In [1]:

```
#Importing Libraries
import warnings
warnings.filterwarnings("ignore")
import csv
import pandas as pd
import datetime
import time
import numpy as np
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns
from matplotlib import rcParams
from sklearn.cluster import MiniBatchKMeans, KMeans
import math
import pickle
import os
import xgboost as xgb
import warnings
import networkx as nx
import pdb
import pickle
from pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score


from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [2]:

```
#Load the graph
if os.path.isfile('data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter=',',create_using=nx.DiGraph()
,nodetype=int)

    print(nx.info(train_graph))
else:
    print("Please run the FB_EDA.ipynb or download the files from drive")
```

```
Name:
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
```

In [3]:

```
#Load stage4 data
from pandas import read_hdf
df_final_train = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'train_df',mode='r')
df_final_test = read_hdf('data/fea_sample/storage_sample_stage4.h5', 'test_df',mode='r')
```

In [4]:

```
#Get feature information
df_final_train.columns
```

Out[4]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees', 'adar_index',
       'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities_s',
       'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4',
       'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
       'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
       'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
       'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6'],
      dtype='object')
```

1.0 Preferential Attachment:

1. https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_prediction.preferential_attachment.html (https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_prediction.preferential_attachment.html)
2. http://be.amazd.com/link-prediction/ (http://be.amazd.com/link-prediction/)
3. D. Liben-Nowell, J. Kleinberg. The Link Prediction Problem for Social Networks (2004). http://www.cs.cornell.edu/home/kleinber/link-pred.pdf (http://www.cs.cornell.edu/home/kleinber/link-pred.pdf)

Preferential Attachment: One well-known concept in social networks is that users with many friends tend to create more connections in the future. This is due to the fact that in some social networks, like in finance, the rich get richer. We estimate how "rich" our two vertices are by calculating the multiplication between the number of friends (|Γ(x)|) or followers each vertex has. It may be noted that the similarity index does not require any node neighbor information; therefore, this similarity index has the lowest computational complexity.

In [5]:

```
#Preferential attachment for followees
def prefer_attach_followees(a,b):
    try:
        if (len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b))) == 0):
            return 0
        else:
            pa_score = len(set(train_graph.successors(a))) * len(set(train_graph.successors(b)))
    except:
        return 0
    return pa_score
```

In [6]:

```
prefer_attach_followees(832016, 1543415)
```

Out[6]:

8662

In [7]:

```
#Preferential attachment for followees
def prefer_attach_followers(a,b):
    try:
        if (len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecessors(b))) == 0):
            return 0
        else:
            pa_score = len(set(train_graph.predecessors(a))) * len(set(train_graph.predecessors(b)))
    except:
        return 0
    return pa_score
```

In [8]:

```
prefer_attach_followers(832016, 1543415)
```

Out[8]:

1598

Adding the new preferential attachment features to the dataset

In [9]:

```
#Adding features to train data
df_final_train['pref_attach_followees'] = df_final_train.apply(lambda row: prefer_attach_followees(row['source_node'],row['destination_node']),axis=1)
df_final_train['pref_attach_followers'] = df_final_train.apply(lambda row: prefer_attach_followers(row['source_node'],row['destination_node']),axis=1)
```

In [10]:

```
#Adding features to test data
df_final_test['pref_attach_followees'] = df_final_test.apply(lambda row: prefer_attach_followees(row['source_node'],row['destination_node']),axis=1)
df_final_test['pref_attach_followers'] = df_final_test.apply(lambda row: prefer_attach_followers(row['source_node'],row['destination_node']),axis=1)
```

In [11]:

```
df_final_train[['pref_attach_followees','pref_attach_followers']].head()
```

Out[11]:

	pref_attach_followees	pref_attach_followers
0	120	66
1	8662	1598
2	902	980
3	35	22
4	33	5

In [12]:

```
df_final_test[['pref_attach_followees','pref_attach_followers']].head()
```

Out[12]:

	pref_attach_followees	pref_attach_followers
0	54	84
1	19	34
2	144	150
3	340	407
4	405	324

In [13]:

```
len(df_final_test.columns)
```

Out[13]:

56

2. SVD Dot

We have already computed the 12 dimensional SVD features (6D for source and 6D for destination). 12 dimensions each for both right as well as left singular matrix. We will now do a dot product of the SVD features of both source and destination for both the right as well as left singular matrices, and also for both train and test data

In [14]:

```
#This function will compute the dot product between source svd(6D) and destination svd(6D)
def svd_dot(svd):
    svd_dot_list=[]
    for index, row in tqdm(svd.iterrows()):
        svd_source = row[0:6].values
        svd_dest = row[6:12].values
        svd_dot = np.dot(svd_source,svd_dest)
        svd_dot_list.append(svd_dot)
    return svd_dot_list
```

Adding the new SVD dot features to the dataset

In [15]:

```
#SVD values for U, for train data
svd_u = df_final_train[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_s_6','svd_u_d_1',
'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_d_6']]
df_final_train['svd_dot_u']=svd_dot(svd_u)
```

```
100002it [00:18, 5303.92it/s]
```

In [16]:

```
#SVD values for v, for train data
svd_v = df_final_train[['svd_v_s_1', 'svd_v_s_2','svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6','svd_v_d_1',
'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v_d_6']]
df_final_train['svd_dot_v']=svd_dot(svd_v)
```

```
100002it [00:18, 5328.08it/s]
```

In [17]:

```
#SVD values for U, for test data
svd_u = df_final_test[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_s_6','svd_u_d_1', '
svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_d_6']]
df_final_test['svd_dot_u']=svd_dot(svd_u)
```

```
50002it [00:09, 5308.69it/s]
```

In [18]:

```
#SVD values for v, for test data
svd_v = df_final_test[['svd_v_s_1', 'svd_v_s_2','svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6','svd_v_d_1', '
svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v_d_6']]
df_final_test['svd_dot_v']=svd_dot(svd_v)
```

```
50002it [00:09, 5301.38it/s]
```

