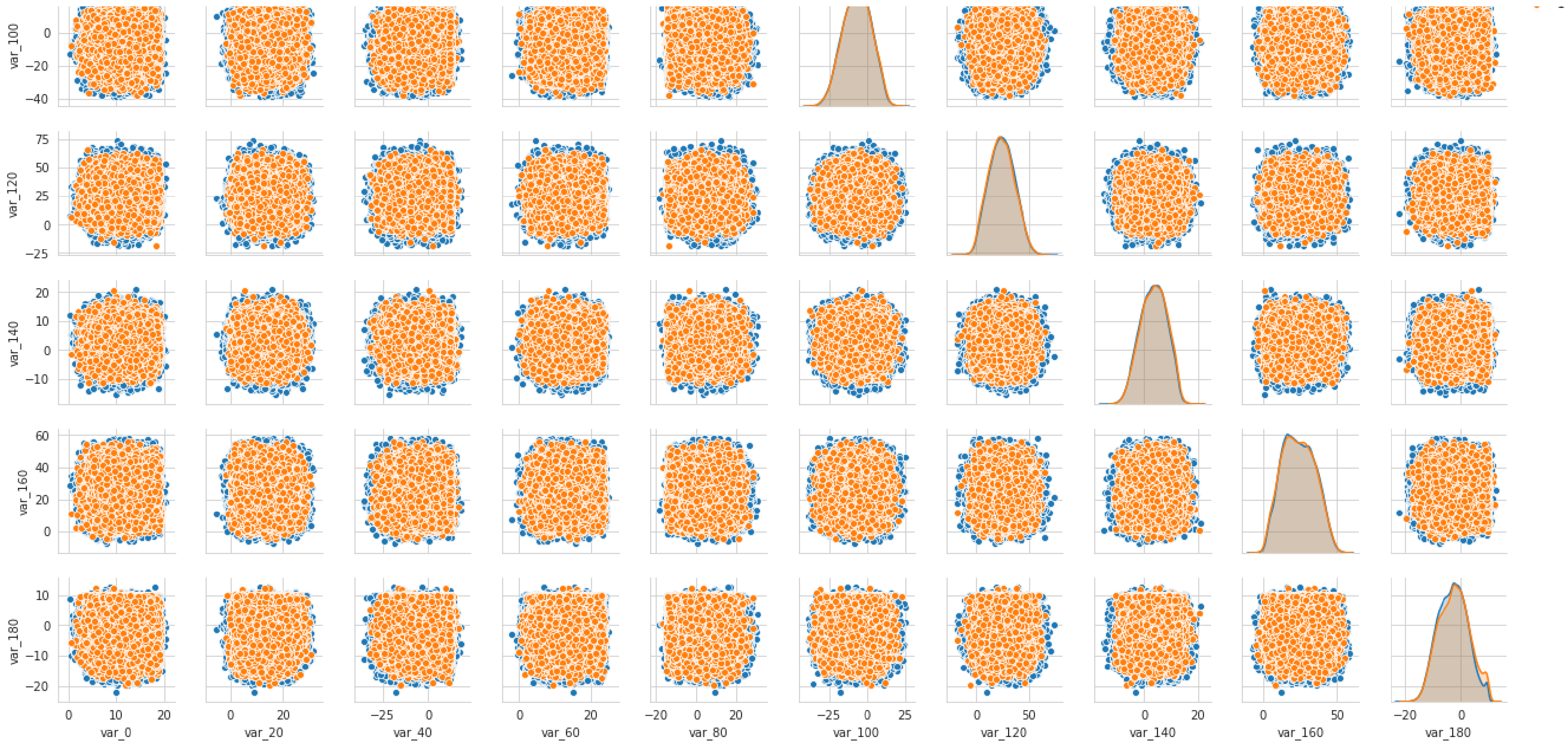
```
# # Violin Plots
# figure, axes = plt.subplots(2, 5, figsize=(23, 10))
```

```
# axes = axes.flatten()

# for idx, col in enumerate(eda_features):
#     sns.violinplot(x='target', y=col, data=df_train, size=8, ax=axes[idx])

# plt.subplots_adjust(wspace=.5, hspace=.5)
# plt.show()
# plt.close()
```

In [0]:

```
sns.pairplot(df_train[[*eda_features, 'target']], vars=eda_features ,hue='target', height=1.8)

plt.show()
```



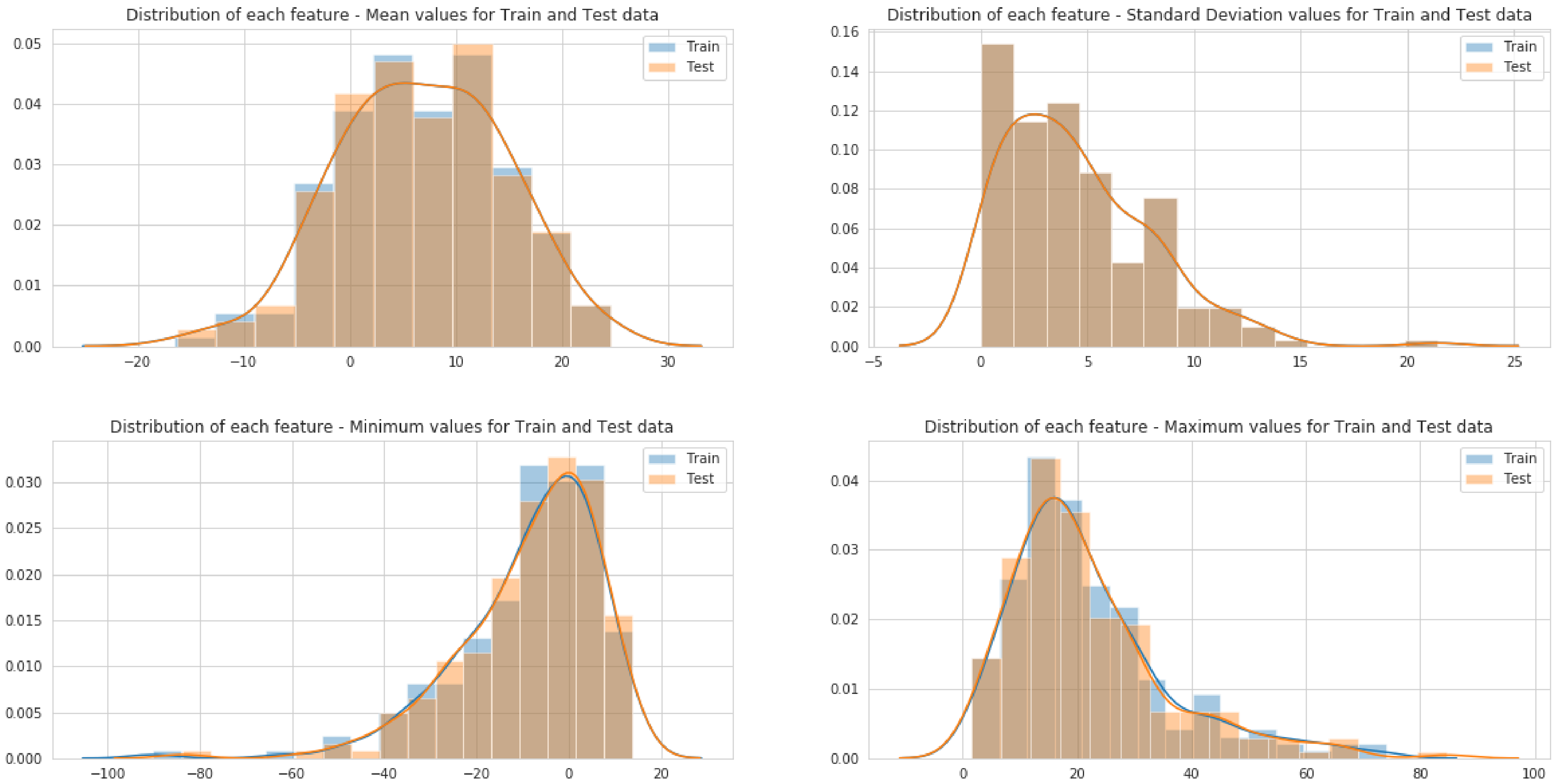


- It's hard to seperate the classes using the linear models as we can see from the pair plots of the features, almost all of them overlap.
- Most of the features are Gaussian distributed except for some like var_0, var_20, var_40, var_60, var_80 and so on with some bumps on left or right of mean value.

In [0]:

```
plots = ['Mean', 'Standard Deviation', 'Minimum', 'Maximum']
plot_funcs = [np.mean, np.std, np.min, np.max]

figure, axes = plt.subplots(2, 2, figsize=(20, 10))
axes = axes.flatten()

for idx, (stat, func) in enumerate(zip(plots, plot_funcs)):
  sns.distplot(df_train[df_train.columns.difference(['target', 'ID_code'])].apply(func), label='Train', ax=axes[idx])
  sns.distplot(df_test[df_train.columns.difference(['target', 'ID_code'])].apply(func), label='Test', ax=axes[idx])
  axes[idx].set_title('Distribution of each feature - {} values for Train and Test data'.format(stat))
```

```
    axes[idx].legend()

plt.subplots_adjust(hspace=.3)
plt.show()
```

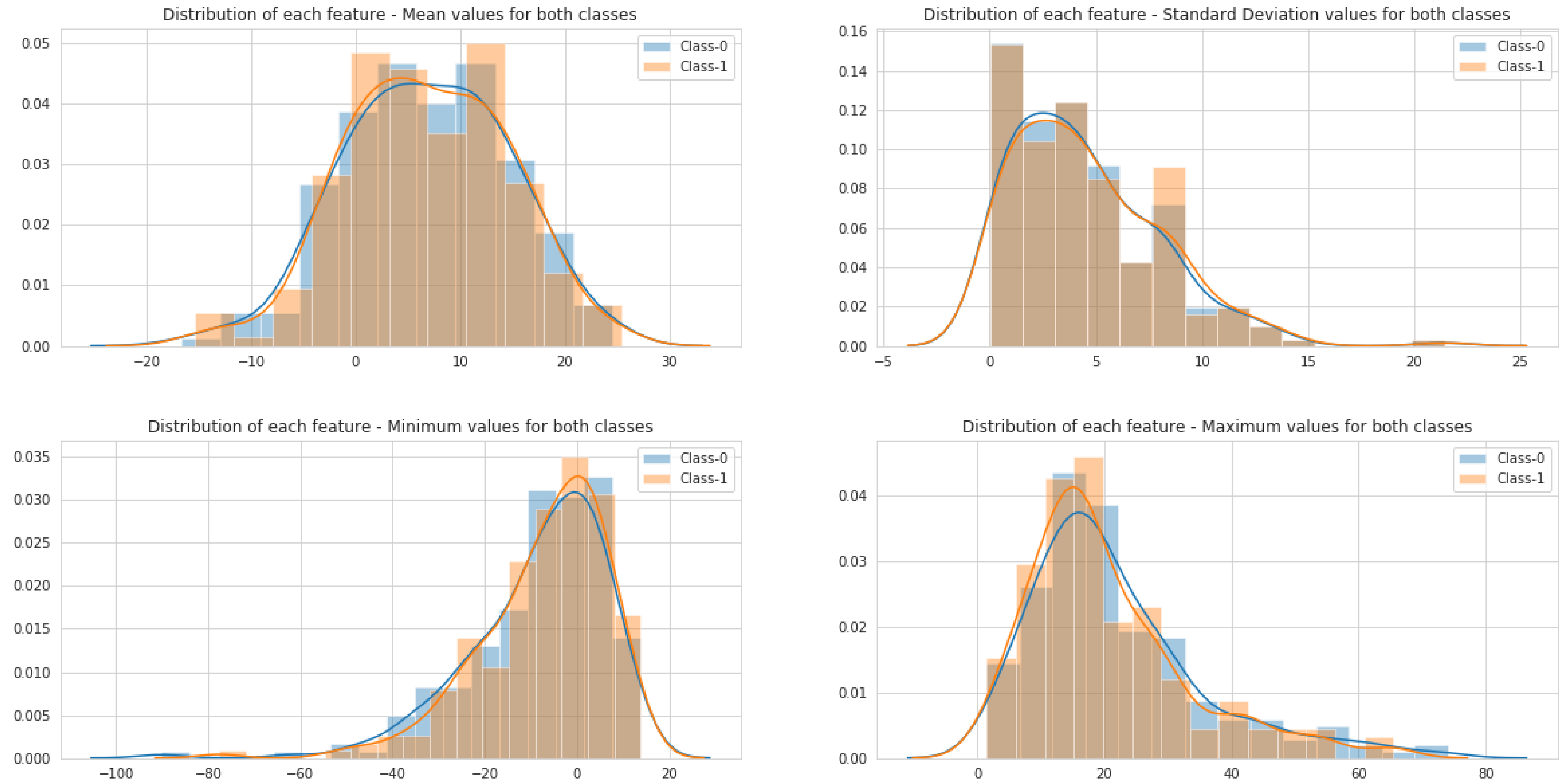


- We can see that the train and test data come from almost similar distrubutions based on mean, standard devation, minimum and maximum values plots for each feature.

In [0]:

```
df_train_0 = df_train[df_train.target == 0][df_train.columns.difference(['target', 'ID_code'])]
df_train_1 = df_train[df_train.target == 1][df_train.columns.difference(['target', 'ID_code'])]

figure, axes = plt.subplots(2, 2, figsize=(20, 10))
axes = axes.flatten()
```

```
for idx, (stat, func) in enumerate(zip(plots, plot_funcs)):
  sns.distplot(df_train_0.apply(func), label='Class-0', ax=axes[idx])
  sns.distplot(df_train_1.apply(func), label='Class-1', ax=axes[idx])
  axes[idx].set_title('Distribution of each feature - {} values for both classes'.format(stat))
  axes[idx].legend()

plt.subplots_adjust(hspace=.3)
plt.show()
```



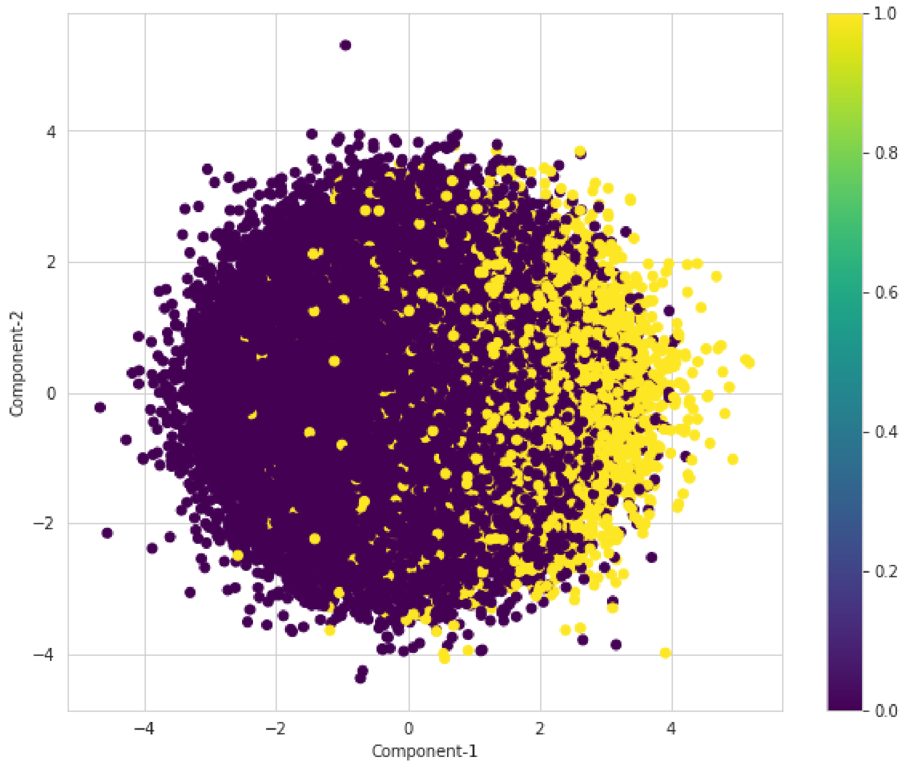
In [0]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

train_scaled = StandardScaler().fit_transform(df_train[df_train.columns.difference(['target', 'ID_code'])])
```

```
train_scaled = StandardScaler().fit_transform(df_train[df_train.columns.difference(['target', 'ID_code'])])

PCA_train = PCA(n_components=2).fit_transform(train_scaled)
```

In [0]:

```
plt.figure(figsize=(10, 8))
plt.scatter(PCA_train[:, 0], PCA_train[:, 1], c=df_train.target.values, cmap='viridis',)

plt.xlabel('Component-1')
plt.ylabel('Component-2')
plt.colorbar()
plt.show()
```

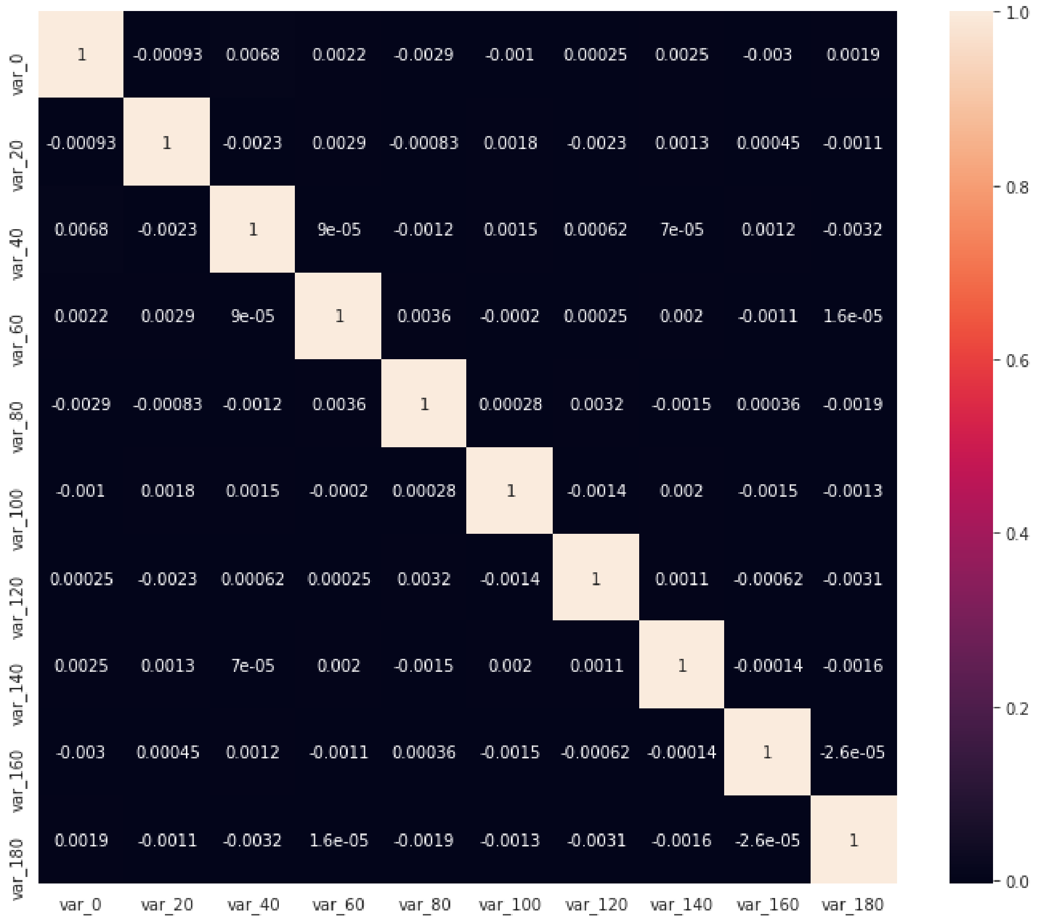


- Even with the PCA the datapoints for both classes are almost overlapping that doesn't seems meaningful.

In [0]:

```
corr = df_train[eda_features].corr()
plt.figure(figsize=(12, 10))
```

```python
sns.heatmap(corr, annot=True) # Correlation heatmap for choosen EDA features.
plt.show()
```



In [0]:

```python
corr = df_train[df_train.columns.difference(['ID_code', 'target'])].corr()
corr = corr.abs().unstack().sort_values(ascending=False).reset_index()
corr = corr[corr['level_0'] != corr['level_1']]
corr.columns = ['Feature_1', 'Feature_2', 'Correlation']

corr.head(10) # Top 10 correlated features.
```

Out[0]:

Feature_1	Feature_2	Correlation

	Feature_1	Feature_2	Correlation
200	var_139	var_26	0.009844
201	var_26	var_139	0.009844
202	var_53	var_148	0.009788
203	var_148	var_53	0.009788
204	var_81	var_165	0.009714
205	var_165	var_81	0.009714
206	var_81	var_174	0.009490
207	var_174	var_81	0.009490
208	var_183	var_189	0.009359
209	var_189	var_183	0.009359

In [0]:

```
corr.tail(10) # Least 10 correlated features.
```

Out[0]:

	Feature_1	Feature_2	Correlation
39990	var_100	var_177	3.116544e-07
39991	var_177	var_100	3.116544e-07
39992	var_27	var_144	1.772502e-07
39993	var_144	var_27	1.772502e-07
39994	var_126	var_109	1.313947e-07
39995	var_109	var_126	1.313947e-07
39996	var_6	var_173	5.942735e-08
39997	var_173	var_6	5.942735e-08
39998	var_75	var_191	2.703975e-08
39999	var_191	var_75	2.703975e-08

- The correlation between the pair of features is less which implies features are independent.

Basic Features :

In [7]:

```
features = df_train.columns.difference(['ID_code', 'target'])
```

```
feat_names = ['mean', 'std', 'max', 'min', 'median']
feat_funcs = [np.mean, np.std, np.max, np.min, np.median]

for feat, func in tqdm(zip(feat_names, feat_funcs)):
  df_train[feat] = df_train[features].apply(func, axis=1)
  df_test[feat] = df_test[features].apply(func, axis=1)
```

```
5it [01:24, 17.99s/it]
```

In [0]:

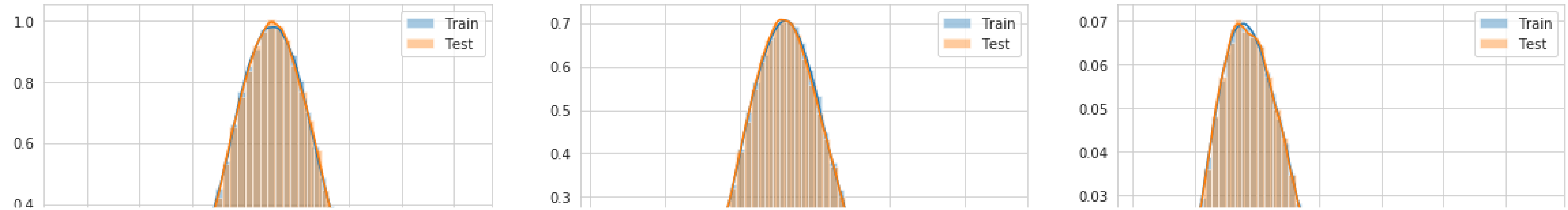
```
df_train[feat_names].head()
```

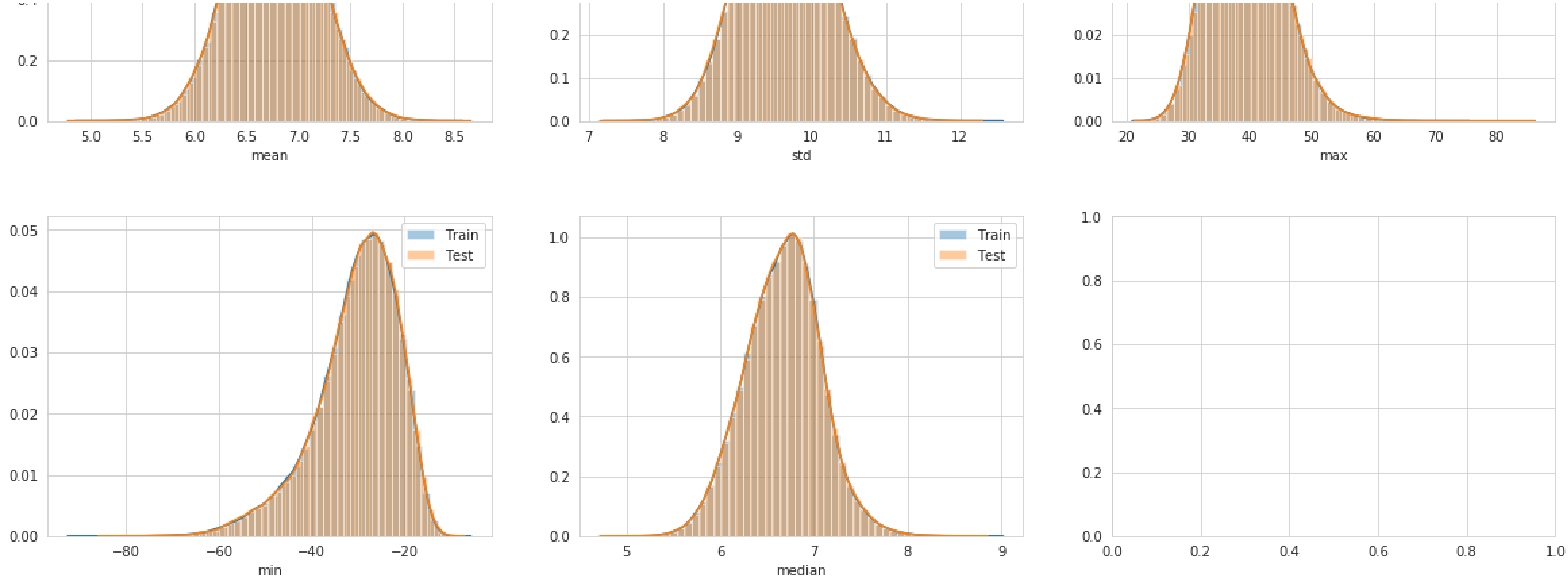
Out[0]:

	mean	std	max	min	median
0	7.281591	9.308182	43.1127	-21.4494	6.77040
1	7.076818	10.310257	40.5632	-47.3797	7.22315
2	6.204483	8.731476	33.8820	-22.4038	5.89940
3	6.441160	9.570048	38.1015	-35.1659	6.70260
4	6.771155	11.258868	41.1037	-65.4863	6.94735

In [0]:

