In [1]:

```
import os
from IPython.display import display, Image
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import colors
from matplotlib.ticker import PercentFormatter
from scipy.stats import linregress
import math
from functools import reduce
import matplotlib
import argparse
from Bio import SeqIO, Entrez, pairwise2
Entrez.email = 'hongyingsun1101@gmail.com'
from Bio.SeqRecord import SeqRecord
import re, time
import os, sys, glob
import random
import uuid
# from skbio.tree import TreeNode
# from skbio import read
# from skbio.stats.distance import DistanceMatrix
# from skbio.stats.distance import DissimilarityMatrix

from scipy import stats
from ast import literal_eval
import sqlite3
# roc curve and auc score
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
df = pd.read_csv("all_data.csv", index_col=0)
score= pd.read_csv("score_merged.csv", index_col=0)
```

In [3]:

```
reference ={'A':"RDP_10398", 'B':'RDP_5224', 'C':"RDP_1017", 'D':'RDP_92', 'E':'RDP_12'}
```

In [4]:

```
#pplacer_ref_list = ['A','B','C', 'D','E'],pplacer_stats_list=['_adcl_log','_edpl', '_pric
hness'],community_list=['0','1','2','3','4'],cutoff_list=['mean','min','25%','50%','75%']
```

In [5]:

```
def is_float(string):
  try:
    return float(string) and '.' in string  # True if string is a number contains a dot
  except ValueError:  # String is not a number
    return False
```

In [6]:

```
reference
```

Out[6]:

```
{'A': 'RDP_10398',
 'B': 'RDP_5224',
 'C': 'RDP_1017',
 'D': 'RDP_92',
 'E': 'RDP_12'}
```

In [7]:

```
columnList=list(df.columns)
```

In [8]:

```
communityList = df.community
```

In [9]:

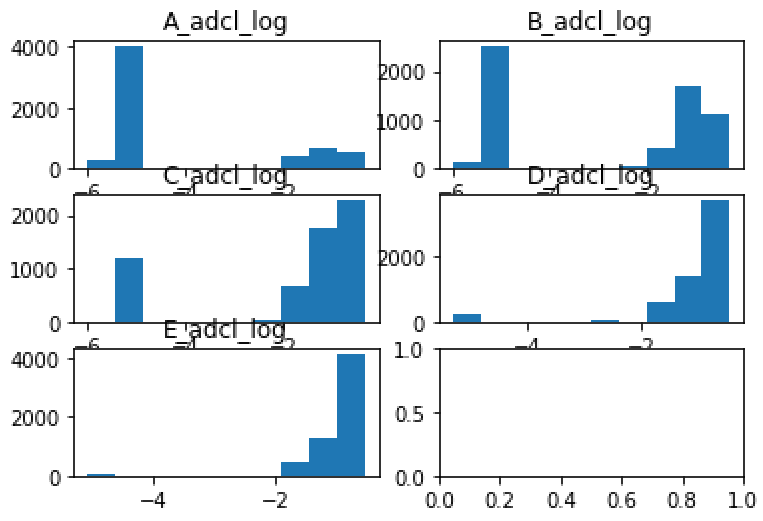
```
# df.describe()
```

In [10]:

```
def plot_pplacer(variable):
    fig, axes = plt.subplots(nrows=3, ncols=2)
    ax0, ax1, ax2, ax3, ax4, ax5 = axes.flatten()

    ax0.hist(df['A'+variable])
    ax0.set_title('A'+variable)

    ax1.hist(df['B'+variable])
    ax1.set_title('B'+variable)

    ax2.hist(df['C'+variable])
    ax2.set_title('C'+variable)

    ax3.hist(df['D'+variable])
    ax3.set_title('D'+variable)
    ax4.hist(df['E'+variable])
    ax4.set_title('E'+variable)
```

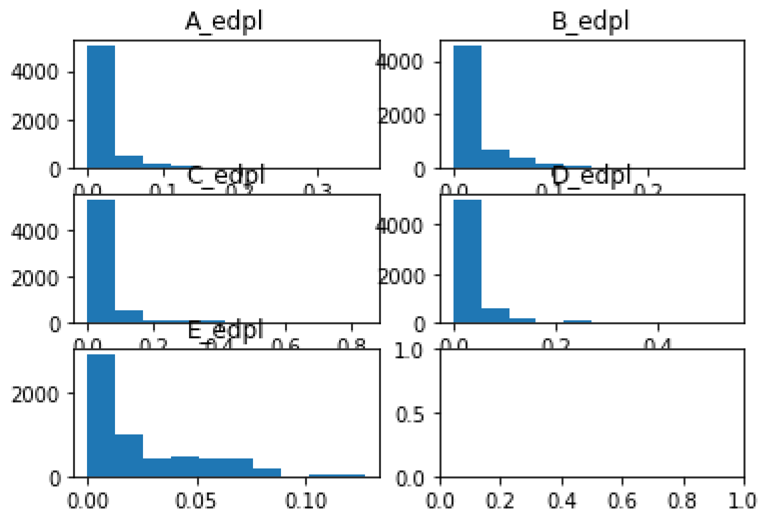
In [11]:

```
plot_pplacer('_adcl_log')
```



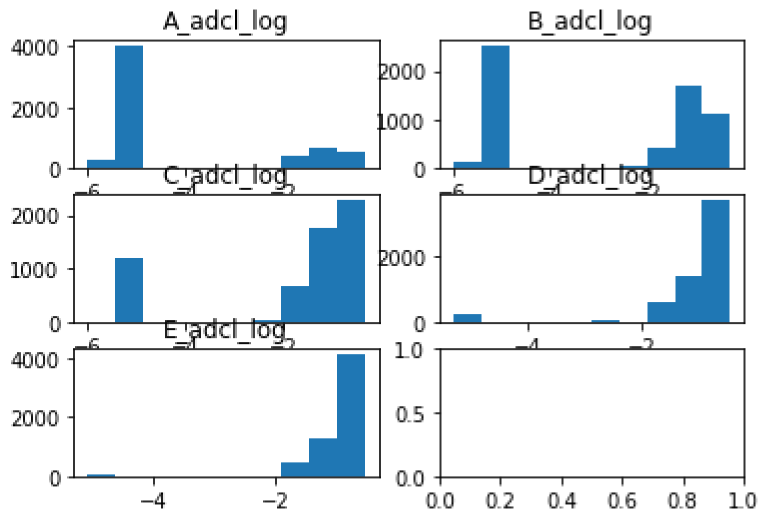
In [12]:

```
plot_pplacer('_edpl')
```



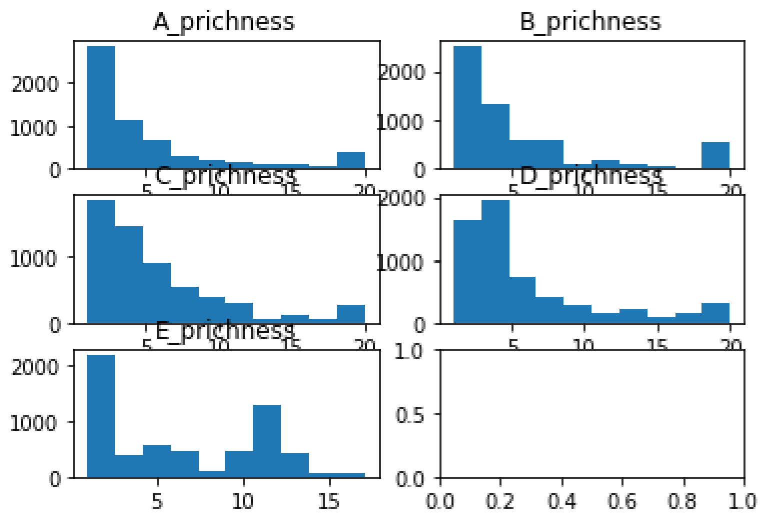
In [13]:

```
plot_pplacer('_adcl_log')
```



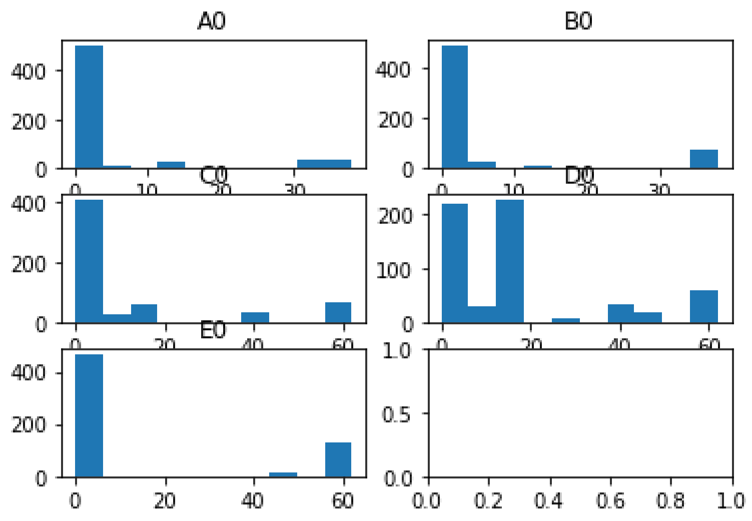
In [14]:

```
plot_pplacer('_prichness')
```



In [15]:

```
plot_pplacer('0');
```

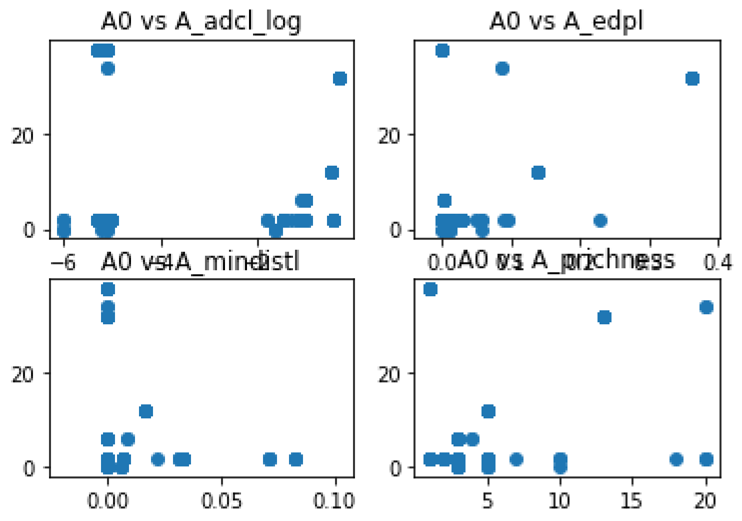


In [16]:

```
def plotScatter(reference,community):
    fig, axes = plt.subplots(nrows=2, ncols=2)
    ax0, ax1, ax2, ax3 = axes.flatten()

    ax0.scatter(df[reference+'_adcl_log'], df[reference+community])
    ax0.set_title(reference+community+' vs '+ reference + '_adcl_log')

    ax1.scatter(df[reference+'_edpl'], df[reference+community])
    ax1.set_title(reference+community+' vs '+ reference + '_edpl')

    ax2.scatter(df[reference+'_mindistl'], df[reference+community])
    ax2.set_title(reference+community+' vs '+ reference + '_mindistl')

    ax3.scatter(df[reference+'_prichness'], df[reference+community])
    ax3.set_title(reference+community+' vs '+ reference + '_prichness')
```

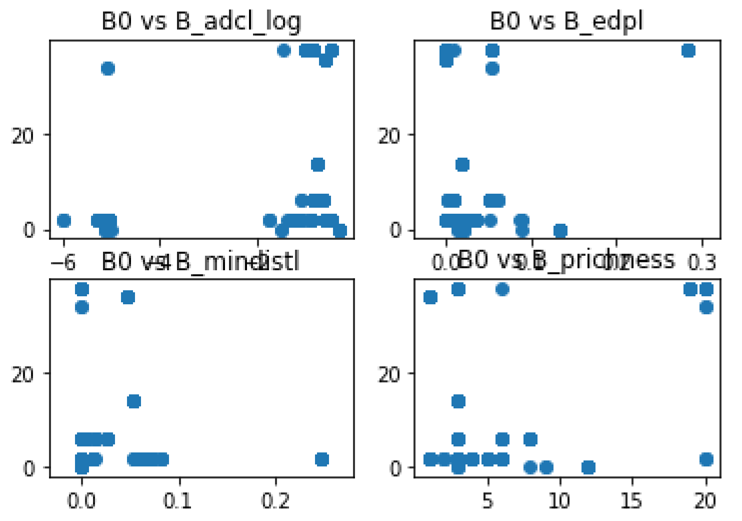
In [17]:

```
plotScatter('A','0')
```



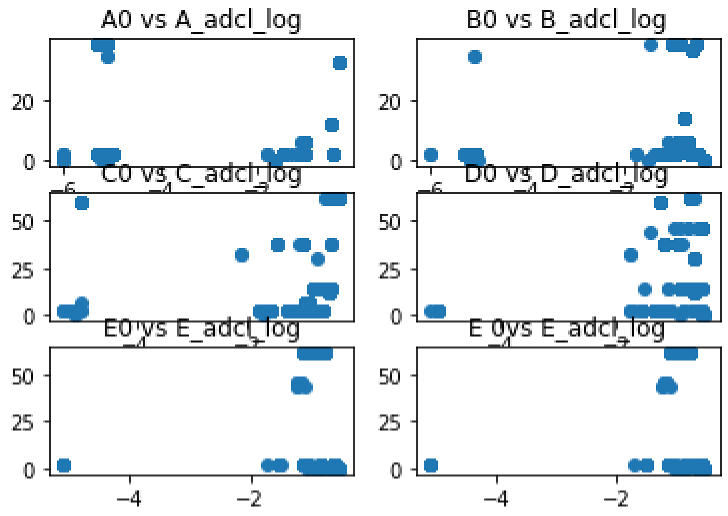
In [18]:

```
plotScatter('B','0')
```



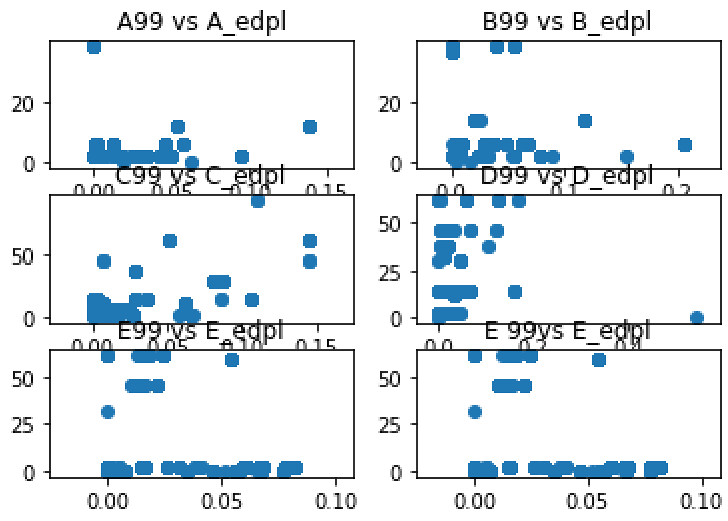
In [19]:

```
def plotScatterRef(variable,community):
    fig, axes = plt.subplots(nrows=3, ncols=2)
    ax0, ax1, ax2, ax3, ax4, ax5 = axes.flatten()

    ax0.scatter(df['A'+variable], df['A'+community])
    ax0.set_title('A' + community +' vs A' + variable)
    ax1.scatter(df['B'+variable], df['B'+community])
    ax1.set_title('B' + community +' vs B' + variable)

    ax2.scatter(df['C'+variable], df['C'+community])
    ax2.set_title('C' + community +' vs C' + variable)

    ax3.scatter(df['D'+variable], df['D'+community])
    ax3.set_title('D' + community +' vs D' + variable)

    ax4.scatter(df['E'+variable], df['E'+community])
    ax4.set_title('E' + community +' vs E' + variable)

    ax5.scatter(df['E'+variable], df['E'+community])
    ax5.set_title('E ' + community +'vs E' + variable)




plotScatterRef('_adcl_log','0');
```