In [21]:

```
df_final_test.columns
```

Out[21]:

```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followees_s',
       'num_followees_d', 'inter_followers', 'inter_followees', 'adar_index',
       'follows_back', 'same_comp', 'shortest_path', 'weight_in', 'weight_out',
       'weight_f1', 'weight_f2', 'weight_f3', 'weight_f4', 'page_rank_s',
       'page_rank_d', 'katz_s', 'katz_d', 'hubs_s', 'hubs_d', 'authorities_s',
       'authorities_d', 'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4',
       'svd_u_s_5', 'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3',
       'svd_u_d_4', 'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2',
       'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1',
       'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6',
       'pref_attach_followees', 'pref_attach_followers', 'svd_dot_u',
       'svd_dot_v'],
      dtype='object')
```

In [22]:

```
df_final_train.to_csv("data/df_final_train_feature_engineered.csv", index=None)
df_final_test.to_csv("data/df_final_test_feature_engineered.csv", index=None)
```

Applying Machine Learning models on the final set of features

In [2]:

```
#Helper function for plotting train and test confusion matrices
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))
    plt.figure(figsize=(20,4))

    labels = [0,1]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

1. Load the feature engineered dataset

In [3]:

```
#Independent variables
df_final_train=pd.read_csv("data/df_final_train_feature_engineered.csv")
df_final_test=pd.read_csv("data/df_final_test_feature_engineered.csv")
```

In [4]:

```
#Target variables
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

In [5]:

```
#Drop the unwanted features + the target variable
df_final_train.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
df_final_test.drop(['source_node', 'destination_node','indicator_link'],axis=1,inplace=True)
```

2. Apply Random Forest Classifier

1. First, we will tune the number of estimators
2. We will tune tune the max depth
3. We will tune all the hyperparameters using Random Search CV

1. In this block we are tuning only the number of estimators.

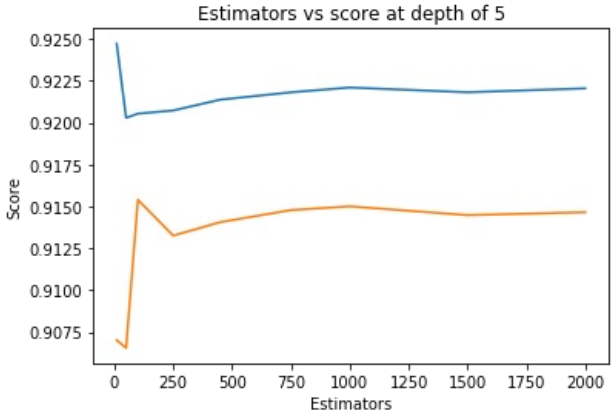
In [11]:

```
estimators = [10,50,100,250,450,750,1000,1500,2000]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=i, n_jobs=-1,random_state=25,verbose=0,warm_start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('Estimators = ',i,'Train F1 Score',train_sc,'Test F1 Score',test_sc)
plt.plot(estimators,train_scores,label='Train Score')
plt.plot(estimators,test_scores,label='Test Score')
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.title('Estimators vs score at depth of 5')
```

```
Estimators =  10 Train F1 Score 0.9247145749410366 Test F1 Score 0.9070220790778489
Estimators =  50 Train F1 Score 0.9202873688697587 Test F1 Score 0.9065528622784703
Estimators =  100 Train F1 Score 0.9205365064453838 Test F1 Score 0.9153941222061627
Estimators =  250 Train F1 Score 0.9207252044081052 Test F1 Score 0.913255647637878
Estimators =  450 Train F1 Score 0.9213715791565144 Test F1 Score 0.9140608560245782
Estimators =  750 Train F1 Score 0.9218091503267974 Test F1 Score 0.9147812177719782
Estimators =  1000 Train F1 Score 0.922095176324131 Test F1 Score 0.9150037888355647
Estimators =  1500 Train F1 Score 0.9218124209183406 Test F1 Score 0.9144793618051316
Estimators =  2000 Train F1 Score 0.9220436856640722 Test F1 Score 0.9146526315789474
```

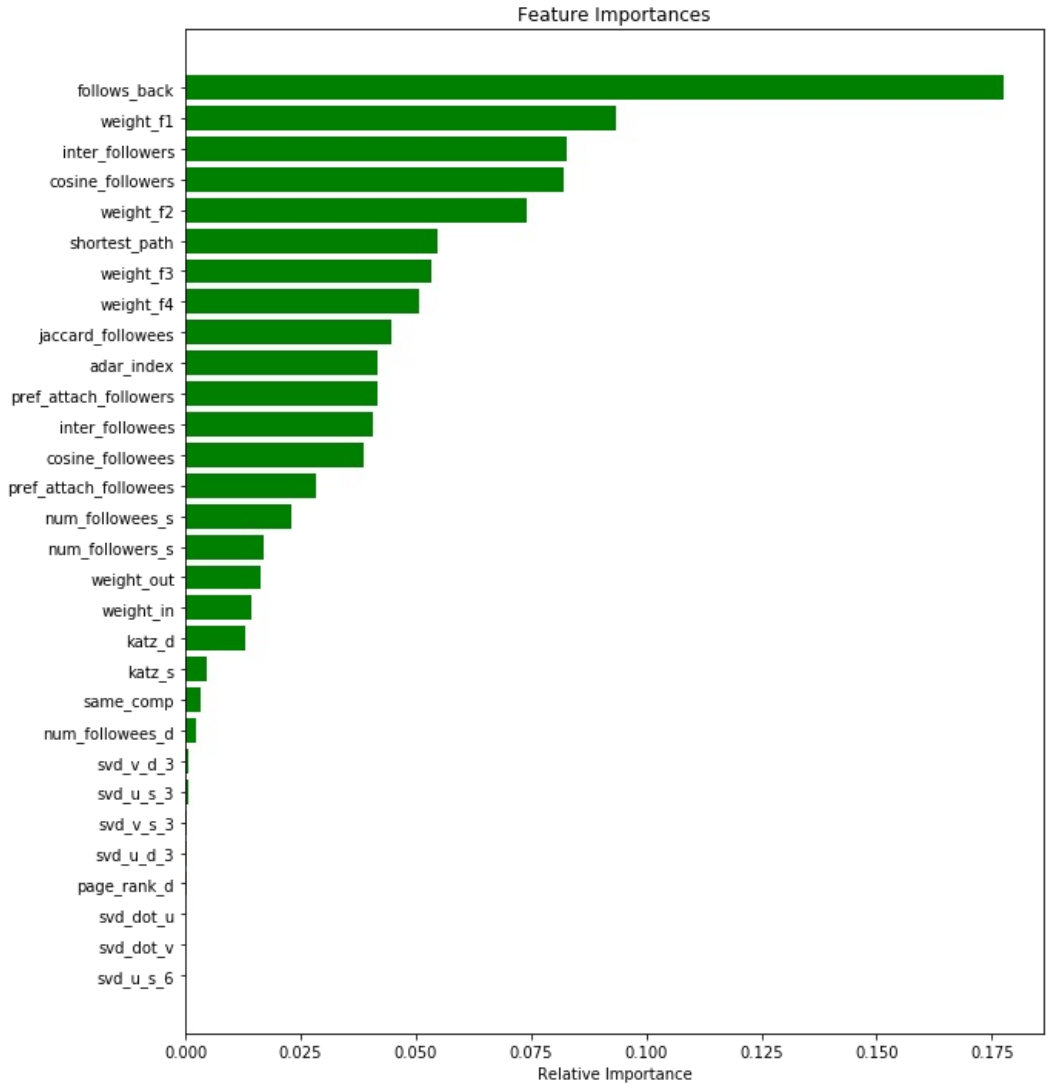
Out[11]:

```
Text(0.5, 1.0, 'Estimators vs score at depth of 5')
```