```
figure, axes = plt.subplots(2, 3, figsize=(20,10))
axes = axes.flatten()

for idx, feat in enumerate(feat_names):
  sns.distplot(df_train[feat], ax=axes[idx], label='Train')
  sns.distplot(df_test[feat], ax=axes[idx], label='Test').set_xlabel(feat)
  axes[idx].legend()

plt.subplots_adjust(hspace=.3)
plt.show()
```

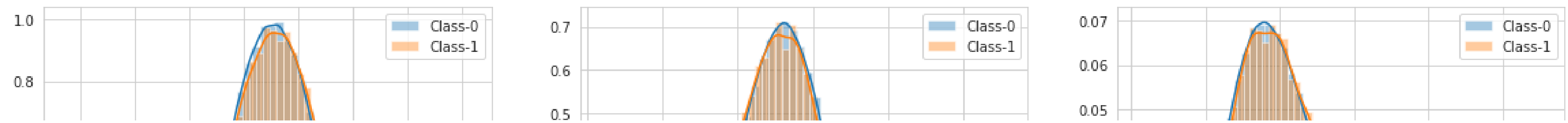


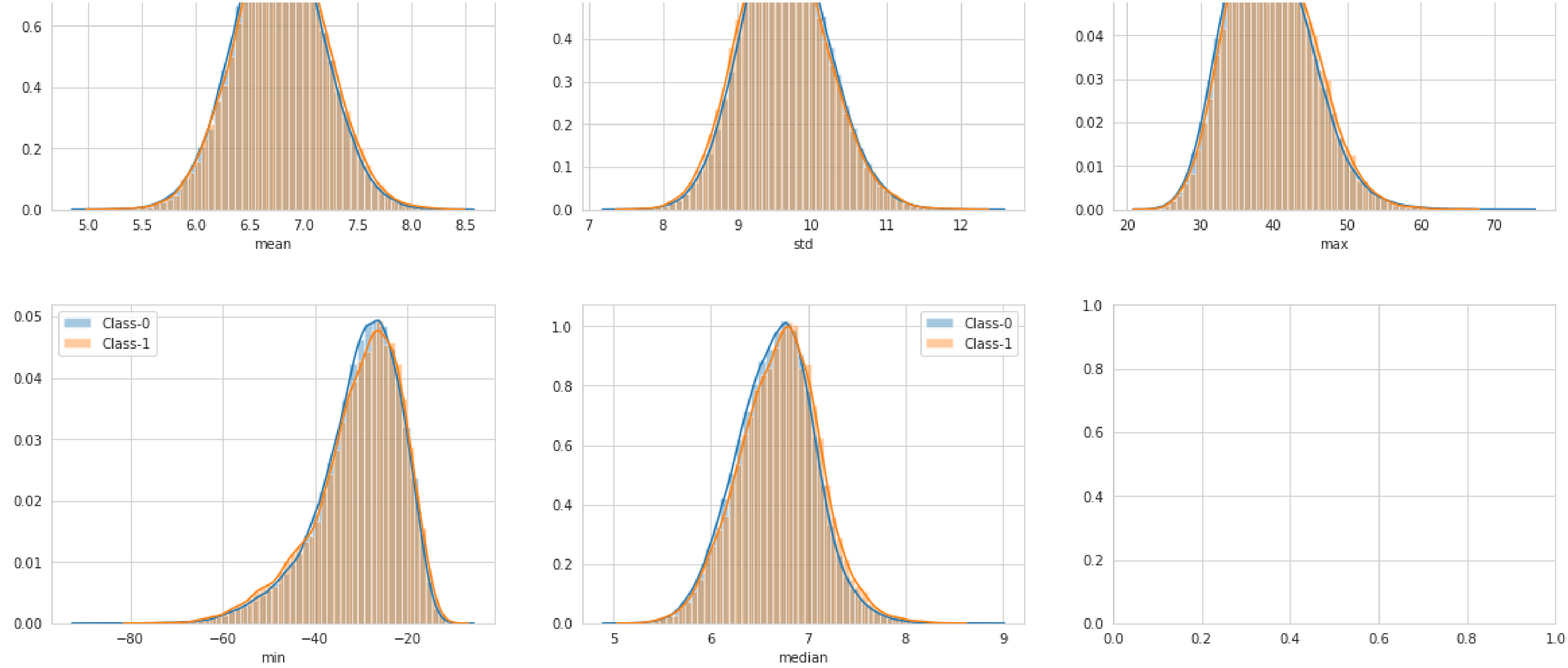


- The basic features seems to follow the gaussian distribution.

In [0]:

```
df_train_0 = df_train[df_train.target == 0][feat_names]
df_train_1 = df_train[df_train.target == 1][feat_names]

figure, axes = plt.subplots(2, 3, figsize=(20,10))
axes = axes.flatten()

for idx, feat in enumerate(feat_names):
  sns.distplot(df_train_0[feat], ax=axes[idx], label='Class-0')
  sns.distplot(df_train_1[feat], ax=axes[idx], label='Class-1').set_xlabel(feat)
  axes[idx].legend()

plt.subplots_adjust(hspace=.3)
plt.show()
```





Modelling [1] - w/ basic features.

In [0]:

```
X = df_train[df_train.columns.difference(['ID_code', 'target'])]
y = df_train.target.values

X_train, X_test, y_train,  y_test = train_test_split(X, y, test_size=.2, random_state=23, stratify=y)

pt_1 = PrettyTable()
pt_1.field_names = ['Model', 'Hyper Parameters', 'Train AUC', 'Test AUC']
```

Logistic Regression(LR) :

RandomSearch :

In [0]:

```
clf = linear_model.SGDClassifier('log', n_jobs=-1, class_weight='balanced')

params = dict(alpha=st.uniform(.00001, .01))
search = RandomizedSearchCV(clf, params, scoring='roc_auc', n_jobs=-1, verbose=10, return_train_score=True, cv=3)
```

In [0]:

```
with parallel_backend('threading'):
  search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[CV] alpha=0.004119678779936287 .....................................
[CV] alpha=0.004119678779936287 .....................................
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
```

```
[CV]  alpha=0.004119678779936287, score=(train=0.859, test=0.855), total=  22.7s
[CV] alpha=0.004119678779936287 .....................................
```

```
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   22.9s
```

```
[CV]  alpha=0.004119678779936287, score=(train=0.857, test=0.855), total=  25.5s
[CV] alpha=0.0020817756520198705 ....................................
[CV]  alpha=0.004119678779936287, score=(train=0.859, test=0.853), total=  23.8s
[CV] alpha=0.0020817756520198705 ....................................
[CV]  alpha=0.0020817756520198705, score=(train=0.857, test=0.857), total=  37.7s
[CV] alpha=0.0020817756520198705 ....................................
```

```
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  1.1min
```

```
[CV]  alpha=0.0020817756520198705, score=(train=0.860, test=0.855), total=  34.2s
[CV] alpha=0.006450444852612077 .....................................
[CV]  alpha=0.0020817756520198705, score=(train=0.856, test=0.851), total=  30.3s
[CV] alpha=0.006450444852612077 .....................................
[CV]  alpha=0.006450444852612077, score=(train=0.857, test=0.854), total=  19.0s
[CV] alpha=0.006450444852612077 .....................................
[CV]  alpha=0.006450444852612077, score=(train=0.858, test=0.854), total=  19.8s
[CV] alpha=0.003828692542999418 .....................................
[CV]  alpha=0.006450444852612077, score=(train=0.860, test=0.856), total=  16.3s
[CV] alpha=0.003828692542999418 .....................................
```

```
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  1.9min
```

```
[CV]  alpha=0.003828692542999418, score=(train=0.857, test=0.853), total=  22.7s
[CV] alpha=0.003828692542999418
```

```
[CV] alpha=0.003828692542999418 ....................................
[CV]  alpha=0.003828692542999418, score=(train=0.858, test=0.857), total=  26.4s
[CV] alpha=0.008794147947487162 ....................................
[CV]  alpha=0.008794147947487162, score=(train=0.857, test=0.855), total=  16.3s
[CV] alpha=0.008794147947487162 ....................................
[CV]  alpha=0.003828692542999418, score=(train=0.859, test=0.855), total=  24.1s
[CV] alpha=0.008794147947487162 ....................................
[CV]  alpha=0.008794147947487162, score=(train=0.860, test=0.857), total=  21.1s
[CV] alpha=0.001390875693309549 ....................................
```

```
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  3.0min
```

```
[CV]  alpha=0.008794147947487162, score=(train=0.859, test=0.853), total=  15.4s
[CV] alpha=0.001390875693309549 ....................................
[CV]  alpha=0.001390875693309549, score=(train=0.859, test=0.855), total=  33.9s
[CV] alpha=0.001390875693309549 ....................................
[CV]  alpha=0.001390875693309549, score=(train=0.856, test=0.854), total=  40.3s
[CV] alpha=0.006254124023839812 ....................................
[CV]  alpha=0.006254124023839812, score=(train=0.855, test=0.853), total=  17.0s
[CV] alpha=0.006254124023839812 ....................................
[CV]  alpha=0.001390875693309549, score=(train=0.857, test=0.853), total=  35.8s
[CV] alpha=0.006254124023839812 ....................................
[CV]  alpha=0.006254124023839812, score=(train=0.859, test=0.855), total=  20.1s
[CV] alpha=0.006669809715255771 ....................................
[CV]  alpha=0.006254124023839812, score=(train=0.859, test=0.855), total=  21.0s
[CV] alpha=0.006669809715255771 ....................................
```

```
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:  4.5min
```

```
[CV]  alpha=0.006669809715255771, score=(train=0.859, test=0.857), total=  20.5s
[CV] alpha=0.006669809715255771 ....................................
[CV]  alpha=0.006669809715255771, score=(train=0.858, test=0.854), total=  16.6s
[CV] alpha=0.005264553242148463 ....................................
[CV]  alpha=0.006669809715255771, score=(train=0.859, test=0.853), total=  19.4s
[CV] alpha=0.005264553242148463 ....................................
[CV]  alpha=0.005264553242148463, score=(train=0.861, test=0.860), total=  22.0s
[CV] alpha=0.005264553242148463 ....................................
[CV]  alpha=0.005264553242148463, score=(train=0.860, test=0.856), total=  23.3s
[CV] alpha=0.005229899758690161 ....................................
[CV]  alpha=0.005264553242148463, score=(train=0.858, test=0.856), total=  17.2s
[CV] alpha=0.005229899758690161 ....................................
[CV]  alpha=0.005229899758690161, score=(train=0.857, test=0.854), total=  22.7s
[CV] alpha=0.005229899758690161 ....................................
[CV]  alpha=0.005229899758690161, score=(train=0.861, test=0.858), total=  20.8s
[CV]  alpha=0.005229899758690161, score=(train=0.858, test=0.854), total=  15.3s
```

```
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  6.0min finished
```

In [0]:

```
search.best_params_
```

Out[0]:

```
{'alpha': 0.005264553242148463}
```

In [0]:

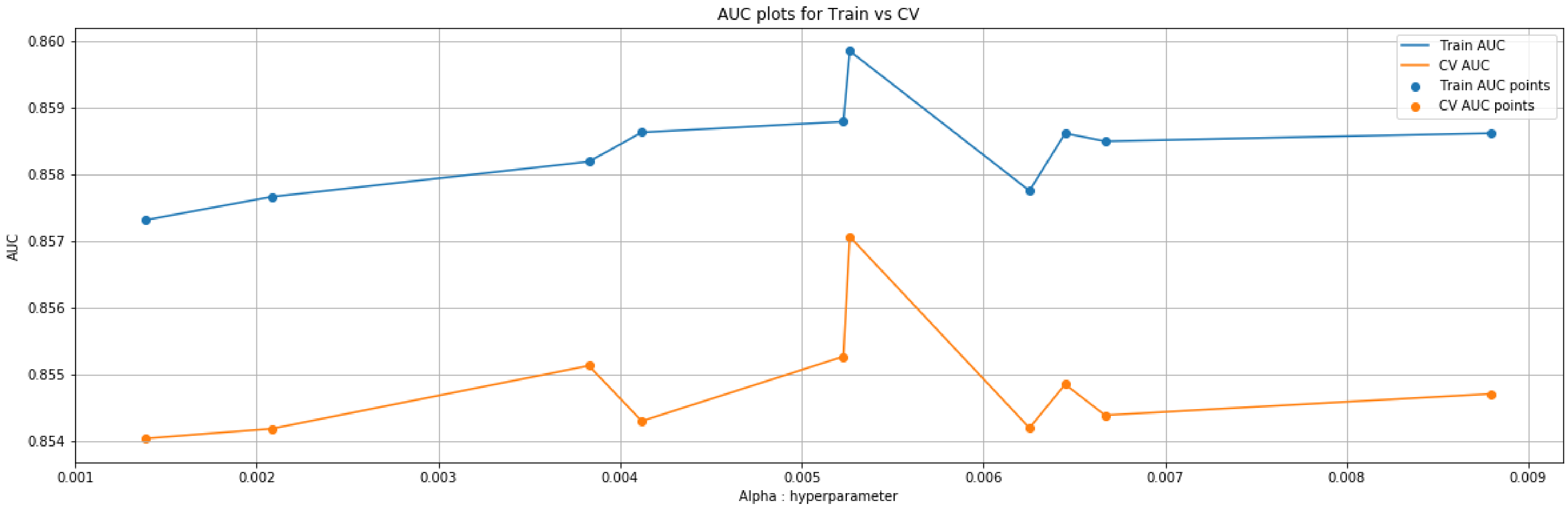
```
res = pd.DataFrame(search.cv_results_)
```

In [0]:

```
idxs = np.argsort(res.param_alpha.values.astype('float64'))
```

In [0]:

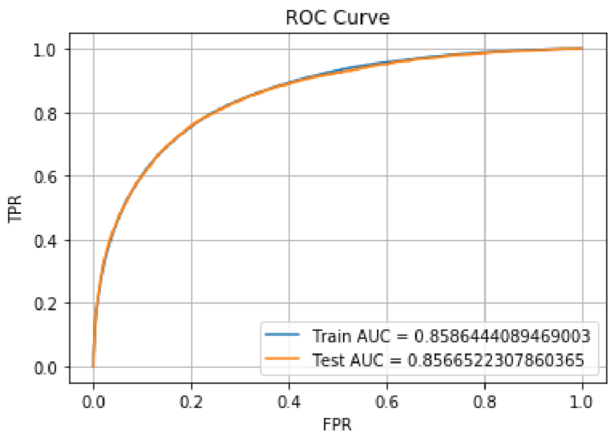
```
auc_plot(res.param_alpha.values.astype('float64')[idxs],\
         res.mean_train_score.values[idxs], res.mean_test_score.values[idxs], 'Alpha')
```



In [0]:

```
y_train_pred = search.best_estimator_.predict_proba(X_train)[:,1]
y_test_pred = search.best_estimator_.predict_proba(X_test)[:,1]
```

```
train_fpr, train_tpr, tr_thresholds = metrics.roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = metrics.roc_curve(y_test, y_test_pred)

roc_plot(train_fpr, train_tpr, test_fpr, test_tpr)
```



GridSearch :

In [0]:

```
clf = linear_model.SGDClassifier('log', n_jobs=-1, class_weight='balanced')

params = dict(alpha=[.00001, .0001, .001, .01, .1, 1, 10])
search = GridSearchCV(clf, params, scoring='roc_auc', n_jobs=-1, verbose=10, return_train_score=True, cv=3)
```

In [0]:

```
with parallel_backend('threading'):
  search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 7 candidates, totalling 21 fits
[CV] alpha=1e-05 .....................................................
[CV] alpha=1e-05 .....................................................
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
```

```
[CV] ..... alpha=1e-05, score=(train=0.841, test=0.838), total=  44.7s
[CV] alpha=1e-05 .....................................................
```

```
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   44.8s
```

```
[CV] ..... alpha=1e-05, score=(train=0.836, test=0.834), total=  48.6s
[CV] alpha=0.0001 ....................................................
[CV] .... alpha=0.0001, score=(train=0.845, test=0.845), total=  49.0s
[CV] alpha=0.0001 ....................................................
[CV] ..... alpha=1e-05, score=(train=0.836, test=0.834), total=  53.7s
[CV] alpha=0.0001 ....................................................
```

```
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  1.6min
```

```
[CV] .... alpha=0.0001, score=(train=0.835, test=0.835), total=  38.9s
[CV] alpha=0.001 .....................................................
[CV] .... alpha=0.0001, score=(train=0.843, test=0.840), total=  55.8s
[CV] alpha=0.001 .....................................................
[CV] ..... alpha=0.001, score=(train=0.851, test=0.848), total=  43.2s
[CV] alpha=0.001 .....................................................
[CV] ..... alpha=0.001, score=(train=0.858, test=0.854), total=  49.4s
[CV] alpha=0.01 ......................................................
[CV] ...... alpha=0.01, score=(train=0.858, test=0.856), total=  16.0s
[CV] alpha=0.01 ......................................................
```

```
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  3.7min
```

```
[CV] ..... alpha=0.001, score=(train=0.850, test=0.849), total=  38.9s
[CV] alpha=0.01 ......................................................
[CV] ...... alpha=0.01, score=(train=0.859, test=0.854), total=  14.8s
[CV] alpha=0.1 .......................................................
[CV] ...... alpha=0.01, score=(train=0.860, test=0.857), total=  17.4s
[CV] alpha=0.1 .......................................................
[CV] ....... alpha=0.1, score=(train=0.848, test=0.846), total=   5.6s
[CV] alpha=0.1 .......................................................
[CV] ....... alpha=0.1, score=(train=0.848, test=0.846), total=   5.3s
[CV] alpha=1 .........................................................
```

```
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  4.0min
```

```
[CV] ......... alpha=1, score=(train=0.830, test=0.829), total=   2.5s
[CV] alpha=1 .........................................................
[CV] ....... alpha=0.1, score=(train=0.849, test=0.843), total=   5.4s
[CV] alpha=1 .........................................................
[CV] ......... alpha=1, score=(train=0.831, test=0.827), total=   2.6s
[CV] alpha=10 ........................................................
[CV] ......... alpha=1, score=(train=0.831, test=0.827), total=   2.4s
[CV] alpha=10 ........................................................
[CV] ........ alpha=10, score=(train=0.796, test=0.795), total=   1.8s
[CV] alpha=10 ........................................................
[CV] ........ alpha=10, score=(train=0.796, test=0.791), total=   1.9s
[CV] ........ alpha=10, score=(train=0.797, test=0.796), total=   1.6s
```

```
[CV] ....... alpha=10, score=(train=0.797, test=0.796), total=   1.6s
```

```
[Parallel(n_jobs=-1)]: Done  21 out of  21 | elapsed:  4.2min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  21 out of  21 | elapsed:  4.2min finished
```

In [0]:

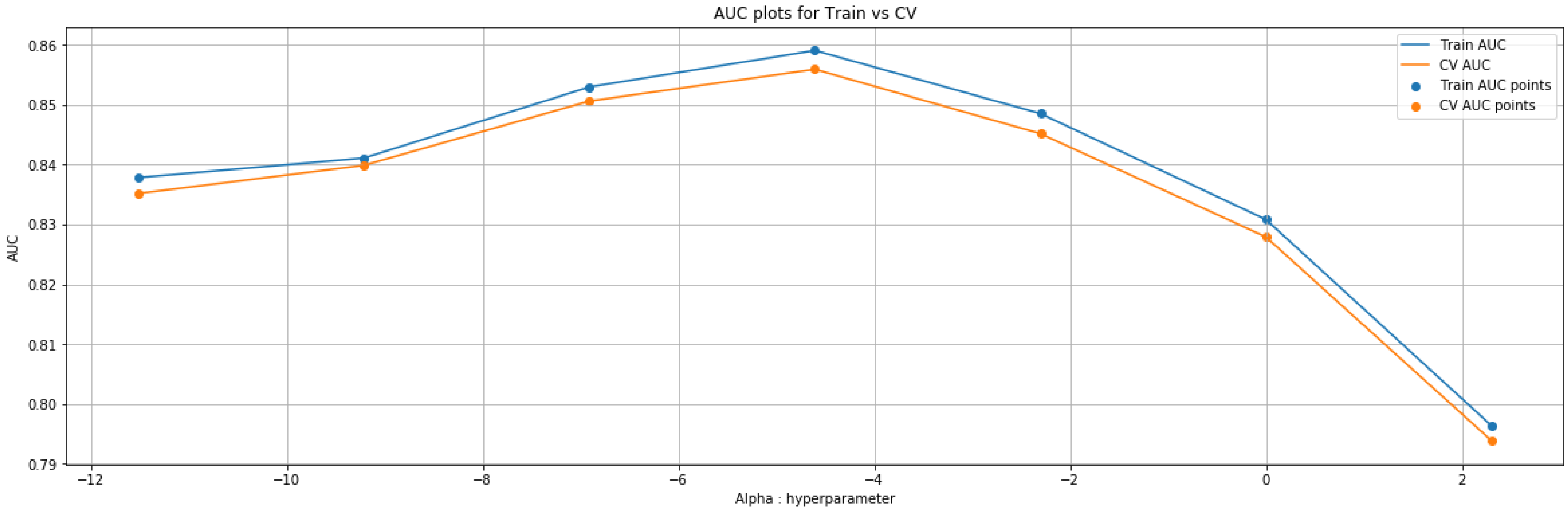
```
search.best_params_
```

Out[0]:

```
{'alpha': 0.01}
```

In [0]:

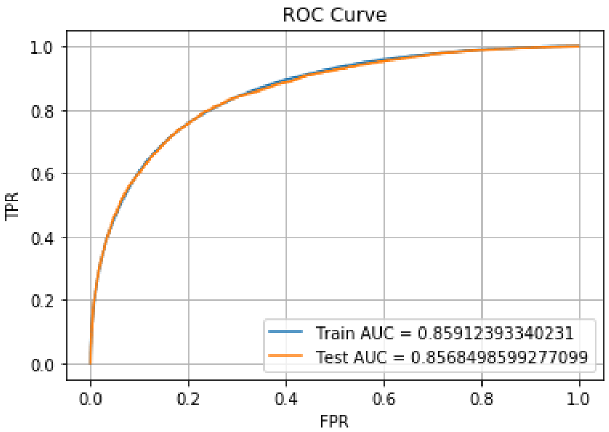
```
res = pd.DataFrame(search.cv_results_)

auc_plot(np.log(res.param_alpha.values.astype('float64')),\
         res.mean_train_score.values, res.mean_test_score.values, 'Alpha')
```



In [0]:

```
y_train_pred = search.best_estimator_.predict_proba(X_train)[:,1]
y_test_pred = search.best_estimator_.predict_proba(X_test)[:,1]
```

```
train_fpr, train_tpr, tr_thresholds = metrics.roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = metrics.roc_curve(y_test, y_test_pred)

roc_plot(train_fpr, train_tpr, test_fpr, test_tpr)
```



In [0]:

```
pt_1 = PrettyTable()
pt_1.field_names = ['Model', 'Hyper Paramters', 'Train AUC', 'Test AUC']

pt_1.add_row(['Logistic Regression', 'alpha = 0.01', np.round(.85912393340231, 3), np.round(.8568498599277099, 3)])
```

- The results were almost similar with simple SGD based Logisitc Regression with test **AUROC** of `".856"`.

Random Forest

In [0]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [0]:

```
clf = RandomForestClassifier(class_weight='balanced', n_jobs=-1, oob_score=True)

params = dict(max_depth=[5, 8, 10], n_estimators=[25, 50, 100], min_samples_split=[5, 10, 100])

search = GridSearchCV(clf, params, scoring='roc_auc', n_jobs=-1, verbose=10, return_train_score=True, cv=2)
```

In [0]:

```
with parallel_backend('threading'):
  search.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 27 candidates, totalling 54 fits
[CV] max_depth=5, min_samples_split=5, n_estimators=25 ...............[CV] max_depth=5, min_samples_split=5, n_estimators=25 ...............
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
```

```
[CV]  max_depth=5, min_samples_split=5, n_estimators=25, score=(train=0.810, test=0.770), total= 1.0min
[CV]  max_depth=5, min_samples_split=5, n_estimators=25, score=(train=0.796, test=0.759), total= 1.0min
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

```
[CV]  max_depth=5, min_samples_split=5, n_estimators=25, score=(train=0.790, test=0.752), total= 1.0min
[CV] max_depth=5, min_samples_split=5, n_estimators=50 ...............
```

```
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  1.0min
```

```
[CV]  max_depth=5, min_samples_split=5, n_estimators=25, score=(train=0.802, test=0.760), total= 1.0min
[CV] max_depth=5, min_samples_split=5, n_estimators=50 ...............
[CV]  max_depth=5, min_samples_split=5, n_estimators=50, score=(train=0.821, test=0.781), total= 1.4min
[CV] max_depth=5, min_samples_split=5, n_estimators=100 ..............
[CV]  max_depth=5, min_samples_split=5, n_estimators=50, score=(train=0.807, test=0.771), total= 1.4min
[CV] max_depth=5, min_samples_split=5, n_estimators=100 ..............
```

```
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  2.4min
```

```
[CV]  max_depth=15, min_samples_split=5, n_estimators=150, score=(train=1.000, test=0.774), total= 6.9min
[CV]  max_depth=15, min_samples_split=5, n_estimators=150, score=(train=1.000, test=0.770), total= 7.2min
[CV]  max_depth=5, min_samples_split=5, n_estimators=100, score=(train=0.834, test=0.794), total= 2.0min
[CV] max_depth=5, min_samples_split=10, n_estimators=25 ..............
[CV]  max_depth=5, min_samples_split=5, n_estimators=100, score=(train=0.824, test=0.783), total= 2.0min
[CV] max_depth=5, min_samples_split=10, n_estimators=25 ..............
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few
trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few
trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