In [20]:

```
plotScatterRef('_edpl','99');
```



In [21]:

```
cols=df.columns.tolist()
# cols[:20]
```

In [22]:

```
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

In [23]:

```
def plot_roc(data_X, class_label):
    trainX, testX, trainy, testy = train_test_split(data_X, class_label, test_size=0.3, ra
ndom_state=1)
    model = RandomForestClassifier()
    model.fit(trainX, trainy)
    probs = model.predict_proba(testX)
    probs = probs[:, 1]
    auc = roc_auc_score(testy, probs)
    fpr, tpr, thresholds = roc_curve(testy, probs)
    optimal_idx = np.argmax(tpr - fpr)
    optimal_threshold = thresholds[optimal_idx]
    print('optimal_threshold: %.2f' % optimal_threshold)
    print('AUC: %.2f' % auc)
    print( thresholds)
#     print( thresholds)
#     print('Model: ')
#     print(model)
    plot_roc_curve(fpr, tpr)
```

In [24]:

```
def makeTable(headerRow,columnizedData,columnSpacing=2):
    """Creates a technical paper style, left justified table"""
    from numpy import array,max,vectorize

    cols = array(columnizedData,dtype=str)
    colSizes = [max(vectorize(len)(col)) for col in cols]

    header = ''
    rows = ['' for i in cols[0]]

    for i in range(0,len(headerRow)):
        if len(headerRow[i]) > colSizes[i]: colSizes[i]=len(headerRow[i])
        headerRow[i]+=' '*(colSizes[i]-len(headerRow[i]))
        header+=headerRow[i]
        if not i == len(headerRow)-1: header+=' '*columnSpacing

        for j in range(0,len(cols[i])):
            if len(cols[i][j]) < colSizes[i]:
                cols[i][j]+=' '*(colSizes[i]-len(cols[i][j])+columnSpacing)
            rows[j]+=cols[i][j]
            if not i == len(headerRow)-1: rows[j]+=' '*columnSpacing

    line = '-'*len(header)
    print(line)
    print(header)
    print(line)
    for row in rows: print(row)
    print(line)
header = ['AUROC','Categoroy']
cutoffs = ['0.9-1.0','0.8-0.9','0.7-0.8','0.6-0.7','0.5-0.6']
evalualtion = ['Very good','Good','Fair', 'Poor', 'Fail']
makeTable(header,[cutoffs,evalualtion])
```

```
------------------
AUROC    Categoroy
------------------
0.9-1.0  Very good
0.8-0.9  Good
0.7-0.8  Fair
0.6-0.7  Poor
0.5-0.6  Fail
------------------
```

In [25]:

```
def plot_roc_microbiome(data_X, class_label, x, y, data_test=False):
    if(not data_test):
#         print("data_set is False")
        trainX, testX, trainy, testy = train_test_split(data_X, class_label, test_size=0.3
, random_state=1)
        model = RandomForestClassifier()
        model.fit(trainX, trainy)
        probs = model.predict_proba(testX)
        probs = probs[:, 1]
        auc = roc_auc_score(testy, probs)
        fpr, tpr, thresholds = roc_curve(testy, probs)
        optimal_idx = np.argmax(tpr - fpr)
        optimal_threshold = thresholds[optimal_idx]
        print('optimal_threshold: %.2f' % optimal_threshold)
        print('AUC: %.2f' % auc)
        print('thresholds: ' + thresholds)

        plot_roc_curve(fpr, tpr)

    else:
        print("data_set is True")
        trainX, testX, trainy, testy = train_test_split(data_X, class_label, test_size=0.3
, random_state=1)
        model = RandomForestClassifier()
        model.fit(trainX, trainy)
        probs1 = model.predict_proba(testX)
        probs1 = probs1[:, 1]
        auc1 = roc_auc_score(testy, probs1)
        fpr1, tpr1, thresholds1 = roc_curve(testy, probs1)
        optimal_idx1 = np.argmax(tpr1 - fpr1)
        optimal_threshold1 = thresholds1[optimal_idx1]
        print('AUC1: %.2f' % auc1)
        print('optimal_threshold1: %.2f' % optimal_threshold1)
        print(thresholds1)
        plot_roc_curve(fpr1, tpr1)

        probs2 = model.predict_proba(x)
        probs2 = probs2[:, 1]
        auc2 = roc_auc_score(y, probs2)
        fpr2, tpr2, thresholds2 = roc_curve(y, probs2)
        optimal_idx2 = np.argmax(tpr2 - fpr2)
        optimal_threshold2 = thresholds2[optimal_idx2]
        print('AUC2: %.2f' % auc2)
        print('optimal_threshold2: %.2f' % optimal_threshold2)
        print( thresholds2)
        plot_roc_curve(fpr2, tpr2)
```

In [26]:

```
def plot_roc_curve_microbiome(pplacer_ref_list, pplacer_stats_list, community_list, cutoff
_list, scoreOption=True):
    for (refIndex,pplacer_ref) in enumerate(pplacer_ref_list):
#     for refIndex in range(len(pplacer_ref_list)):
#         pplacer_ref = pplacer_ref_list[refIndex]
        for (statsIndex,pplacer_stats) in enumerate(pplacer_stats_list):
#         for statsIndex in range(len(pplacer_stats_list)):
#             pplacer_stats = pplacer_stats_list[statsIndex]
            for (communityIndex,community) in enumerate(community_list):
#             for communityIndex in range(len(community_list)):
#                 community = community_list[communityIndex]
                for (i, cutoff) in enumerate(cutoff_list):
#                 for i in range(len(cutoff_list)):
#                     cutoff=cutoff_list[i]
                    if(is_float(cutoff)):
                        cutoff_binary=float(cutoff)
                    else:
                        if(scoreOption):
                            cutoff_binary=float(df[pplacer_ref+community].describe().loc[[
cutoff]])

                        else:
                            cutoff_binary = float(df[pplacer_ref+pplacer_stats].describe()
.loc[[cutoff]])
                    if(scoreOption):
                        mask = df[pplacer_ref+community] <=  cutoff_binary
                        df.loc[mask, pplacer_ref+community+'_binary'] = 1
                        mask = df[pplacer_ref+community] >cutoff_binary
                        df.loc[mask, pplacer_ref+community+'_binary'] = 0
                        df_binary = df[[pplacer_ref+pplacer_stats, pplacer_ref+community+
'_binary']].dropna()
                        data_stats = df_binary[pplacer_ref+pplacer_stats].to_numpy().resha
pe(-1,1)
                        binary_label =  df_binary[pplacer_ref+community+'_binary'].to_nump
y()
                        print(' The score cutoff '+ cutoff +' for Reference ' + pplacer_re
f +' community ' + community   + ' with pplacer_stats '+ pplacer_stats[1:] + ': %.2f' % cu
toff_binary )
                        plot_roc(data_stats,binary_label)
                    else:
                        mask = df[pplacer_ref+pplacer_stats] <=  cutoff_binary
                        df.loc[mask, pplacer_ref+pplacer_stats+'_binary'] = 1
                        mask = df[pplacer_ref+pplacer_stats] >cutoff_binary
                        df.loc[mask, pplacer_ref+pplacer_stats+'_binary'] = 0
                        df_binary = df[[pplacer_ref+community, pplacer_ref+pplacer_stats+
'_binary']].dropna()
                        data_stats = df_binary[pplacer_ref+community].to_numpy().reshape(-
1,1)
                        binary_label =  df_binary[pplacer_ref+pplacer_stats+'_binary'].to_
numpy()
                        print(' The pplacer_stats_cutoff '+ cutoff +' for Reference ' + pp
lacer_ref +' community ' + community + ' pplacer_stats '  + pplacer_stats[1:]  + ': %.2f'
% cutoff_binary )
                        plot_roc(data_stats,binary_label)
```

different reference same pplacer stats same community to test different cutoffs and different references for score

In [27]:

```
# plot_roc_curve_microbiome(pplacer_ref_list = ['A','B','C','D','E'],pplacer_stats_list=
['_adcl_log'],community_list=['A'],cutoff_list=['mean', 'min','25%','50%','75%'],scoreOpti
on=False)
```

In [28]:

```
df['E0'].describe()
```

Out[28]:

```
count    605.000000
mean      14.601653
std       25.354082
min        0.000000
25%        0.000000
50%        2.000000
75%        2.000000
max       62.000000
Name: E0, dtype: float64
```

Different reference same pplacer stats same community to test different cutoffs and different references for adcl_log

Fitting on large reference and test on small reference datasets

In [29]:

```
def plot_roc_curve_microbiome_test2(pplacer_ref_list, pplacer_stats_list, community_list,
cutoff_list, test_data_list, scoreOption=True, testOption=False):
    for refIndex in range(len(pplacer_ref_list)):
        pplacer_ref = pplacer_ref_list[refIndex]
        for statsIndex in range(len(pplacer_stats_list)):
            pplacer_stats = pplacer_stats_list[statsIndex]
            for communityIndex in range(len(community_list)):
                community = community_list[communityIndex]
                for i in range(len(cutoff_list)):
                    cutoff=cutoff_list[i]
                    if(is_float(cutoff)):
                        cutoff_binary=float(cutoff)
                    else:
                        if(scoreOption):
                            cutoff_binary=float(df[pplacer_ref+community].describe().loc[[
cutoff]])

                        else:
                            cutoff_binary = float(df[pplacer_ref+pplacer_stats].describe()
.loc[[cutoff]])
                    # no test situation, which is the default option
                    if (not testOption):

                        if(scoreOption):
                            mask = df[pplacer_ref+community] <=  cutoff_binary
                            df.loc[mask, pplacer_ref+community+'_binary'] = 1
                            mask = df[pplacer_ref+community] >cutoff_binary
                            df.loc[mask, pplacer_ref+community+'_binary'] = 0
                            df_binary = df[[pplacer_ref+pplacer_stats, pplacer_ref+communi
ty+'_binary']].dropna()
                            data_stats = df_binary[pplacer_ref+pplacer_stats].to_numpy().r
eshape(-1,1)
                            binary_label =  df_binary[pplacer_ref+community+'_binary'].to_
numpy()
                            print(' The score cutoff '+ cutoff +' for Reference ' + pplace
r_ref +' community ' + community   + ' with pplacer_stats '+ pplacer_stats[1:] + ': %.2f'
% cutoff_binary )
                            # plot_roc(data_stats,binary_label)
                            plot_roc_microbiome(data_stats,binary_label,x=None,y=None,data
_test=False)
                        else:
                            mask = df[pplacer_ref+pplacer_stats] <=  cutoff_binary
                            df.loc[mask, pplacer_ref+pplacer_stats+'_binary'] = 1
                            mask = df[pplacer_ref+pplacer_stats] >cutoff_binary
                            df.loc[mask, pplacer_ref+pplacer_stats+'_binary'] = 0
                            df_binary = df[[pplacer_ref+community, pplacer_ref+pplacer_sta
ts+'_binary']].dropna()
                            data_stats = df_binary[pplacer_ref+community].to_numpy().resha
pe(-1,1)
                            binary_label =  df_binary[pplacer_ref+pplacer_stats+'_binary']
.to_numpy()
                            print(' The pplacer_stats_cutoff '+ cutoff +' for Reference '
+ pplacer_ref +' community ' + community + ' pplacer_stats '  + pplacer_stats[1:]  + ': %.
2f' % cutoff_binary )
                            # plot_roc(data_stats,binary_label)
                            plot_roc_microbiome(data_stats,binary_label,x=None,y=None,data
_test=False)

                    # if there is test
                    else:
                        for j in range(len(test_data_list)):
                            test=test_data_list[j]
                            if(scoreOption):
                                mask = df[pplacer_ref+community] <=  cutoff_binary
                                df.loc[mask, pplacer_ref+community+'_binary'] = 1
                                mask = df[pplacer_ref+community] >cutoff_binary
                                df.loc[mask, pplacer_ref+community+'_binary'] = 0
                                df_binary = df[[pplacer_ref+pplacer_stats, pplacer_ref+com
munity+'_binary']].dropna()
                                data_stats = df_binary[pplacer_ref+pplacer_stats].to_numpy
().reshape(-1,1)
                                binary_label =  df_binary[pplacer_ref+community+'_binary']
.to_numpy()

                                mask_test = df[test+community] <=  cutoff_binary
                                df.loc[mask_test, test+community+'_binary'] = 1
                                mask_test = df[test+community] >cutoff_binary
                                df.loc[mask_test, test+community+'_binary'] = 0
                                df_binary = df[[test+pplacer_stats, test+community+'_binar
y']].dropna()
                                x = df_binary[test+pplacer_stats].to_numpy().reshape(-1,1)
                                y =  df_binary[test+community+'_binary'].to_numpy()

                                print(' The score cutoff '+ cutoff +' for Reference ' + pp
lacer_ref +' community ' + community   + ' with pplacer_stats '+ pplacer_stats[1:] + ' com
pared with test ' + test +': %.2f' % cutoff_binary )
                                plot_roc_microbiome(data_stats,binary_label,x,y,data_test=
True)
                            else:

                                mask = df[pplacer_ref+pplacer_stats] <=  cutoff_binary
                                df.loc[mask, pplacer_ref+pplacer_stats+'_binary'] = 1
                                mask = df[pplacer_ref+pplacer_stats] >cutoff_binary
                                df.loc[mask, pplacer_ref+pplacer_stats+'_binary'] = 0
                                df_binary = df[[pplacer_ref+community, pplacer_ref+pplacer
_stats+'_binary']].dropna()
                                data_stats = df_binary[pplacer_ref+community].to_numpy().r
eshape(-1,1)
                                binary_label =  df_binary[pplacer_ref+pplacer_stats+'_bina
ry'].to_numpy()

                                mask_test = df[test+pplacer_stats] <=  cutoff_binary
                                df.loc[mask_test, test+pplacer_stats+'_binary'] = 1
                                mask_test = df[test+pplacer_stats] >cutoff_binary
                                df.loc[mask_test, test+pplacer_stats+'_binary'] = 0
                                df_binary = df[[test+community, test+pplacer_stats+'_binar
y']].dropna()
                                x = df_binary[test+community].to_numpy().reshape(-1,1)
                                y =  df_binary[test+pplacer_stats+'_binary'].to_numpy()

                                print(' The pplacer_stats_cutoff '+ cutoff +' for Referenc
e ' + pplacer_ref +' community ' + community + ' pplacer_stats '  + pplacer_stats[1:] + '
```

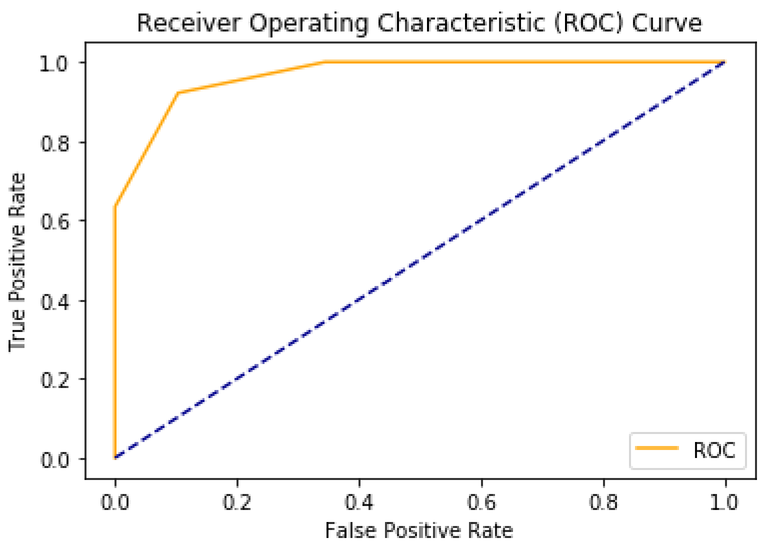
```
 compared with test ' + test  + ': %.2f' % cutoff_binary )
                                plot_roc_microbiome(data_stats,binary_label,x,y,data_test=
True)
```

Model from larger reference sets to fit data used small reference set. Could be worse on both directions

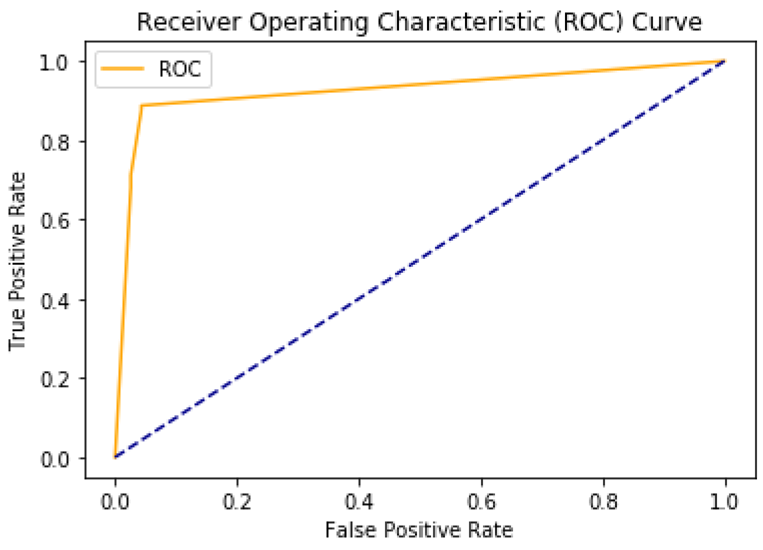
In [30]:

```
plot_roc_curve_microbiome_test2(pplacer_ref_list = ['A'],pplacer_stats_list=['_adcl_log',
'_edpl','_prichness','_mindistl'],community_list=['0'],cutoff_list=['2.00'], test_data_lis
t=['B','C','D','E'],testOption=True, scoreOption=True)
```

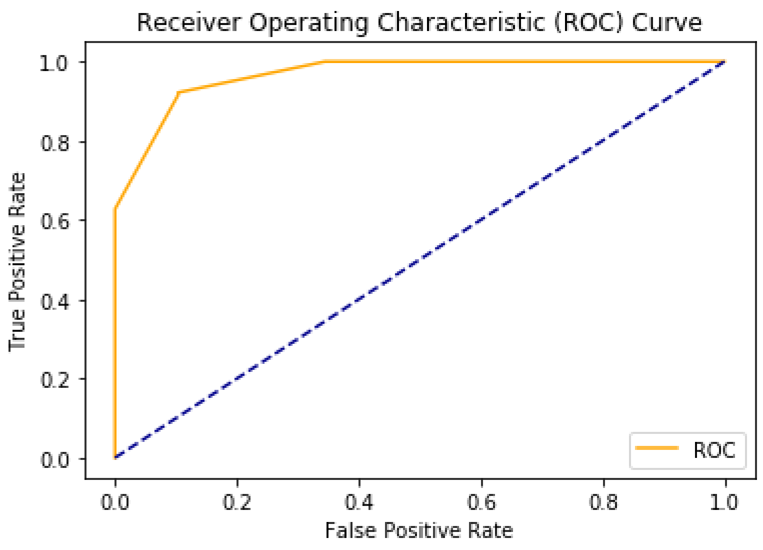
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats adcl_lo
g compared with test B: 2.00
data_set is True
AUC1: 0.97
optimal_threshold1: 0.87
[2.         1.         0.9        0.86736831 0.62544164 0.3
 0.        ]
```



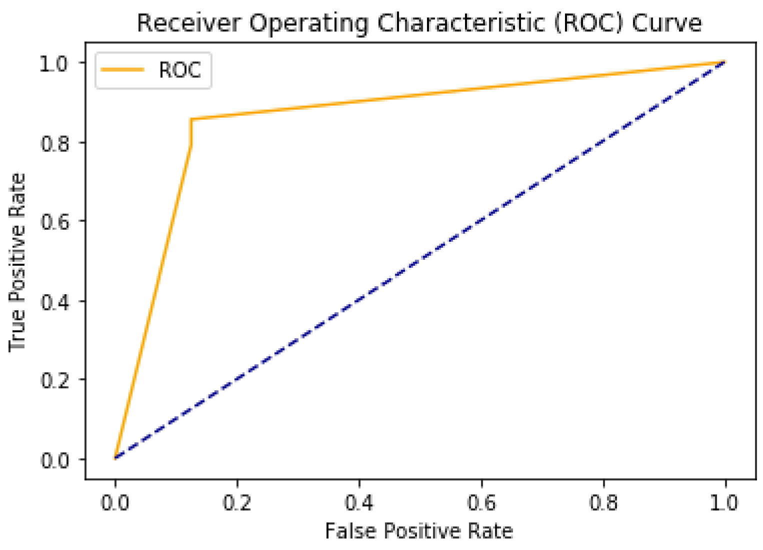
```
AUC2: 0.93
optimal_threshold2: 0.63
[2.         1.         0.9        0.86736831 0.62544164 0.        ]
```



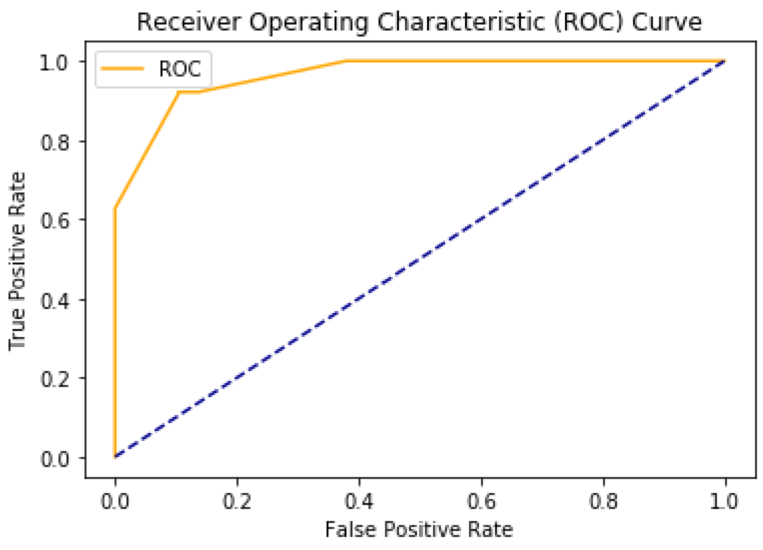
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats adcl_lo
g compared with test C: 2.00
data_set is True
AUC1: 0.97
optimal_threshold1: 0.80
[2.         1.         0.87278263 0.8        0.62028391 0.4
 0.        ]
```



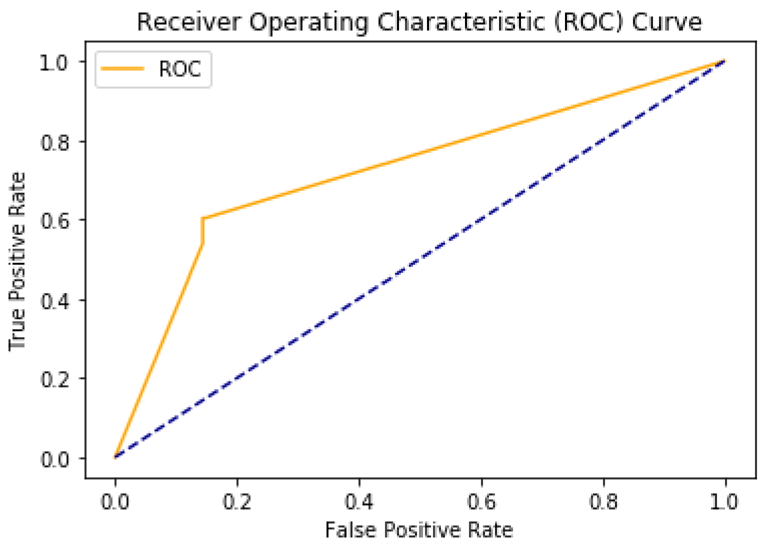
```
AUC2: 0.86
optimal_threshold2: 0.62
[2.         1.         0.87278263 0.62028391 0.4        0.        ]
```



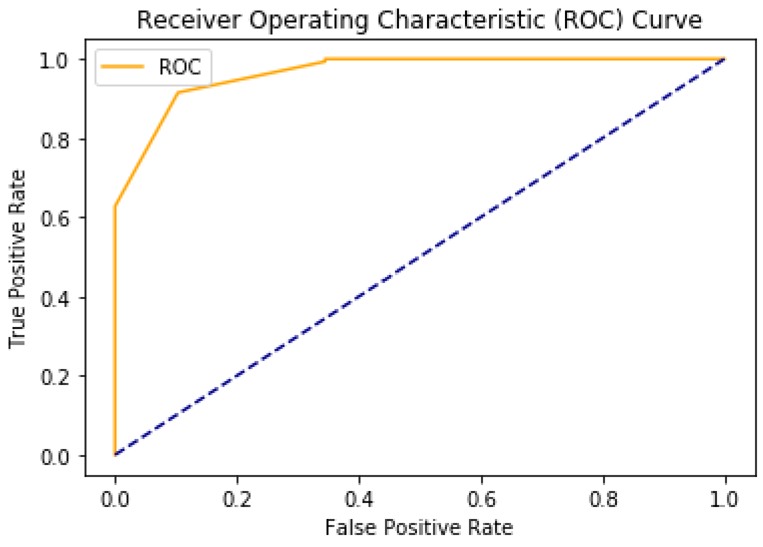
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats adcl_lo
g compared with test D: 2.00
data_set is True
AUC1: 0.96
optimal_threshold1: 0.70
[2.         1.         0.84484447 0.7        0.6        0.57612807
 0.        ]
```



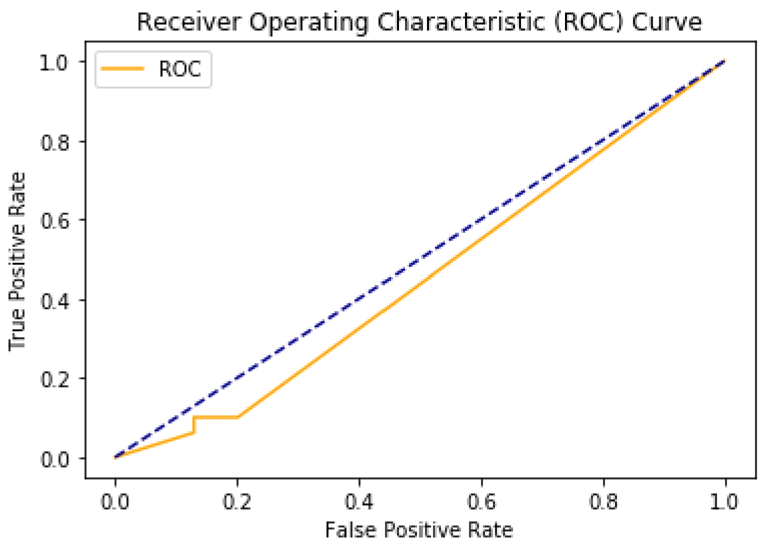
```
AUC2: 0.72
optimal_threshold2: 0.70
[2.         1.         0.84484447 0.7        0.6        0.        ]
```



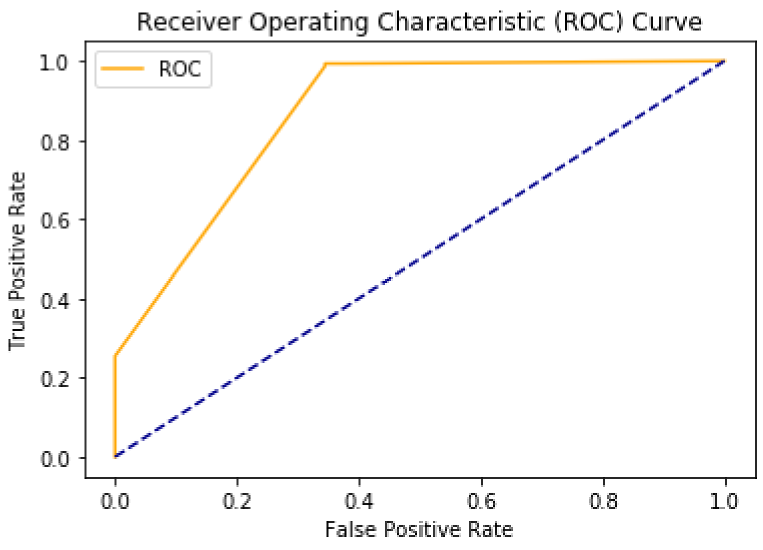
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats adcl_lo
g compared with test E: 2.00
data_set is True
AUC1: 0.97
optimal_threshold1: 0.85
[2.         1.         0.84685514 0.60397569 0.6        0.3
 0.        ]
```



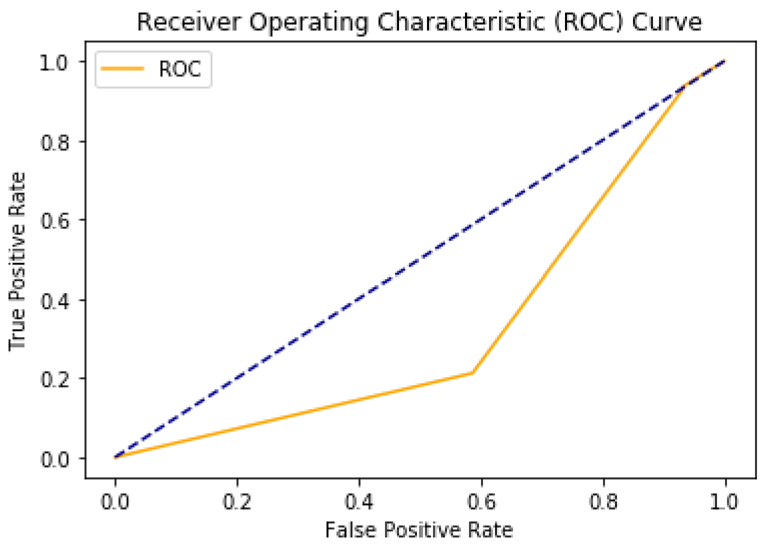
```
AUC2: 0.45
optimal_threshold2: 2.00
[2.         1.         0.84685514 0.6        0.3        0.         ]
```



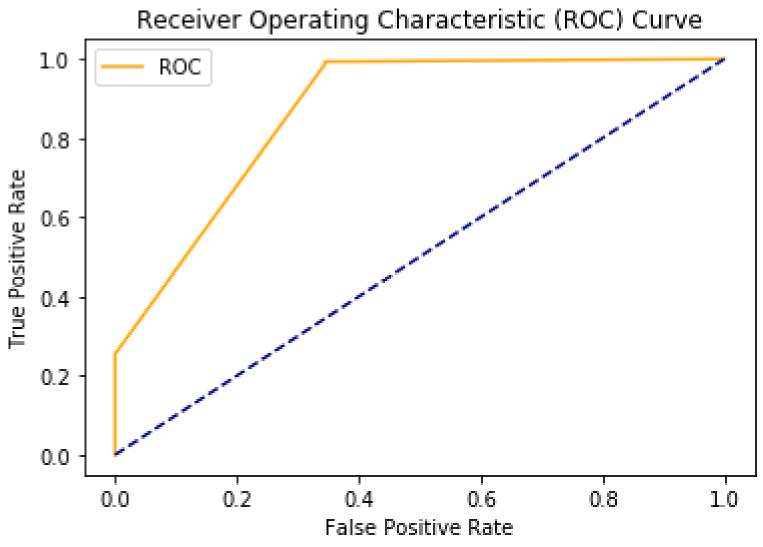
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats edpl co
mpared with test B: 2.00
data_set is True
AUC1: 0.87
optimal_threshold1: 0.80
[2.         1.         0.9        0.89528812 0.8        0.4
 0.        ]
```