In [16]:

```
#Get feature importances
features = df_final_train.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-30:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='g', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

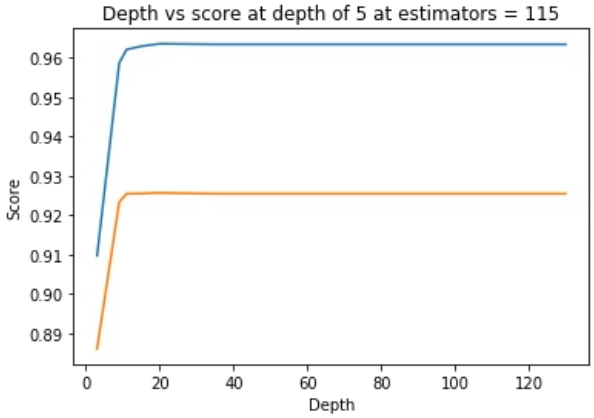


2. Tune the max depth parameter

In [18]:

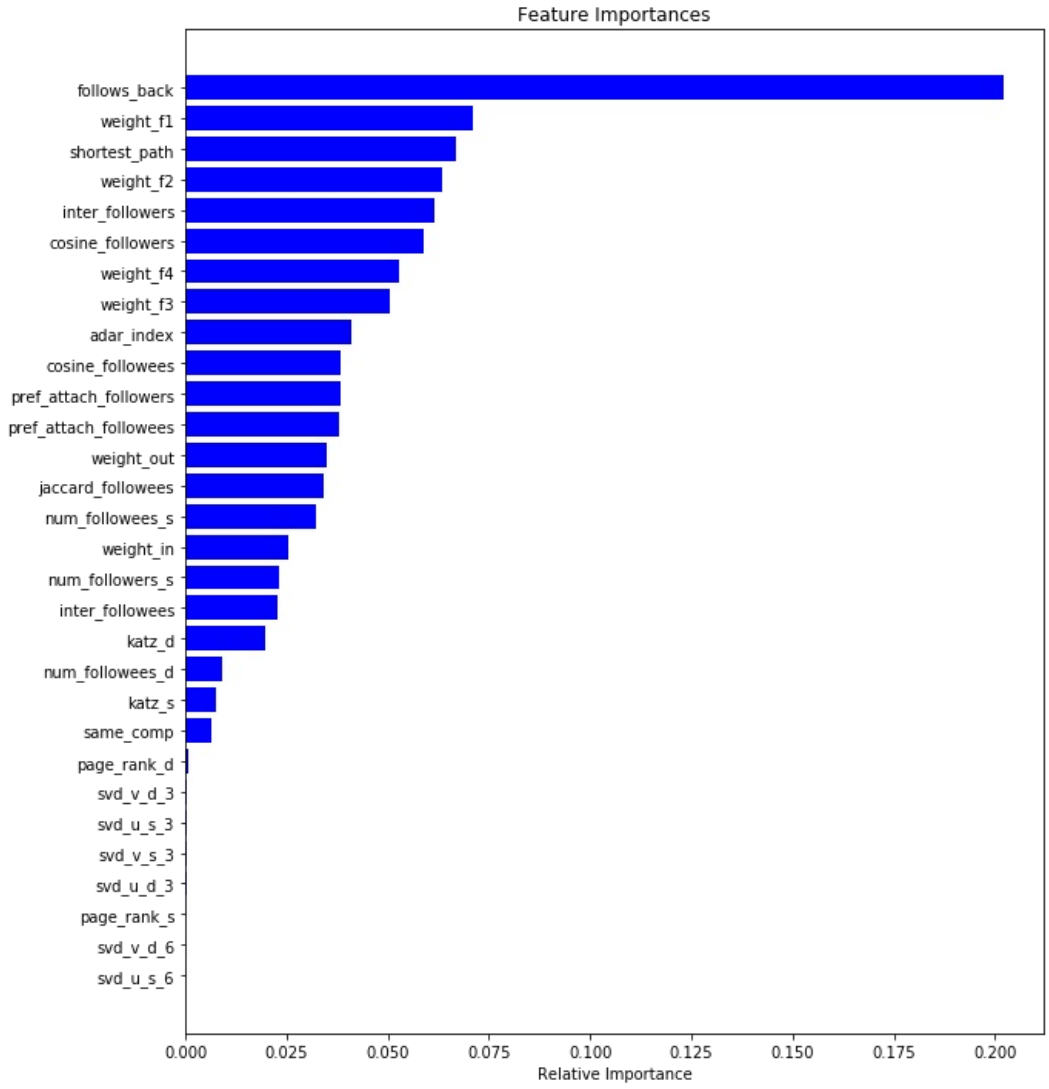
```
depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=i, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=115, n_jobs=-1,random_state=25,verbose=0,warm_start=False)
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('Depth = ',i,'Train F1 Score',train_sc,'Test F1 Score',test_sc)
plt.plot(depths,train_scores,label='Train Score')
plt.plot(depths,test_scores,label='Test Score')
plt.xlabel('Depth')
plt.ylabel('Score')
plt.title('Depth vs score at depth of 5 at estimators = 115')
plt.show()
```