```
[CV]  max_depth=5, min_samples_split=10, n_estimators=25, score=(train=0.798, test=0.759), total=  20.7s
[CV] max_depth=5, min_samples_split=10, n_estimators=50 .............
[CV]  max_depth=5, min_samples_split=10, n_estimators=25, score=(train=0.811, test=0.773), total=  20.8s
[CV] max_depth=5, min_samples_split=10, n_estimators=50 .............
[CV]  max_depth=5, min_samples_split=10, n_estimators=50, score=(train=0.814, test=0.773), total=  40.6s
[CV] max_depth=5, min_samples_split=10, n_estimators=100 ............
```

```
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  5.5min
```

```
[CV]  max_depth=5, min_samples_split=10, n_estimators=50, score=(train=0.821, test=0.781), total=  40.7s
[CV] max_depth=5, min_samples_split=10, n_estimators=100 ............
[CV]  max_depth=5, min_samples_split=10, n_estimators=100, score=(train=0.830, test=0.790), total= 1.3min
[CV] max_depth=5, min_samples_split=100, n_estimators=25 ............
[CV]  max_depth=5, min_samples_split=10, n_estimators=100, score=(train=0.825, test=0.786), total= 1.3min
[CV] max_depth=5, min_samples_split=100, n_estimators=25 ............
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few
trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

```
[CV]  max_depth=5, min_samples_split=100, n_estimators=25, score=(train=0.795, test=0.760), total=  20.2s
[CV] max_depth=5, min_samples_split=100, n_estimators=50 ............
[CV]  max_depth=5, min_samples_split=100, n_estimators=25, score=(train=0.796, test=0.760), total=  20.2s
[CV] max_depth=5, min_samples_split=100, n_estimators=50 ............
```

```
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  7.2min
```

```
[CV]  max_depth=5, min_samples_split=100, n_estimators=50, score=(train=0.814, test=0.780), total=  40.2s
[CV] max_depth=5, min_samples_split=100, n_estimators=100 ...........
[CV]  max_depth=5, min_samples_split=100, n_estimators=50, score=(train=0.819, test=0.780), total=  40.3s
[CV] max_depth=5, min_samples_split=100, n_estimators=100 ...........
[CV]  max_depth=5, min_samples_split=100, n_estimators=100, score=(train=0.822, test=0.785), total= 1.3min
[CV] max_depth=8, min_samples_split=5, n_estimators=25 ..............
[CV]  max_depth=5, min_samples_split=100, n_estimators=100, score=(train=0.826, test=0.787), total= 1.3min
[CV] max_depth=8, min_samples_split=5, n_estimators=25 ..............
[CV]  max_depth=8, min_samples_split=5, n_estimators=25, score=(train=0.886, test=0.758), total=  30.6s
```

```
[CV] max_depth=8, min_samples_split=5, n_estimators=50 ..............
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

```
[CV]  max_depth=8, min_samples_split=5, n_estimators=25, score=(train=0.876, test=0.758), total=  30.8s
[CV] max_depth=8, min_samples_split=5, n_estimators=50 ...............
[CV]  max_depth=8, min_samples_split=5, n_estimators=50, score=(train=0.899, test=0.783), total= 1.0min
[CV] max_depth=8, min_samples_split=5, n_estimators=100 .............
```

```
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 10.8min
```

```
[CV]  max_depth=8, min_samples_split=5, n_estimators=50, score=(train=0.898, test=0.786), total= 1.0min
[CV] max_depth=8, min_samples_split=5, n_estimators=100 .............
[CV]  max_depth=8, min_samples_split=5, n_estimators=100, score=(train=0.909, test=0.796), total= 2.0min
[CV] max_depth=8, min_samples_split=10, n_estimators=25 .............
[CV]  max_depth=8, min_samples_split=5, n_estimators=100, score=(train=0.910, test=0.796), total= 2.0min
[CV] max_depth=8, min_samples_split=10, n_estimators=25 .............
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

```
[CV]  max_depth=8, min_samples_split=10, n_estimators=25, score=(train=0.877, test=0.758), total=  30.5s
[CV] max_depth=8, min_samples_split=10, n_estimators=50 .............
[CV]  max_depth=8, min_samples_split=10, n_estimators=25, score=(train=0.880, test=0.766), total=  30.8s
[CV] max_depth=8, min_samples_split=10, n_estimators=50 .............
[CV]  max_depth=8, min_samples_split=10, n_estimators=50, score=(train=0.904, test=0.784), total= 1.0min
[CV] max_depth=8, min_samples_split=10, n_estimators=100 ............
[CV]  max_depth=8, min_samples_split=10, n_estimators=50, score=(train=0.903, test=0.786), total= 1.0min
[CV] max_depth=8, min_samples_split=10, n_estimators=100 ............
```

```
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed: 14.5min
```

```
[CV]  max_depth=8, min_samples_split=10, n_estimators=100, score=(train=0.908, test=0.796), total= 2.0min
[CV] max_depth=8, min_samples_split=100, n_estimators=25 ............
[CV]  max_depth=8, min_samples_split=10, n_estimators=100, score=(train=0.911, test=0.796), total= 2.1min
[CV] max_depth=8, min_samples_split=100, n_estimators=25 ............
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
```

```
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

```
[CV]  max_depth=8, min_samples_split=100, n_estimators=25, score=(train=0.877, test=0.782), total=  29.6s
[CV] max_depth=8, min_samples_split=100, n_estimators=50 .............
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

```
[CV]  max_depth=8, min_samples_split=100, n_estimators=25, score=(train=0.870, test=0.769), total=  30.9s
[CV] max_depth=8, min_samples_split=100, n_estimators=50 .............
[CV]  max_depth=8, min_samples_split=100, n_estimators=50, score=(train=0.892, test=0.796), total= 1.0min
[CV] max_depth=8, min_samples_split=100, n_estimators=100 ............
[CV]  max_depth=8, min_samples_split=100, n_estimators=50, score=(train=0.884, test=0.788), total= 1.0min
[CV] max_depth=8, min_samples_split=100, n_estimators=100 ............
[CV]  max_depth=8, min_samples_split=100, n_estimators=100, score=(train=0.896, test=0.804), total= 2.0min
[CV] max_depth=10, min_samples_split=5, n_estimators=25 ..............
[CV]  max_depth=8, min_samples_split=100, n_estimators=100, score=(train=0.899, test=0.805), total= 2.0min
[CV] max_depth=10, min_samples_split=5, n_estimators=25 ..............
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

```
[CV]  max_depth=10, min_samples_split=5, n_estimators=25, score=(train=0.941, test=0.743), total=  34.7s
[CV] max_depth=10, min_samples_split=5, n_estimators=50 ..............
```

```
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 20.6min
```

```
[CV]  max_depth=10, min_samples_split=5, n_estimators=25, score=(train=0.944, test=0.743), total=  36.8s
[CV] max_depth=10, min_samples_split=5, n_estimators=50 ..............
[CV]  max_depth=10, min_samples_split=5, n_estimators=50, score=(train=0.958, test=0.770), total= 1.2min
[CV] max_depth=10, min_samples_split=5, n_estimators=100 .............
[CV]  max_depth=10, min_samples_split=5, n_estimators=50, score=(train=0.961, test=0.776), total= 1.2min
[CV] max_depth=10, min_samples_split=5, n_estimators=100 .............
[CV]  max_depth=10, min_samples_split=5, n_estimators=100, score=(train=0.969, test=0.792), total= 2.5min
[CV] max_depth=10, min_samples_split=10, n_estimators=25 .............
[CV]  max_depth=10, min_samples_split=5, n_estimators=100, score=(train=0.964, test=0.793), total= 2.5min
[CV] max_depth=10, min_samples_split=10, n_estimators=25 .............
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
```

```
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

```
[CV]  max_depth=10, min_samples_split=10, n_estimators=25, score=(train=0.943, test=0.747), total=  34.8s
[CV] max_depth=10, min_samples_split=10, n_estimators=50 .............
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:460: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

```
[CV]  max_depth=10, min_samples_split=10, n_estimators=25, score=(train=0.938, test=0.744), total=  37.4s
[CV] max_depth=10, min_samples_split=10, n_estimators=50 .............
[CV]  max_depth=10, min_samples_split=10, n_estimators=50, score=(train=0.956, test=0.778), total= 1.2min
[CV] max_depth=10, min_samples_split=10, n_estimators=100 ............
[CV]  max_depth=10, min_samples_split=10, n_estimators=50, score=(train=0.952, test=0.773), total= 1.2min
[CV] max_depth=10, min_samples_split=10, n_estimators=100 ............
```

```
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 26.5min
```

```
[CV]  max_depth=10, min_samples_split=10, n_estimators=100, score=(train=0.968, test=0.792), total= 2.4min
[CV] max_depth=10, min_samples_split=100, n_estimators=25 ............
[CV]  max_depth=10, min_samples_split=10, n_estimators=100, score=(train=0.968, test=0.795), total= 2.5min
[CV] max_depth=10, min_samples_split=100, n_estimators=25 ............
[CV]  max_depth=10, min_samples_split=100, n_estimators=25, score=(train=0.917, test=0.775), total=  34.6s
[CV] max_depth=10, min_samples_split=100, n_estimators=50 ............
[CV]  max_depth=10, min_samples_split=100, n_estimators=25, score=(train=0.908, test=0.775), total=  36.9s
[CV] max_depth=10, min_samples_split=100, n_estimators=50 ............
[CV]  max_depth=10, min_samples_split=100, n_estimators=50, score=(train=0.934, test=0.798), total= 1.2min
[CV] max_depth=10, min_samples_split=100, n_estimators=100 ...........
[CV]  max_depth=10, min_samples_split=100, n_estimators=50, score=(train=0.933, test=0.796), total= 1.2min
[CV] max_depth=10, min_samples_split=100, n_estimators=100 ...........
[CV]  max_depth=10, min_samples_split=100, n_estimators=100, score=(train=0.941, test=0.810), total= 2.4min
[CV]  max_depth=10, min_samples_split=100, n_estimators=100, score=(train=0.939, test=0.809), total= 2.3min
```

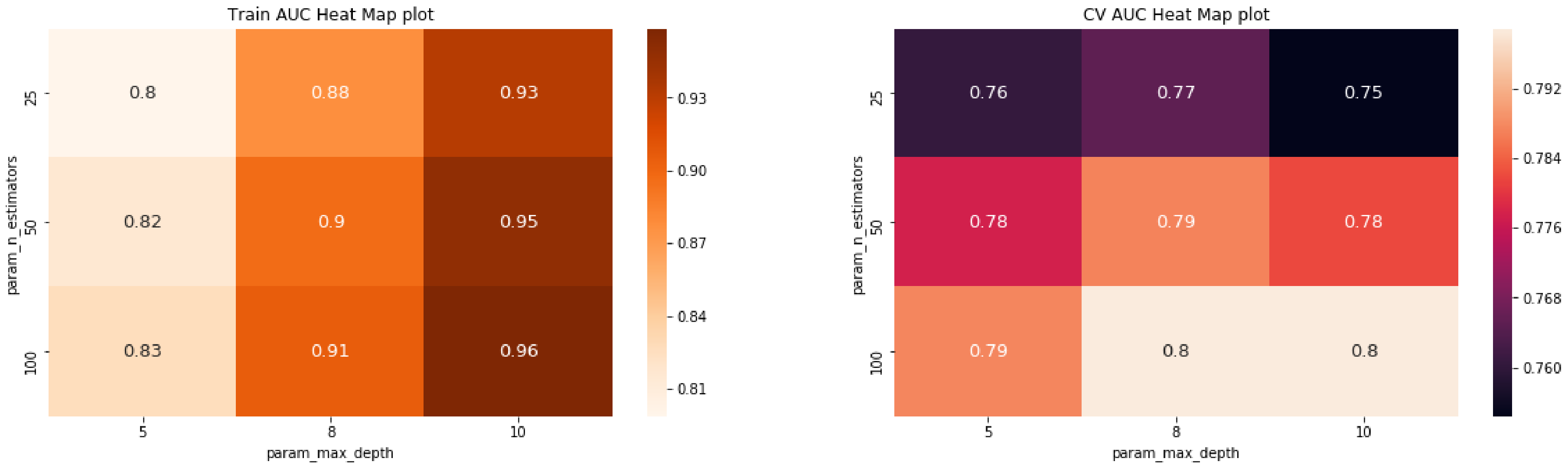
```
[Parallel(n_jobs=-1)]: Done  54 out of  54 | elapsed: 33.1min finished
```

In [0]:

```
res = pd.DataFrame(search.cv_results_)
```

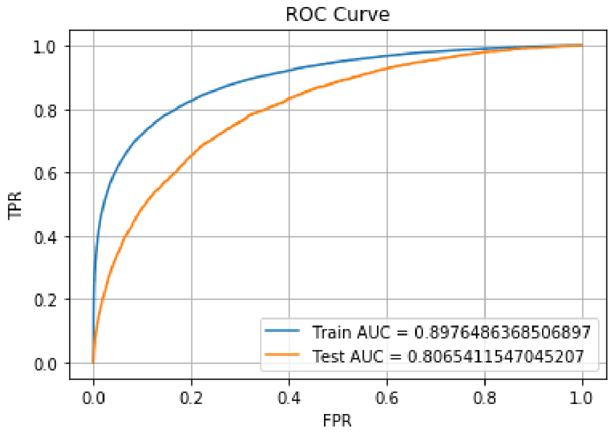
In [0]:

```
plotTrainVsCV_AUC(search, subplots=(1, 2), figsize=(20, 5) idx='param_n_estimators', cols='param_max_depth')
```



In [0]:

```
y_train_pred = search.best_estimator_.predict_proba(X_train)[:,1]
y_test_pred = search.best_estimator_.predict_proba(X_test)[:,1]

train_fpr, train_tpr, tr_thresholds = metrics.roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = metrics.roc_curve(y_test, y_test_pred)

roc_plot(train_fpr, train_tpr, test_fpr, test_tpr)
```



- Tried Decision Tree and Random Forest but didn't gave better results as both were completely over-fitting on training data even with the various hyper parameters

- Tried Decision Tree and Random Forest but didn't gave better results as both were completely over fitting on training data even with the various hyper parameters.

In [0]:

```
pt_1.add_row(['', '', '', ''])
pt_1.add_row(['Random Forest', 'Max_depth = 8, n_estimators = 100',\
              np.round(.8976486368506897, 3), np.round(.8065411547045207, 3)])
```

XGBOOST

In [0]:

```
one_to_left = st.beta(10, 1)
from_zero_positive = st.expon(0, 50)

params = {
    "n_estimators": st.randint(3, 200),
    "max_depth": st.randint(3, 10),
    "learning_rate": st.uniform(0.001, 0.1),
    "colsample_bytree": one_to_left,
    "subsample": one_to_left,
    "gamma": st.uniform(0, 10),
    "reg_lambda": st.uniform(0.001, 0.1),
    "min_child_weight": from_zero_positive,
}

search = RandomizedSearchCV(xgb.XGBClassifier(eval_metric='auc', objective='binary:logistic', n_jobs=-1, tree_method='auto'),\
                            params, scoring='roc_auc', n_jobs=-1, verbose=10, return_train_score=True, cv=2)
```

In [0]:

```
with parallel_backend('multiprocessing'):
  search.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
```

```
[Parallel(n_jobs=-1)]: Using backend MultiprocessingBackend with 2 concurrent workers.
```

```
[CV] colsample_bytree=0.892207906442659, gamma=1.7488800353518186, learning_rate=0.06747751081039131, max_depth=8,
min_child_weight=21.851606190465088, n_estimators=93, reg_lambda=0.08253053315245326, subsample=0.9226774110370277
[CV] colsample_bytree=0.892207906442659, gamma=1.7488800353518186, learning_rate=0.06747751081039131, max_depth=8,
min_child_weight=21.851606190465088, n_estimators=93, reg_lambda=0.08253053315245326, subsample=0.9226774110370277
[CV]  colsample_bytree=0.892207906442659, gamma=1.7488800353518186, learning_rate=0.06747751081039131, max_depth=8,
min_child_weight=21.851606190465088, n_estimators=93, reg_lambda=0.08253053315245326, subsample=0.9226774110370277, score=(train=0.954,
test=0.849), total= 7.8min
[CV]  colsample_bytree=0.892207906442659, gamma=1.7488800353518186, learning_rate=0.06747751081039131, max_depth=8,
min_child_weight=21.851606190465088, n_estimators=93, reg_lambda=0.08253053315245326, subsample=0.9226774110370277, score=(train=0.956,
test=0.849), total= 7.8min
```

```
[CV] colsample_bytree=0.8554388572158456, gamma=1.2808291901922786, learning_rate=0.06480135327008976, max_depth=4,
min_child_weight=63.22383782769299, n_estimators=91, reg_lambda=0.016279529095083668, subsample=0.6932577829777565
[CV] colsample_bytree=0.8554388572158456, gamma=1.2808291901922786, learning_rate=0.06480135327008976, max_depth=4,
min_child_weight=63.22383782769299, n_estimators=91, reg_lambda=0.016279529095083668, subsample=0.6932577829777565
```

```
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  7.8min
```

```
[CV]  colsample_bytree=0.8554388572158456, gamma=1.2808291901922786, learning_rate=0.06480135327008976, max_depth=4,
min_child_weight=63.22383782769299, n_estimators=91, reg_lambda=0.016279529095083668, subsample=0.6932577829777565, score=(train=0.860,
test=0.824), total= 3.3min
[CV] colsample_bytree=0.9644454614980406, gamma=6.075913532620509, learning_rate=0.002032830201545121, max_depth=7,
min_child_weight=55.72368578803438, n_estimators=68, reg_lambda=0.0358647229261306, subsample=0.9921905942236565
[CV]  colsample_bytree=0.8554388572158456, gamma=1.2808291901922786, learning_rate=0.06480135327008976, max_depth=4,
min_child_weight=63.22383782769299, n_estimators=91, reg_lambda=0.016279529095083668, subsample=0.6932577829777565, score=(train=0.860,
test=0.820), total= 3.3min
[CV] colsample_bytree=0.9644454614980406, gamma=6.075913532620509, learning_rate=0.002032830201545121, max_depth=7,
min_child_weight=55.72368578803438, n_estimators=68, reg_lambda=0.0358647229261306, subsample=0.9921905942236565
```

```
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 11.1min
```

```
[CV]  colsample_bytree=0.9644454614980406, gamma=6.075913532620509, learning_rate=0.002032830201545121, max_depth=7,
min_child_weight=55.72368578803438, n_estimators=68, reg_lambda=0.0358647229261306, subsample=0.9921905942236565, score=(train=0.732, test=0.698),
total= 5.2min
[CV] colsample_bytree=0.9552976445386667, gamma=8.070588074129297, learning_rate=0.03867883887926241, max_depth=4,
min_child_weight=38.298202983871086, n_estimators=120, reg_lambda=0.03309635037246653, subsample=0.9746209401382341
[CV]  colsample_bytree=0.9644454614980406, gamma=6.075913532620509, learning_rate=0.002032830201545121, max_depth=7,
min_child_weight=55.72368578803438, n_estimators=68, reg_lambda=0.0358647229261306, subsample=0.9921905942236565, score=(train=0.732, test=0.697),
total= 5.3min
[CV] colsample_bytree=0.9552976445386667, gamma=8.070588074129297, learning_rate=0.03867883887926241, max_depth=4,
min_child_weight=38.298202983871086, n_estimators=120, reg_lambda=0.03309635037246653, subsample=0.9746209401382341
[CV]  colsample_bytree=0.9552976445386667, gamma=8.070588074129297, learning_rate=0.03867883887926241, max_depth=4,
min_child_weight=38.298202983871086, n_estimators=120, reg_lambda=0.03309635037246653, subsample=0.9746209401382341, score=(train=0.852,
test=0.805), total= 5.3min
[CV] colsample_bytree=0.9760882562642434, gamma=7.33899070008214, learning_rate=0.07511994983852938, max_depth=6,
min_child_weight=52.05549446335297, n_estimators=16, reg_lambda=0.016381614223396755, subsample=0.9846398442555937
[CV]  colsample_bytree=0.9552976445386667, gamma=8.070588074129297, learning_rate=0.03867883887926241, max_depth=4,
min_child_weight=38.298202983871086, n_estimators=120, reg_lambda=0.03309635037246653, subsample=0.9746209401382341, score=(train=0.852,
test=0.803), total= 5.3min
[CV] colsample_bytree=0.9760882562642434, gamma=7.33899070008214, learning_rate=0.07511994983852938, max_depth=6,
min_child_weight=52.05549446335297, n_estimators=16, reg_lambda=0.016381614223396755, subsample=0.9846398442555937
[CV]  colsample_bytree=0.9760882562642434, gamma=7.33899070008214, learning_rate=0.07511994983852938, max_depth=6,
min_child_weight=52.05549446335297, n_estimators=16, reg_lambda=0.016381614223396755, subsample=0.9846398442555937, score=(train=0.779,
test=0.734), total= 1.1min
[CV] colsample_bytree=0.9392374918228288, gamma=1.2976168531850774, learning_rate=0.0033690080424075832, max_depth=9,
min_child_weight=121.04345829562762, n_estimators=189, reg_lambda=0.0017891873407416261, subsample=0.8649567989813004
```

```
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 22.8min
```

```
[CV]  colsample_bytree=0.9760882562642434, gamma=7.33899070008214, learning_rate=0.07511994983852938, max_depth=6,
min_child_weight=52.05549446335297, n_estimators=16, reg_lambda=0.016381614223396755, subsample=0.9846398442555937, score=(train=0.777,
test=0.731), total= 1.1min
[CV] colsample_bytree=0.9392374918228288, gamma=1.2976168531850774, learning_rate=0.0033690080424075832, max_depth=9,
min_child_weight=121.04345829562762, n_estimators=189, reg_lambda=0.0017891873407416261, subsample=0.8649567989813004
[CV]  colsample_bytree=0.9392374918228288, gamma=1.2976168531850774, learning_rate=0.0033690080424075832, max_depth=9,
min_child_weight=121.04345829562762, n_estimators=189, reg_lambda=0.0017891873407416261, subsample=0.8649567989813004, score=(train=0.795,
test=0.748), total=16.8min
[CV] colsample_bytree=0.9905388970901187, gamma=0.9553491010166437, learning_rate=0.08297699202864298, max_depth=7,
min_child_weight=33.349588022354965, n_estimators=39, reg_lambda=0.09579148177344904, subsample=0.9969684752561796
[CV]  colsample_bytree=0.9392374918228288, gamma=1.2976168531850774, learning_rate=0.0033690080424075832, max_depth=9,
min_child_weight=121.04345829562762, n_estimators=189, reg_lambda=0.0017891873407416261, subsample=0.8649567989813004, score=(train=0.794,
test=0.746), total=16.8min
[CV] colsample_bytree=0.9905388970901187, gamma=0.9553491010166437, learning_rate=0.08297699202864298, max_depth=7,
min_child_weight=33.349588022354965, n_estimators=39, reg_lambda=0.09579148177344904, subsample=0.9969684752561796
[CV]  colsample_bytree=0.9905388970901187, gamma=0.9553491010166437, learning_rate=0.08297699202864298, max_depth=7,
min_child_weight=33.349588022354965, n_estimators=39, reg_lambda=0.09579148177344904, subsample=0.9969684752561796, score=(train=0.880,
test=0.804), total= 3.2min
[CV] colsample_bytree=0.6822568644110761, gamma=0.21387817694568323, learning_rate=0.06766228327528674, max_depth=5,
min_child_weight=17.747049596571745, n_estimators=42, reg_lambda=0.055513040244731464, subsample=0.9330200170360701
[CV]  colsample_bytree=0.9905388970901187, gamma=0.9553491010166437, learning_rate=0.08297699202864298, max_depth=7,
min_child_weight=33.349588022354965, n_estimators=39, reg_lambda=0.09579148177344904, subsample=0.9969684752561796, score=(train=0.878,
test=0.800), total= 3.2min
[CV] colsample_bytree=0.6822568644110761, gamma=0.21387817694568323, learning_rate=0.06766228327528674, max_depth=5,
min_child_weight=17.747049596571745, n_estimators=42, reg_lambda=0.055513040244731464, subsample=0.9330200170360701
```

```
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 42.9min
```

```
[CV]  colsample_bytree=0.6822568644110761, gamma=0.21387817694568323, learning_rate=0.06766228327528674, max_depth=5,
min_child_weight=17.747049596571745, n_estimators=42, reg_lambda=0.055513040244731464, subsample=0.9330200170360701, score=(train=0.833,
test=0.779), total= 1.7min
[CV] colsample_bytree=0.9086723055091218, gamma=0.21216894437701028, learning_rate=0.0722791991688925, max_depth=9,
min_child_weight=37.48636405680028, n_estimators=181, reg_lambda=0.014456759006000242, subsample=0.7045874614416053
[CV]  colsample_bytree=0.6822568644110761, gamma=0.21387817694568323, learning_rate=0.06766228327528674, max_depth=5,
min_child_weight=17.747049596571745, n_estimators=42, reg_lambda=0.055513040244731464, subsample=0.9330200170360701, score=(train=0.836,
test=0.785), total= 1.7min
[CV] colsample_bytree=0.9086723055091218, gamma=0.21216894437701028, learning_rate=0.0722791991688925, max_depth=9,
min_child_weight=37.48636405680028, n_estimators=181, reg_lambda=0.014456759006000242, subsample=0.7045874614416053
[CV]  colsample_bytree=0.9086723055091218, gamma=0.21216894437701028, learning_rate=0.0722791991688925, max_depth=9,
min_child_weight=37.48636405680028, n_estimators=181, reg_lambda=0.014456759006000242, subsample=0.7045874614416053, score=(train=0.971,
test=0.877), total=14.6min
[CV] colsample_bytree=0.8760430563901844, gamma=8.437628384358698, learning_rate=0.02274271827471186, max_depth=6,
min_child_weight=41.31544825645659, n_estimators=157, reg_lambda=0.031149218614658903, subsample=0.9020335523195462
[CV]  colsample_bytree=0.9086723055091218, gamma=0.21216894437701028, learning_rate=0.0722791991688925, max_depth=9,
min_child_weight=37.48636405680028, n_estimators=181, reg_lambda=0.014456759006000242, subsample=0.7045874614416053, score=(train=0.972,
test=0.876), total=14.7min
[CV] colsample_bytree=0.8760430563901844, gamma=8.437628384358698, learning_rate=0.02274271827471186, max_depth=6,
min_child_weight=41.31544825645659, n_estimators=157, reg_lambda=0.031149218614658903, subsample=0.9020335523195462
[CV]  colsample_bytree=0.8760430563901844, gamma=8.437628384358698, learning_rate=0.02274271827471186, max_depth=6,
min_child_weight=41.31544825645659, n_estimators=157, reg_lambda=0.031149218614658903, subsample=0.9020335523195462, score=(train=0.874,
test=0.812), total= 9.1min
[CV]  colsample_bytree=0.8760430563901844, gamma=8.437628384358698, learning_rate=0.02274271827471186, max_depth=6,
```