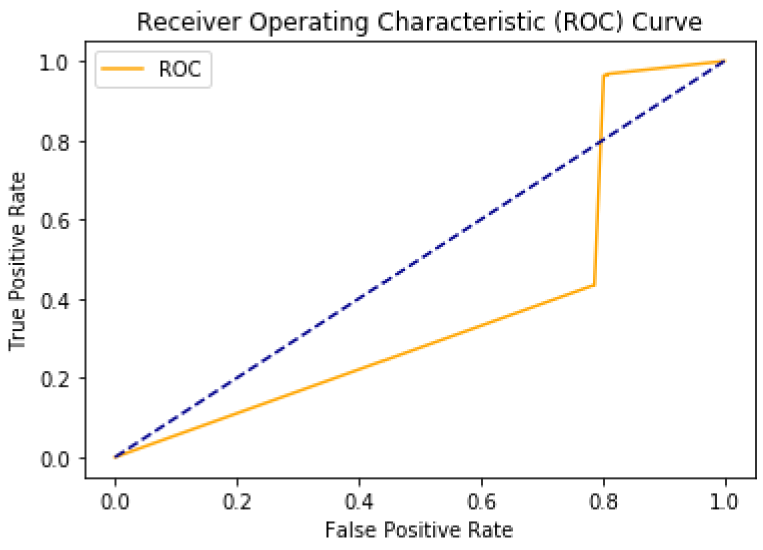
```
AUC2: 0.33
optimal_threshold2: 0.20
[2.         1.         0.89528812 0.2        0.        ]
```



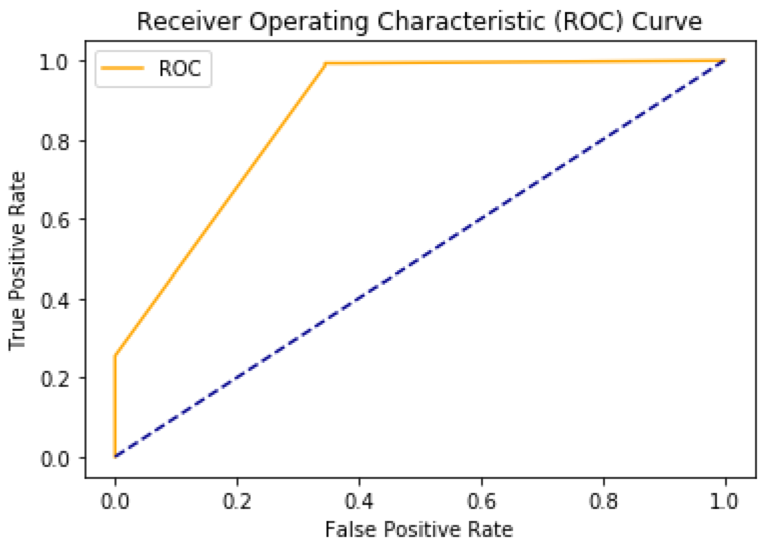
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats edpl co
mpared with test C: 2.00
data_set is True
AUC1: 0.87
optimal_threshold1: 0.60
[2.          1.          0.9         0.89523861 0.6        0.5
 0.        ]
```



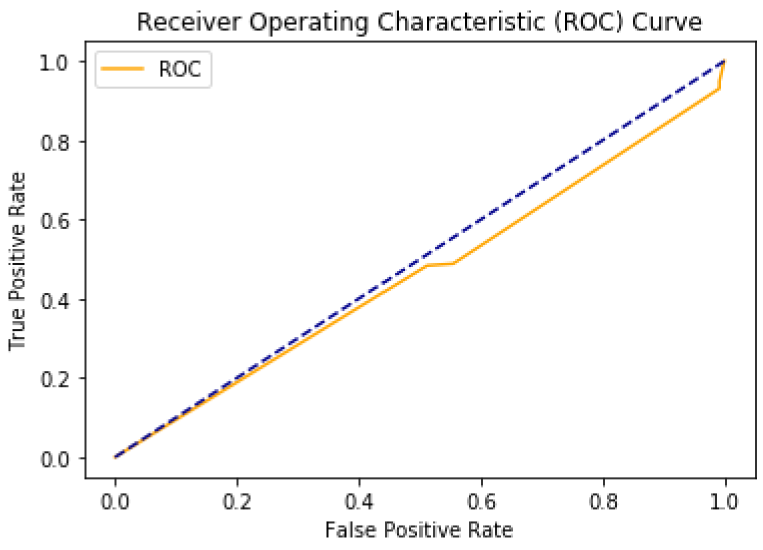
```
AUC2: 0.38
optimal_threshold2: 0.90
[2.          1.          0.9         0.89523861 0.8        0.6
 0.        ]
```



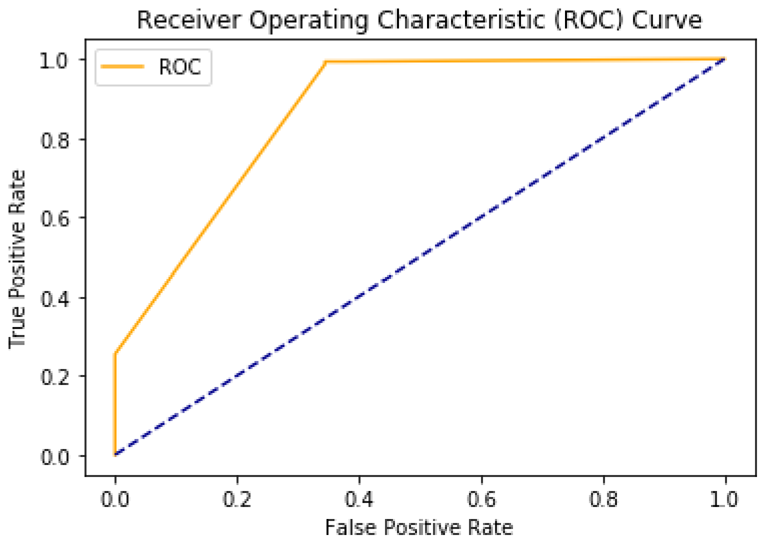
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats edpl co
mpared with test D: 2.00
data_set is True
AUC1: 0.87
optimal_threshold1: 0.60
[2.         1.         0.9        0.89400043 0.6        0.3
 0.        ]
```



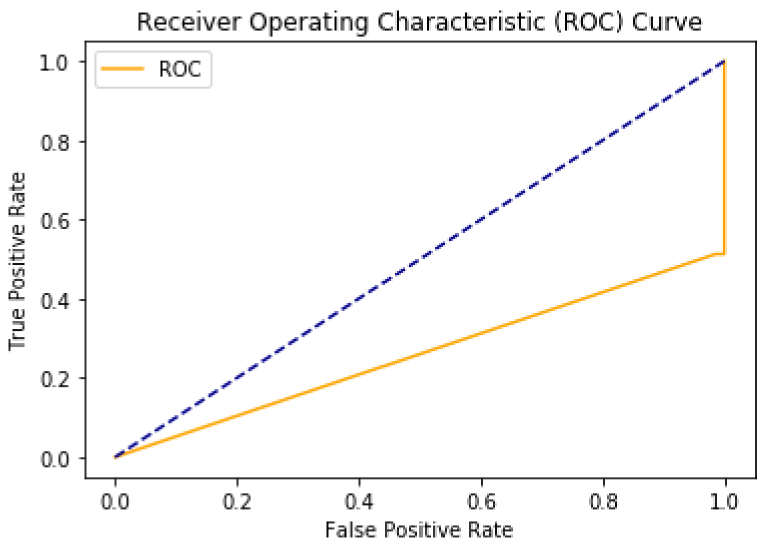
```
AUC2: 0.46
optimal_threshold2: 2.00
[2.         1.         0.9        0.89400043 0.3        0.        ]
```



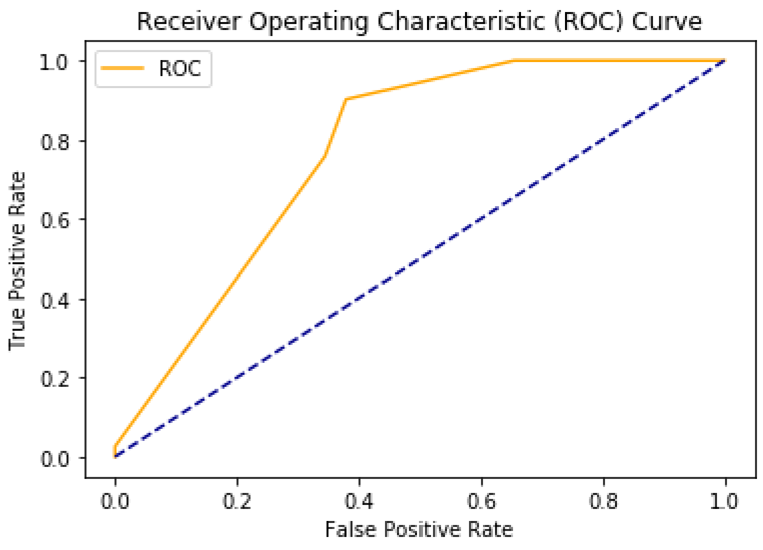
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats edpl co
mpared with test E: 2.00
data_set is True
AUC1: 0.87
optimal_threshold1: 0.50
[2.         1.         0.89581838 0.5        0.2        0.        ]
```



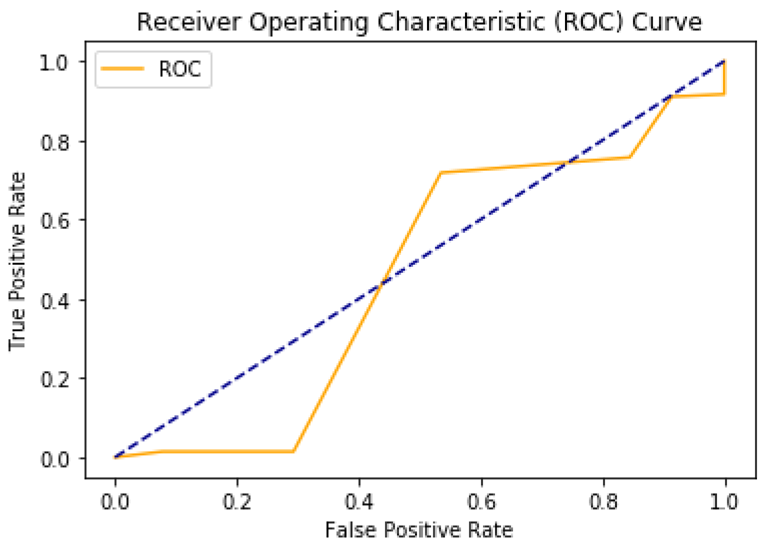
```
AUC2: 0.26
optimal_threshold2: 2.00
[2.         1.         0.9        0.89581838 0.2        0.1
 0.        ]
```



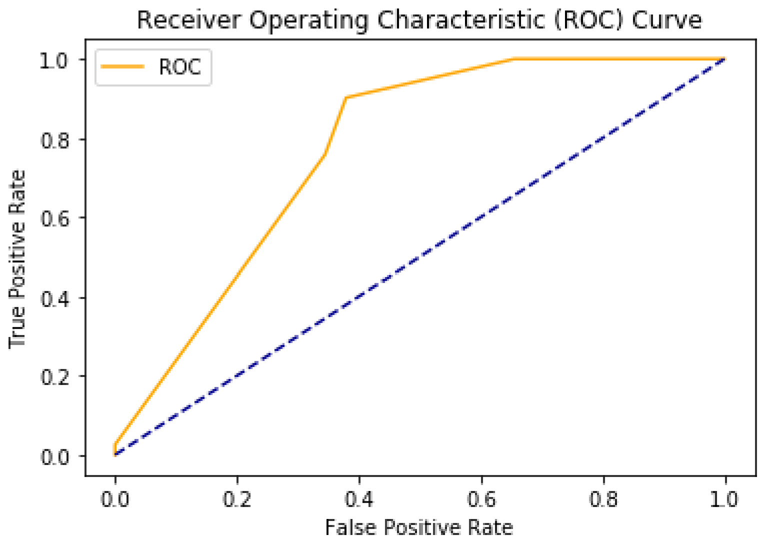
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats prichne
ss compared with test B: 2.00
data_set is True
AUC1: 0.77
optimal_threshold1: 0.86
[2.         1.         0.89550854 0.85524673 0.7327125  0.34720664
 0.        ]
```



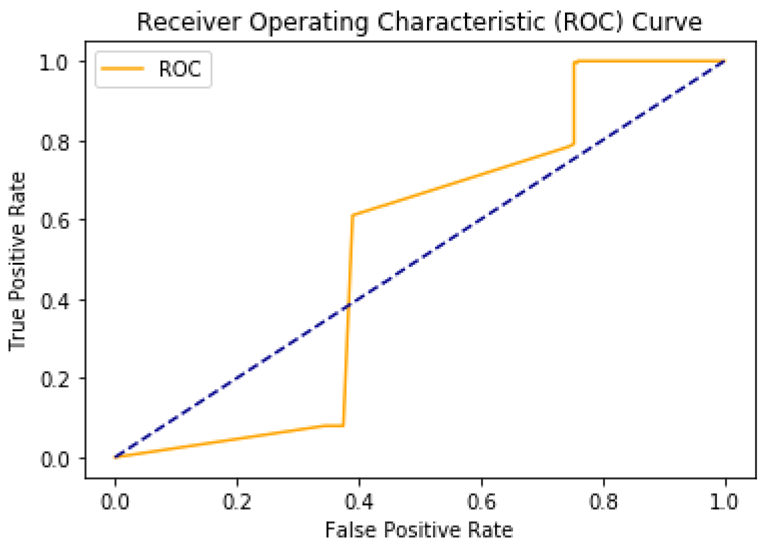
```
AUC2: 0.46
optimal_threshold2: 0.90
[2.         1.         0.96       0.89550854 0.85524673 0.7327125
 0.52583333 0.34720664 0.        ]
```