```
Depth =  3 Train F1 Score 0.9097409508457588 Test F1 Score 0.8861138650410276
Depth =  9 Train F1 Score 0.9586007375262311 Test F1 Score 0.9233071297603743
Depth =  11 Train F1 Score 0.9620353443022547 Test F1 Score 0.9254529004565826
Depth =  15 Train F1 Score 0.96287804234683 Test F1 Score 0.9255030593578503
Depth =  20 Train F1 Score 0.9635026141936529 Test F1 Score 0.9256746823479995
Depth =  35 Train F1 Score 0.9633638381320118 Test F1 Score 0.9254490074571998
Depth =  50 Train F1 Score 0.9633638381320118 Test F1 Score 0.9254490074571998
Depth =  70 Train F1 Score 0.9633638381320118 Test F1 Score 0.9254490074571998
Depth =  130 Train F1 Score 0.9633638381320118 Test F1 Score 0.9254490074571998
```



In [20]:

```
#Get feature importances
features = df_final_train.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-30:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



3. Hyper-parameter tuning using Random Search CV

In [22]:

```
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform

param_dist = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(10,20),
              "min_samples_split": sp_randint(110,190),
              "min_samples_leaf": sp_randint(25,65)}

clf = RandomForestClassifier(random_state=25,n_jobs=-1)

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,n_iter=10,cv=10,scoring='f1',random_state=25)

rf_random.fit(df_final_train,y_train)
print('Mean test scores: ',rf_random.cv_results_['mean_test_score'])
print('Mean train scores: ',rf_random.cv_results_['mean_train_score'])
```

```
Mean test scores:  [0.96212041 0.96204688 0.96202542 0.96321151 0.96303087 0.96268266
 0.96217733 0.96102021 0.96171663 0.9617239 ]
Mean train scores:  [0.96310717 0.96276998 0.9627793  0.9644579  0.96405141 0.96357035
 0.96310741 0.96163364 0.9627062  0.96282698]
```

In [23]:

```
#Print the best estimator
print(rf_random.best_estimator_)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=17, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=28, min_samples_split=165,
            min_weight_fraction_leaf=0.0, n_estimators=108, n_jobs=-1,
            oob_score=False, random_state=25, verbose=0, warm_start=False)
```

In [26]:

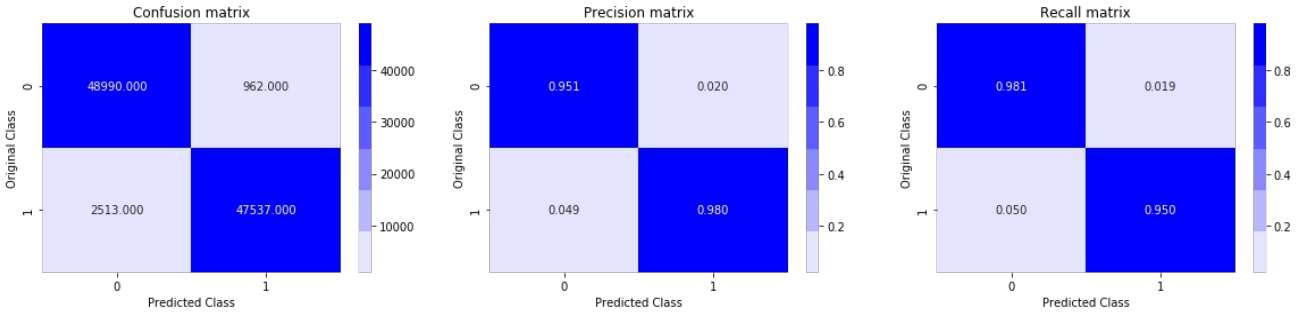
```
#Train a model with the best estimator obtained using random search
clf=rf_random.best_estimator_

clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)

from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9647383535094217
Test f1 score 0.9265374894692502
```

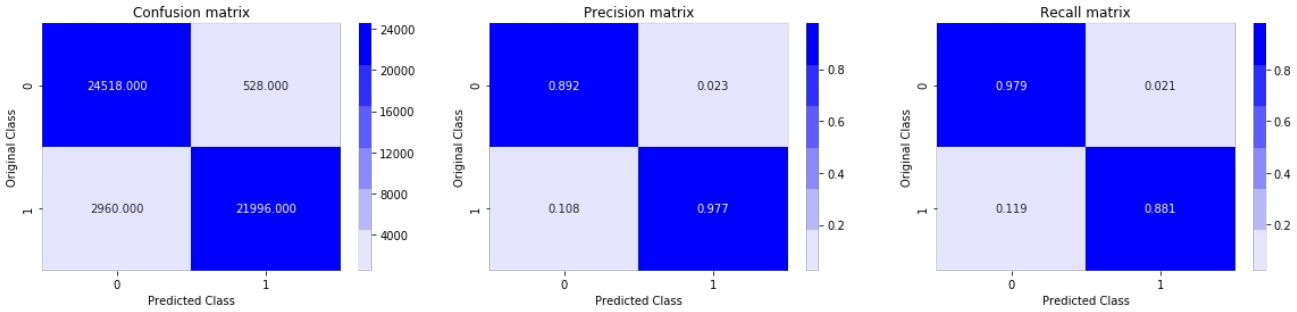
In [27]:

```
#Plot the confusion matrix for train and test data
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