```
min_child_weight=41.31544825645659, n_estimators=157, reg_lambda=0.031149218614658903, subsample=0.9020335523195462, score=(train=0.872,
test=0.809), total= 9.1min
```

```
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 68.4min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 68.4min finished
```

In [0]:

```
search.best_params_
```

Out[0]:

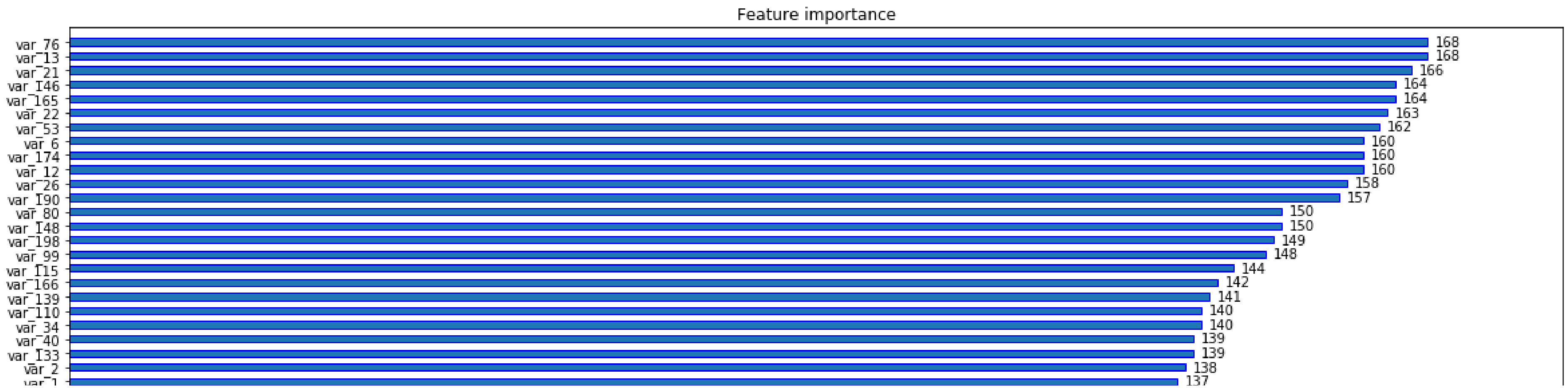
```
{'colsample_bytree': 0.9086723055091218,
 'gamma': 0.21216894437701028,
 'learning_rate': 0.07222791991688925,
 'max_depth': 9,
 'min_child_weight': 37.48636405680028,
 'n_estimators': 181,
 'reg_lambda': 0.014456759006000242,
 'subsample': 0.7045874614416053}
```

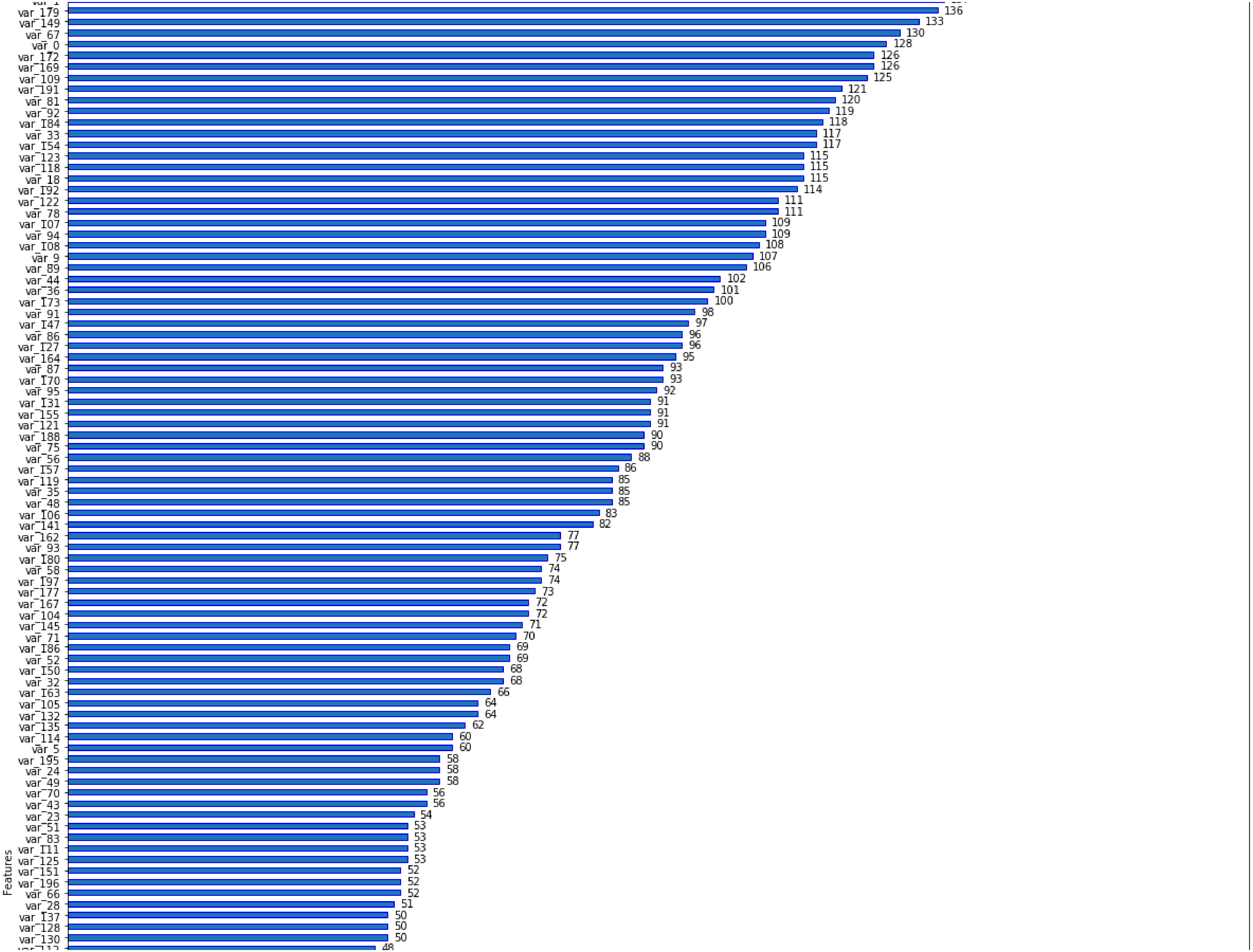
In [0]:

```
xgb.plot_importance?
```

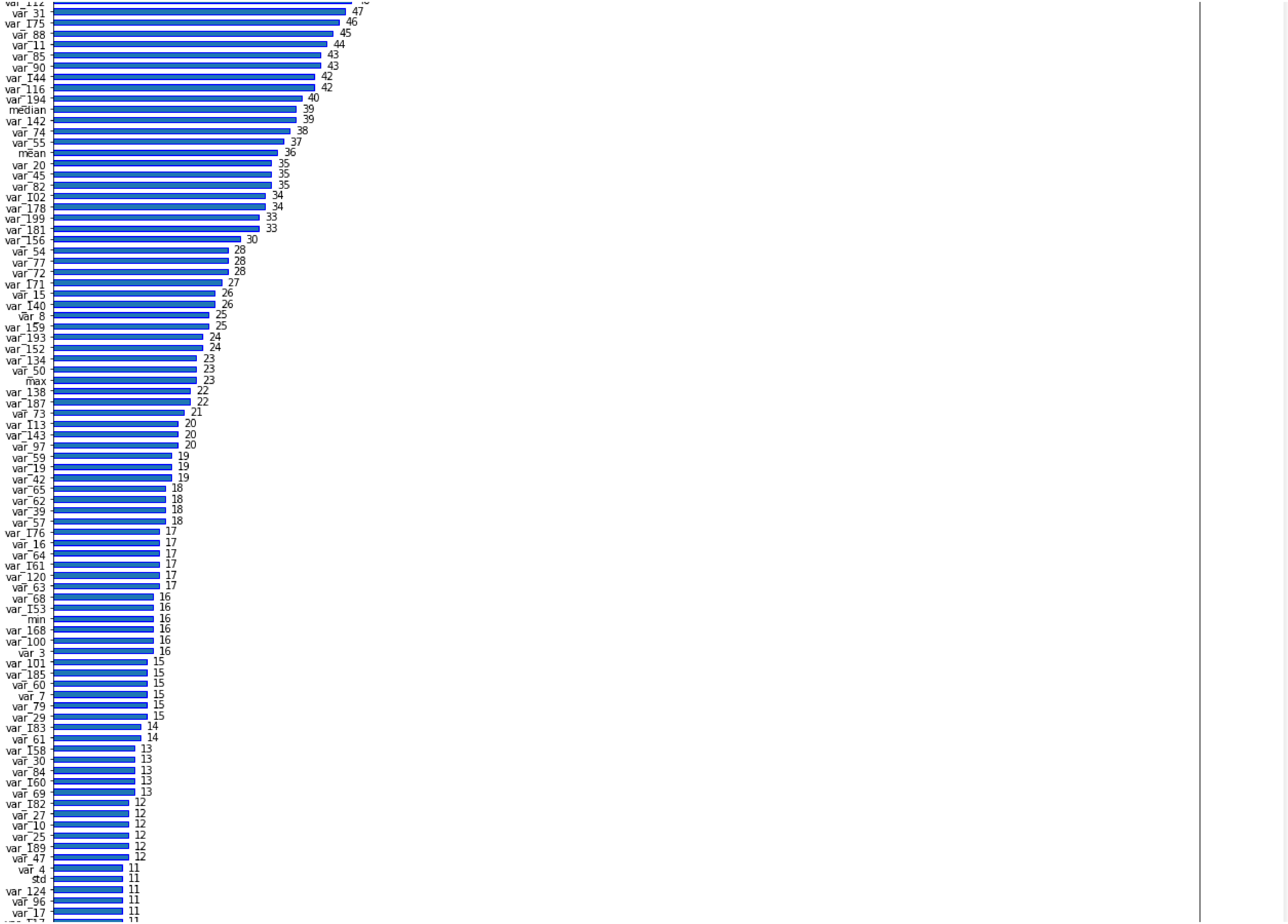
In [0]:

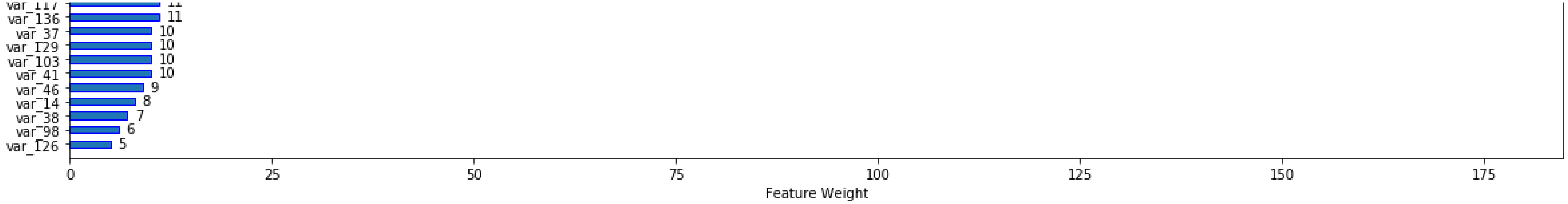
```
plt.figure(figsize=(20, 40))
xgb.plot_importance(search.best_estimator_, plt.gca(), grid=False, height=.4, edgecolor='blue', xlabel='Feature Weight')
plt.show()
```





Features
var_179 136
var_149 133
var_67 130
var_0 128
var_172 126
var_169 126
var_109 125
var_191 121
var_81 120
var_92 119
var_184 118
var_33 117
var_154 117
var_123 115
var_118 115
var_18 115
var_192 114
var_122 111
var_78 111
var_107 109
var_94 109
var_108 108
var_9 107
var_89 106
var_44 102
var_36 101
var_173 100
var_91 98
var_147 97
var_86 96
var_127 96
var_164 95
var_87 93
var_170 93
var_95 92
var_131 91
var_155 91
var_121 91
var_188 90
var_75 90
var_56 88
var_157 86
var_119 85
var_35 85
var_48 85
var_106 83
var_141 82
var_162 77
var_93 77
var_180 75
var_58 74
var_197 74
var_177 73
var_167 72
var_104 72
var_145 71
var_71 70
var_186 69
var_52 69
var_150 68
var_32 68
var_163 66
var_105 64
var_132 64
var_135 62
var_114 60
var_5 60
var_195 58
var_24 58
var_49 58
var_70 56
var_43 56
var_23 54
var_51 53
var_83 53
var_111 53
var_125 53
var_151 52
var_196 52
var_66 52
var_28 51
var_137 50
var_128 50
var_130 50
48


var_31 47
var_175 46
var_88 45
var_11 44
var_85 43
var_90 43
var_144 42
var_116 42
var_194 40
median 39
var_142 39
var_74 38
var_55 37
mean 36
var_20 35
var_45 35
var_82 35
var_102 34
var_178 34
var_199 33
var_181 33
var_156 30
var_54 28
var_77 28
var_72 28
var_171 27
var_15 26
var_140 26
var_8 25
var_159 25
var_193 24
var_152 24
var_134 23
var_50 23
max 23
var_138 22
var_187 22
var_73 21
var_113 20
var_143 20
var_97 20
var_59 19
var_19 19
var_42 19
var_65 18
var_62 18
var_39 18
var_57 18
var_176 17
var_16 17
var_64 17
var_161 17
var_120 17
var_63 17
var_68 16
var_153 16
min 16
var_168 16
var_100 16
var_3 16
var_101 15
var_185 15
var_60 15
var_7 15
var_79 15
var_29 15
var_183 14
var_61 14
var_158 13
var_30 13
var_84 13
var_160 13
var_69 13
var_182 12
var_27 12
var_10 12
var_25 12
var_189 12
var_47 12
var_4 11
std 11
var_124 11
var_96 11
var_17 11

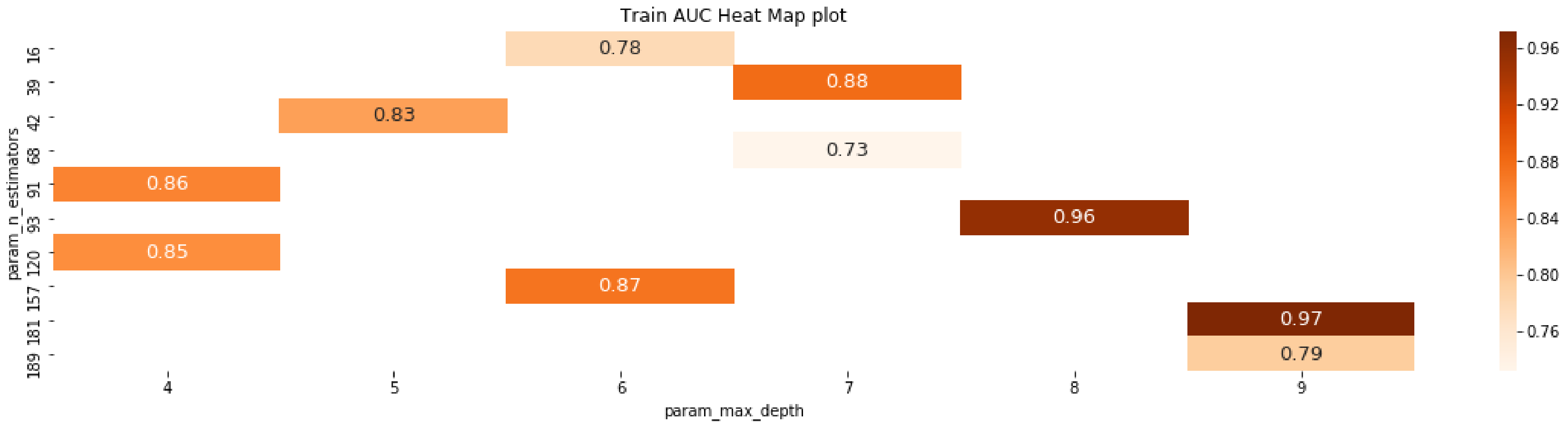


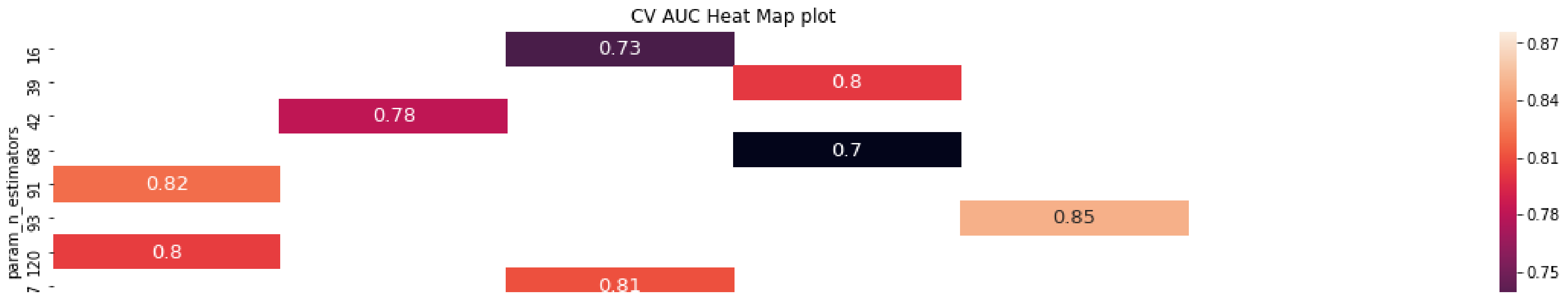
In [0]:

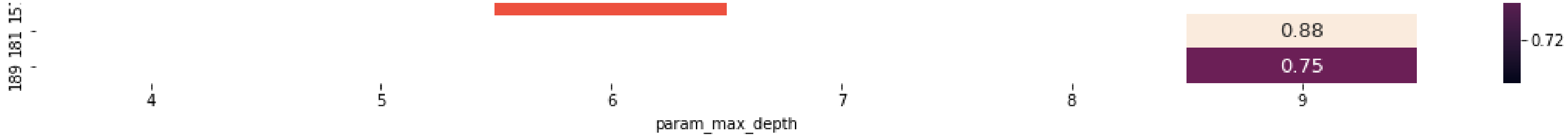
```
res = pd.DataFrame(search.cv_results_)
res
```

In [0]:

```
plotTrainVsCV_AUC(search, subplots=(2, 1), figsize=(20, 10), idx='param_n_estimators', cols='param_max_depth')
```

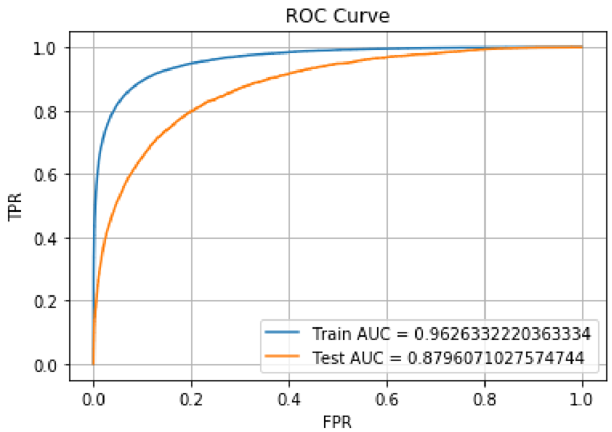






In [0]:

```
y_train_pred = search.best_estimator_.predict_proba(X_train)[:,1]
y_test_pred = search.best_estimator_.predict_proba(X_test)[:,1]

train_fpr, train_tpr, tr_thresholds = metrics.roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = metrics.roc_curve(y_test, y_test_pred)

roc_plot(train_fpr, train_tpr, test_fpr, test_tpr)
```



In [0]:

```
pt_1.add_row(['', '', '', ''])
pt_1.add_row(['XGBoost', 'Max_depth = 9, n_estimators = 181',\
              np.round(.9626332220363334, 3), np.round(.8796071027574744, 3)])
```

Light GBM

In [0]:

```
one_to_left = st.beta(10, 1)
from_zero_positive = st.expon(0, 50)

params = {
    "num_leaves": st.randint(100, 200)
```

```
    "num_leaves": st.randint(100, 200),
    "min_data_in_leaf": st.randint(50, 100),
    "max_depth": st.randint(5, 10),
    "learning_rate": st.uniform(0.001, 0.1),
    "colsample_bytree": one_to_left,
    # "subsample": one_to_left,
    "reg_lambda": st.uniform(0.0001, 0.1),
    "reg_alpha": st.uniform(0.0001, 0.01),
    "max_bin": st.randint(67, 100),
}

search = RandomizedSearchCV(lgb.LGBMClassifier(objective='binary', n_jobs=-1), params, n_iter=20,\
                            scoring='roc_auc', n_jobs=-1, verbose=10, return_train_score=True, cv=2, error_score='raise')
```

In [0]:

```
# with parallel_backend('multiprocessing'):
search.fit(X_train, y_train, eval_metric='auc')
```

```
Fitting 2 folds for each of 20 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   28.9s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:   56.3s
/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 worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:  6.1min
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  8.6min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:  9.2min finished
```

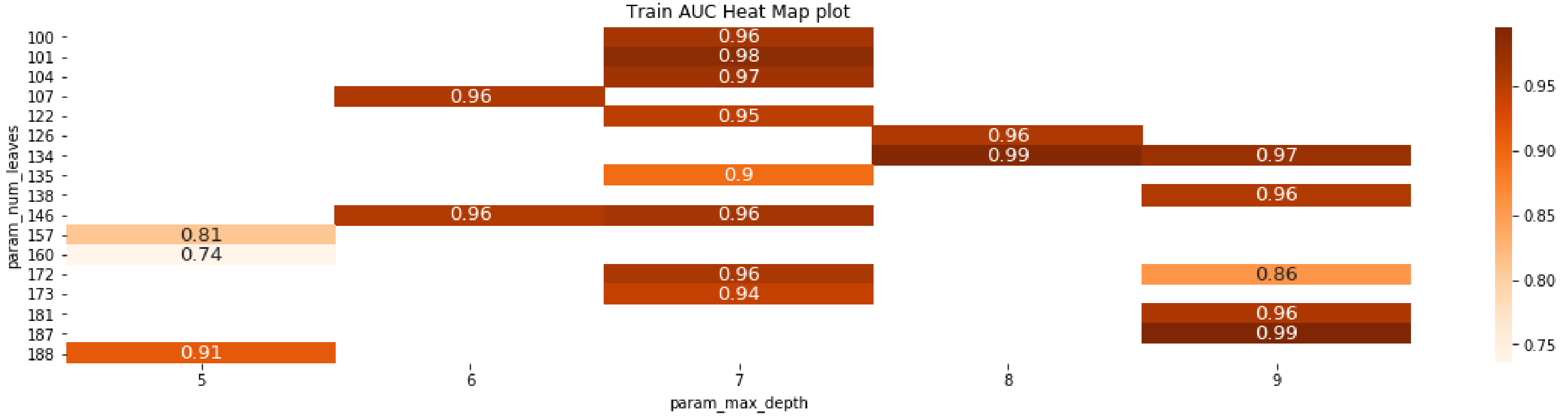
Out[0]:

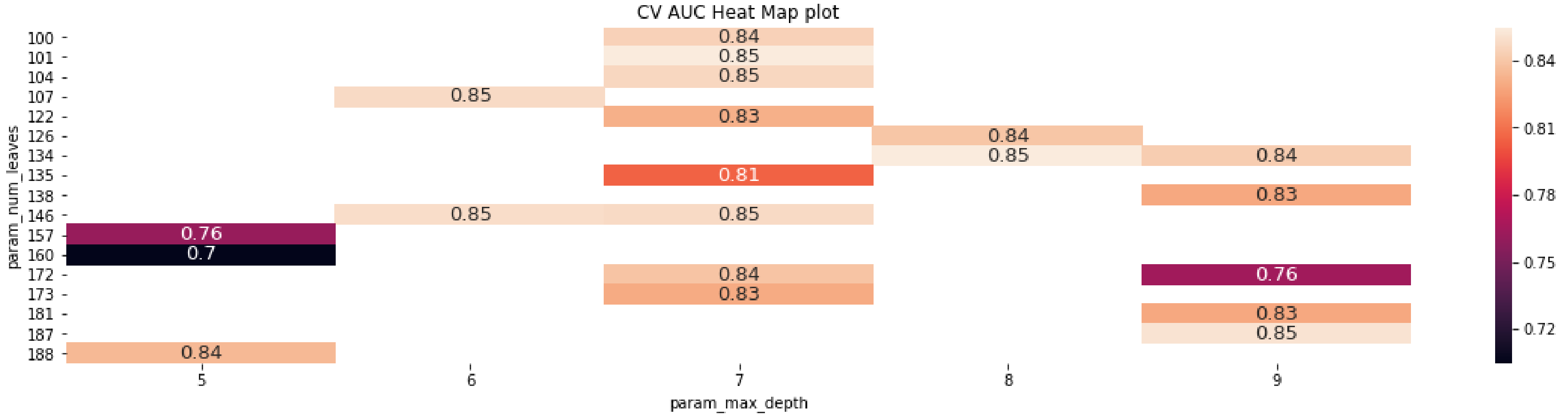
```
RandomizedSearchCV(cv=2, error_score='raise',
                   estimator=LGBMClassifier(boosting_type='gbdt',
                                            class_weight=None,
                                            colsample_bytree=1.0,
                                            importance_type='split',
                                            learning_rate=0.1, max_depth=-1,
                                            min_child_samples=20,
                                            min_child_weight=0.001,
                                            min_split_gain=0.0,
                                            n_estimators=100, n_jobs=-1,
                                            num_leaves=31, objective='binary',
                                            random_state=None, reg_alpha=0.0,
                                            reg_lambda=...
                                        'min_data_in_leaf': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f3329cf7b00>,
                                        'num_leaves': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f332b899ac8>,
```

```
                                   'reg_alpha': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f3329d008d0>,
                                   'reg_lambda': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f3329d00668>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='roc_auc', verbose=10)
```

In [0]:

```
plotTrainVsCV_AUC(search, (2,1), (20, 10), idx='param_num_leaves', cols='param_max_depth')
```

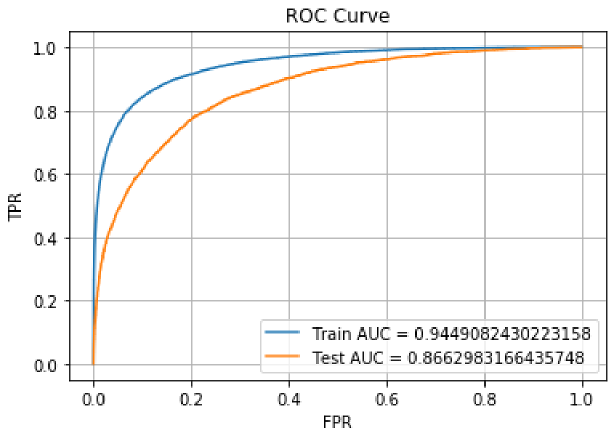




In [0]:

```
y_train_pred = search.best_estimator_.predict_proba(X_train)[:,1]
```

```
y_test_pred = search.best_estimator_.predict_proba(X_test)[:,1]

train_fpr, train_tpr, tr_thresholds = metrics.roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = metrics.roc_curve(y_test, y_test_pred)

roc_plot(train_fpr, train_tpr, test_fpr, test_tpr)
```



In [0]:

```
pt_1.add_row(['', '', '', ''])
pt_1.add_row(['Light GBM', 'Max_depth = 6, num_leaves = 181',\
              np.round(.9449082430223158, 3), np.round(.8662983166435748, 3)])
```

New Features

Type - 1 | Count Features :

In [0]:

```
total_vars = X_train.columns.difference(['mean', 'std', 'max', 'min', 'median']).size

X_train_cnt = np.zeros((len(X_train), total_vars * 4))
X_test_cnt = np.zeros((len(X_test), total_vars * 4))

for j in tqdm(range(total_vars)):
    for i in range(1, 4):
        x = np.round(X_train.iloc[:, j], i+1)
        dic = pd.value_counts(x)
        missedValue = dic.idxmax() #np.median(dic.index)
        dic = dic.to_dict()
        X_train_cnt[:, j*4 + i] = pd.Series(x).map(dic)
```

```
        X_train_cnt[:, j*4 + i] = pd.Series(x).map(dic)
        X_test_cnt[:, j*4 + i] = pd.Series(np.round(X_test.iloc[:, j], i+1)).map(lambda p: dic.get(p, missedValue))

    x = X_train.iloc[:, j]
    dic = pd.value_counts(x)
    missedValue = dic.idxmax() #np.median(dic.index)
    dic = dic.to_dict()

    X_train_cnt[:, j*4] = pd.Series(x).map(dic)
    X_test_cnt[:, j*4] = pd.Series(X_test.iloc[:, j]).map(lambda p: dic.get(p, missedValue))

X_train_basic_feats = X_train[['mean', 'std', 'max', 'min', 'median']].copy()
X_test_basic_feats = X_test[['mean', 'std', 'max', 'min', 'median']].copy()

X_train_raw = X_train[X_train.columns.difference(['mean', 'std', 'max', 'min', 'median'])].copy()
X_test_raw = X_test[X_test.columns.difference(['mean', 'std', 'max', 'min', 'median'])].copy()

X_train_new_feats = np.zeros((len(X_train_raw), total_vars * 5))
X_test_new_feats = np.zeros((len(X_test_raw), total_vars * 5))

# raw + count of rounded feature
old_cols = X_train_raw.columns
new_cols = list()

for idx in tqdm(range(total_vars)):
    X_train_new_feats[:, 5*idx] = X_train_raw.iloc[:, idx]
    new_cols.extend([old_cols[idx], '{}_count'.format(old_cols[idx])])

    X_train_new_feats[:, 5*idx+1:5*idx+5] = X_train_cnt[:, 4*idx:4*idx+4]
    new_cols.extend(['{}_roundedTo_{}'.format(old_cols[idx], rnd) for rnd in range(2, 5)])

    X_test_new_feats[:, 5*idx] = X_test_raw.iloc[:, idx]
    X_test_new_feats[:, 5*idx+1:5*idx+5] = X_test_cnt[:, 4*idx:4*idx+4]

X_train_new_feats = pd.DataFrame(X_train_new_feats, columns=new_cols)
X_test_new_feats = pd.DataFrame(X_test_new_feats, columns=new_cols)

del X_train_cnt, X_test_cnt, X_train_raw, X_test_raw; gc.collect()
```

```
100%|██████████| 200/200 [01:53<00:00,  2.19it/s]
100%|██████████| 200/200 [00:05<00:00, 36.48it/s]
```

Out[0]:

```
0
```

In [0]:

```
basic_feats_also = str(input('Do you want to use basic features also (y/n) : ')) == 'y'

X_tr = pd.concat([X_train_new_feats.reset_index(drop=True), X_train_basic_feats.reset_index(drop=True)],\
                 axis=1, sort=False) if basic_feats_also else X_train_new_feats
```

```
                 axis=1, sort=False) if basic_feats_also else X_train_new_feats
X_te = pd.concat([X_test_new_feats.reset_index(drop=True), X_test_basic_feats.reset_index(drop=True)],\
                 axis=1, sort=False) if basic_feats_also else X_test_new_feats
```

```
Do you want to use basic features also (y/n) : n
```

In [0]:

```
X_tr.shape, X_te.shape
```

Out[0]:

```
((160000, 1000), (40000, 1000))
```

In [0]:

```
y_train.shape, y_test.shape
```

Out[0]:

```
((160000,), (40000,))
```

Type - 2 | Rounded features :

In [13]:

```
X_tr_2 = X_train[X_train.columns.difference(['mean', 'std', 'max', 'min', 'median'])].copy()
X_te_2 = X_test[X_test.columns.difference(['mean', 'std', 'max', 'min', 'median'])].copy()

for feature in tqdm(X_tr_2.columns):
    X_tr_2[feature+'_r2'] = np.round(X_tr_2[feature], 2)
    X_tr_2[feature+'_r1'] = np.round(X_tr_2[feature], 1)

    X_te_2[feature+'_r2'] = np.round(X_te_2[feature], 2)
    X_te_2[feature+'_r1'] = np.round(X_te_2[feature], 1)
```

```
100%|██████████| 200/200 [00:02<00:00, 69.28it/s]
```

In [14]:

```
basic_feats_also = str(input('Do you want to use basic features also (y/n) : ')) == 'y'

X_tr_2 = pd.concat([X_tr_2.reset_index(drop=True),\
                    X_train[['mean', 'std', 'max', 'min', 'median']].copy().reset_index(drop=True)],\
                 axis=1, sort=False) if basic_feats_also else X_train_new_feats
X_te_2 = pd.concat([X_te_2.reset_index(drop=True),\
```

```
            X_test[['mean', 'std', 'max', 'min', 'median']].copy().reset_index(drop=True)],\
          axis=1, sort=False) if basic_feats_also else X_test_new_feats
```

```
Do you want to use basic features also (y/n) : y
```

Modelling [2] - w/ new[, basic] features.

In [0]:

```
pt_2 = PrettyTable()
pt_2.field_names = ['Model', 'Hyper Parameters', 'Train AUC', 'Test AUC']
```

Logistic Regression :

In [0]:

```
clf = linear_model.LogisticRegression(penalty='elasticnet', n_jobs=-1, class_weight='balanced', solver='saga', fit_intercept=False)

params = dict(C=(.0001, .001, .01, .1), max_iter=st.randint(800, 1000), l1_ratio=st.uniform(.15, .5))
search = RandomizedSearchCV(clf, params, scoring='roc_auc', n_jobs=-1, verbose=10, return_train_score=True, cv=2, n_iter=20)
```

In [0]:

```
with parallel_backend('multiprocessing'):
  search.fit(scaler.fit_transform(X_tr), y_train)
```

```
Fitting 2 folds for each of 20 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=-1)]: Using backend MultiprocessingBackend with 4 concurrent workers.
```

```
[CV] C=0.0001, l1_ratio=0.40540059812874696, max_iter=849 ...........
[CV] C=0.0001, l1_ratio=0.40540059812874696, max_iter=849 ...........
[CV] C=0.01, l1_ratio=0.28284730554247794, max_iter=902 .............
[CV] C=0.01, l1_ratio=0.28284730554247794, max_iter=902 .............
[CV]  C=0.0001, l1_ratio=0.40540059812874696, max_iter=849, score=(train=0.649, test=0.641), total=  45.6s
[CV] C=0.1, l1_ratio=0.6347826726244412, max_iter=841 ...............
[CV]  C=0.0001, l1_ratio=0.40540059812874696, max_iter=849, score=(train=0.639, test=0.639), total=  47.8s
[CV] C=0.1, l1_ratio=0.6347826726244412, max_iter=841 ...............
[CV]  C=0.01, l1_ratio=0.28284730554247794, max_iter=902, score=(train=0.893, test=0.877), total= 1.3min
[CV] C=0.001, l1_ratio=0.5839910295748246, max_iter=923 .............
[CV]  C=0.01, l1_ratio=0.28284730554247794, max_iter=902, score=(train=0.891, test=0.879), total= 1.9min
[CV] C=0.001, l1_ratio=0.5839910295748246, max_iter=923 .............
[CV]  C=0.1, l1_ratio=0.6347826726244412, max_iter=841, score=(train=0.892, test=0.878), total= 1.2min
[CV] C=0.0001, l1_ratio=0.5667287045097704, max_iter=900 ............
```

```
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  2.0min
```

```
[CV]  C=0.1, l1_ratio=0.6347826726244412, max_iter=841, score=(train=0.893, test=0.876), total= 1.3min
[CV] C=0.0001, l1_ratio=0.5667287045097704, max_iter=900 .............
[CV]  C=0.001, l1_ratio=0.5839910295748246, max_iter=923, score=(train=0.873, test=0.869), total=  56.2s
[CV] C=0.01, l1_ratio=0.45957488288800297, max_iter=943 ..............
[CV]  C=0.0001, l1_ratio=0.5667287045097704, max_iter=900, score=(train=0.500, test=0.500), total=  24.7s
[CV] C=0.01, l1_ratio=0.45957488288800297, max_iter=943 ..............
[CV]  C=0.0001, l1_ratio=0.5667287045097704, max_iter=900, score=(train=0.500, test=0.500), total=  26.5s
[CV] C=0.1, l1_ratio=0.5933255298866175, max_iter=863 ................
[CV]  C=0.001, l1_ratio=0.5839910295748246, max_iter=923, score=(train=0.875, test=0.867), total=  52.3s
[CV] C=0.1, l1_ratio=0.5933255298866175, max_iter=863 ................
```

```
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  2.8min
```

```
[CV]  C=0.01, l1_ratio=0.45957488288800297, max_iter=943, score=(train=0.890, test=0.880), total= 1.2min
[CV] C=0.001, l1_ratio=0.2894029134510545, max_iter=802 ..............
[CV]  C=0.01, l1_ratio=0.45957488288800297, max_iter=943, score=(train=0.892, test=0.877), total= 1.2min
[CV] C=0.001, l1_ratio=0.2894029134510545, max_iter=802 ..............
[CV]  C=0.1, l1_ratio=0.5933255298866175, max_iter=863, score=(train=0.892, test=0.878), total= 1.2min
[CV] C=0.1, l1_ratio=0.22196647609466505, max_iter=946 ...............
[CV]  C=0.1, l1_ratio=0.5933255298866175, max_iter=863, score=(train=0.893, test=0.876), total= 1.3min
[CV] C=0.1, l1_ratio=0.22196647609466505, max_iter=946 ...............
[CV]  C=0.001, l1_ratio=0.2894029134510545, max_iter=802, score=(train=0.883, test=0.877), total=  55.2s
[CV] C=0.0001, l1_ratio=0.4202726590010638, max_iter=871 .............
[CV]  C=0.001, l1_ratio=0.2894029134510545, max_iter=802, score=(train=0.885, test=0.875), total=  55.9s
[CV] C=0.0001, l1_ratio=0.4202726590010638, max_iter=871 .............
[CV]  C=0.0001, l1_ratio=0.4202726590010638, max_iter=871, score=(train=0.624, test=0.625), total=  43.8s
[CV] C=0.001, l1_ratio=0.1545037567823819, max_iter=898 ..............
```

```
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  5.0min
```

```
[CV]  C=0.1, l1_ratio=0.22196647609466505, max_iter=946, score=(train=0.892, test=0.878), total= 1.3min
[CV] C=0.001, l1_ratio=0.1545037567823819, max_iter=898 ..............
[CV]  C=0.0001, l1_ratio=0.4202726590010638, max_iter=871, score=(train=0.632, test=0.624), total=  44.0s
[CV] C=0.01, l1_ratio=0.5816246884542369, max_iter=883 ...............
[CV]  C=0.001, l1_ratio=0.1545037567823819, max_iter=898, score=(train=0.887, test=0.879), total=  59.0s
[CV] C=0.01, l1_ratio=0.5816246884542369, max_iter=883 ...............
[CV]  C=0.001, l1_ratio=0.1545037567823819, max_iter=898, score=(train=0.889, test=0.877), total= 1.1min
[CV] C=0.001, l1_ratio=0.3932018575524343, max_iter=921 ..............
[CV]  C=0.01, l1_ratio=0.5816246884542369, max_iter=883, score=(train=0.890, test=0.880), total= 1.1min
[CV] C=0.001, l1_ratio=0.3932018575524343, max_iter=921 ..............
[CV]  C=0.1, l1_ratio=0.22196647609466505, max_iter=946, score=(train=0.893, test=0.876), total= 2.2min
[CV] C=0.1, l1_ratio=0.45391957123442916, max_iter=983 ...............
[CV]  C=0.001, l1_ratio=0.3932018575524343, max_iter=921, score=(train=0.880, test=0.875), total=  53.0s
[CV] C=0.1, l1_ratio=0.45391957123442916, max_iter=983 ...............
```

```
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed:  7.1min
```

```
[CV]  C=0.001, l1_ratio=0.3932018575524343, max_iter=921, score=(train=0.882, test=0.873), total=  58.0s
[CV] C=0.001, l1_ratio=0.35302373545693966, max_iter=882 ..............
[CV]  C=0.01, l1_ratio=0.5816246884542369, max_iter=883, score=(train=0.892, test=0.877), total= 1.3min
[CV] C=0.001, l1_ratio=0.35302373545693966, max_iter=882 ..............
[CV]  C=0.1, l1_ratio=0.45391957123442916, max_iter=983, score=(train=0.892, test=0.878), total= 1.2min
[CV] C=0.0001, l1_ratio=0.5563723898738865, max_iter=974 ..............
[CV]  C=0.0001, l1_ratio=0.5563723898738865, max_iter=974, score=(train=0.500, test=0.500), total=  25.7s
[CV] C=0.0001, l1_ratio=0.5563723898738865, max_iter=974 ..............
[CV]  C=0.001, l1_ratio=0.35302373545693966, max_iter=882, score=(train=0.881, test=0.876), total=  55.3s
[CV] C=0.0001, l1_ratio=0.5447952733137348, max_iter=862 ..............
[CV]  C=0.001, l1_ratio=0.35302373545693966, max_iter=882, score=(train=0.883, test=0.874), total=  57.3s
[CV] C=0.0001, l1_ratio=0.5447952733137348, max_iter=862 ..............
[CV]  C=0.1, l1_ratio=0.45391957123442916, max_iter=983, score=(train=0.893, test=0.876), total= 1.3min
[CV] C=0.001, l1_ratio=0.4740555376479476, max_iter=834 ..............
[CV]  C=0.0001, l1_ratio=0.5563723898738865, max_iter=974, score=(train=0.500, test=0.500), total=  23.9s
[CV] C=0.001, l1_ratio=0.4740555376479476, max_iter=834 ..............
[CV]  C=0.0001, l1_ratio=0.5447952733137348, max_iter=862, score=(train=0.500, test=0.500), total=  28.5s
[CV] C=0.01, l1_ratio=0.4964518773723896, max_iter=951 ...............
```

```
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:  8.8min
```

```
[CV]  C=0.0001, l1_ratio=0.5447952733137348, max_iter=862, score=(train=0.570, test=0.564), total=  49.5s
[CV] C=0.01, l1_ratio=0.4964518773723896, max_iter=951 ...............
[CV]  C=0.001, l1_ratio=0.4740555376479476, max_iter=834, score=(train=0.877, test=0.872), total=  54.3s
[CV] C=0.1, l1_ratio=0.5723260976994545, max_iter=901 ................
[CV]  C=0.001, l1_ratio=0.4740555376479476, max_iter=834, score=(train=0.879, test=0.870), total=  55.7s
[CV] C=0.1, l1_ratio=0.5723260976994545, max_iter=901 ................
[CV]  C=0.01, l1_ratio=0.4964518773723896, max_iter=951, score=(train=0.890, test=0.880), total= 1.1min
[CV]  C=0.01, l1_ratio=0.4964518773723896, max_iter=951, score=(train=0.892, test=0.877), total= 1.1min
```

```
[Parallel(n_jobs=-1)]: Done  38 out of  40 | elapsed: 10.3min remaining:   32.5s
```

```
[CV]  C=0.1, l1_ratio=0.5723260976994545, max_iter=901, score=(train=0.892, test=0.878), total= 1.1min
[CV]  C=0.1, l1_ratio=0.5723260976994545, max_iter=901, score=(train=0.893, test=0.876), total= 1.1min
```

```
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 10.5min finished
```

In [0]:

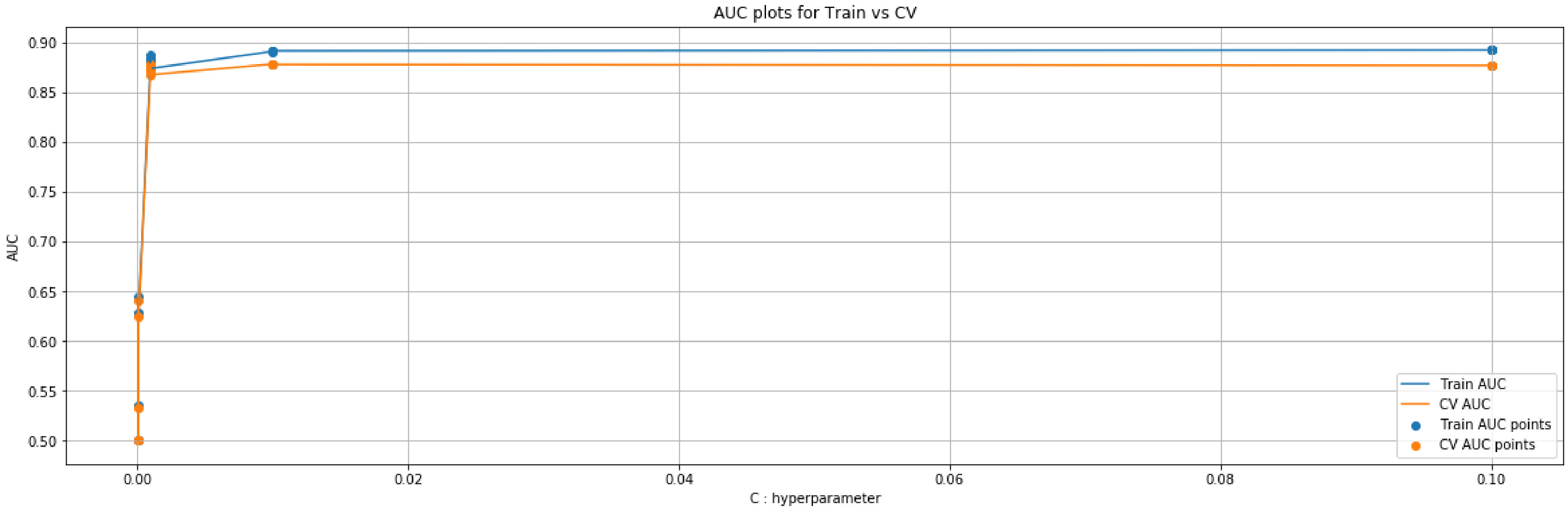
```
search.best_params_
```

Out[0]:

```
{'C': 0.01, 'l1_ratio': 0.5816246884542369, 'max_iter': 883}
```

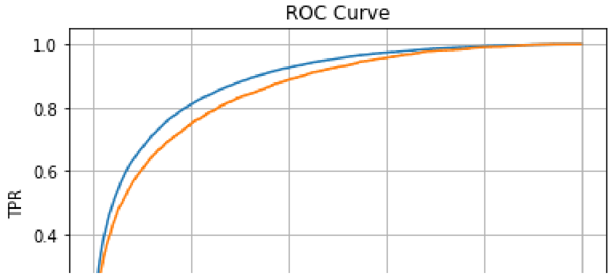
In [0]:

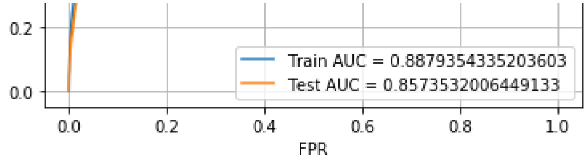
```
res = pd.DataFrame(search.cv_results_)
idxs = np.argsort(res.param_C.values.astype('float64'))

auc_plot(res.param_C.values.astype('float64')[idxs], res.mean_train_score.values[idxs], res.mean_test_score.values[idxs], 'C')
```



In [0]:

```
y_train_pred = search.best_estimator_.predict_proba(scaler.transform(X_tr))[:,1]
y_test_pred = search.best_estimator_.predict_proba(scaler.transform(X_te))[:,1]

train_fpr, train_tpr, tr_thresholds = metrics.roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = metrics.roc_curve(y_test, y_test_pred)

roc_plot(train_fpr, train_tpr, test_fpr, test_tpr)
```





In [0]:

```
pt_2.add_row(['Logistic Regression', 'C = 0.01, l1_ratio = 0.5816246884542369, max_iter = 883',\
              np.round(.887993, 3), np.round(.857353, 3)])
```

XGBoost :

In [0]:

```
# https://github.com/dmlc/xgboost/blob/master/demo/kaggle-higgs/higgs-numpy.py

weight = (X_train.var_76 * float(X_test.shape[0]) / len(y_train))

sum_wpos = sum( weight.iloc[i] for i in range(len(y_train)) if y_train[i] )
sum_wneg = sum( weight.iloc[i] for i in range(len(y_train)) if not y_train[i] )
```

In [0]:

```
xgb.XGBClassifier?
```

In [0]:

```
# https://xgboost.readthedocs.io/en/latest/tutorials/param_tuning.html

one_to_left = st.beta(5, 1)
from_zero_positive = st.expon(0, 50)

params = {
    "n_estimators": st.randint(100, 123),
    "max_depth": st.randint(2, 5),
    "colsample_bytree": one_to_left,
    "subsample": one_to_left,
    "gamma": st.uniform(0, 10),
    "reg_lambda": (.001, .01, .1),
    "reg_alpha": (.001, .01, .1),
    "min_child_weight": from_zero_positive,
    "learning_rate": (.01, .1)
}

search = RandomizedSearchCV(xgb.XGBClassifier(max_delta_step=10, eval_metric='auc', objective='binary:logistic',\
                                              n_jobs=-1, scale_pos_weight=sum_wneg/sum_wpos, num_boost_round=100000),\
```

```
                                    n_jobs=-1, scale_pos_weight=sum_wneg/sum_wpos, num_boost_round=100000),(
                            params, scoring='roc_auc', n_jobs=-1, verbose=23, return_train_score=True, cv=2)
```

In [19]:

```
with parallel_backend('threading'):
  search.fit(X_tr_2, y_train)
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
[CV] colsample_bytree=0.9795385580301809, gamma=7.48368178462424, learning_rate=0.1, max_depth=3, min_child_weight=26.306065256124405, n_estimators=
116, reg_alpha=0.01, reg_lambda=0.01, subsample=0.8693109798363479 [CV] colsample_bytree=0.9795385580301809, gamma=7.48368178462424,
learning_rate=0.1, max_depth=3, min_child_weight=26.306065256124405, n_estimators=116, reg_alpha=0.01, reg_lambda=0.01,
subsample=0.8693109798363479 [CV] colsample_bytree=0.981991935471296, gamma=3.133728648190235, learning_rate=0.01, max_depth=2,
min_child_weight=6.56542037919276, n_estimators=122, reg_alpha=0.01, reg_lambda=0.01, subsample=0.9185191139938026 [CV]
colsample_bytree=0.981991935471296, gamma=3.133728648190235, learning_rate=0.01, max_depth=2, min_child_weight=6.56542037919276, n_estimators=122,
reg_alpha=0.01, reg_lambda=0.01, subsample=0.9185191139938026
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent workers.
```

```
[CV]  colsample_bytree=0.981991935471296, gamma=3.133728648190235, learning_rate=0.01, max_depth=2, min_child_weight=6.56542037919276,
n_estimators=122, reg_alpha=0.01, reg_lambda=0.01, subsample=0.9185191139938026, score=(train=0.702, test=0.690), total= 6.4min
[CV] colsample_bytree=0.7306922599120634, gamma=8.954793152617174, learning_rate=0.1, max_depth=3, min_child_weight=86.38868548503523,
n_estimators=122, reg_alpha=0.1, reg_lambda=0.001, subsample=0.5839620442647151
```

```
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  6.5min
```

```
[CV]  colsample_bytree=0.981991935471296, gamma=3.133728648190235, learning_rate=0.01, max_depth=2, min_child_weight=6.56542037919276,
n_estimators=122, reg_alpha=0.01, reg_lambda=0.01, subsample=0.9185191139938026, score=(train=0.712, test=0.696), total= 6.4min
[CV] colsample_bytree=0.7306922599120634, gamma=8.954793152617174, learning_rate=0.1, max_depth=3, min_child_weight=86.38868548503523,
n_estimators=122, reg_alpha=0.1, reg_lambda=0.001, subsample=0.5839620442647151
```

```
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  6.5min
```

```
[CV]  colsample_bytree=0.9795385580301809, gamma=7.48368178462424, learning_rate=0.1, max_depth=3, min_child_weight=26.306065256124405, n_estimators
=116, reg_alpha=0.01, reg_lambda=0.01, subsample=0.8693109798363479, score=(train=0.885, test=0.840), total= 8.8min
[CV] colsample_bytree=0.97312978438908, gamma=5.242335416132442, learning_rate=0.1, max_depth=2, min_child_weight=104.78643562746936,
n_estimators=116, reg_alpha=0.01, reg_lambda=0.001, subsample=0.9904962826855939
```

```
[Parallel(n_jobs=-1)]: Done   3 tasks      | elapsed:  8.8min
```

```
[CV]  colsample_bytree=0.9795385580301809, gamma=7.48368178462424, learning_rate=0.1, max_depth=3, min_child_weight=26.306065256124405, n_estimators
=116, reg_alpha=0.01, reg_lambda=0.01, subsample=0.8693109798363479, score=(train=0.885, test=0.838), total= 8.8min
[CV] colsample_bytree=0.97312978438908, gamma=5.242335416132442, learning_rate=0.1, max_depth=2, min_child_weight=104.78643562746936,
n_estimators=116, reg_alpha=0.01, reg_lambda=0.001, subsample=0.9904962826855939
```

```
n_estimators=116, reg_alpha=0.01, reg_lambda=0.001, subsample=0.9904962826855939
```

```
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:  8.9min
```

```
[CV]  colsample_bytree=0.7306922599120634, gamma=8.954793152617174, learning_rate=0.1, max_depth=3, min_child_weight=86.38868548503523,
n_estimators=122, reg_alpha=0.1, reg_lambda=0.001, subsample=0.5839620442647151, score=(train=0.885, test=0.845), total= 6.6min
[CV] colsample_bytree=0.964207487966292, gamma=0.7612741260974909, learning_rate=0.1, max_depth=3, min_child_weight=81.85692285482902,
n_estimators=103, reg_alpha=0.001, reg_lambda=0.001, subsample=0.7881643463534637
```

```
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed: 13.1min
```

```
[CV]  colsample_bytree=0.7306922599120634, gamma=8.954793152617174, learning_rate=0.1, max_depth=3, min_child_weight=86.38868548503523,
n_estimators=122, reg_alpha=0.1, reg_lambda=0.001, subsample=0.5839620442647151, score=(train=0.885, test=0.843), total= 6.6min
[CV] colsample_bytree=0.964207487966292, gamma=0.7612741260974909, learning_rate=0.1, max_depth=3, min_child_weight=81.85692285482902,
n_estimators=103, reg_alpha=0.001, reg_lambda=0.001, subsample=0.7881643463534637
```

```
[Parallel(n_jobs=-1)]: Done   6 tasks      | elapsed: 13.2min
```

```
[CV]  colsample_bytree=0.97312978438908, gamma=5.242335416132442, learning_rate=0.1, max_depth=2, min_child_weight=104.78643562746936,
n_estimators=116, reg_alpha=0.01, reg_lambda=0.001, subsample=0.9904962826855939, score=(train=0.851, test=0.825), total= 6.1min
[CV] colsample_bytree=0.7768314957295858, gamma=2.246418634135564, learning_rate=0.1, max_depth=4, min_child_weight=19.924485703424306,
n_estimators=105, reg_alpha=0.1, reg_lambda=0.01, subsample=0.7651510127958602
```

```
[Parallel(n_jobs=-1)]: Done   7 tasks      | elapsed: 14.9min
```

```
[CV]  colsample_bytree=0.97312978438908, gamma=5.242335416132442, learning_rate=0.1, max_depth=2, min_child_weight=104.78643562746936,
n_estimators=116, reg_alpha=0.01, reg_lambda=0.001, subsample=0.9904962826855939, score=(train=0.850, test=0.820), total= 6.0min
[CV] colsample_bytree=0.7768314957295858, gamma=2.246418634135564, learning_rate=0.1, max_depth=4, min_child_weight=19.924485703424306,
n_estimators=105, reg_alpha=0.1, reg_lambda=0.01, subsample=0.7651510127958602
```

```
[Parallel(n_jobs=-1)]: Done   8 tasks      | elapsed: 15.0min
```

```
[CV]  colsample_bytree=0.964207487966292, gamma=0.7612741260974909, learning_rate=0.1, max_depth=3, min_child_weight=81.85692285482902,
n_estimators=103, reg_alpha=0.001, reg_lambda=0.001, subsample=0.7881643463534637, score=(train=0.878, test=0.837), total= 7.5min
[CV] colsample_bytree=0.8619262389413318, gamma=4.074731259525173, learning_rate=0.01, max_depth=4, min_child_weight=50.30903142680757,
n_estimators=117, reg_alpha=0.01, reg_lambda=0.1, subsample=0.9138171108843222
```

```
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 20.7min
```

```
[CV]  colsample_bytree=0.964207487966292, gamma=0.7612741260974909, learning_rate=0.1, max_depth=3, min_child_weight=81.85692285482902,
n_estimators=103, reg_alpha=0.001, reg_lambda=0.001, subsample=0.7881643463534637, score=(train=0.878, test=0.835), total= 7.5min
[CV] colsample_bytree=0.8619262389413318, gamma=4.074731259525173, learning_rate=0.01, max_depth=4, min_child_weight=50.30903142680757,
n_estimators=117, reg_alpha=0.01, reg_lambda=0.1, subsample=0.9138171108843222
```

```
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed: 20.7min
```

```
[CV]  colsample_bytree=0.7768314957295858, gamma=2.246418634135564, learning_rate=0.1, max_depth=4, min_child_weight=19.924485703424306,
n_estimators=105, reg_alpha=0.1, reg_lambda=0.01, subsample=0.7651510127958602, score=(train=0.912, test=0.845), total= 8.4min
[CV] colsample_bytree=0.6594019736141953, gamma=2.6035533500979504, learning_rate=0.01, max_depth=4, min_child_weight=61.193191400991445,
n_estimators=116, reg_alpha=0.001, reg_lambda=0.1, subsample=0.6278599077588631
```

```
[Parallel(n_jobs=-1)]: Done  11 tasks      | elapsed: 23.4min
```

```
[CV]  colsample_bytree=0.7768314957295858, gamma=2.246418634135564, learning_rate=0.1, max_depth=4, min_child_weight=19.924485703424306,
n_estimators=105, reg_alpha=0.1, reg_lambda=0.01, subsample=0.7651510127958602, score=(train=0.911, test=0.843), total= 8.4min
[CV] colsample_bytree=0.6594019736141953, gamma=2.6035533500979504, learning_rate=0.01, max_depth=4, min_child_weight=61.193191400991445,
n_estimators=116, reg_alpha=0.001, reg_lambda=0.1, subsample=0.6278599077588631
```

```
[Parallel(n_jobs=-1)]: Done  12 tasks      | elapsed: 23.4min
```

```
[CV]  colsample_bytree=0.6594019736141953, gamma=2.6035533500979504, learning_rate=0.01, max_depth=4, min_child_weight=61.193191400991445,
n_estimators=116, reg_alpha=0.001, reg_lambda=0.1, subsample=0.6278599077588631, score=(train=0.773, test=0.743), total= 7.6min
[CV] colsample_bytree=0.9074312279518009, gamma=8.441913379007856, learning_rate=0.01, max_depth=2, min_child_weight=22.495303701236598,
n_estimators=118, reg_alpha=0.01, reg_lambda=0.01, subsample=0.955217118357863
```

```
[Parallel(n_jobs=-1)]: Done  13 tasks      | elapsed: 31.1min
```

```
[CV]  colsample_bytree=0.6594019736141953, gamma=2.6035533500979504, learning_rate=0.01, max_depth=4, min_child_weight=61.193191400991445,
n_estimators=116, reg_alpha=0.001, reg_lambda=0.1, subsample=0.6278599077588631, score=(train=0.778, test=0.744), total= 7.7min
[CV] colsample_bytree=0.9074312279518009, gamma=8.441913379007856, learning_rate=0.01, max_depth=2, min_child_weight=22.495303701236598,
n_estimators=118, reg_alpha=0.01, reg_lambda=0.01, subsample=0.955217118357863
```

```
[Parallel(n_jobs=-1)]: Done  14 out of  20 | elapsed: 31.2min remaining: 13.4min
```

```
[CV]  colsample_bytree=0.8619262389413318, gamma=4.074731259525173, learning_rate=0.01, max_depth=4, min_child_weight=50.30903142680757,
n_estimators=117, reg_alpha=0.01, reg_lambda=0.1, subsample=0.9138171108843222, score=(train=0.769, test=0.737), total=10.5min
[CV] colsample_bytree=0.8705828787827952, gamma=6.169139480861725, learning_rate=0.01, max_depth=4, min_child_weight=74.85042440808952,
n_estimators=112, reg_alpha=0.1, reg_lambda=0.01, subsample=0.48538140978171307
```

```
[Parallel(n_jobs=-1)]: Done  15 out of  20 | elapsed: 31.3min remaining: 10.4min
```

```
[CV]  colsample_bytree=0.8619262389413318, gamma=4.074731259525173, learning_rate=0.01, max_depth=4, min_child_weight=50.30903142680757,
n_estimators=117, reg_alpha=0.01, reg_lambda=0.1, subsample=0.9138171108843222, score=(train=0.774, test=0.737), total=10.5min
[CV] colsample_bytree=0.8705828787827952, gamma=6.169139480861725, learning_rate=0.01, max_depth=4, min_child_weight=74.85042440808952,
n_estimators=112, reg_alpha=0.1, reg_lambda=0.01, subsample=0.48538140978171307
```

```
[Parallel(n_jobs=-1)]: Done  16 out of  20 | elapsed: 31.3min remaining:  7.8min
```

```
[CV]  colsample_bytree=0.9074312279518009, gamma=8.441913379007856, learning_rate=0.01, max_depth=2, min_child_weight=22.495303701236598,
```

```
n_estimators=118, reg_alpha=0.01, reg_lambda=0.01, subsample=0.955217118357863, score=(train=0.700, test=0.689), total= 5.8min
```

```
[Parallel(n_jobs=-1)]: Done  17 out of  20 | elapsed: 36.9min remaining:  6.5min
```

```
[CV]  colsample_bytree=0.9074312279518009, gamma=8.441913379007856, learning_rate=0.01, max_depth=2, min_child_weight=22.495303701236598,
n_estimators=118, reg_alpha=0.01, reg_lambda=0.01, subsample=0.955217118357863, score=(train=0.708, test=0.692), total= 5.8min
```

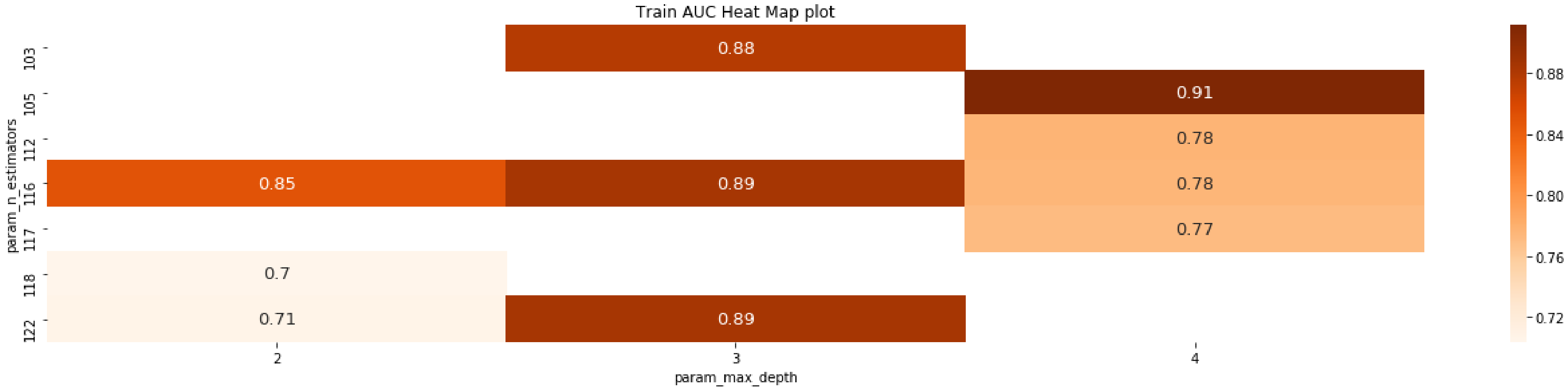
```
[Parallel(n_jobs=-1)]: Done  18 out of  20 | elapsed: 37.0min remaining:  4.1min
```

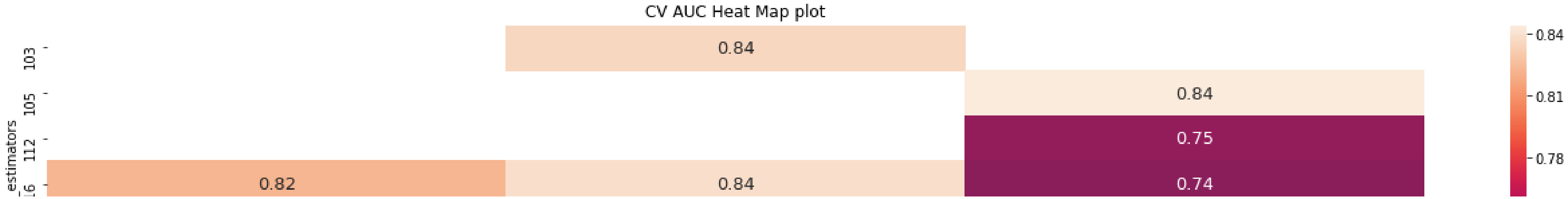
```
[CV]  colsample_bytree=0.8705828787827952, gamma=6.169139480861725, learning_rate=0.01, max_depth=4, min_child_weight=74.85042440808952,
n_estimators=112, reg_alpha=0.1, reg_lambda=0.01, subsample=0.48538140978171307, score=(train=0.776, test=0.748), total= 7.3min
[CV]  colsample_bytree=0.8705828787827952, gamma=6.169139480861725, learning_rate=0.01, max_depth=4, min_child_weight=74.85042440808952,
n_estimators=112, reg_alpha=0.1, reg_lambda=0.01, subsample=0.48538140978171307, score=(train=0.779, test=0.745), total= 7.3min
```

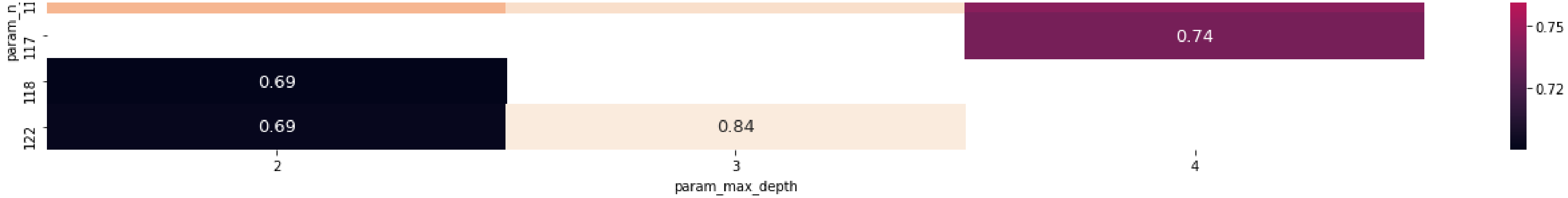
```
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 38.6min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 38.6min finished
```

In [20]:

```
plotTrainVsCV_AUC(search, subplots=(2, 1), figsize=(23, 10), idx='param_n_estimators', cols='param_max_depth')
```







In [23]:

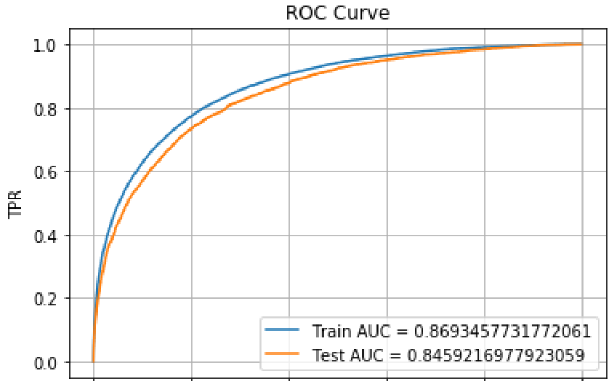
```
search.best_params
```

Out[23]:

```
{'colsample_bytree': 0.7306922599120634,
 'gamma': 8.954793152617174,
 'learning_rate': 0.1,
 'max_depth': 3,
 'min_child_weight': 86.38868548503523,
 'n_estimators': 122,
 'reg_alpha': 0.1,
 'reg_lambda': 0.001,
 'subsample': 0.5839620442647151}
```

In [21]:

```
y_train_pred = search.best_estimator_.predict_proba(X_tr_2)[:,1]
y_test_pred = search.best_estimator_.predict_proba(X_te_2)[:,1]

train_fpr, train_tpr, tr_thresholds = metrics.roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = metrics.roc_curve(y_test, y_test_pred)

roc_plot(train_fpr, train_tpr, test_fpr, test_tpr)
```



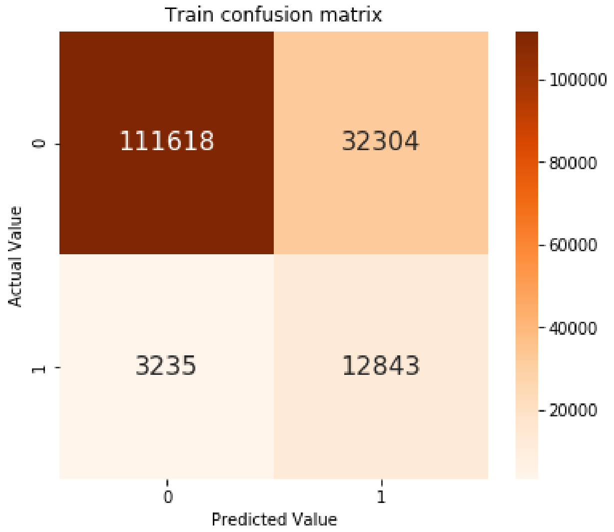
In [0]:

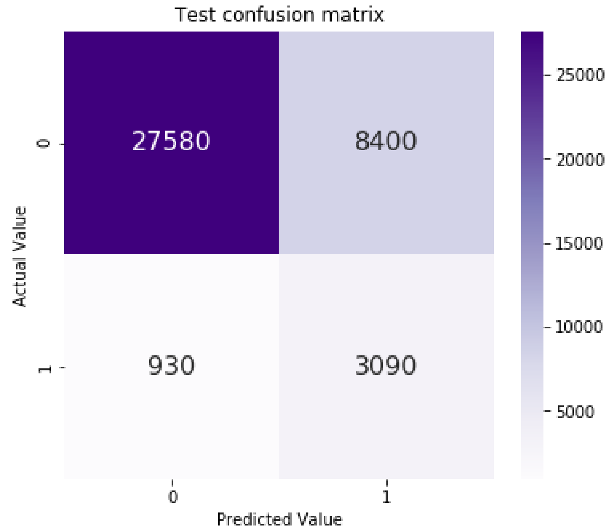
```
pt_2.add_row(['', '', '', ''])
pt_2.add_row(['XGBoost', 'Max_depth = 3, n_estimators = 122', np.round(.86934577, 3), np.round(.84592169, 3)])
```

In [22]:

```
printConfusionMatrix(y_train, y_test, y_train_pred, y_test_pred, tr_thresholds, te_thresholds, train_fpr, train_tpr, test_fpr, test_tpr)
```

```
The maximum value of Train tpr*(1-fpr) is 0.6195002830787604 for threshold 0.546999990940094

The maximum value of Test tpr*(1-fpr) is 0.5892037865148962 for threshold 0.5440000295639038
```





Light GBM

In [0]:

```
# Hyperparameter tuning in light gbm : https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html
params = {
    "num_leaves": st.randint(30, 80),
    "min_data_in_leaf": st.randint(80, 100),
    "max_depth": st.randint(2, 7),
```

```
    "learning_rate": (.01, .1),
    "reg_lambda": (.001, .01, .1),
    "reg_alpha": (.001, .01, .1),
    "max_bin": st.randint(67, 100),
    "min_gain_to_split": st.uniform(0, 10)
}

search = RandomizedSearchCV(lgb.LGBMClassifier(objective='binary', n_jobs=-1, min_sum_hessian_in_leaf=10., bagging_freq=5, bagging_fraction=.4, boos
t_from_average=False,\
                                               feature_fraction=.05, num_round=100000),\
                             params, n_iter=10, scoring='roc_auc', verbose=10, return_train_score=True, cv=2, error_score='raise')
```

In [26]:

```
# with parallel_backend('multiprocessing'):
search.fit(X_tr_2, y_train, eval_metric='auc')
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
[CV] learning_rate=0.1, max_bin=79, max_depth=5, min_data_in_leaf=80, min_gain_to_split=5.756727905110919, num_leaves=45, reg_alpha=0.1,
reg_lambda=0.01

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV]  learning_rate=0.1, max_bin=79, max_depth=5, min_data_in_leaf=80, min_gain_to_split=5.756727905110919, num_leaves=45, reg_alpha=0.1,
reg_lambda=0.01, score=(train=0.946, test=0.890), total= 3.6min
[CV] learning_rate=0.1, max_bin=79, max_depth=5, min_data_in_leaf=80, min_gain_to_split=5.756727905110919, num_leaves=45, reg_alpha=0.1,
reg_lambda=0.01

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.6min remaining:    0.0s

[CV]  learning_rate=0.1, max_bin=79, max_depth=5, min_data_in_leaf=80, min_gain_to_split=5.756727905110919, num_leaves=45, reg_alpha=0.1,
reg_lambda=0.01, score=(train=0.949, test=0.888), total= 3.6min
[CV] learning_rate=0.1, max_bin=79, max_depth=6, min_data_in_leaf=91, min_gain_to_split=2.15719108068663, num_leaves=43, reg_alpha=0.01,
reg_lambda=0.001

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  7.3min remaining:    0.0s

[CV]  learning_rate=0.1, max_bin=79, max_depth=6, min_data_in_leaf=91, min_gain_to_split=2.15719108068663, num_leaves=43, reg_alpha=0.01,
reg_lambda=0.001, score=(train=0.999, test=0.873), total= 4.3min
[CV] learning_rate=0.1, max_bin=79, max_depth=6, min_data_in_leaf=91, min_gain_to_split=2.15719108068663, num_leaves=43, reg_alpha=0.01,
reg_lambda=0.001

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 11.9min remaining:    0.0s

[CV]  learning_rate=0.1, max_bin=79, max_depth=6, min_data_in_leaf=91, min_gain_to_split=2.15719108068663, num_leaves=43, reg_alpha=0.01,
reg_lambda=0.001, score=(train=0.999, test=0.872), total= 4.3min
```

```
[CV] learning_rate=0.1, max_bin=81, max_depth=6, min_data_in_leaf=86, min_gain_to_split=6.340819585601512, num_leaves=62, reg_alpha=0.001,
reg_lambda=0.01
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 16.5min remaining:    0.0s
```

```
[CV]  learning_rate=0.1, max_bin=81, max_depth=6, min_data_in_leaf=86, min_gain_to_split=6.340819585601512, num_leaves=62, reg_alpha=0.001,
reg_lambda=0.01, score=(train=0.942, test=0.891), total= 3.5min
[CV] learning_rate=0.1, max_bin=81, max_depth=6, min_data_in_leaf=86, min_gain_to_split=6.340819585601512, num_leaves=62, reg_alpha=0.001,
reg_lambda=0.01
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed: 20.1min remaining:    0.0s
```

```
[CV]  learning_rate=0.1, max_bin=81, max_depth=6, min_data_in_leaf=86, min_gain_to_split=6.340819585601512, num_leaves=62, reg_alpha=0.001,
reg_lambda=0.01, score=(train=0.944, test=0.890), total= 3.5min
[CV] learning_rate=0.01, max_bin=96, max_depth=2, min_data_in_leaf=88, min_gain_to_split=4.431550819941017, num_leaves=73, reg_alpha=0.001,
reg_lambda=0.01
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed: 23.7min remaining:    0.0s
```

```
[CV]  learning_rate=0.01, max_bin=96, max_depth=2, min_data_in_leaf=88, min_gain_to_split=4.431550819941017, num_leaves=73, reg_alpha=0.001,
reg_lambda=0.01, score=(train=0.964, test=0.891), total=10.0min
[CV] learning_rate=0.01, max_bin=96, max_depth=2, min_data_in_leaf=88, min_gain_to_split=4.431550819941017, num_leaves=73, reg_alpha=0.001,
reg_lambda=0.01
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed: 35.7min remaining:    0.0s
```

```
[CV]  learning_rate=0.01, max_bin=96, max_depth=2, min_data_in_leaf=88, min_gain_to_split=4.431550819941017, num_leaves=73, reg_alpha=0.001,
reg_lambda=0.01, score=(train=0.965, test=0.888), total= 9.6min
[CV] learning_rate=0.1, max_bin=96, max_depth=6, min_data_in_leaf=89, min_gain_to_split=2.1284522761940607, num_leaves=31, reg_alpha=0.001,
reg_lambda=0.01
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed: 47.5min remaining:    0.0s
```

```
[CV]  learning_rate=0.1, max_bin=96, max_depth=6, min_data_in_leaf=89, min_gain_to_split=2.1284522761940607, num_leaves=31, reg_alpha=0.001,
reg_lambda=0.01, score=(train=0.999, test=0.874), total= 4.1min
[CV] learning_rate=0.1, max_bin=96, max_depth=6, min_data_in_leaf=89, min_gain_to_split=2.1284522761940607, num_leaves=31, reg_alpha=0.001,
reg_lambda=0.01
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed: 51.8min remaining:    0.0s
```

```
[CV]  learning_rate=0.1, max_bin=96, max_depth=6, min_data_in_leaf=89, min_gain_to_split=2.1284522761940607, num_leaves=31, reg_alpha=0.001,
reg_lambda=0.01, score=(train=0.999, test=0.872), total= 4.4min
[CV] learning_rate=0.1, max_bin=79, max_depth=5, min_data_in_leaf=84, min_gain_to_split=6.486166274120549, num_leaves=57, reg_alpha=0.1,
reg_lambda=0.01
[CV]  learning_rate=0.1, max_bin=79, max_depth=5, min_data_in_leaf=84, min_gain_to_split=6.486166274120549, num_leaves=57, reg_alpha=0.1,
```

```
reg_lambda=0.01, score=(train=0.939, test=0.891), total= 3.5min
[CV] learning_rate=0.1, max_bin=79, max_depth=5, min_data_in_leaf=84, min_gain_to_split=6.486166274120549, num_leaves=57, reg_alpha=0.1,
reg_lambda=0.01
[CV]  learning_rate=0.1, max_bin=79, max_depth=5, min_data_in_leaf=84, min_gain_to_split=6.486166274120549, num_leaves=57, reg_alpha=0.1,
reg_lambda=0.01, score=(train=0.940, test=0.890), total= 3.5min
[CV] learning_rate=0.1, max_bin=96, max_depth=2, min_data_in_leaf=97, min_gain_to_split=9.293773454804837, num_leaves=32, reg_alpha=0.001,
reg_lambda=0.01
[CV]  learning_rate=0.1, max_bin=96, max_depth=2, min_data_in_leaf=97, min_gain_to_split=9.293773454804837, num_leaves=32, reg_alpha=0.001,
reg_lambda=0.01, score=(train=0.919, test=0.894), total= 3.4min
[CV] learning_rate=0.1, max_bin=96, max_depth=2, min_data_in_leaf=97, min_gain_to_split=9.293773454804837, num_leaves=32, reg_alpha=0.001,
reg_lambda=0.01
[CV]  learning_rate=0.1, max_bin=96, max_depth=2, min_data_in_leaf=97, min_gain_to_split=9.293773454804837, num_leaves=32, reg_alpha=0.001,
reg_lambda=0.01, score=(train=0.921, test=0.893), total= 3.4min
[CV] learning_rate=0.01, max_bin=84, max_depth=6, min_data_in_leaf=90, min_gain_to_split=1.172521298504613, num_leaves=34, reg_alpha=0.1,
reg_lambda=0.01
[CV]  learning_rate=0.01, max_bin=84, max_depth=6, min_data_in_leaf=90, min_gain_to_split=1.172521298504613, num_leaves=34, reg_alpha=0.1,
reg_lambda=0.01, score=(train=1.000, test=0.890), total=13.3min
[CV] learning_rate=0.01, max_bin=84, max_depth=6, min_data_in_leaf=90, min_gain_to_split=1.172521298504613, num_leaves=34, reg_alpha=0.1,
reg_lambda=0.01
[CV]  learning_rate=0.01, max_bin=84, max_depth=6, min_data_in_leaf=90, min_gain_to_split=1.172521298504613, num_leaves=34, reg_alpha=0.1,
reg_lambda=0.01, score=(train=1.000, test=0.887), total=12.7min
[CV] learning_rate=0.1, max_bin=75, max_depth=2, min_data_in_leaf=87, min_gain_to_split=0.3018309256258689, num_leaves=70, reg_alpha=0.1,
reg_lambda=0.001
[CV]  learning_rate=0.1, max_bin=75, max_depth=2, min_data_in_leaf=87, min_gain_to_split=0.3018309256258689, num_leaves=70, reg_alpha=0.1,
reg_lambda=0.001, score=(train=1.000, test=0.871), total= 7.7min
[CV] learning_rate=0.1, max_bin=75, max_depth=2, min_data_in_leaf=87, min_gain_to_split=0.3018309256258689, num_leaves=70, reg_alpha=0.1,
reg_lambda=0.001
[CV]  learning_rate=0.1, max_bin=75, max_depth=2, min_data_in_leaf=87, min_gain_to_split=0.3018309256258689, num_leaves=70, reg_alpha=0.1,
reg_lambda=0.001, score=(train=1.000, test=0.870), total= 7.4min
[CV] learning_rate=0.01, max_bin=80, max_depth=3, min_data_in_leaf=93, min_gain_to_split=3.4441851833627535, num_leaves=41, reg_alpha=0.001,
reg_lambda=0.001
[CV]  learning_rate=0.01, max_bin=80, max_depth=3, min_data_in_leaf=93, min_gain_to_split=3.4441851833627535, num_leaves=41, reg_alpha=0.001,
reg_lambda=0.001, score=(train=0.984, test=0.889), total=10.4min
[CV] learning_rate=0.01, max_bin=80, max_depth=3, min_data_in_leaf=93, min_gain_to_split=3.4441851833627535, num_leaves=41, reg_alpha=0.001,
reg_lambda=0.001
[CV]  learning_rate=0.01, max_bin=80, max_depth=3, min_data_in_leaf=93, min_gain_to_split=3.4441851833627535, num_leaves=41, reg_alpha=0.001,
reg_lambda=0.001, score=(train=0.985, test=0.887), total=10.5min
```

```
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed: 148.5min finished
```

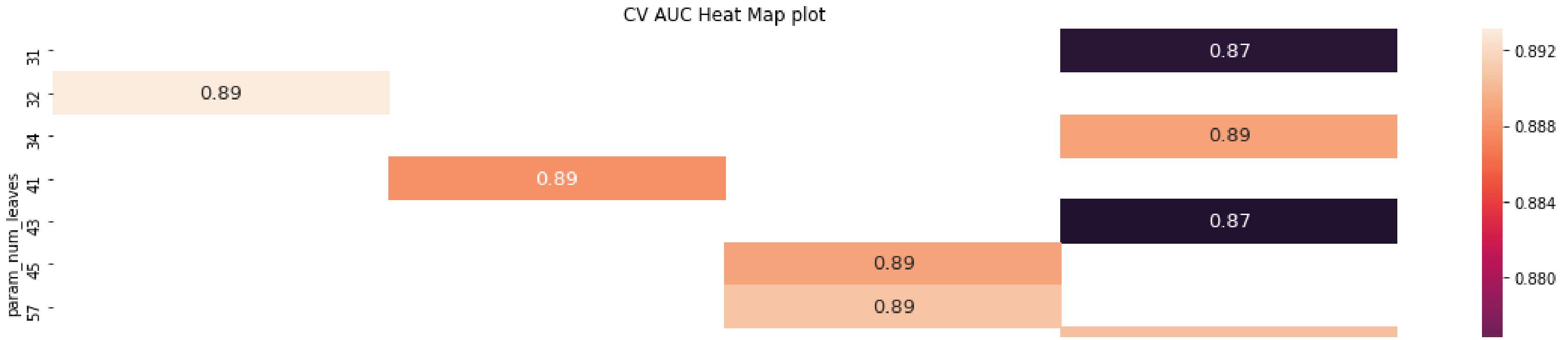
Out[26]:

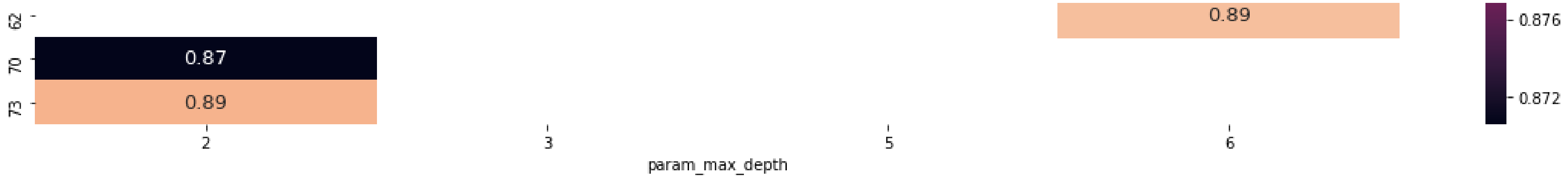
```
RandomizedSearchCV(cv=2, error_score='raise',
                   estimator=LGBMClassifier(bagging_fraction=0.4,
                                            bagging_freq=5,
                                            boost_from_average=False,
                                            boosting_type='gbdt',
                                            class_weight=None,
                                            colsample_bytree=1.0,
                                            feature_fraction=0.05,
                                            importance_type='split',
                                            learning_rate=0.1, max_depth=-1,
```

```
                                    min_child_samples=20,
                                    min_child_weight=0.001,
                                    min_split_gain=0.0,
                                    min_sum_hessian_in_l...
                                'min_data_in_leaf': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f49c029ddd8>,
                                'min_gain_to_split': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f49c03c92e8>,
                                'num_leaves': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f49c029da90>,
                                'reg_alpha': (0.001, 0.01, 0.1),
                                'reg_lambda': (0.001, 0.01, 0.1)},
             pre_dispatch='2*n_jobs', random_state=None, refit=True,
             return_train_score=True, scoring='roc_auc', verbose=10)
```

In [27]:

```
plotTrainVsCV_AUC(search, (2,1), (20, 12), idx='param_num_leaves', cols='param_max_depth')
```





In [30]:

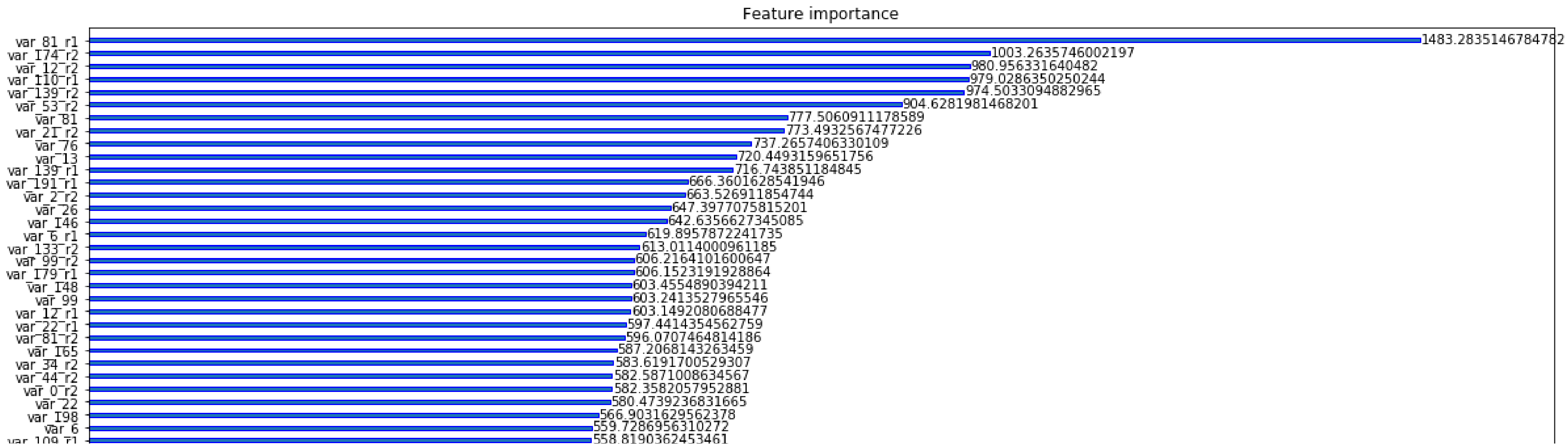
```
search.best_params_
```

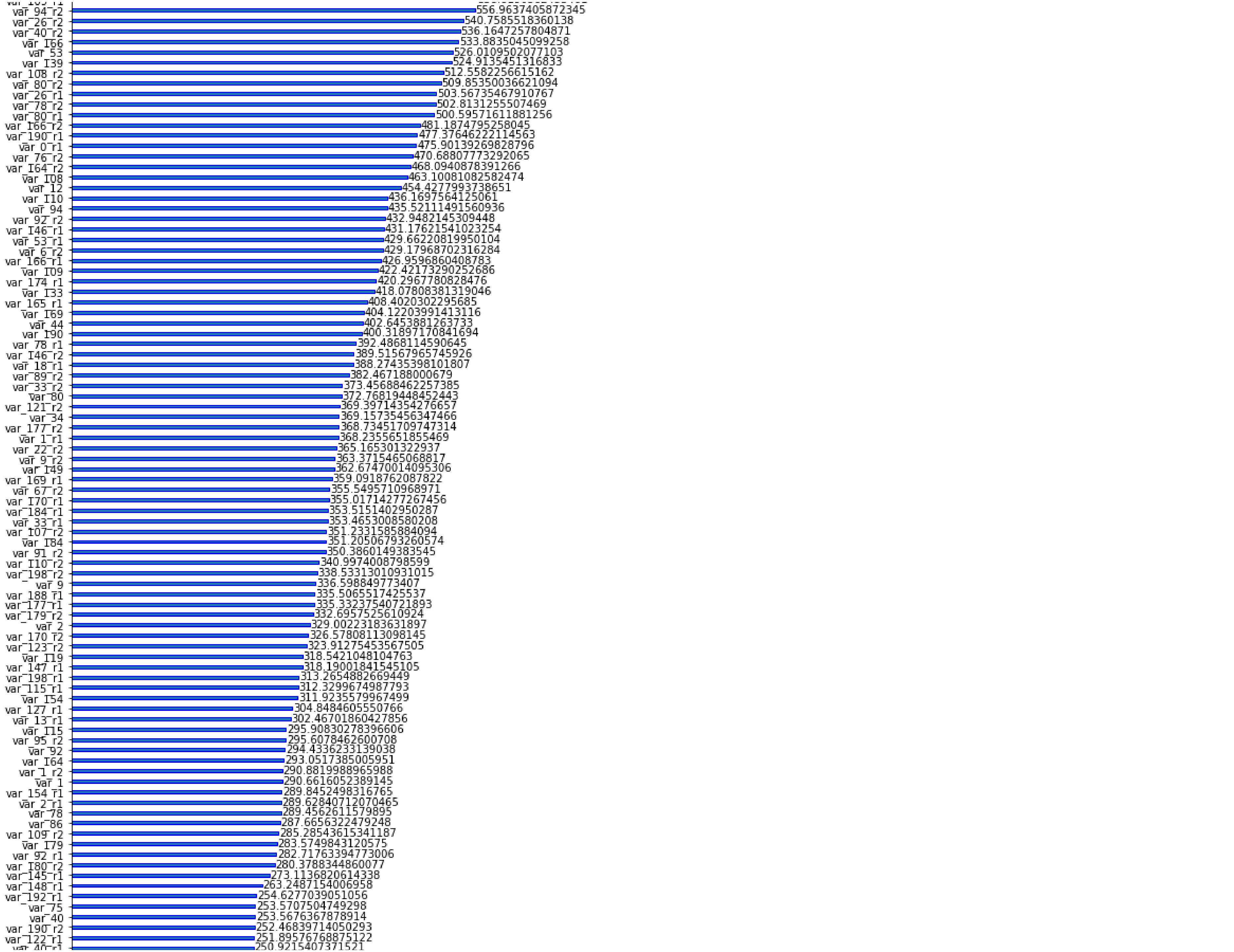
Out[30]:

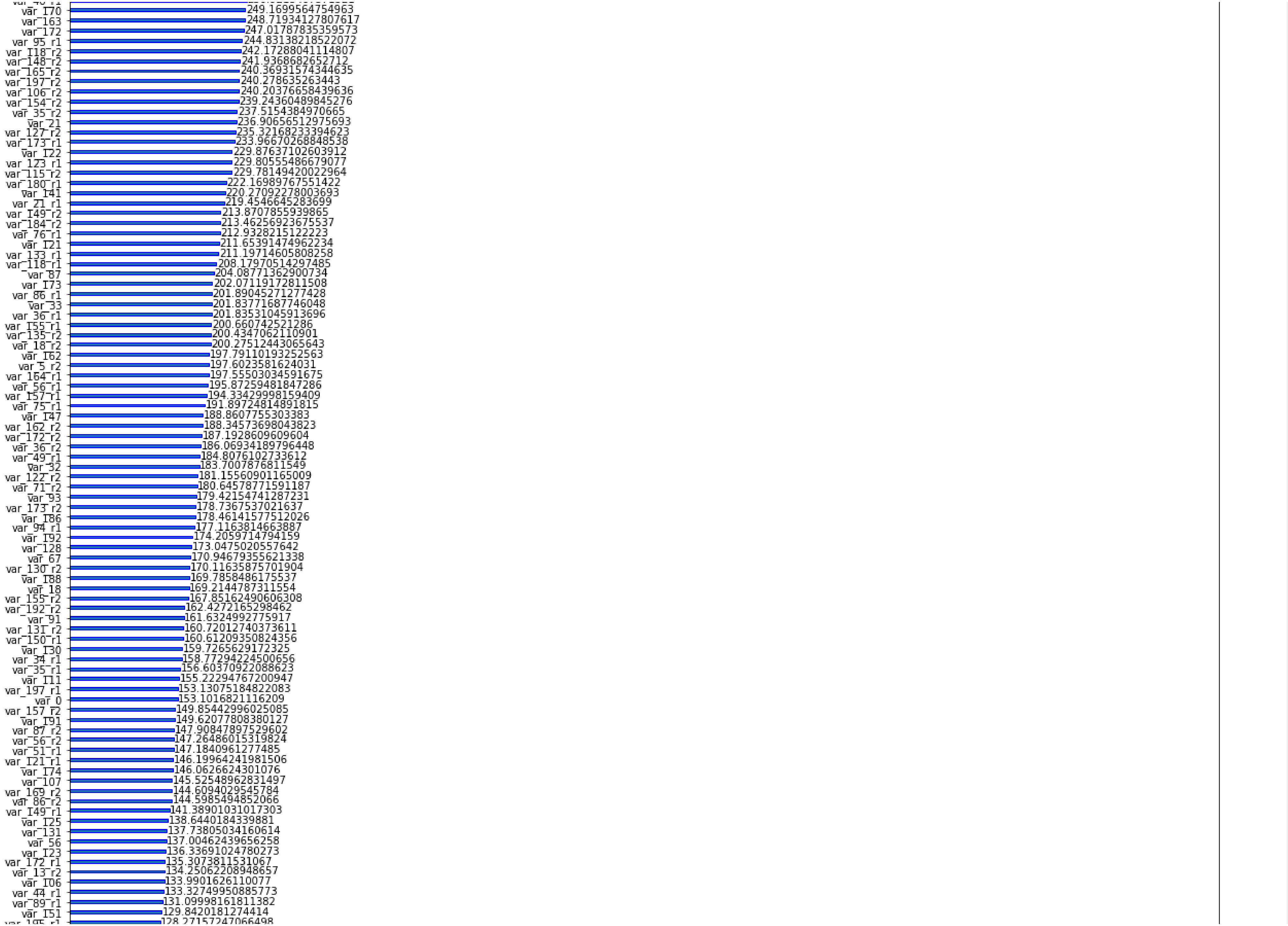
```
{'learning_rate': 0.1,
 'max_bin': 96,
 'max_depth': 2,
 'min_data_in_leaf': 97,
 'min_gain_to_split': 9.293773454804837,
 'num_leaves': 32,
 'reg_alpha': 0.001,
 'reg_lambda': 0.01}
```

In [62]:

```
plt.figure(figsize=(20, 100))
lgb.plot_importance(search.best_estimator_, plt.gca(), grid=False, height=.3, edgecolor='blue',\
                    xlabel='Feature Gain', importance_type='gain')
plt.show()
```

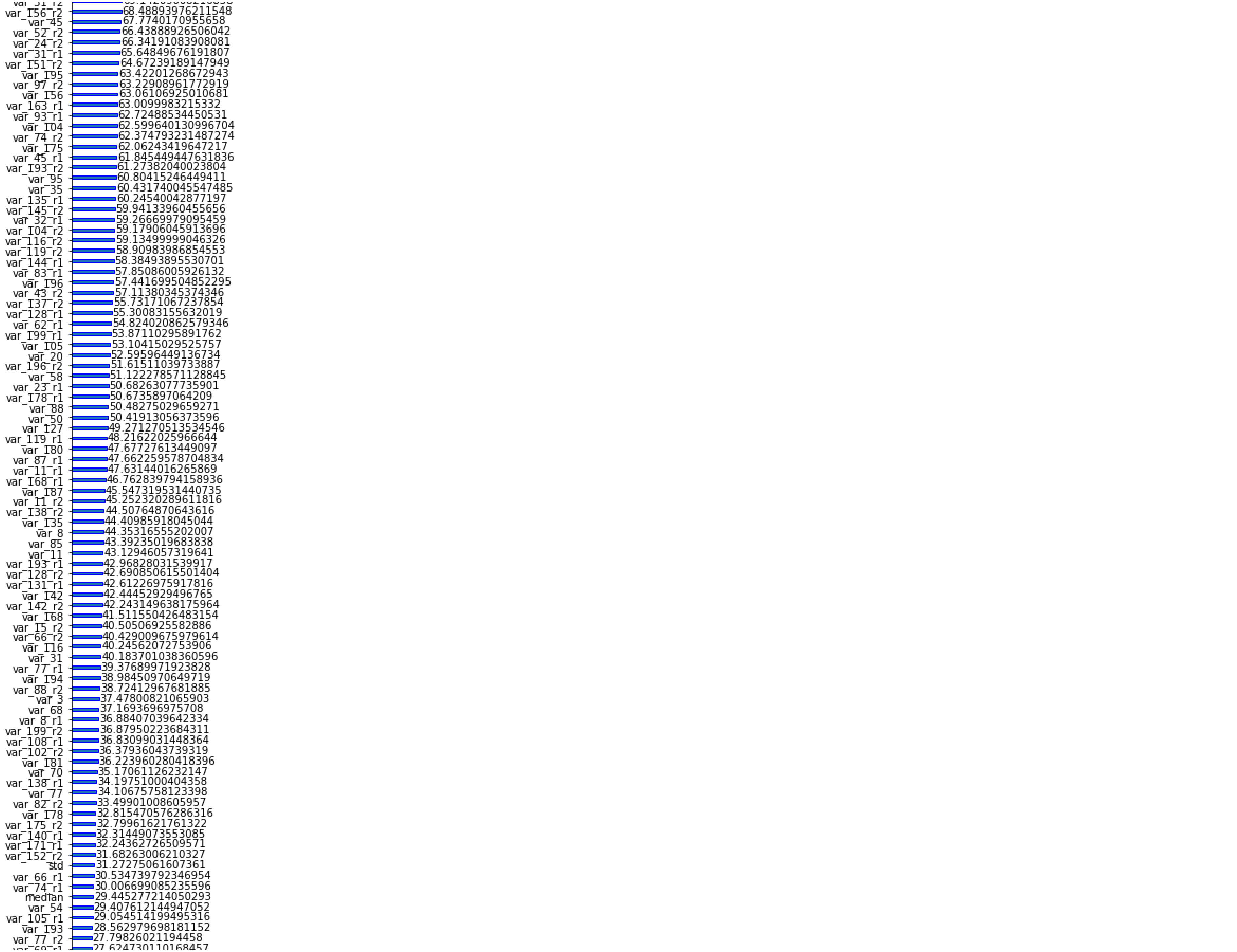



var_94_r2 556.9637405872345
var_26_r2 540.7585518360138
var_40_r2 536.1647257804871
var_166 533.8835045099258
var_53 526.0109502077103
var_139 524.9135451316833
var_108_r2 512.5582256615162
var_80_r2 509.85350036621094
var_26_r1 503.56735467910767
var_78_r2 502.8131255507469
var_80_r1 500.59571611881256
var_166_r2 481.1874795258045
var_190_r1 477.37646222114563
var_0_r1 475.90139269828796
var_76_r2 470.68807773292065
var_164_r2 468.0940878391266
var_108 463.10081082582474
var_12 454.4277993738651
var_110 436.1697564125061
var_94 435.52111491560936
var_92_r2 432.9482145309448
var_146_r1 431.17621541023254
var_53_r1 429.66220819950104
var_6_r2 429.17968702316284
var_166_r1 426.9596860408783
var_109 422.42173290252686
var_174_r1 420.2967780828476
var_133 418.07808381319046
var_165_r1 408.4020302295685
var_169 404.12203991413116
var_44 402.6453881263733
var_190 400.31897170841694
var_78_r1 392.4868114590645
var_146_r2 389.51567965745926
var_18_r1 388.27435398101807
var_89_r2 382.467188000679
var_33_r2 373.45688462257385
var_80 372.76819448452443
var_121_r2 369.39714354276657
var_34 369.15735456347466
var_177_r2 368.73451709747314
var_1_r1 368.2355651855469
var_22_r2 365.165301322937
var_9_r2 363.3715465068817
var_149 362.67470014095306
var_169_r1 359.0918762087822
var_67_r2 355.5495710968971
var_170_r1 355.01714277267456
var_184_r1 353.5151402950287
var_33_r1 353.4653008580208
var_107_r2 351.2331585884094
var_184 351.20506793260574
var_91_r2 350.3860149383545
var_110_r2 340.9974008798599
var_198_r2 338.53313010931015
var_9 336.598849773407
var_188_r1 335.5065517425537
var_177_r1 335.33237540721893
var_179_r2 332.6957525610924
var_2 329.00223183631897
var_170_r2 326.57808113098145
var_123_r2 323.91275453567505
var_119 318.5421048104763
var_147_r1 318.19001841545105
var_198_r1 313.2654882669449
var_115_r1 312.3299674987793
var_154 311.9235579967499
var_127_r1 304.8484605550766
var_13_r1 302.46701860427856
var_115 295.90830278396606
var_95_r2 295.6078462600708
var_92 294.4336233139038
var_164 293.0517385005951
var_1_r2 290.8819988965988
var_1 290.6616052389145
var_154_r1 289.8452498316765
var_2_r1 289.62840712070465
var_78 289.4562611579895
var_86 287.6656322479248
var_109_r2 285.28543615341187
var_179 283.5749843120575
var_92_r1 282.71763394773006
var_180_r2 280.3788344860077
var_145_r1 273.1136820614338
var_148_r1 263.2487154006958
var_192_r1 254.6277039051056
var_75 253.5707504749298
var_40 253.5676367878914
var_190_r2 252.46839714050293
var_122_r1 251.89576768875122

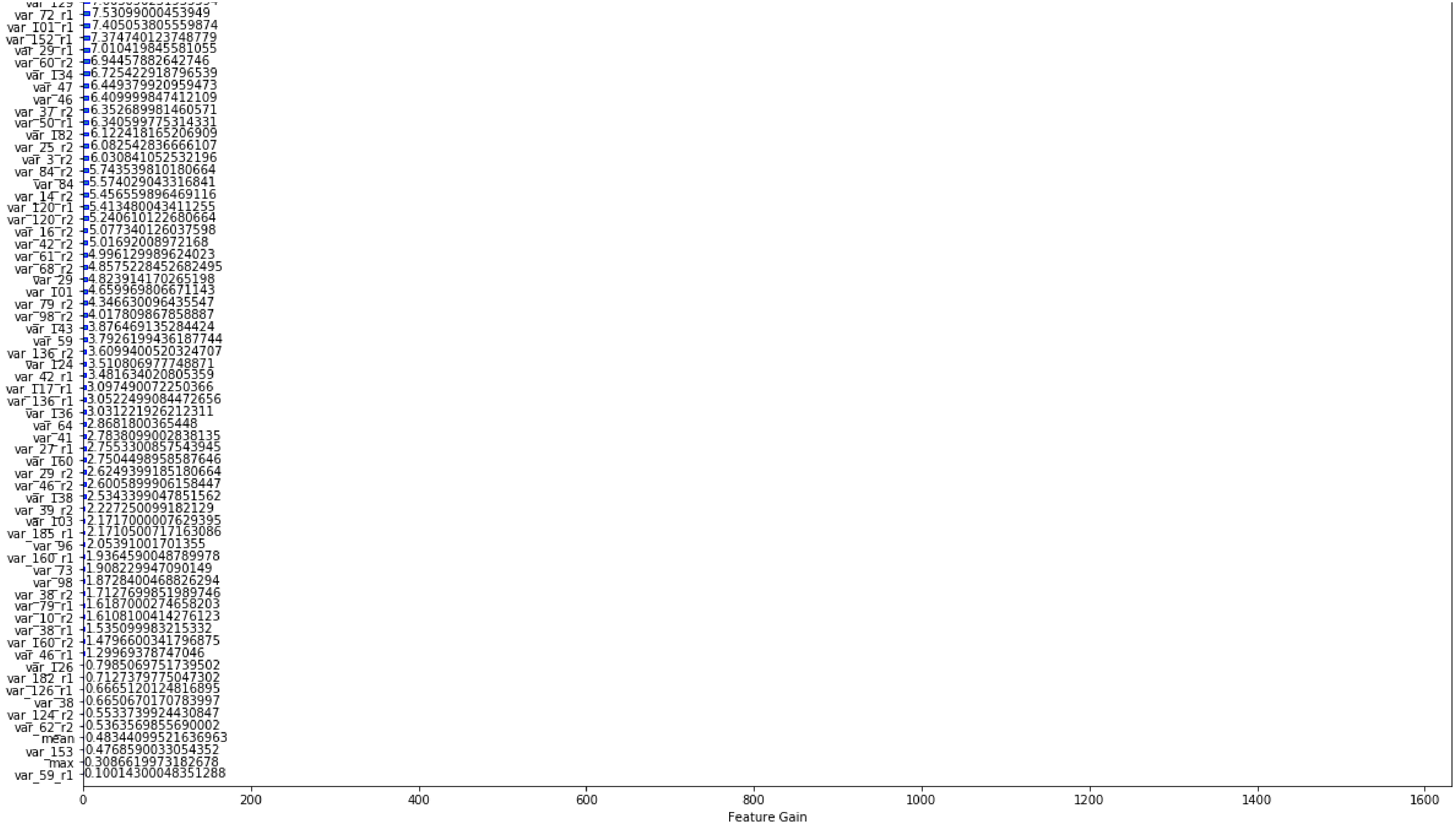


var_170
249.1699564754963
var_163
248.71934127807617
var_172
247.01787835359573
var_95_r1
244.83138218522072
var_118_r2
242.17288041114807
var_148_r2
241.9368682652712
var_165_r2
240.36931574344635
var_197_r2
240.278635263443
var_106_r2
240.20376658439636
var_154_r2
239.24360489845276
var_35_r2
237.5154384970665
var_21
236.90656512975693
var_127_r2
235.32168233394623
var_173_r1
233.96670268848538
var_122
229.87637102603912
var_123_r1
229.80555486679077
var_115_r2
229.78149420022964
var_180_r1
222.16989767551422
var_141
220.27092278003693
var_21_r1
219.4546645283699
var_149_r2
213.8707855939865
var_184_r2
213.46256923675537
var_76_r1
212.9328215122223
var_121
211.65391474962234
var_133_r1
211.19714605808258
var_118_r1
208.17970514297485
var_87
204.08771362900734
var_173
202.07119172811508
var_86_r1
201.89045271277428
var_33
201.83771687746048
var_36_r1
201.83531045913696
var_155_r1
200.660742521286
var_135_r2
200.4347062110901
var_18_r2
200.27512443065643
var_162
197.79110193252563
var_5_r2
197.6023581624031
var_164_r1
197.55503034591675
var_56_r1
195.87259481847286
var_157_r1
194.33429998159409
var_75_r1
191.89724814891815
var_147
188.8607755303383
var_162_r2
188.34573698043823
var_172_r2
187.1928609609604
var_36_r2
186.06934189796448
var_49_r1
184.8076102733612
var_32
183.7007876811549
var_122_r2
181.15560901165009
var_71_r2
180.64578771591187
var_93
179.42154741287231
var_173_r2
178.7367537021637
var_186
178.46141577512026
var_94_r1
177.1163814663887
var_192
174.2059714794159
var_128
173.0475020557642
var_67
170.94679355621338
var_130_r2
170.11635875701904
var_188
169.7858486175537
var_18
169.2144787311554
var_155_r2
167.85162490606308
var_192_r2
162.4272165298462
var_91
161.6324992775917
var_131_r2
160.72012740373611
var_150_r1
160.61209350824356
var_130
159.7265629172325
var_34_r1
158.77294224500656
var_35_r1
156.60370922088623
var_111
155.22294767200947
var_197_r1
153.13075184822083
var_0
153.1016821116209
var_157_r2
149.85442996025085
var_191
149.62077808380127
var_87_r2
147.90847897529602
var_56_r2
147.26486015319824
var_51_r1
147.1840961277485
var_121_r1
146.19964241981506
var_174
146.0626624301076
var_107
145.52548962831497
var_169_r2
144.6094029545784
var_86_r2
144.5985494852066
var_149_r1
141.38901031017303
var_125
138.6440184339881
var_131
137.73805034160614
var_56
137.00462439656258
var_123
136.33691024780273
var_172_r1
135.3073811531067
var_13_r2
134.25062208948657
var_106
133.9901626110077
var_44_r1
133.32749950885773
var_89_r1
131.09998161811382
var_151
129.8420181274414
128.27157247066498

Features
var_5 128.12905222177505
var_32_r2 127.48407729528844
var_5_r1 127.03726105391979
var_48_r2 125.14178228378296
var_51_r2 124.91010993719101
var_93_r2 124.87282931804657
var_82_r1 124.75474571436644
var_177 124.23970770835876
var_155 123.67181766033173
var_67_r1 123.53988367319107
var_141_r2 123.3283234834671
var_111_r2 120.70903015136719
var_186_r2 118.97615273296833
var_99_r1 118.33571696281433
var_191_r2 117.81040000915527
var_167_r2 117.27293992042542
var_43 116.67534148693085
var_114 115.76733994483948
var_107_r1 115.64649939537048
var_71 115.12785053253174
var_105_r2 114.95912927389145
var_137 113.9918992370367
var_167_r1 113.39760184288025
var_151_r1 110.88831049203873
var_196_r1 110.82242953777313
var_85_r2 110.20278739929199
var_125_r2 110.0612123310566
var_89 109.42323064804077
var_58_r1 107.12097942829132
var_132_r1 106.46422731876373
var_24_r1 106.18482041358948
var_188_r2 105.86531090736389
var_48 105.396599650383
var_197 103.64380133152008
var_195_r2 100.85279858112335
var_52_r1 100.19020891189575
var_23 99.99805709719658
var_141_r1 99.83214826136827
var_163_r2 99.60552144050598
var_194_r1 98.55936908721924
var_150_r2 97.3594001531601
var_130_r1 96.32316136360168
var_114_r2 95.53905242681503
var_116_r1 95.09249067306519
var_114_r1 94.3465805053711
var_112 92.85462671518326
var_9_r1 92.03960132598877
var_20_r1 91.38598775863647
var_28 91.08643698692322
var_118 89.8343816101551
var_112_r1 89.23413014411926
var_70_r2 88.51081842184067
var_145 87.39289128780365
var_75_r2 86.96111035346985
var_70_r1 86.30511206388474
var_51 85.75560212135315
var_167 85.66692066192627
var_24 85.55473899841309
var_175_r1 85.12090873718262
var_102_r1 85.00720989704132
var_83 84.50357937812805
var_66 84.15380835533142
var_28_r1 84.00885045528412
var_58_r2 82.8863000869751
var_125_r1 82.32683789730072
var_144_r2 81.53288102149963
var_23_r2 81.50030052661896
var_55_r1 80.9423816204071
var_132 80.30776888132095
var_199 80.20202112197876
var_162_r1 80.09373092651367
var_49_r2 79.51211071014404
var_90_r1 79.17926144599915
var_137_r1 78.8737211227417
var_150 78.81406655907631
var_52 77.52337944507599
var_49 77.24759984016418
var_112_r2 76.76957893371582
var_104_r1 76.50605905056
var_36 76.3929193019867
var_157 75.39267086982727
var_132_r2 75.32687997817993
var_134_r1 74.97945004701614
var_43_r1 74.33635950088501
var_106_r1 73.69708120822906
var_90_r2 73.56552493572235
var_48_r1 71.83264994621277
var_85_r1 71.58482050895691
var_83_r2 70.83020615577698
var_147_r2 69.82772481441498
69.14209008216858

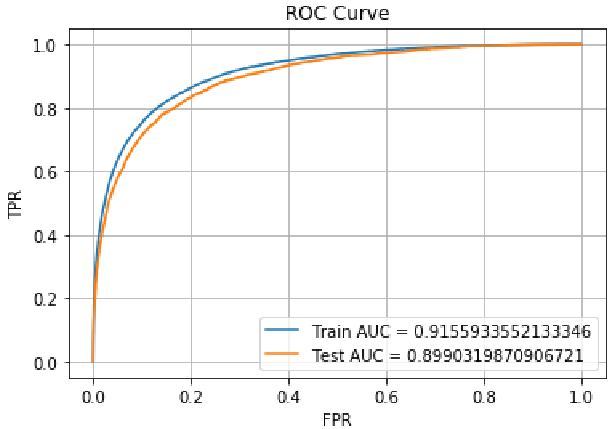

var_156_r2 68.48893976211548
var_45 67.7740170955658
var_52_r2 66.43888926506042
var_24_r2 66.34191083908081
var_31_r1 65.64849676191807
var_151_r2 64.67239189147949
var_195 63.42201268672943
var_97_r2 63.22908961772919
var_156 63.06106925010681
var_163_r1 63.0099983215332
var_93_r1 62.72488534450531
var_104 62.599640130996704
var_74_r2 62.374793231487274
var_175 62.06243419647217
var_45_r1 61.845449447631836
var_193_r2 61.27382040023804
var_95 60.80415246449411
var_35 60.431740045547485
var_135_r1 60.24540042877197
var_145_r2 59.94133960455656
var_32_r1 59.26669979095459
var_104_r2 59.17906045913696
var_116_r2 59.13499999046326
var_119_r2 58.90983986854553
var_144_r1 58.38493895530701
var_83_r1 57.85086005926132
var_196 57.441699504852295
var_43_r2 57.11380345374346
var_137_r2 55.73171067237854
var_128_r1 55.30083155632019
var_62_r1 54.824020862579346
var_199_r1 53.87110295891762
var_105 53.10415029525757
var_20 52.59596449136734
var_196_r2 51.61511039733887
var_58 51.122278571128845
var_23_r1 50.68263077735901
var_178_r1 50.6735897064209
var_88 50.48275029659271
var_50 50.41913056373596
var_127 49.271270513534546
var_119_r1 48.21622025966644
var_180 47.67727613449097
var_87_r1 47.662259578704834
var_11_r1 47.63144016265869
var_168_r1 46.762839794158936
var_187 45.547319531440735
var_11_r2 45.252320289611816
var_138_r2 44.50764870643616
var_135 44.40985918045044
var_8 44.35316555202007
var_85 43.39235019683838
var_11 43.12946057319641
var_193_r1 42.96828031539917
var_128_r2 42.690850615501404
var_131_r1 42.61226975917816
var_142 42.44452929496765
var_142_r2 42.243149638175964
var_168 41.511550426483154
var_15_r2 40.50506925582886
var_66_r2 40.429009675979614
var_116 40.24562072753906
var_31 40.183701038360596
var_77_r1 39.37689971923828
var_194 38.98450970649719
var_88_r2 38.72412967681885
var_3 37.47800821065903
var_68 37.1693696975708
var_8_r1 36.88407039642334
var_199_r2 36.87950223684311
var_108_r1 36.83099031448364
var_102_r2 36.37936043739319
var_181 36.223960280418396
var_70 35.17061126232147
var_138_r1 34.19751000404358
var_77 34.10675758123398
var_82_r2 33.49901008605957
var_178 32.815470576286316
var_175_r2 32.79961621761322
var_140_r1 32.31449073553085
var_171_r1 32.24362726509571
var_152_r2 31.68263006210327
std 31.27275061607361
var_66_r1 30.534739792346954
var_74_r1 30.006699085235596
median 29.445277214050293
var_54 29.407612144947052
var_105_r1 29.054514199495316
var_193 28.562979698181152
var_77_r2 27.79826021194458


var_55_r2 27.532390117645264
var_63_r2 27.405786961317062
var_69_r2 27.27226960659027
var_91_r1 27.11603021621704
var_194_r2 26.846639752388
var_54_r2 26.667489767074585
var_54_r1 26.47509002685547
var_186_r1 26.224831461906433
var_57 26.19958996772766
var_45_r2 25.42628026008606
var_156_r1 25.330950260162354
var_187_r1 24.965855214744806
var_113_r1 24.797169148921967
var_61 24.73486065864563
var_143_r1 24.67559051513672
var_113_r2 24.44711485505104
var_25 24.339500427246094
var_171_r2 23.78552007675171
var_97_r1 23.48861026763916
var_8_r2 23.30532994866371
var_187_r2 22.915729999542236
var_88_r1 22.877708230167627
var_20_r2 22.873239636421204
var_71_r1 22.656479835510254
var_4_r1 22.525870323181152
var_143_r2 22.05776023864746
var_142_r1 21.83076000213623
var_57_r2 21.219655960798264
var_72_r2 20.756102204322815
var_90 20.475979804992676
var_74 19.243520259857178
var_3_r1 19.224290251731873
var_134_r2 19.154069900512695
var_159_r1 19.13116955757141
var_64_r1 18.9623062312603
var_181_r1 18.633569464087486
var_178_r2 18.478822946548462
var_60_r1 18.287110209465027
var_176 17.992100596427917
var_50_r2 17.91514015197754
var_37_r1 17.660368591547012
var_55 17.625839948654175
var_28_r2 17.39963048696518
var_168_r2 17.04518985748291
var_15 16.35330079495907
var_144 16.324759483337402
var_97 15.815430164337158
var_140 15.812930345535278
var_61_r1 15.408140063285828
var_4_r2 15.400540113449097
var_82 15.359994411468506
var_4 15.344800472259521
var_189_r2 15.065060019493103
var_15_r1 14.682305872440338
var_101_r2 14.582570314407349
var_72 14.473740100860596
var_181_r2 14.367660284042358
var_171 14.353876650333405
var_113 13.768283247947693
var_16_r1 12.648499488830566
var_19 12.31565997004509
var_64_r2 11.941399574279785
var_59_r2 11.891049981117249
min 11.745559692382812
var_102 11.455499649047852
var_65 11.443639993667603
var_63_r1 11.233974933624268
var_60 11.215749859809875
var_140_r2 11.031369805335999
var_14_r1 10.978609800338745
var_159_r2 10.93529987335205
var_19_r1 10.736100196838379
var_152 10.578300476074219
var_153_r1 10.532600402832031
var_120 9.906049966812134
var_159 9.566500067710876
var_73_r2 9.559209823608398
var_27_r2 9.364969968795776
var_153_r2 9.006579875946045
var_65_r1 8.9757399559021
var_16 8.889770030975342
var_19_r2 8.579850196838379
var_69 8.530179977416992
var_189 8.471030235290527
var_65_r2 8.421059608459473
var_63 8.21517014503479
var_161 8.193869590759277
var_129_r2 8.182149887084961
var_73_r1 7.875024974346161
var_14 7.635739803314209



In [28]:

```
y_train_pred = search.best_estimator_.predict_proba(X_tr_2)[:,1]
y_test_pred = search.best_estimator_.predict_proba(X_te_2)[:,1]

train_fpr, train_tpr, tr_thresholds = metrics.roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = metrics.roc_curve(y_test, y_test_pred)

roc_plot(train_fpr, train_tpr, test_fpr, test_tpr)
```



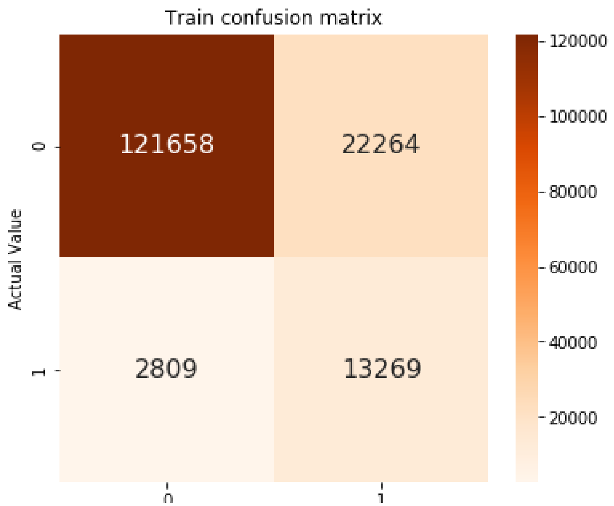
In [0]:

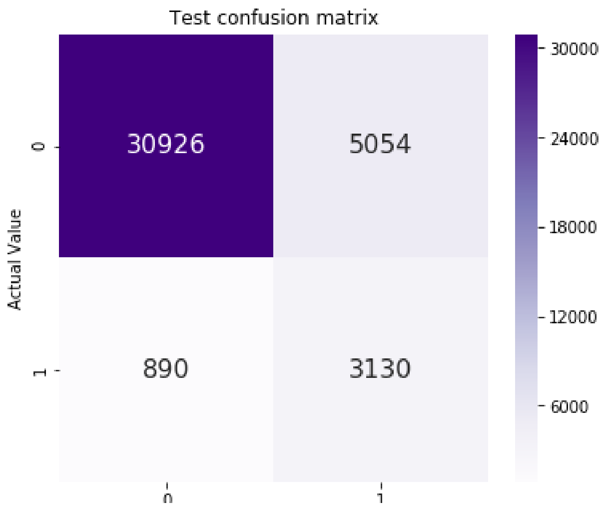
```
pt_2.add_row(['', '', '', ''])
pt_2.add_row(['Light GBM', 'Max_depth = 2, num_leaves = 32', np.round(.9155933, 2), np.round(.89903198, 2)])
```

In [31]:

```
printConfusionMatrix(y_train, y_test, y_train_pred, y_test_pred, tr_thresholds, te_thresholds, train_fpr, train_tpr, test_fpr, test_tpr)
```

```
The maximum value of Train tpr*(1-fpr) is 0.69762117902598 for threshold 0.116

The maximum value of Test tpr*(1-fpr) is 0.6692384381593975 for threshold 0.128
```





Predicted Value Predicted Value

Comparision of Models :

In [80]:

```
print('Modelling With the basic features (mean, std, median, min, max) :')
print(pt_1)
```

```
Modelling With the basic features (mean, std, median, min, max) :
+---------------------+-----------------------------------+-----------+----------+
|        Model        |          Hyper Parameters         | Train AUC | Test AUC |
+---------------------+-----------------------------------+-----------+----------+
| Logistic Regression |            alpha = 0.01           |   0.859   |  0.857   |
|                     |                                   |           |          |
|    Random Forest    | Max_depth = 8, n_estimators = 100 |   0.898   |  0.807   |
|                     |                                   |           |          |
|       XGBoost       | Max_depth = 9, n_estimators = 181 |   0.963   |   0.88   |
|                     |                                   |           |          |
|      Light GBM      |  Max_depth = 6, num_leaves = 181  |   0.945   |  0.866   |
+---------------------+-----------------------------------+-----------+----------+
```

In [78]:

```
print('Modelling With the new features(rounding floats) :')
print(pt_2)
```

```
Modelling With the new features(rounding floats) :
+---------------------+---------------------------------------------------------+-----------+----------+
|        Model        |                     Hyper Parameters                    | Train AUC | Test AUC |
+---------------------+---------------------------------------------------------+-----------+----------+
| Logistic Regression | C = 0.01, l1_ratio = 0.5816246884542369, max_iter = 883 |   0.888   |  0.857   |
|                     |                                                         |           |          |
|       XGBoost       |           Max_depth = 3, n_estimators = 122             |   0.869   |  0.846   |
|                     |                                                         |           |          |
|      Light GBM      |             Max_depth = 2, num_leaves = 32              |    0.92   |   0.9    |
+---------------------+---------------------------------------------------------+-----------+----------+
```

References :

[1] https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/89034#latest-548982

[2] https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/89003#latest-638601

[3] https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/88939#latest-637010

Light GBM :

[1] https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/

[2] https://lightgbm.readthedocs.io/en/latest/Python-API.html#

[3] https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html

[4] https://www.kaggle.com/roydatascience/eda-pca-simple-lgbm-on-kfold-technique

Steps followed :

1. Collected the data from the kaggle website of the competition Santander customer transaction prediction (classification) dataset. The performance metric used for the classification task is **Area Under ROC**.
2. Performed the data cleaning, processing checking for unknown values, filling the missing values. But found that dataset is complete without any missing values, that made the preprocessing simple.
3. Then performed the high level analysis of the dataset like,
   - No. of Datapoints,
   - No. of unique class labels and
   - No. of datapoints per unique class label.
4. Found that the dataset is highly imbalanced with no. of datapoints per class,
   - **Class - 1 (Customers who made txn) : 10%**
   - **Class - 0 (Customers who didn't made txn) : 90%**
5. Performed the EDA on the raw features associated with the dataset and found that features are independent of each other. Written some of my observations inline in the EDA.
6. Applied the dimensionality reduction using the PCA to 2 dimnesions and plotted the same. It's hard in separating the classes using linear models as both were completely overlapping.
7. Then I've added the basic features such as mean, standard deviaiton, median, minimum, maximum of 200 numerical features per data point.
8. Made the train and test split of ratio 80 : 20. Used the train data with the basic features for modelling - I.
9. The basic features gave a reasonable **AUROC** of around **0.85** using the Logistic Regression, Random Forest, XGBoost and Light GBM with hyperparameter tuning. Performed hyperparameter tuning for all the models using RandomizedSearch Cross-Validaiton.
10. Referrring to some of the kaggle discussions I've came across and learnt a new boosting algorithm **Light Gradient boosted Machines(LGBM)** which does splitting leaf-wise instead of depth-wise and gave better predictions than XGBoost.
11. Now I wanted to increase **AUROC** a bit further and gone through some of the kaggle discussions that I referred above. Added some of the count and rounding features for each raw feature as per the discussions.
12. Did the modelling - II with these new count, round features and performed the hyperparameter tuning with previous LR, XGBoost, LGBM.
13. The overfitting problem seems to be reduced for all the models with the new count, rounf features.
14. LGBM seems to be outperform the other models without overfitting to the training data with the new count, round features which gave an **AUROC** of,
    - `Train data` -> `0.92`.
    - `Test data` -> `0.9`.
15. plotted the feature importance for the LGBM and found that the round features contributed most at each split.
16. Finally compared the all the models for basic and the round features in a table with their train and test **AUROC** and hyperparameters.