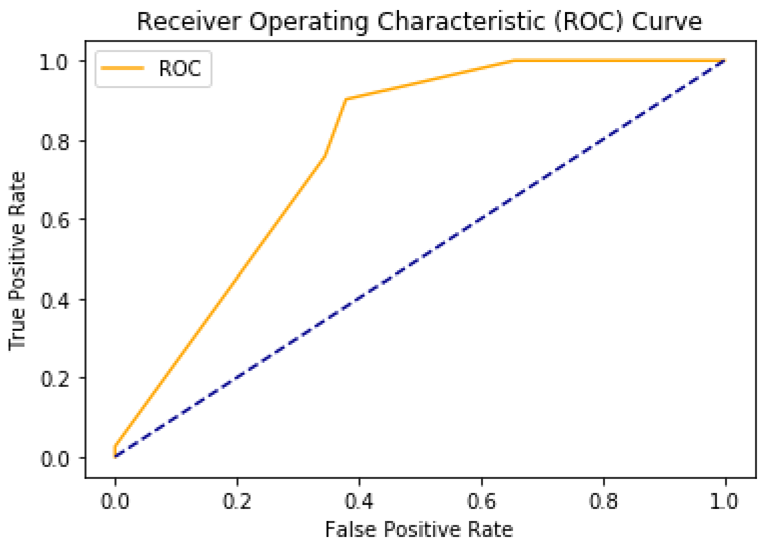
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats prichne
ss compared with test C: 2.00
data_set is True
AUC1: 0.77
optimal_threshold1: 0.83
[2.         1.         0.8935953  0.82751093 0.7407789  0.5793377
 0.        ]
```



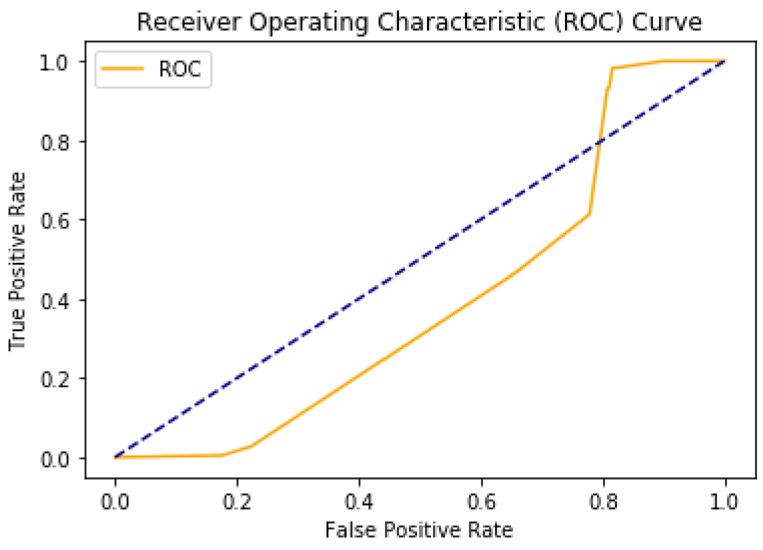
```
AUC2: 0.52
optimal_threshold2: 0.74
[2.         1.         0.90579373 0.8935953  0.82751093 0.79
 0.7407789  0.7        0.5793377  0.4        0.        ]
```



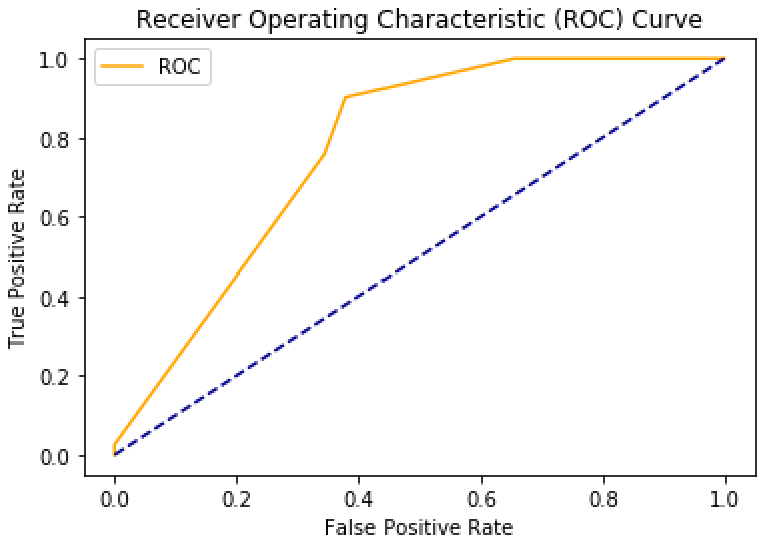
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats prichne
ss compared with test D: 2.00
data_set is True
AUC1: 0.77
optimal_threshold1: 0.84
[2.         1.         0.89510428 0.83632515 0.78098906 0.49408046
 0.        ]
```



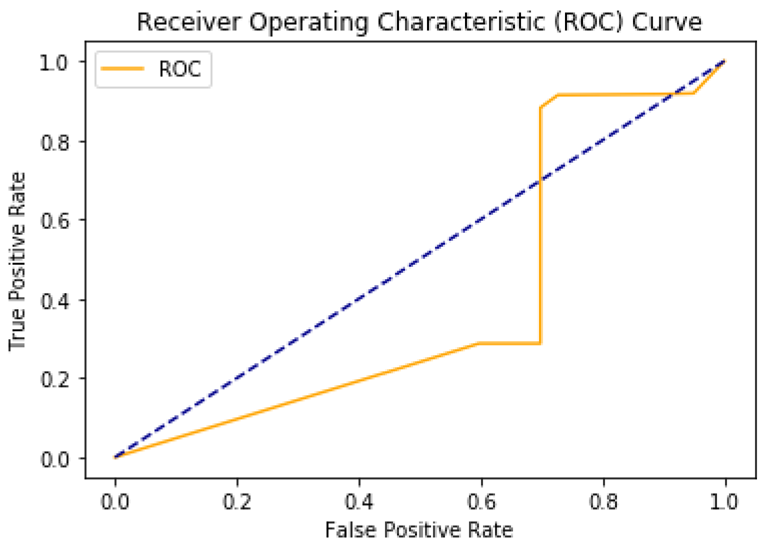
```
AUC2: 0.39
optimal_threshold2: 0.56
[2.         1.         0.91135512 0.89510428 0.83632515 0.78098906
 0.6        0.55916667 0.49408046 0.        ]
```



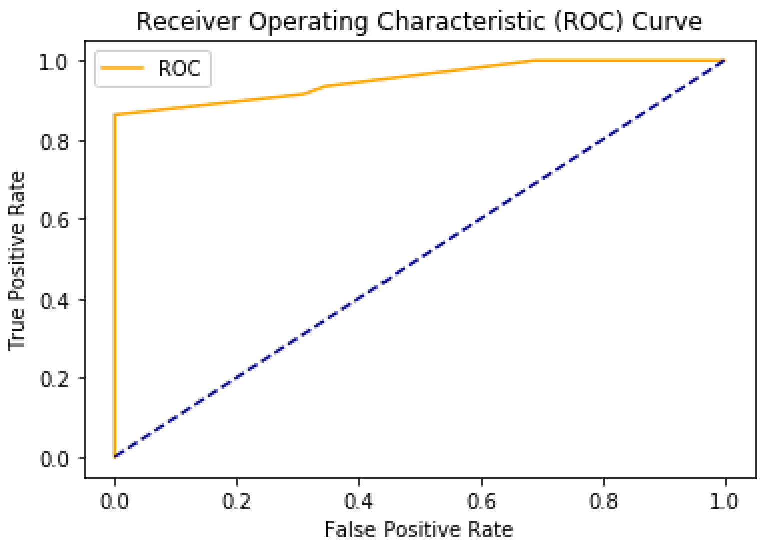
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats prichne
ss compared with test E: 2.00
data_set is True
AUC1: 0.77
optimal_threshold1: 0.86
[2.         1.         0.89748917 0.86226382 0.7512946  0.26714635
 0.        ]
```



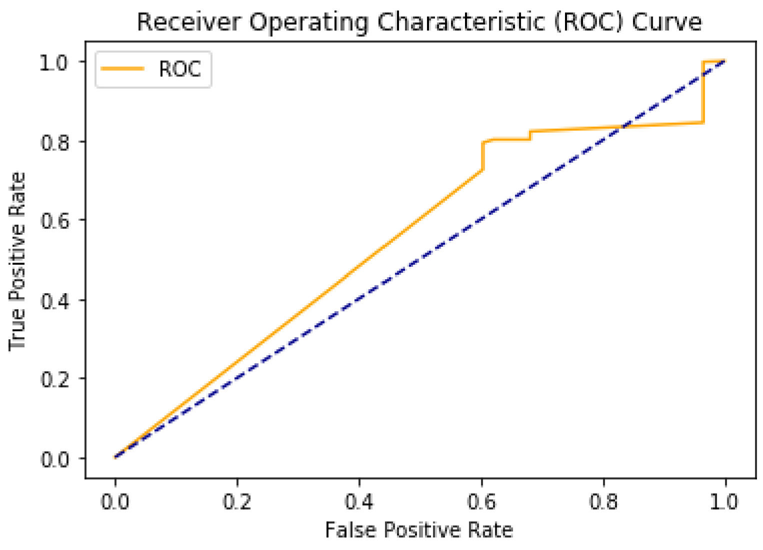
```
AUC2: 0.39
optimal_threshold2: 0.86
[2.         1.         0.91143239 0.9        0.89748917 0.86226382
 0.7512946  0.26714635 0.        ]
```



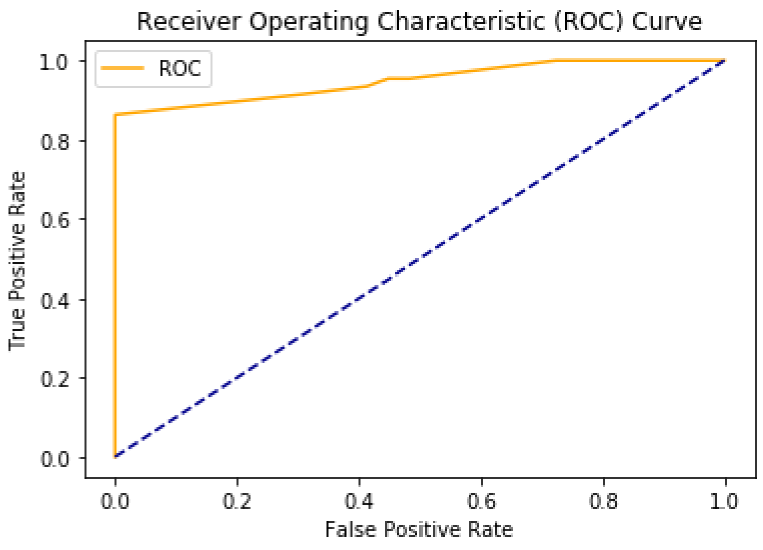
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats mindist
l compared with test B: 2.00
data_set is True
AUC1: 0.95
optimal_threshold1: 0.61
[2.         1.         0.80222763 0.60913004 0.5200068  0.45413004
 0.42523488 0.20420889 0.2        0.        ]
```



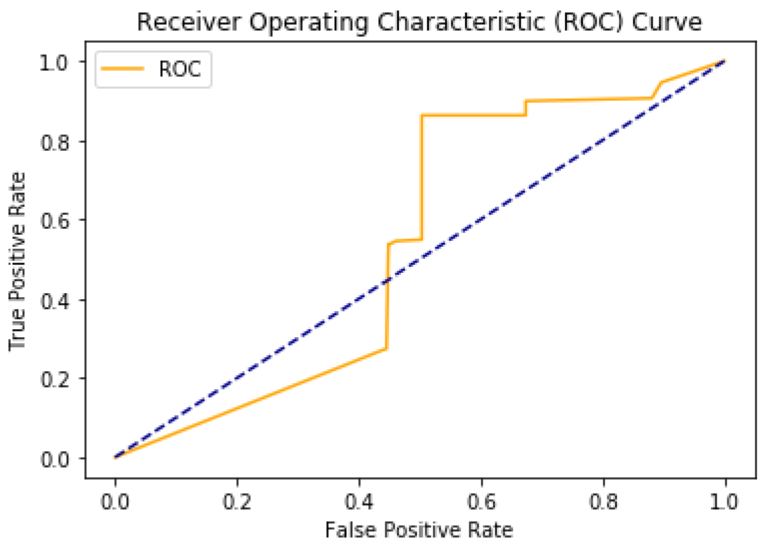
```
AUC2: 0.55
optimal_threshold2: 0.85
[2.         1.         0.85       0.80222763 0.6706685  0.60913004
 0.5200068  0.49389569 0.42523488 0.2332655  0.20420889 0.2
 0.        ]
```



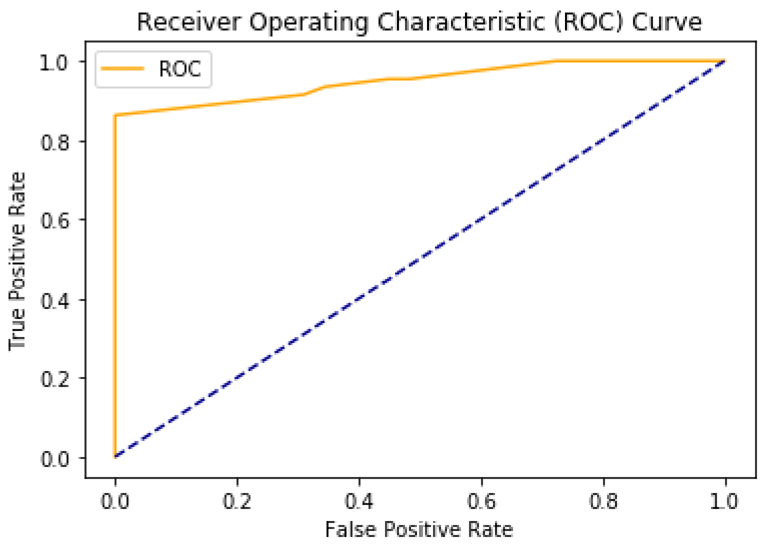
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats mindist
l compared with test C: 2.00
data_set is True
AUC1: 0.95
optimal_threshold1: 0.68
[2.         1.         0.76499695 0.67653954 0.55932033 0.50726323
 0.50420442 0.4        0.21885908 0.        ]
```



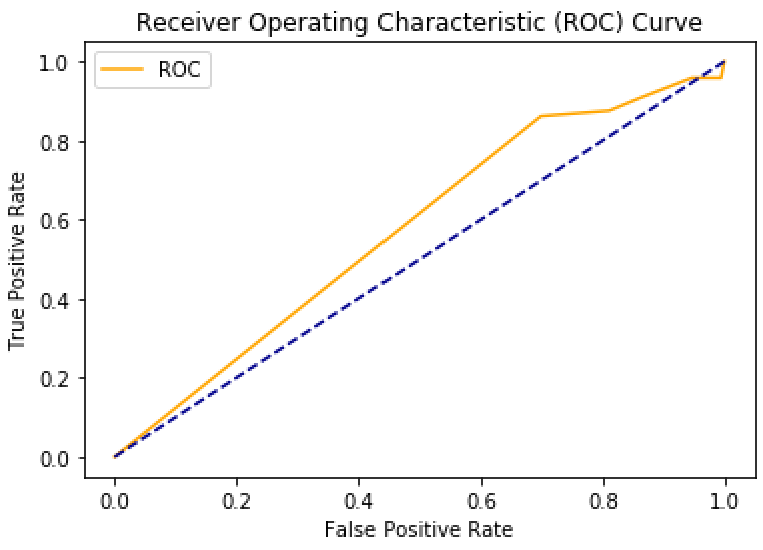
```
AUC2: 0.54
optimal_threshold2: 0.60
[2.         1.         0.86170569 0.8        0.76499695 0.67653954
 0.6        0.55932033 0.50726323 0.4        0.21885908 0.        ]
```



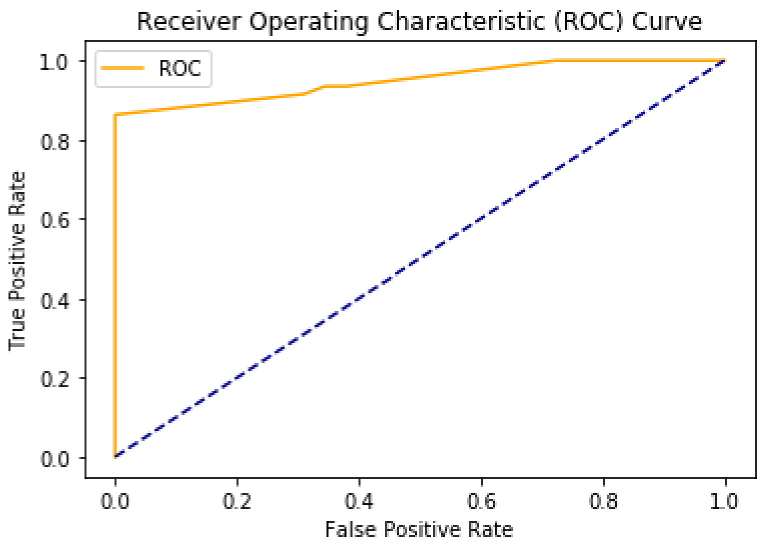
```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats mindist
l compared with test D: 2.00
data_set is True
AUC1: 0.95
optimal_threshold1: 0.77
[2.         1.         0.78809524 0.7676713  0.51811533 0.44875189
 0.44670924 0.4        0.19152626 0.        ]
```



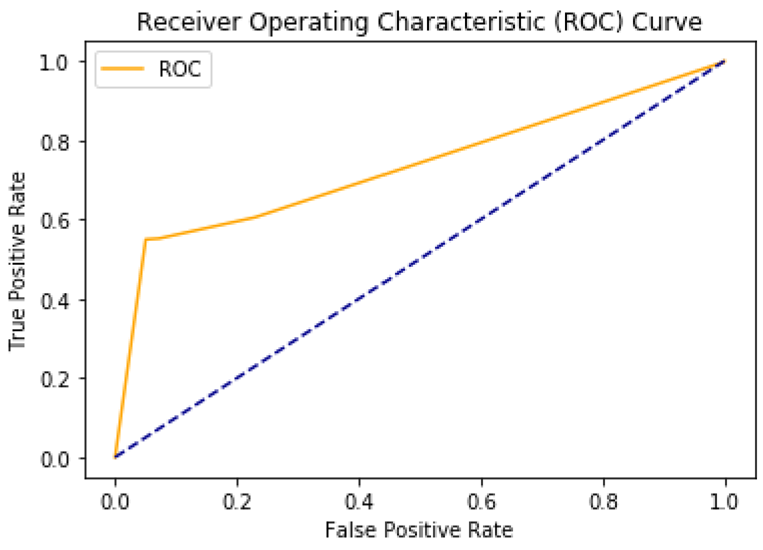
```
AUC2: 0.57
optimal_threshold2: 1.00
[2.         1.         0.6        0.44875189 0.4        0.19152626
 0.        ]
```