```
Train confusion_matrix
```

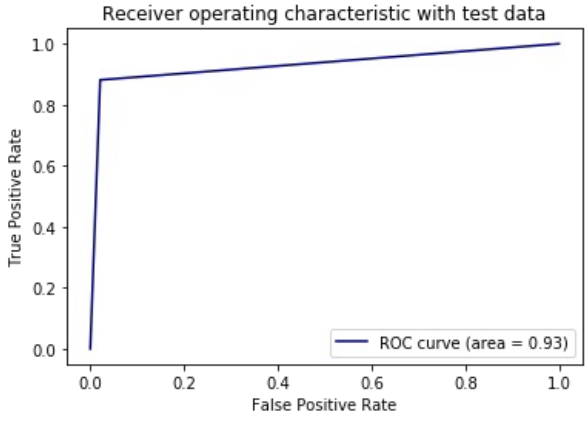



```
Test confusion_matrix
```



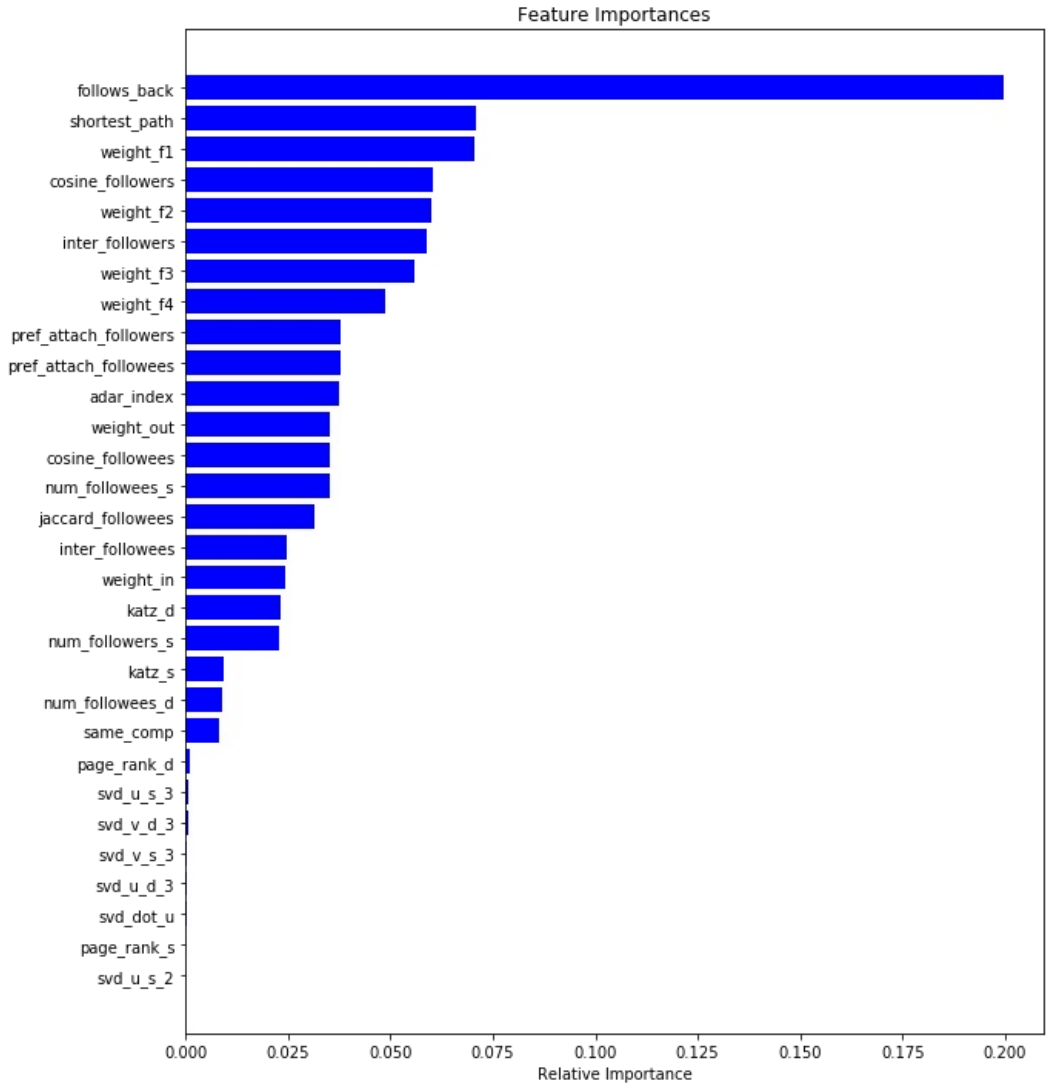
In [28]:

```
#Plot ROC Curve for the best model obtained using random Search
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```



In [29]:

```
#Get the feature importance for the best model trained using random search cross validation
features = df_final_train.columns
importances = clf.feature_importances_
indices = (np.argsort(importances))[-30:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



3. Modeling with XGBoost Classifier

1. First, we will tune the number of estimators
2. We will tune tune the max depth
3. We will tune all the hyperparameters using Random Search CV

1. In this block we are tuning only the number of estimators.

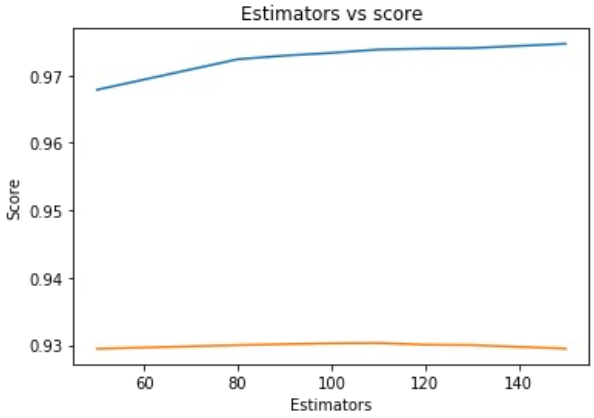
In [14]:

```
estimators = [50,80,90,100,110,120,130,150]
train_scores = []
test_scores = []
for i in estimators:
    clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.8,colsample_bynode=0.7, colsamp
le_bytree=1, gamma=0.8832009558840542,learning_rate=0.1,
                        max_delta_step=0, max_depth=6,min_child_weight=6, missing=None, n_estimators=i, n_jobs=-1
,nthread=None, objective='binary:logistic', random_state=25,
                        reg_alpha=46, reg_lambda=399.7173121530736, scale_pos_weight=1,
                        seed=None, silent=True, subsample=0.7)

    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('Estimators = ',i,'Train F1 Score',train_sc,'Test F1 Score',test_sc)

plt.plot(estimators,train_scores,label='Train Score')
plt.plot(estimators,test_scores,label='Test Score')
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.title('Estimators vs score')
```

```
Estimators =  50 Train F1 Score 0.9678700949431565 Test F1 Score 0.9294719918767981
Estimators =  80 Train F1 Score 0.9723919530663202 Test F1 Score 0.930012063236757
Estimators =  90 Train F1 Score 0.9729323918513108 Test F1 Score 0.9301587301587301
Estimators =  100 Train F1 Score 0.9733337390581099 Test F1 Score 0.9302876982026801
Estimators =  110 Train F1 Score 0.9738344096268287 Test F1 Score 0.9303329875738268
Estimators =  120 Train F1 Score 0.973976489663559 Test F1 Score 0.9300759997459619
Estimators =  130 Train F1 Score 0.9740476890181493 Test F1 Score 0.9300177920867575
Estimators =  150 Train F1 Score 0.9746918768925392 Test F1 Score 0.9294975002118464
```

Out[14]:

```
Text(0.5, 1.0, 'Estimators vs score')
```

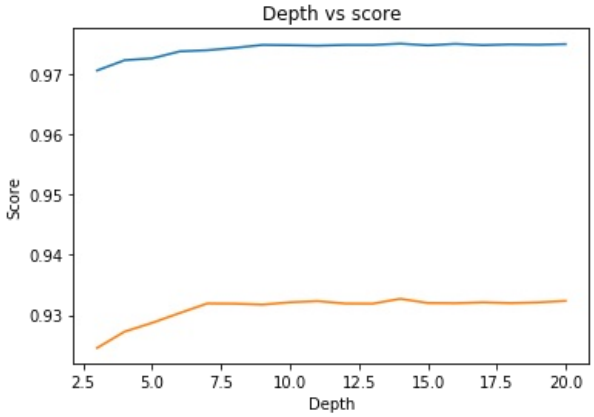


2. Tune the max depth parameter

In [20]:

```
depths = [3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
train_scores = []
test_scores = []
for i in depths:
    clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.8,colsample_bynode=0.7, colsamp
le_bytree=1, gamma=0.8832009558840542,learning_rate=0.1,
                        max_delta_step=0, max_depth=i,min_child_weight=6, missing=None, n_estimators=110, n_jobs=
-1,nthread=None, objective='binary:logistic', random_state=25,
                        reg_alpha=46, reg_lambda=399.7173121530736, scale_pos_weight=1,
                        seed=None, silent=True, subsample=0.7)

    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('Depth = ',i,'Train F1 Score',train_sc,'Test F1 Score',test_sc)

plt.plot(depths,train_scores,label='Train Score')
plt.plot(depths,test_scores,label='Test Score')
plt.xlabel('Depth')
plt.ylabel('Score')
plt.title('Depth vs score')
plt.show()
```

```
Depth =  3 Train F1 Score 0.9706605176127916 Test F1 Score 0.9245275041224472
Depth =  4 Train F1 Score 0.9723695284061527 Test F1 Score 0.9272461248118069
Depth =  5 Train F1 Score 0.9726821444161329 Test F1 Score 0.9287032519462889
Depth =  6 Train F1 Score 0.9738344096268287 Test F1 Score 0.9303329875738268
Depth =  7 Train F1 Score 0.9740284744388711 Test F1 Score 0.9319243368910494
Depth =  8 Train F1 Score 0.9744275311680287 Test F1 Score 0.9318911610613486
Depth =  9 Train F1 Score 0.9749129343160282 Test F1 Score 0.9317326795451663
Depth =  10 Train F1 Score 0.9748648046503149 Test F1 Score 0.9321188554038634
Depth =  11 Train F1 Score 0.9747880433131083 Test F1 Score 0.9323190702588483
Depth =  12 Train F1 Score 0.9749050969276712 Test F1 Score 0.9319141746916709
Depth =  13 Train F1 Score 0.9749040807442726 Test F1 Score 0.931890703915809
Depth =  14 Train F1 Score 0.9751250835578423 Test F1 Score 0.9327022629788493
Depth =  15 Train F1 Score 0.9748384352018801 Test F1 Score 0.9320031282365624
Depth =  16 Train F1 Score 0.9750881030501883 Test F1 Score 0.9319632965453084
Depth =  17 Train F1 Score 0.9748711144422726 Test F1 Score 0.9321131166251004
Depth =  18 Train F1 Score 0.974996202339359 Test F1 Score 0.9319805537941239
Depth =  19 Train F1 Score 0.9749430523917996 Test F1 Score 0.9321024167652529
Depth =  20 Train F1 Score 0.9750450687678504 Test F1 Score 0.9323530654497824
```



In [27]:

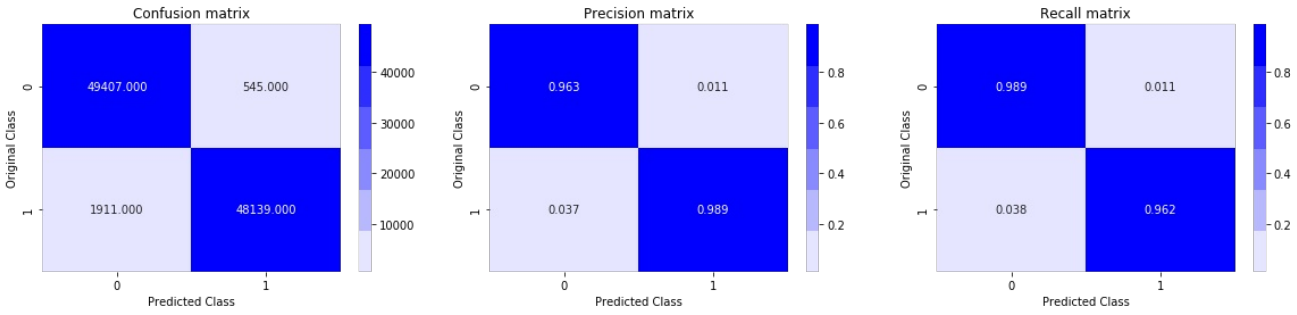
```
#Train a model with the best estimator obtained from above cross validation. n_estimators = 110, max_depth=
clf=xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.8,colsample_bynode=0.7, colsample_byt
ree=1, gamma=0.8832009558840542,learning_rate=0.1,
                      max_delta_step=0, max_depth=14,min_child_weight=6, missing=None, n_estimators=110, n_jobs=-
1,nthread=None, objective='binary:logistic', random_state=25,
                      reg_alpha=46, reg_lambda=399.7173121530736, scale_pos_weight=1,seed=None, silent=True, subs
ample=0.7)

clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)

from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9751250835578423
Test f1 score 0.9327022629788493
```

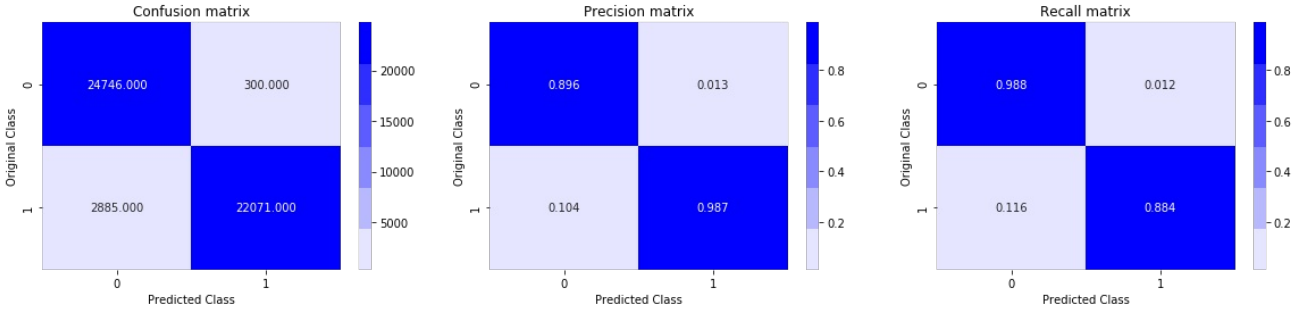
In [28]:

```
#Plot the confusion matrix for train and test data
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

```
Train confusion_matrix
```




```
Test confusion_matrix
```



3. Hyper-parameter tuning using Random Search CV

In [29]:

```
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
import xgboost as xgb

params = {'learning_rate' :[0.00001,0.001,0.01,0.1,0.3],
          'n_estimators':sp_randint(105,120),
          'gamma':uniform(0,1),
          'subsample':(0.4, 0.5, 0.6, 0.7, 0.8, 1),
          'reg_alpha':sp_randint(0,600),
          'reg_lambda':uniform(0,600),
          'max_depth':np.arange(6,15),
          'colsample_bytree':[0.5,0.6,0.7,0.8,1],
          'colsample_bylevel':[0.5,0.6,0.7,0.8,1],
          'colsample_bynode':[0.5,0.6,0.7,0.8,1],
          'min_child_weight':np.arange(1,11)}


clf = xgb.XGBClassifier(random_state=25,n_jobs=-1)

xgb_random = RandomizedSearchCV(clf, param_distributions=params,n_iter=25,cv=10,scoring='f1',random_state=25, n_j
obs=-1)

xgb_random.fit(df_final_train,y_train)

print('Mean test scores: ',xgb_random.cv_results_['mean_test_score'])
print('Mean train scores: ',xgb_random.cv_results_['mean_train_score'])
```

```
Mean test scores:  [0.91952995 0.93207382 0.9306585  0.96355921 0.96578825 0.94968838
 0.94275011 0.9593975  0.92304224 0.92118385 0.92168851 0.92897182
 0.92935871 0.96756342 0.9431952  0.93728678 0.92257958 0.94064258
 0.965886   0.92739883 0.93177614 0.97366573 0.92695431 0.97003725
 0.93022402]
Mean train scores:  [0.91957705 0.93255795 0.93097488 0.96415292 0.96634738 0.950254
 0.94315417 0.95966309 0.92341304 0.92127187 0.92234405 0.92919271
 0.92951859 0.96847824 0.94386712 0.9377575  0.92300498 0.94115811
 0.96641357 0.92759639 0.93210043 0.97435548 0.92726129 0.97094396
 0.93037081]
```

In [30]:

```
#Print the best estimator
print(xgb_random.best_estimator_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.5,
       colsample_bynode=0.6, colsample_bytree=0.7, gamma=0.922923328367314,
       learning_rate=0.1, max_delta_step=0, max_depth=8,
       min_child_weight=6, missing=None, n_estimators=108, n_jobs=-1,
       nthread=None, objective='binary:logistic', random_state=25,
       reg_alpha=59, reg_lambda=196.0533323996213, scale_pos_weight=1,
       seed=None, silent=True, subsample=0.6)
```

In [31]:

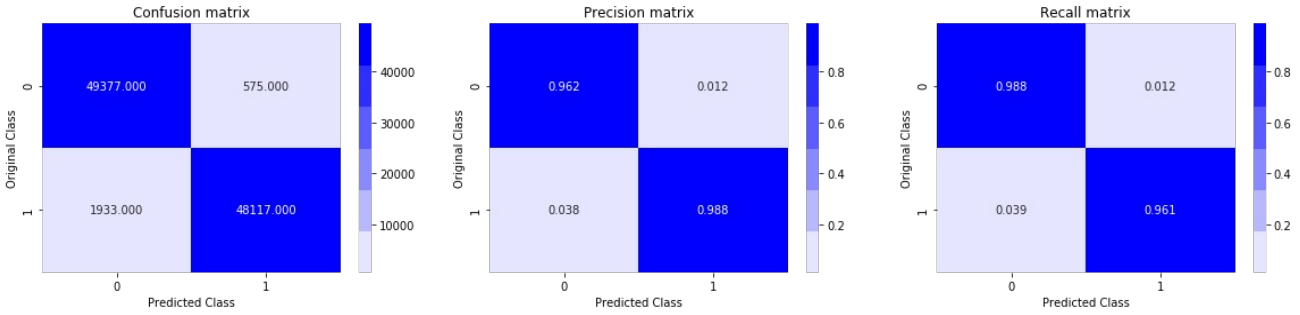
```
#Train a model with the best estimator obtained using random search
clf=xgb_random.best_estimator_

clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)

from sklearn.metrics import f1_score
print('Train f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))
```

```
Train f1 score 0.9746004739624475
Test f1 score 0.9302857142857143
```

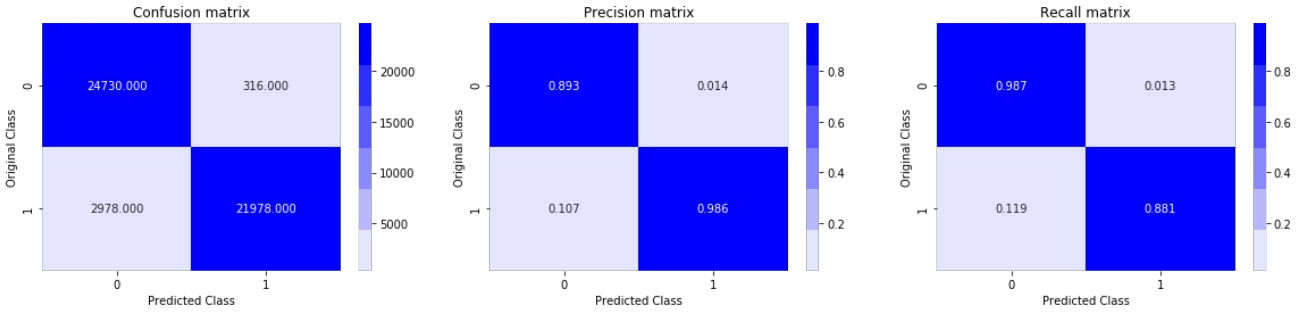
In [32]:

```
#Plot the confusion matrix for train and test data
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

```
Train confusion_matrix
```

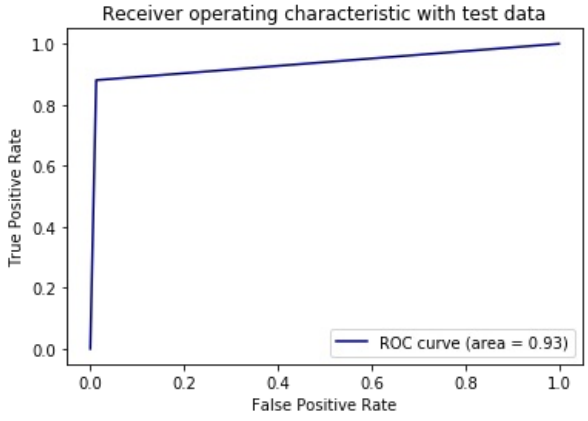



```
Test confusion_matrix
```



In [33]:

```
#Plot ROC Curve for the best model obtained using random Search
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```



Model Comparison.

In [38]:

```
from prettytable import PrettyTable

table =PrettyTable()
table.field_names = ["Model", "Train F1", "Test F1"]
table.add_row(["Random Forest (ver1)", 0.965, 0.924])
table.add_row(["Random Forest (ver2)", 0.964, 0.926])
table.add_row(["XGBoost (ver1)", 0.975, 0.932])
table.add_row(["XGBoost (ver2)", 0.974, 0.931])

print(table)
```

```
+----------------------+----------+---------+
|        Model         | Train F1 | Test F1 |
+----------------------+----------+---------+
| Random Forest (ver1) |  0.965   |  0.924  |
| Random Forest (ver2) |  0.964   |  0.926  |
|    XGBoost (ver1)    |  0.975   |  0.932  |
|    XGBoost (ver2)    |  0.974   |  0.931  |
+----------------------+----------+---------+
```

What we did throughout this experiment?

In this case study given a directed social graph, the task is to predict missing links to recommend friends/followers to new users. The dataset was taken from the facebook recruiting challenge hosted at kaggle at this link: https://www.kaggle.com/c/FacebookRecruiting (https://www.kaggle.com/c/FacebookRecruiting). The dataset only has source node information and destinatin node information. Here each node represents an user. Basically a link between source to destination means that a user follows another user.

In the given data, only those source and destination nodes are given for which an edge exists. There is no information about the nodes whcih does not have an edge between them. So, in order to map this problem to a binary classification problem of whether or not an edge exists in the graph, we need to create training and testing sample which has a class label of 0 (0 means that there are no edges present between source to destination).

NOTE: In the given dataset, we have roughly 9.43 million edges and 1.93 million nodes (vertices or users). For the given data, all the links are present and hence the class label will be 1. However, for classification we also need 0 class labels. How do we generate the 0 class labels?

1. Create the same number of 0 labeled pairs of vertices that we have for class 1 labeled pairs of vertices.
2. Randomly sample a pair of vertices
3. Check if the path length is greater than 2.
4. Check if no edge connection exists between the pairs of vertices.
5. If both the above conditions are satisfied then we will have a new pair of edges which will have a class label 0.

Coming to business constraints, there are no low latency requirements. We need to use probability estimates to predict edges between two nodes. This will give us more interpretability in terms of which edge connections are more important. The metric we have chosen is F1 score and binary confusion matrix.

We curated features like number of followers and followees of each node, whether or not a node is followed back by any other nodes, page rank of individual nodes, katz score, jaccard index, preferential attachment,svd features, svd dot features, adar index and so on. There were a total of 59 features on which we train and test our model.

There is not timestamp provided for this data. Ideally, if you think about it, we have the dataset for a given time stap t. However, the graph is evolving and changing over time. After 30 days the edge connections might change, because people might have new followers and they may even start to follow new people. In the real word, we would split the data according to time. But, since we do not have any information about time stamp, we will split the data randomly in 80:20 ratio. 80% for training data and 20% for cross validation data.

Hyper-parameter tuning using cross validation is done for Random Forest classifier. Feature importance and feature score is calculated for all the final sets of features. A further hyper-parameter tuning was done using cross validation using xgbclassifier, as well as obtaining the best set of hyperparameters using random search cv. The best F1 score we obtained using hyperparameter tuning is 0.932. The train F1 score of 0.97 suggests that there exists a very very small amount of overfitting. However, that's ok. The difference is less than 0.05.

In order to improve the f1 score, we can add more graph based features and perform more aggresive hyperparameter tuning.