```
 The score cutoff 2.00 for Reference A community 0 with pplacer_stats mindist
l compared with test E: 2.00
data_set is True
AUC1: 0.95
optimal_threshold1: 0.66
[2.         1.         0.78656593 0.65941742 0.52136951 0.51950145
 0.5        0.43225864 0.24644268 0.        ]
```



```
AUC2: 0.73
optimal_threshold2: 1.00
[2.         1.         0.7        0.52136951 0.24644268 0.        ]
```

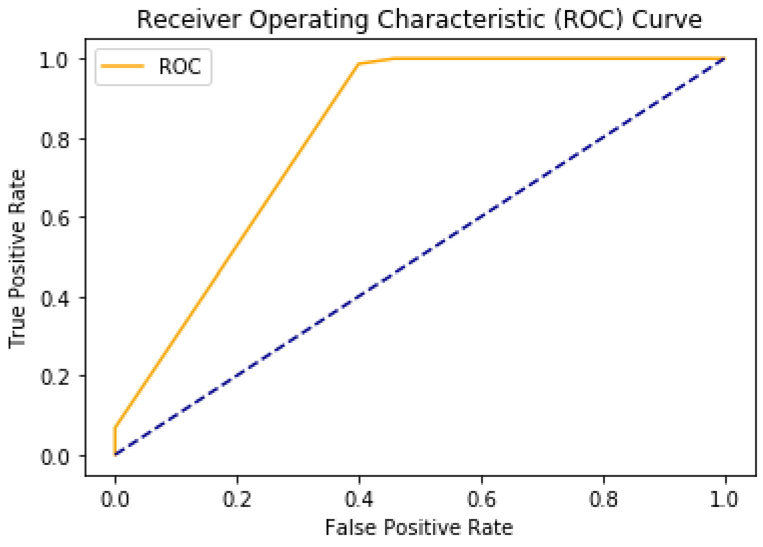


Receiver Operating Characteristic (ROC) Curve
ROC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

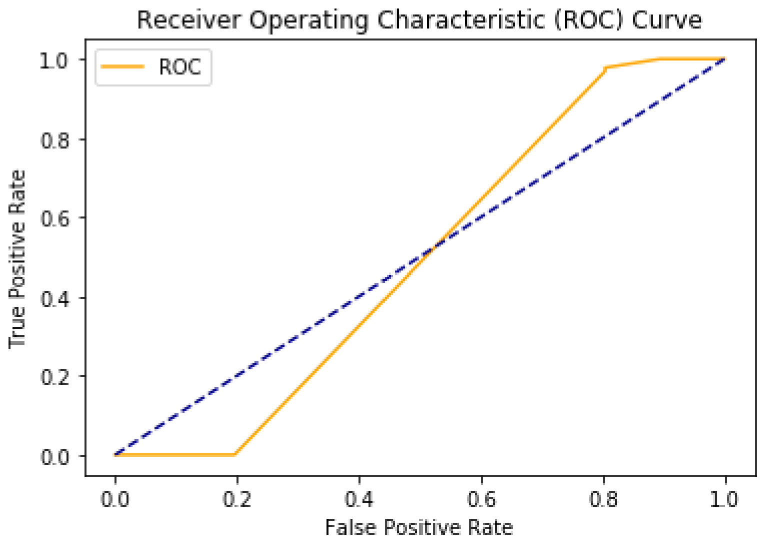
In [31]:

```
plot_roc_curve_microbiome_test2(pplacer_ref_list = ['A'],pplacer_stats_list=['_adcl_log'],
community_list=['0'],cutoff_list=['-4.00'], test_data_list=['B','C','D','E'],testOption=Tr
ue, scoreOption=False)
```

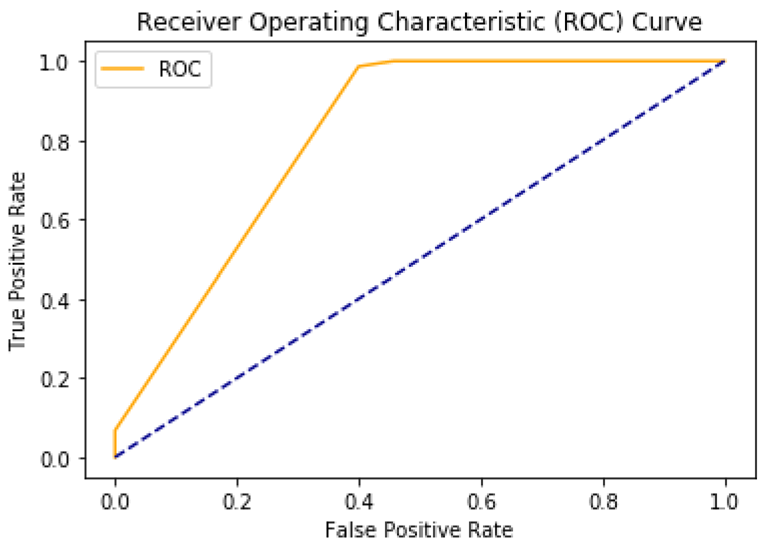
```
 The pplacer_stats_cutoff -4.00 for Reference A community 0 pplacer_stats adc
l_log compared with test B: -4.00
data_set is True
AUC1: 0.81
optimal_threshold1: 0.92
[2.         1.         0.91910309 0.51261905 0.        ]
```



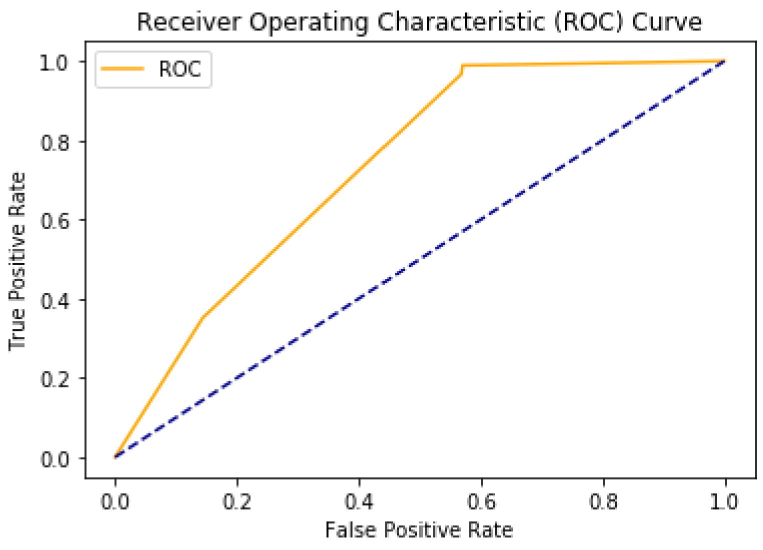
```
AUC2: 0.49
optimal_threshold2: 0.90
[2.         1.         0.91910309 0.9        0.51261905 0.        ]
```



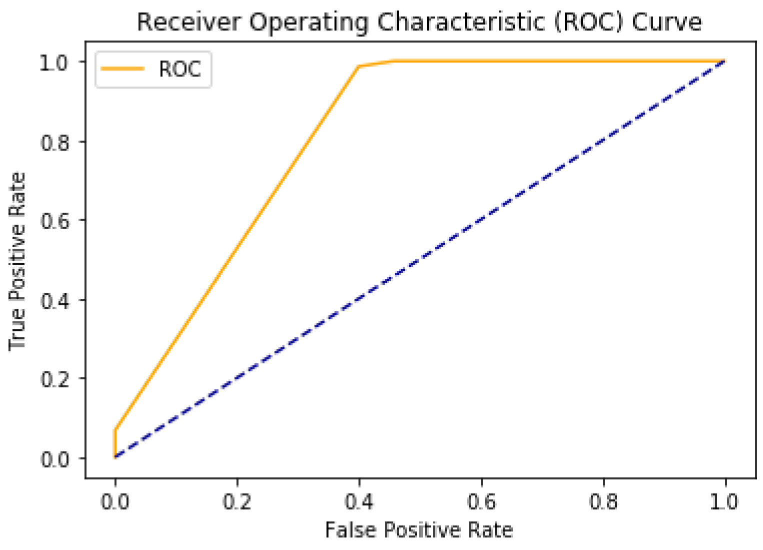
```
 The pplacer_stats_cutoff -4.00 for Reference A community 0 pplacer_stats adc
l_log compared with test C: -4.00
data_set is True
AUC1: 0.81
optimal_threshold1: 0.92
[2.         1.         0.92038132 0.62708874 0.        ]
```



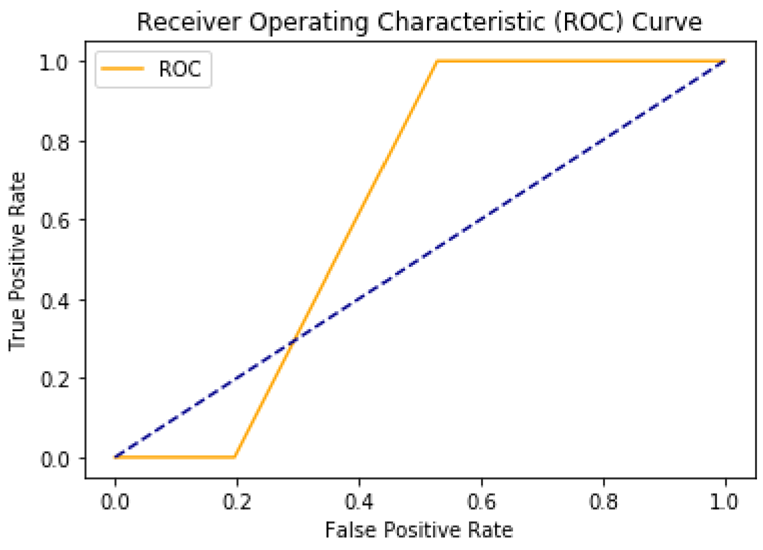
```
AUC2: 0.73
optimal_threshold2: 0.63
[2.         1.         0.92038132 0.62708874 0.        ]
```



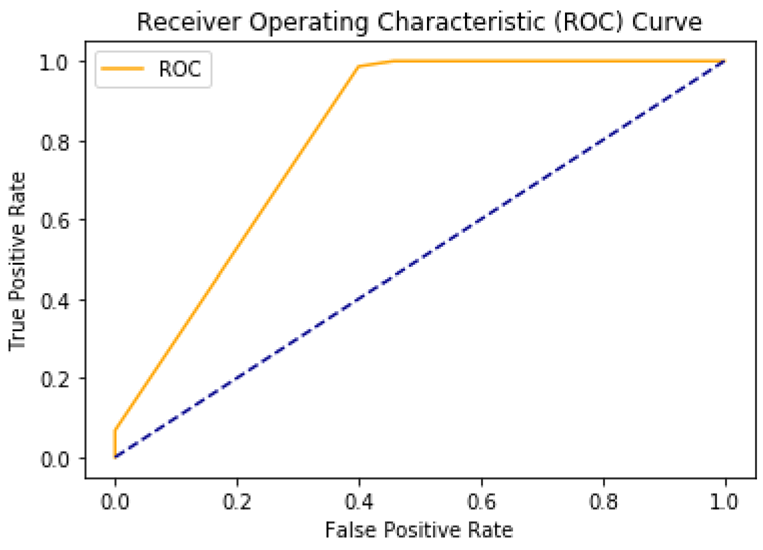
```
 The pplacer_stats_cutoff -4.00 for Reference A community 0 pplacer_stats adc
l_log compared with test D: -4.00
data_set is True
AUC1: 0.81
optimal_threshold1: 0.92
[2.         1.         0.91961409 0.69247863 0.        ]
```



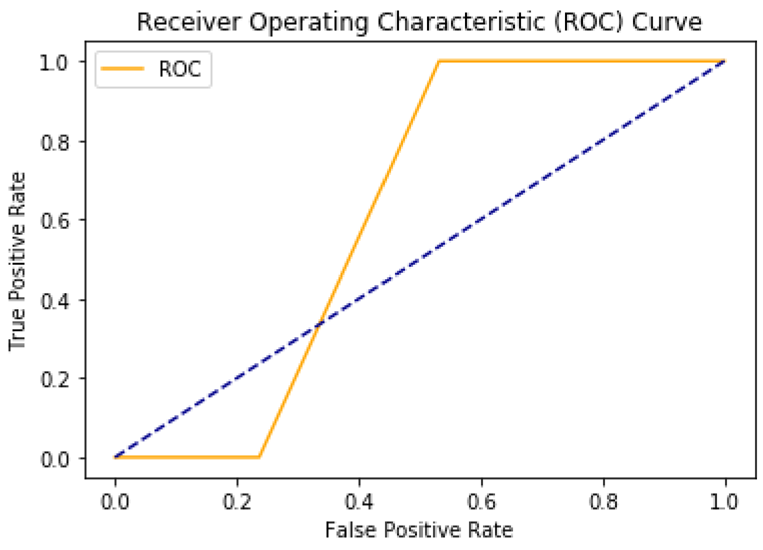
```
AUC2: 0.64
optimal_threshold2: 0.92
[2.         1.         0.91961409 0.69247863 0.        ]
```



```
 The pplacer_stats_cutoff -4.00 for Reference A community 0 pplacer_stats adc
l_log compared with test E: -4.00
data_set is True
AUC1: 0.81
optimal_threshold1: 0.91
[2.         1.         0.91372875 0.60600122 0.        ]
```



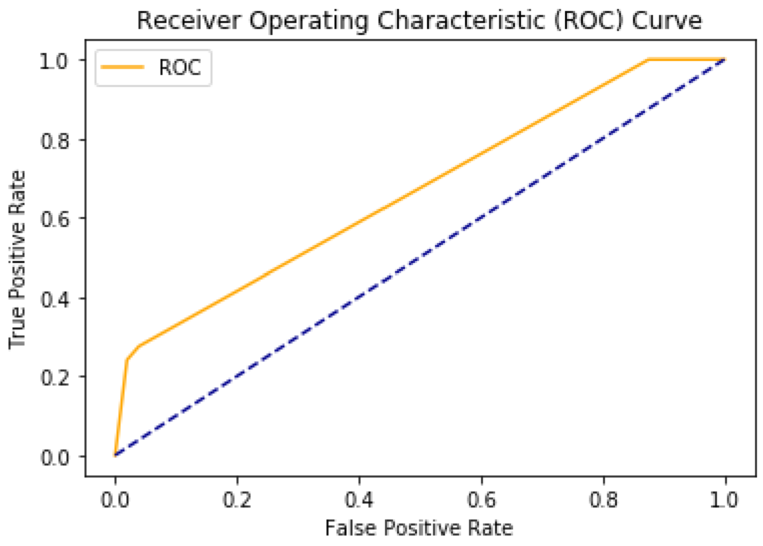
```
AUC2: 0.62
optimal_threshold2: 0.91
[2.         1.         0.91372875 0.60600122]
```



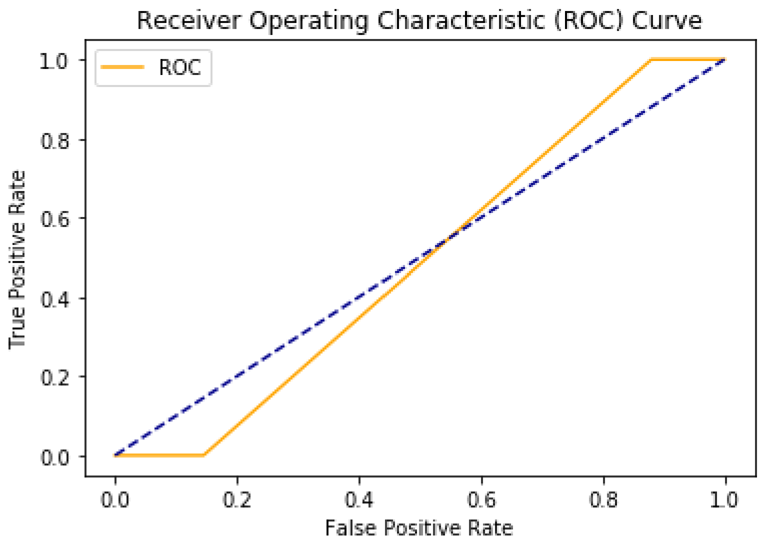
In [32]:

```
plot_roc_curve_microbiome_test2(pplacer_ref_list = ['A'],pplacer_stats_list=['_adcl_log'],
community_list=['0'],cutoff_list=['25%'], test_data_list=['B','C'],testOption=True, scoreO
ption=False)
```

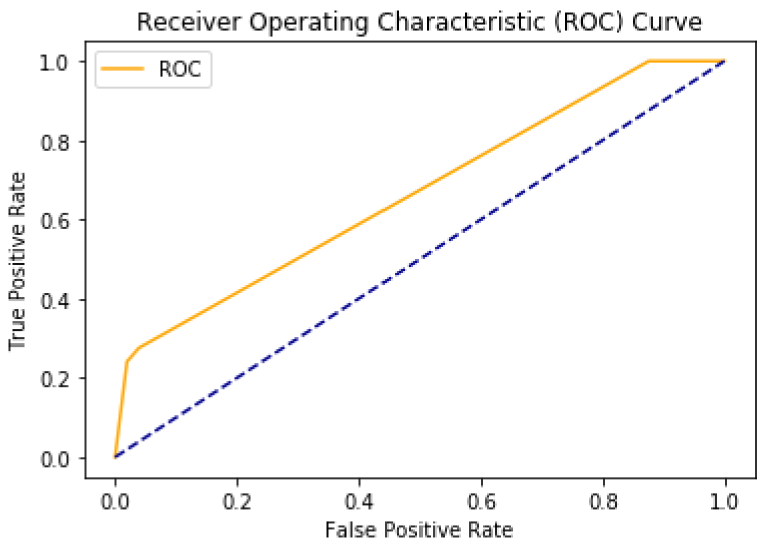
```
 The pplacer_stats_cutoff 25% for Reference A community 0 pplacer_stats adcl_
log compared with test B: -5.22
data_set is True
AUC1: 0.67
optimal_threshold1: 0.62
[1.69069337 0.69069337 0.62349206 0.1344266  0.        ]
```



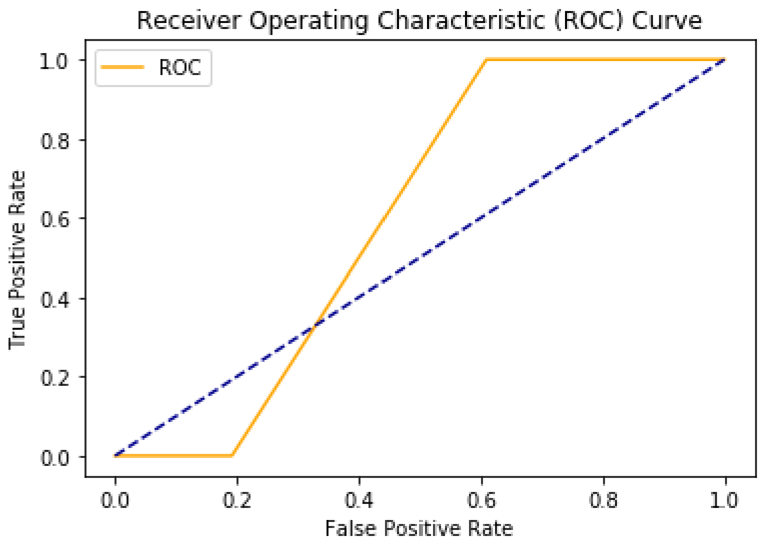
```
AUC2: 0.49
optimal_threshold2: 0.13
[1.69069337 0.69069337 0.62349206 0.1344266  0.        ]
```



```
 The pplacer_stats_cutoff 25% for Reference A community 0 pplacer_stats adcl_
log compared with test C: -5.22
data_set is True
AUC1: 0.67
optimal_threshold1: 0.50
[1.70657894 0.70657894 0.4980303  0.14889904 0.        ]
```



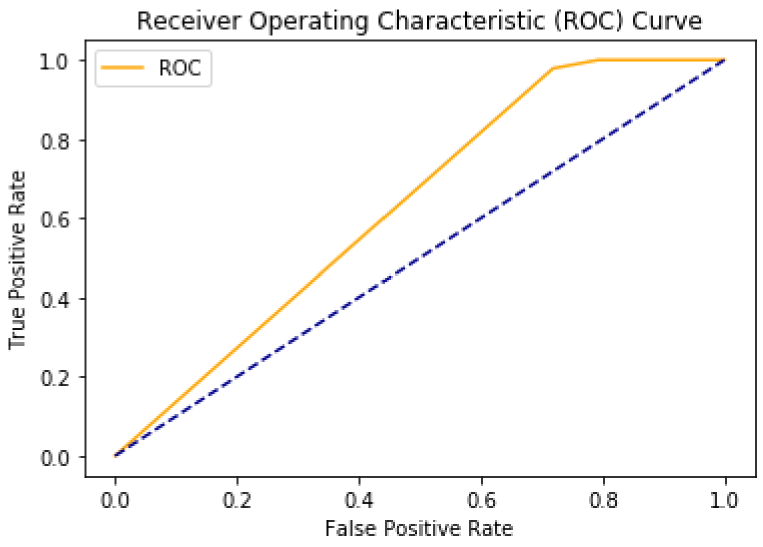
```
AUC2: 0.60
optimal_threshold2: 0.15
[1.70657894 0.70657894 0.4980303  0.14889904 0.        ]
```



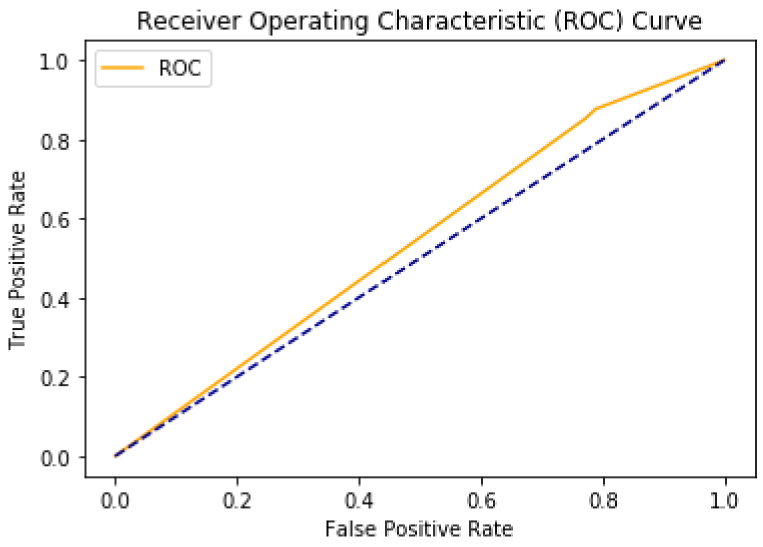
In [33]:

```
plot_roc_curve_microbiome_test2(pplacer_ref_list = ['B'],pplacer_stats_list=['_adcl_log'],
community_list=['0'],cutoff_list=['25%'], test_data_list=['A','C'],testOption=True, scoreO
ption=False)
```

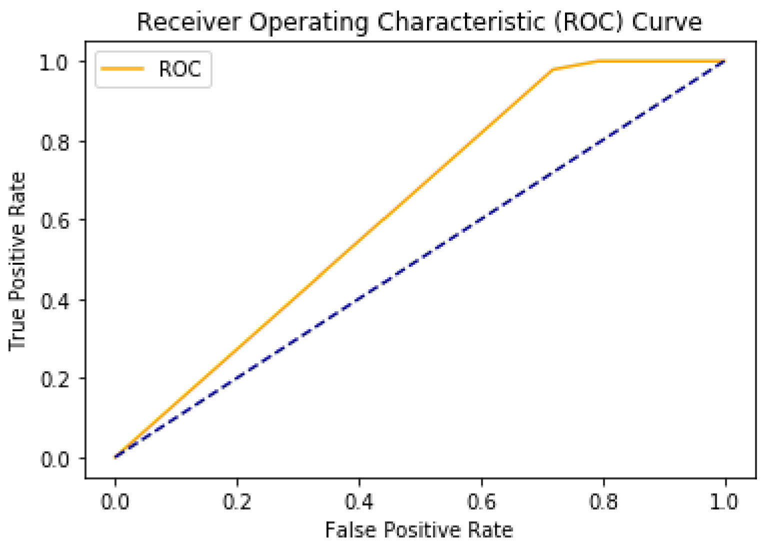
```
 The pplacer_stats_cutoff 25% for Reference B community 0 pplacer_stats adcl_
log compared with test A: -5.15
data_set is True
AUC1: 0.63
optimal_threshold1: 0.25
[1.24654057 0.24654057 0.10306062 0.        ]
```



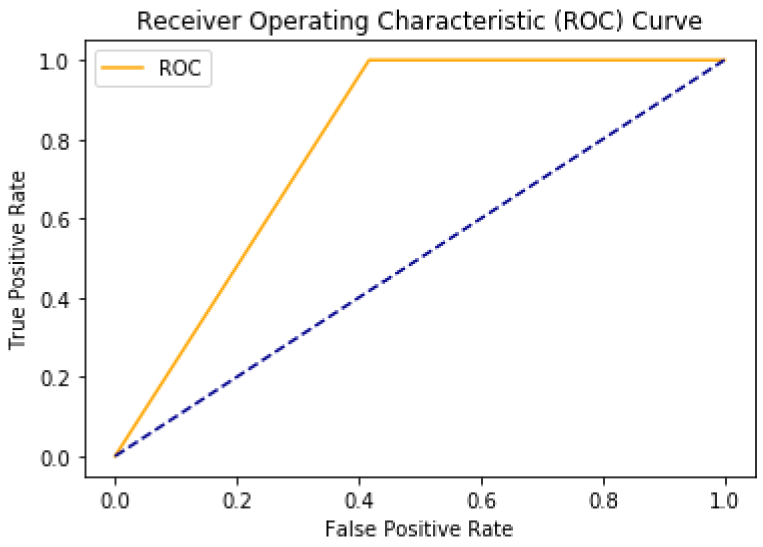
```
AUC2: 0.54
optimal_threshold2: 0.10
[1.24654057 0.24654057 0.10306062 0.        ]
```



```
 The pplacer_stats_cutoff 25% for Reference B community 0 pplacer_stats adcl_
log compared with test C: -5.15
data_set is True
AUC1: 0.63
optimal_threshold1: 0.22
[1.21930663 0.21930663 0.1113958  0.        ]
```



```
AUC2: 0.79
optimal_threshold2: 0.22
[1.21930663 0.21930663 0.1113958  0.        ]
```



In [34]:

```
# print("the head for df is {}".format(df.head)+ " the columns of the df is {}".format(df.
columns))
#
```

In [35]:

```
# df['A0'].describe(), df['B0'].describe(), df['C0'].describe(), df['D0'].describe(),df['E
0'].describe()
```

In [36]:

```
# for community in ['A','B','C','D','E']:
#     for i in range(10):
#             print(df[community+str(i)].describe())
```

In [37]:

```
df_0 = df
```

In [38]:

```
# plot_pplacer('90')
```

In [39]:

```
# plotScatter('B','0')
```

In [40]:

```
# plotScatterRef('_adcl_log','0')
```

In [41]:

```
# plot_pplacer('_adcl_log')
```

In [42]:

```
df['A_adcl_log'].describe()
```

Out[42]:

```
count    5974.000000
mean       -4.083366
std         1.837510
min        -5.995679
25%        -5.221126
50%        -5.096367
75%        -1.706947
max        -0.344675
Name: A_adcl_log, dtype: float64
```

In [43]:

```
# plot_pplacer('0')
```

In [44]:

```
df['A0'].describe()
```

Out[44]:

```
count    605.000000
mean       6.390083
std       10.778008
min        0.000000
25%        2.000000
50%        2.000000
75%        2.000000
max       38.000000
Name: A0, dtype: float64
```

In [45]:

```
df1 = df[(df['A0']>10)]
```

In [46]:

```
df1['A0'].describe()
```

Out[46]:

```
count    99.000000
mean     28.626263
std      10.791707
min      12.000000
25%      12.000000
50%      32.000000
75%      38.000000
max      38.000000
Name: A0, dtype: float64
```

In [47]:

```
99/605
```

Out[47]:

```
0.16363636363636364
```

In [48]:

```
df1['B0'].describe()
```

Out[48]:

```
count    99.000000
mean     25.070707
std      17.438670
min       0.000000
25%       0.000000
50%      36.000000
75%      38.000000
max      38.000000
Name: B0, dtype: float64
```

In [49]:

```
df2=df[['seqID','A0','B0', 'C0','D0','E0']].dropna()
```

In [50]:

```
df3 = df2[(df2['A0']>10) & (df2['B0']>10)  & (df2['C0']>10)  & (df2['D0']>10) & (df2['E0']
>10)]
```

In [51]:

```
# df2.describe(), df3.describe()
```

In [52]:

```
df3
```

Out[52]:

	seqID	A0	B0	C0	D0	E0
5313	CC11CM5SCR137ef78188b94db7b59504dc64363aa3	34.0	34.0	32.0	32.0	44.0
5314	CC11CM0SCR35529da454f0497fa16e04841e8e1639	34.0	34.0	32.0	32.0	44.0

In [53]:

```
2/605
```

Out[53]:

```
0.003305785123966942
```

In [54]:

```
dfc90 = df[(df['A90']>10) & (df['B90']>10)  & (df['C90']>10)  & (df['D90']>10) & (df['E90'
]>10)]
```

In [55]:

```
dfc90['B0'].describe()
```

Out[55]:

```
count    0.0
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
75%      NaN
max      NaN
Name: B0, dtype: float64
```

In [56]:

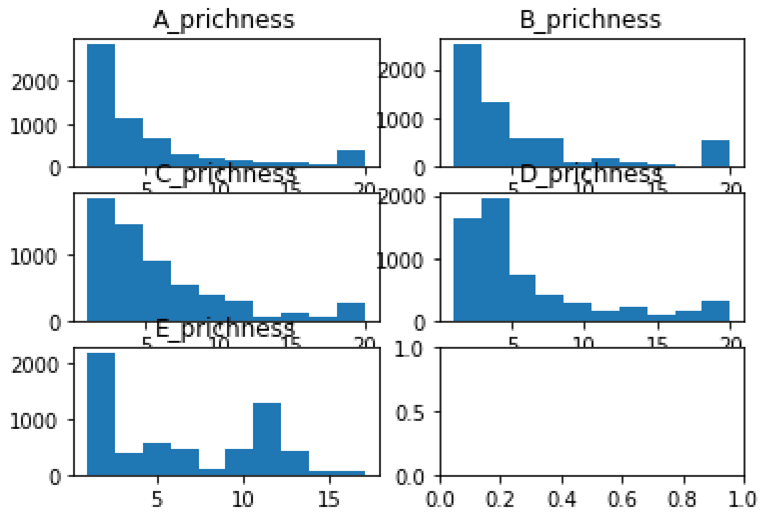
```
df[(df.community=='CC11CM0')]['C_adcl_log'].dropna().describe()
```

Out[56]:

```
count    55.000000
mean     -1.885119
std       1.762885
min      -5.300162
25%      -1.773077
50%      -1.040954
75%      -0.682030
max      -0.373058
Name: C_adcl_log, dtype: float64
```

In [57]:

```
plot_pplacer('_prichness')
```



In [58]:

```
df['A_prichness'].describe()
```

Out[58]:

```
count    5974.000000
mean        4.727653
std         5.465596
min         1.000000
25%         1.000000
50%         3.000000
75%         5.000000
max        20.000000
Name: A_prichness, dtype: float64
```

In [59]:

```
df[df.A0>10].A0.count()
```

Out[59]:

```
99
```

In [60]:

```
df[df.A0>10].A0.count()/df.A0.count()
```

Out[60]:

```
0.16363636363636364
```

In [61]:

```
# df.head()
```

In [62]:

```
df.A_adcl.count()
```

Out[62]:

```
5974
```

In [63]:

```
d={"a":1, "b":2}
```

In [64]:

```
d
```

Out[64]:

```
{'a': 1, 'b': 2}
```

In [65]:

```
dd = pd.Series(d, name='score')
```

In [66]:

```
dd.index.name="community"
```

In [67]:

```
dd.reset_index()
```

Out[67]:

	community	score
0	a	1
1	b	2

In [68]:

```
"CC11CM"+str(0)
```

Out[68]:

```
'CC11CM0'
```

In [69]:

```
c0=df['A0'][df['community']=='CC11CM0']
```

In [70]:

```
per=c0[c0>10].count()/c0.count()
```

In [71]:

```
def generateScore(stats, referenceID,scorecutoff,statscutoff):
    d1={}
    d2={}
    for i in range(100):
        values = df[referenceID+str(i)][df.community=='CC11CM'+str(i)]
        statsvalues = df[stats][df['community']=='CC11CM'+str(i)]
        d1['CC11CM'+str(i)] = values[values>scorecutoff].count()/values.count()
        d2['CC11CM'+str(i)] =statsvalues[statsvalues>statscutoff].count()/statsvalues.coun
t()
    d1=pd.Series(d1, name=referenceID)
    d1.index.name='community'
    d1=d1.reset_index()
    d2=pd.Series(d2, name=stats)
    d2.index.name='community'
    d2=d2.reset_index()
    dt = pd.concat([d1,d2], axis=1)
#     dt=dt.set_index('community')
    return (dt)
```

In [72]:

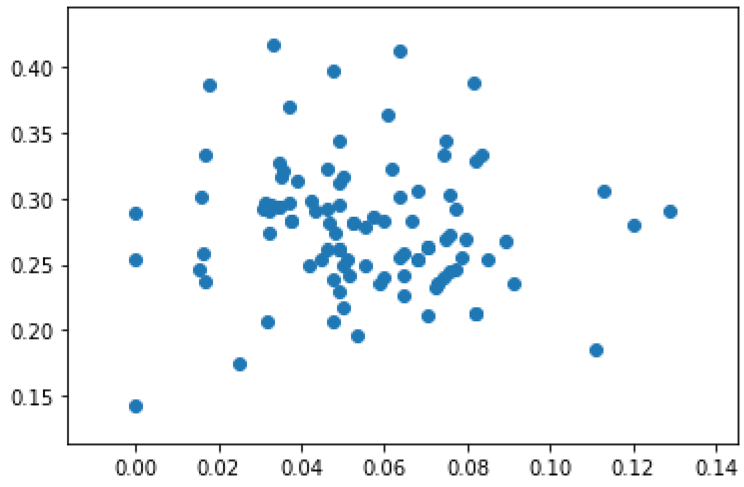
```
dt=generateScore('A_adcl', 'A', 10, 0.001)
```

In [73]:

```
plt.scatter(dt.A, dt.A_adcl)
```

Out[73]:

```
<matplotlib.collections.PathCollection at 0x1a976c50>
```



In [74]:

```
t=[]
for referenceID in ['A','B','C','D','E']:
    t.append(generateScore('A_adcl', referenceID, 10, 0.001))
```

In [75]:

```
t[0].head()
```

Out[75]:

	community	A	community	A_adcl
0	CC11CM0	0.090909	CC11CM0	0.236364
1	CC11CM1	0.082192	CC11CM1	0.328767
2	CC11CM2	0.025000	CC11CM2	0.175000
3	CC11CM3	0.050847	CC11CM3	0.254237
4	CC11CM4	0.000000	CC11CM4	0.288462

In [76]:

```
tt = pd.concat([t[0],t[1],t[2],t[3],t[4]], axis=1)
```

In [77]:

```
def generateScoreu(stats, referenceID,scorecutoff,statscutoff):
    d1={}
    d2={}
    for i in range(100):
        values = df[referenceID+str(i)][df.community=='CC11CM'+str(i)]
        statsvalues = df[referenceID+stats][df['community']=='CC11CM'+str(i)]
        d1['CC11CM'+str(i)] = values[values>scorecutoff].count()/values.count()
        d2['CC11CM'+str(i)] =statsvalues[statsvalues>statscutoff].count()/statsvalues.coun
t()
    d1=pd.Series(d1, name=referenceID)
    d1.index.name='community'
    d1=d1.reset_index()
    d2=pd.Series(d2, name=referenceID+stats)
    d2.index.name='community'
    d2=d2.reset_index()
    dt = pd.concat([d1,d2], axis=1)
    dt=dt.loc[:, ~dt.columns.duplicated()]
    dt=dt.set_index('community')
    return (dt)
```

In [78]:

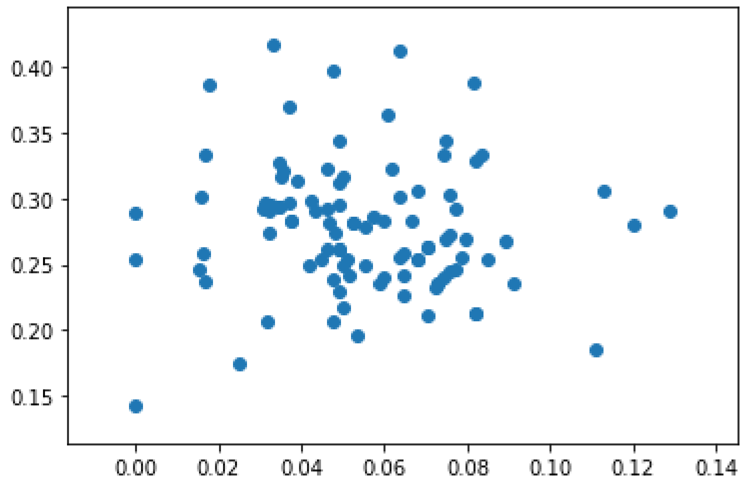
```
dtu=generateScoreu('_adcl', 'A', 10, 0.001)
```

In [79]:

```
plt.scatter(dtu.A, dtu.A_adcl)
```

Out[79]:

```
<matplotlib.collections.PathCollection at 0x1bd43a30>
```



In [ ]:

In [80]:

```
t=[]
statsdir= {'_adcl':0.0001, '_edpl':0,'_prichness':10,'_mindistl':0.05}
for stats in statsdir.keys():

    for referenceID in ['A','B','C','D','E']:
        t.append(generateScoreu(stats, referenceID, 10, statsdir[stats]))
```

In [ ]:

```
```

In [81]:

```
ttt = pd.concat([t[0],t[1],t[2],t[3],t[4],t[5],t[6],t[7],t[8],t[9],t[10],t[11],t[12],t[13
],t[14],t[15],t[16],t[17],t[18],t[19]], axis=1)
ttt=ttt.loc[:, ~ttt.columns.duplicated()]
```

In [82]:

```
# ttt.describe()
```

In [83]:

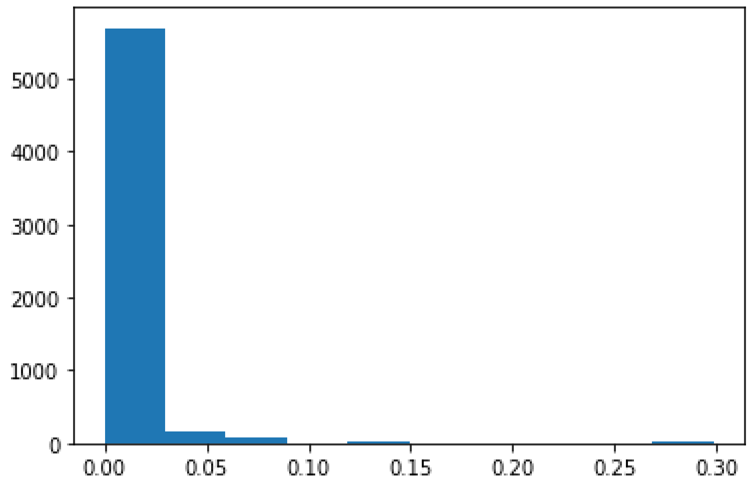
```
ttt.to_csv("community-based.csv")
```

In [84]:

```
plt.hist(df.A_mindistl)
```

Out[84]:

```
(array([5.696e+03, 1.570e+02, 6.800e+01, 0.000e+00, 2.200e+01, 0.000e+00,
        0.000e+00, 5.000e+00, 0.000e+00, 2.600e+01]),
 array([3.48920365e-07, 2.98496853e-02, 5.96990217e-02, 8.95483581e-02,
        1.19397695e-01, 1.49247031e-01, 1.79096367e-01, 2.08945704e-01,
        2.38795040e-01, 2.68644377e-01, 2.98493713e-01]),
 <a list of 10 Patch objects>)
```



In [85]:

```
dp =pd.read_csv("community-based.csv", index_col=0)
```

In [86]:

```
dp.describe()
```

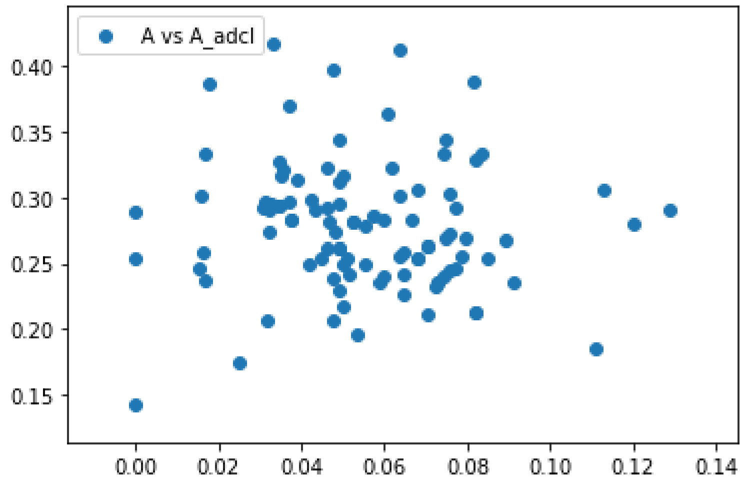
Out[86]:

	A	A_adcl	B	B_adcl	C	C_adcl	D	
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100
mean	0.055034	0.277586	0.116043	0.557102	0.318386	0.799584	0.647292	0
std	0.024318	0.049060	0.032415	0.054352	0.048526	0.044362	0.051736	0
min	0.000000	0.142857	0.051724	0.406780	0.216667	0.666667	0.491803	0
25%	0.037736	0.246154	0.095238	0.516532	0.285119	0.773585	0.612455	0
50%	0.052178	0.275986	0.112007	0.563333	0.315789	0.800000	0.649561	0
75%	0.072530	0.301587	0.136310	0.600000	0.346392	0.830769	0.682738	0
max	0.129032	0.416667	0.209677	0.682540	0.448980	0.888889	0.786885	1

8 rows × 25 columns

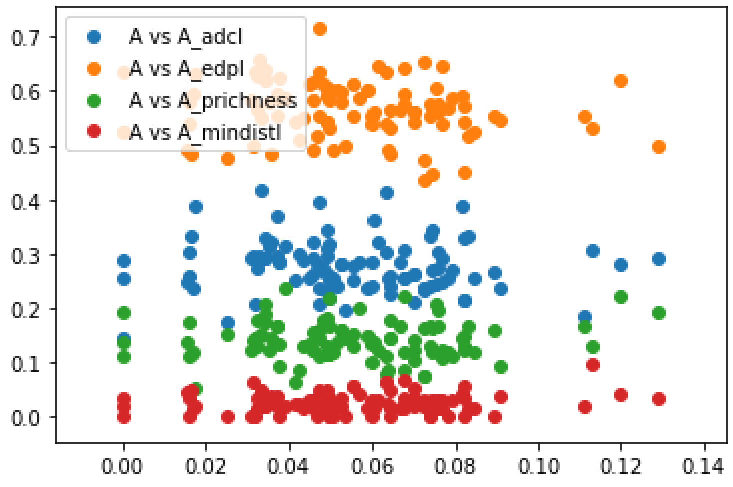
In [87]:

```
for score in ['A','B','C','D','E'][0:1]:
    for stats in ['_adcl', '_edpl','_prichness','_mindistl'][0:1]:
        plt.scatter(dp[score], dp[score+stats], label=score + ' vs '+ score+stats)
        plt.legend(loc='upper left')
        plt.show
```



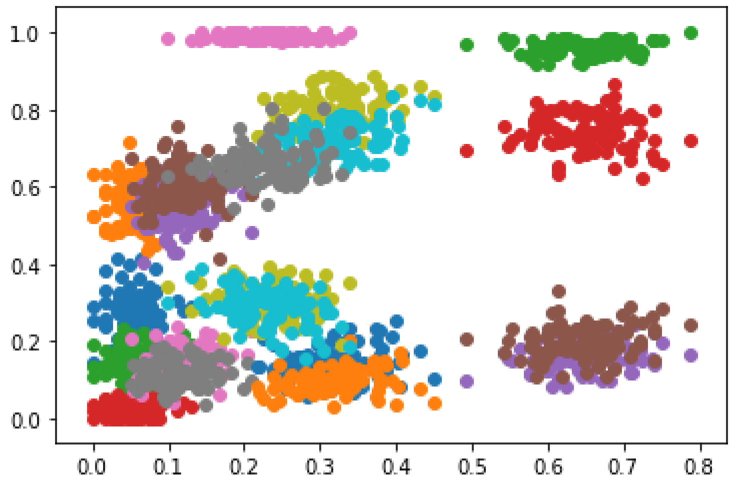
In [88]:

```
for score in ['A','B','C','D','E'][0:1]:
    for stats in ['_adcl', '_edpl','_prichness','_mindistl'][0:5]:
        plt.scatter(dp[score], dp[score+stats], label=score + ' vs '+ score+stats)
        plt.legend(loc='upper left')
        plt.show
```



In [89]:

```
for score in ['A','B','C','D','E']:
    for stats in ['_adcl', '_edpl','_prichness','_mindistl']:
        plt.scatter(dp[score], dp[score+stats], label=score + ' vs '+ score+stats)
#         plt.legend(loc='upper left')
        plt.show
```



In [90]:

```
dp.describe()
```

Out[90]:

	A	A_adcl	B	B_adcl	C	C_adcl	D	
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100
mean	0.055034	0.277586	0.116043	0.557102	0.318386	0.799584	0.647292	0
std	0.024318	0.049060	0.032415	0.054352	0.048526	0.044362	0.051736	0
min	0.000000	0.142857	0.051724	0.406780	0.216667	0.666667	0.491803	0
25%	0.037736	0.246154	0.095238	0.516532	0.285119	0.773585	0.612455	0
50%	0.052178	0.275986	0.112007	0.563333	0.315789	0.800000	0.649561	0
75%	0.072530	0.301587	0.136310	0.600000	0.346392	0.830769	0.682738	0
max	0.129032	0.416667	0.209677	0.682540	0.448980	0.888889	0.786885	1

8 rows × 25 columns

In [ ]: