all_data

February 26, 2020

[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")
```

[2]:
```
df = pd.read_csv("all_data.csv", index_col=0)
score= pd.read_csv("score_merged.csv", index_col=0)
```

[3]:
```
reference ={'A':"RDP_10398", 'B':'RDP_5224', 'C':"RDP_1017", 'D':'RDP_92', 'E':
 ↪'RDP_12'}
```

[4]:
```
#pplacer_ref_list = ['A','B','C',␣
 ↪'D','E'],pplacer_stats_list=['_adcl_log','_edpl',␣
 ↪'_prichness'],community_list=['0','1','2','3','4'],cutoff_list=['mean','min','25%','50%','7
```

[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
```

[6]:
```
reference
```

[6]:
```
{'A': 'RDP_10398',
 'B': 'RDP_5224',
 'C': 'RDP_1017',
 'D': 'RDP_92',
 'E': 'RDP_12'}
```

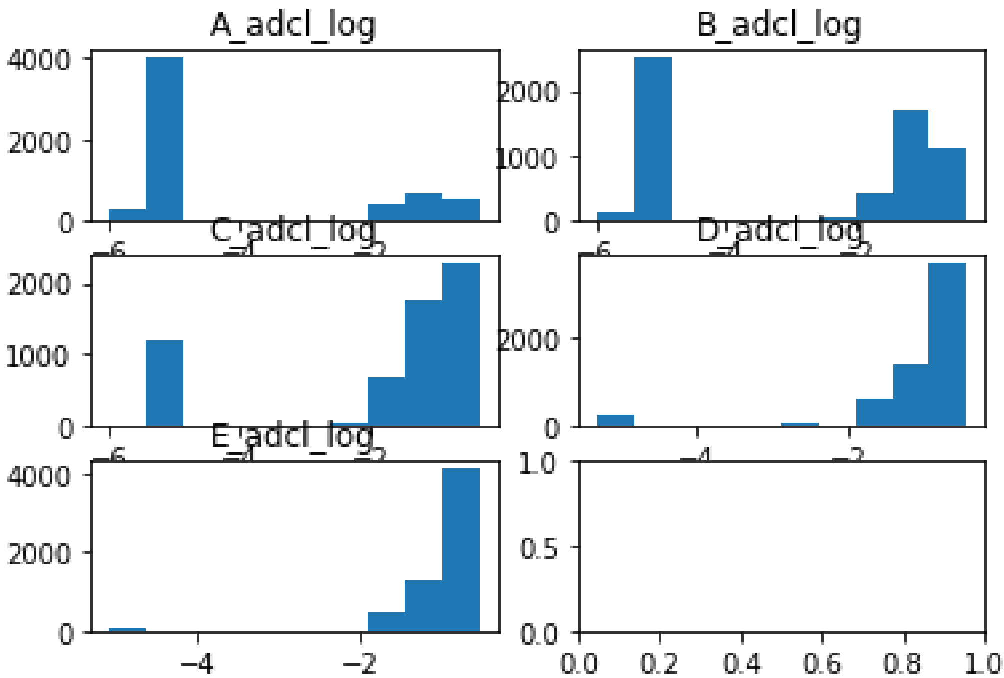
[7]:
```
columnList=list(df.columns)
```

[8]:
```
communityList = df.community
```

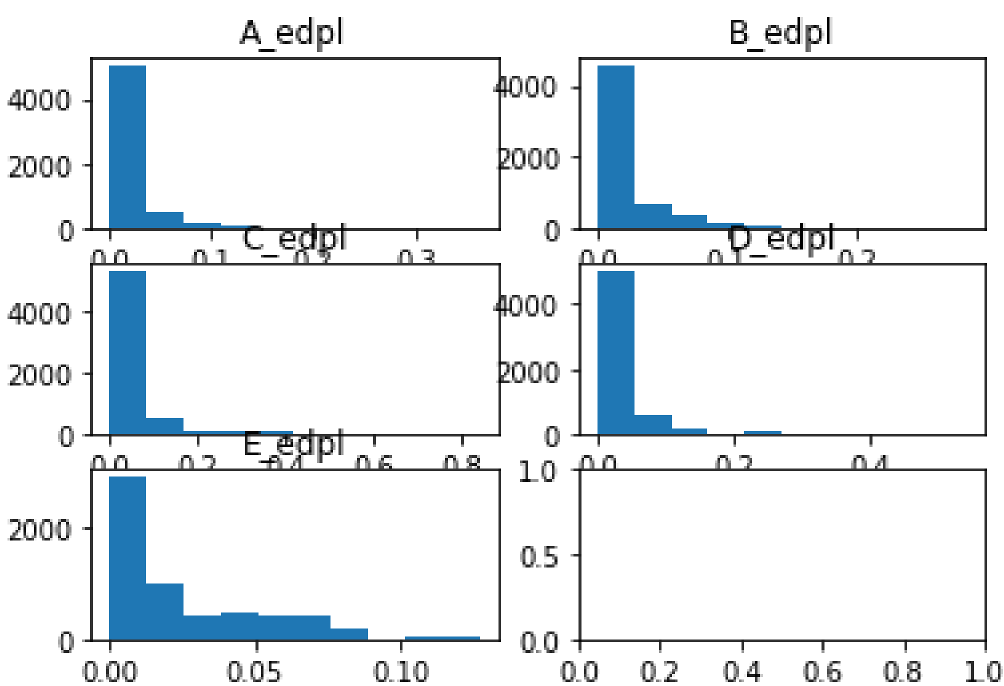
[9]:
```
# df.describe()
```

[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)
```

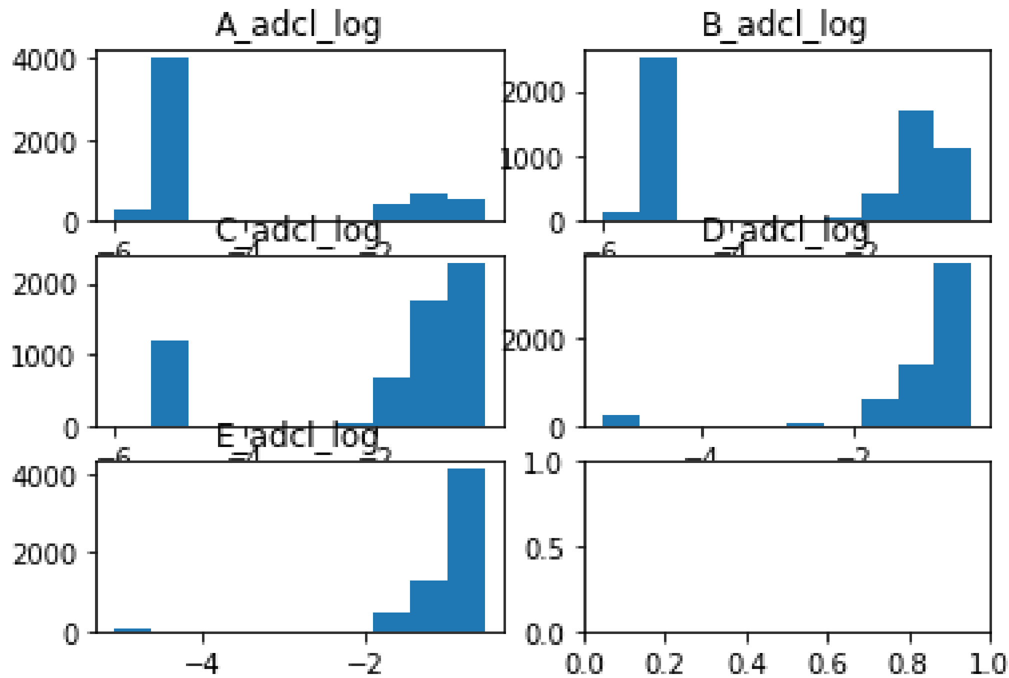
[11]:
```
plot_pplacer('_adcl_log')
```



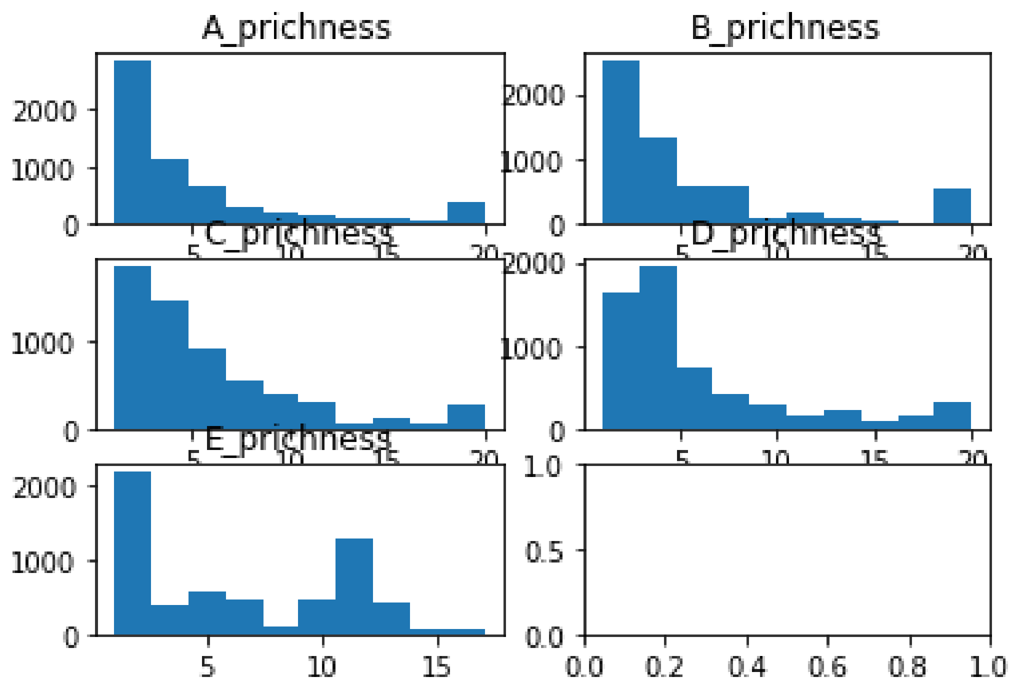
[12]:
```
plot_pplacer('_edpl')
```



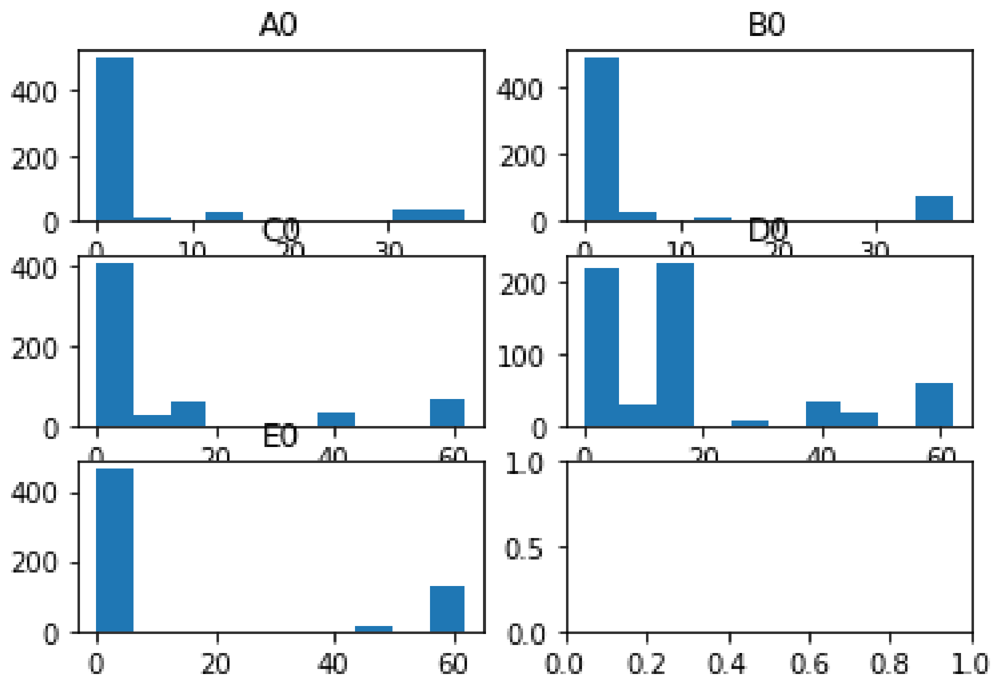
[13]:
```
plot_pplacer('_adcl_log')
```



[14]:
```
plot_pplacer('_prichness')
```

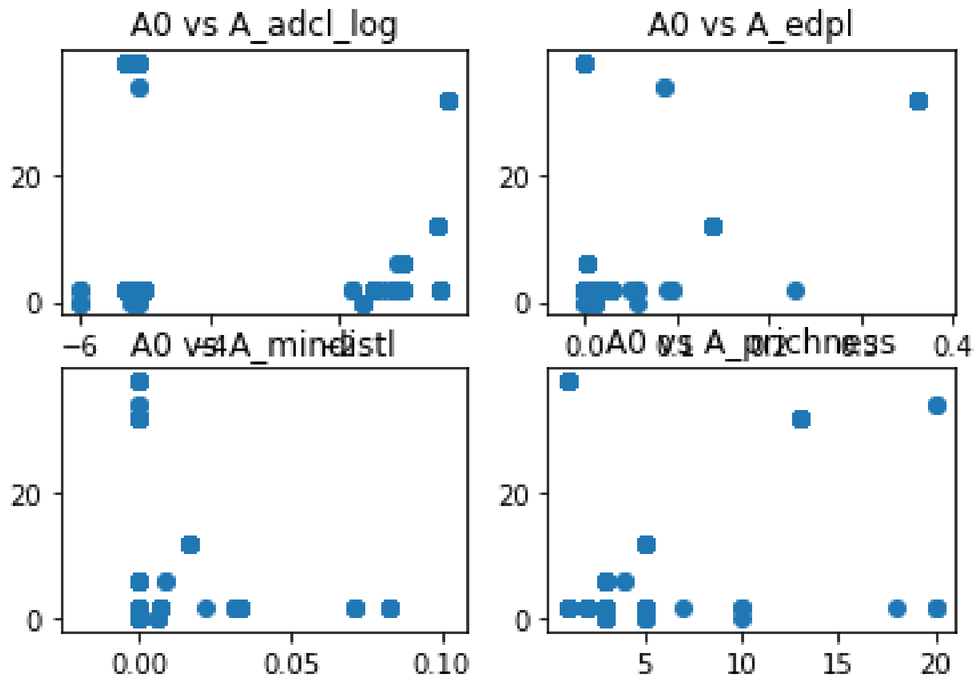


[15]:
```
plot_pplacer('0');
```



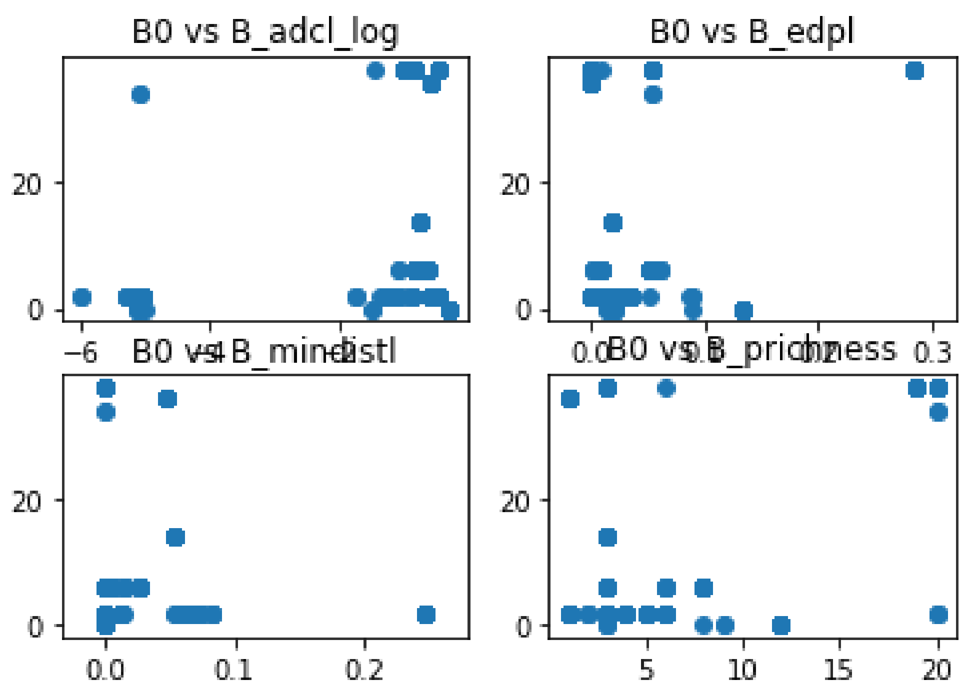

```
[16]: def plotScatter(reference,community):
          fig, axes = plt.subplots(nrows=2, ncols=2)
          ax0, ax1, ax2, ax3 = axes.flatten()

          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')
```

```
[17]: plotScatter('A','O')
```

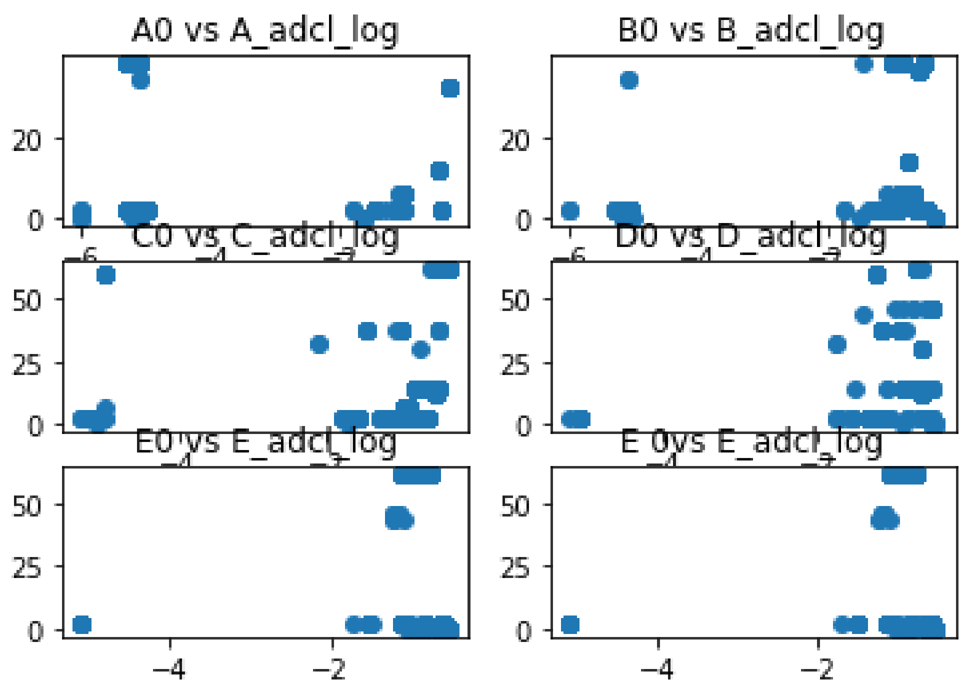


[18]:

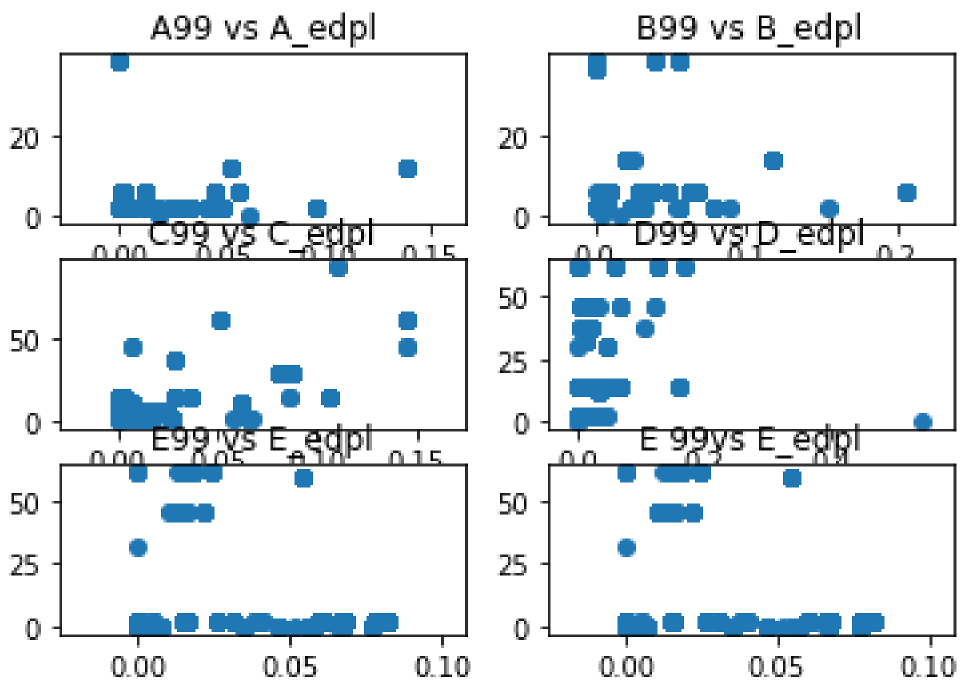
```
plotScatter('B','0')
```



[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');
```



[20]:
```
plotScatterRef('_edpl','99');
```



[21]:
```
cols=df.columns.tolist()
# cols[:20]
```

[164]:
```
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()
```

[169]:
```
def plot_roc(data_X, class_label):
    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)
#     print( thresholds)
#     print('Model: ')
#     print(model)
    plot_roc_curve(fpr, tpr)
```

[170]:
```
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
------------------
```

[171]:
```python
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)
```

[172]:

```
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().reshape(-1,1)
                        binary_label = ␣
 ↪df_binary[pplacer_ref+community+'_binary'].to_numpy()
                        print(' The score cutoff '+ cutoff +' for Reference ' +␣
 ↪pplacer_ref +' community ' + community   + ' with pplacer_stats '+␣
 ↪pplacer_stats[1:] + ': %.2f' % cutoff_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 ' + pplacer_ref +' community ' + community + ' pplacer_stats '  +␣
↪pplacer_stats[1:]  + ': %.2f' % cutoff_binary )
                        plot_roc(data_stats,binary_label)
```

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

[173]:
```
# plot_roc_curve_microbiome(pplacer_ref_list =␣
↪['A','B','C','D','E'],pplacer_stats_list=['_adcl_log'],community_list=['A'],cutoff_list=['m
↪'min','25%','50%','75%'],scoreOption=False)
```

[174]:
```
df['E0'].describe()
```

[174]:
```
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
```

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

3 Fitting on large reference and test on small reference datasets

[175]:
```
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+community+'_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()
                            print(' The score cutoff '+ cutoff +' for Reference␣
↪' + pplacer_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_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 ' + 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+community+'_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+'_binary']].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 ' + pplacer_ref +' community ' + community   + ' with␣
↪pplacer_stats '+ pplacer_stats[1:] + ' compared with test ' + test +': %.2f'␣
↪% cutoff_binary )
                               ␣
↪plot_roc_microbiome(data_stats,binary_label,x,y,data_test=True)
                            else:
```

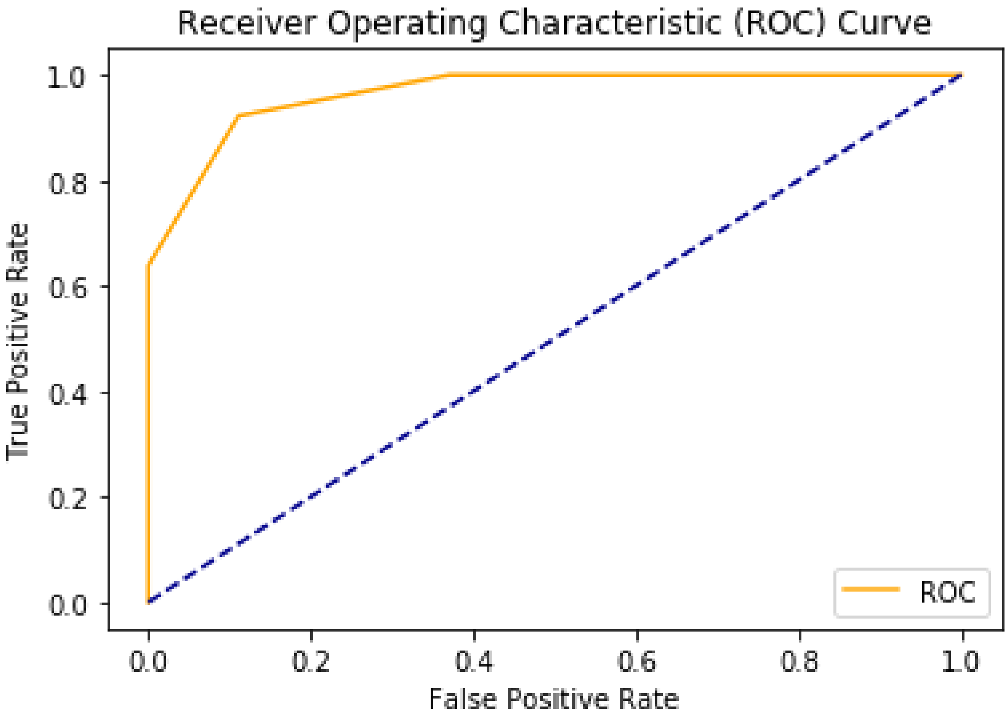
```
                                mask = df[pplacer_ref+pplacer_stats] <= ␣
↪cutoff_binary
                                df.loc[mask,␣
↪pplacer_ref+pplacer_stats+'_binary'] = 1
                                mask = df[pplacer_ref+pplacer_stats]␣
↪>cutoff_binary
                                df.loc[mask,␣
↪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().reshape(-1,1)
                                binary_label = ␣
↪df_binary[pplacer_ref+pplacer_stats+'_binary'].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+'_binary']].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 Reference ' + 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)
```

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

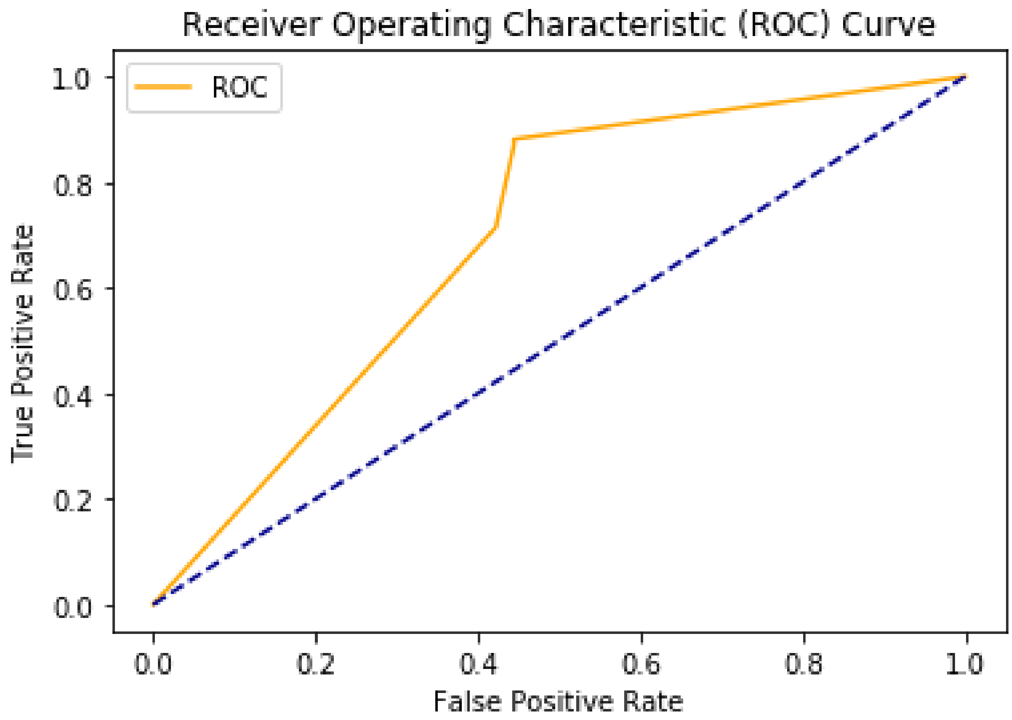
[176]:
```
plot_roc_curve_microbiome_test2(pplacer_ref_list =␣
↪['A'],pplacer_stats_list=['_adcl_log','_edpl'],community_list=['0'],cutoff_list=['10.
↪00'], test_data_list=['B','C','D','E'],testOption=True, scoreOption=True)
```

```
The score cutoff 10.00 for Reference A community 0 with pplacer_stats adcl_log
```

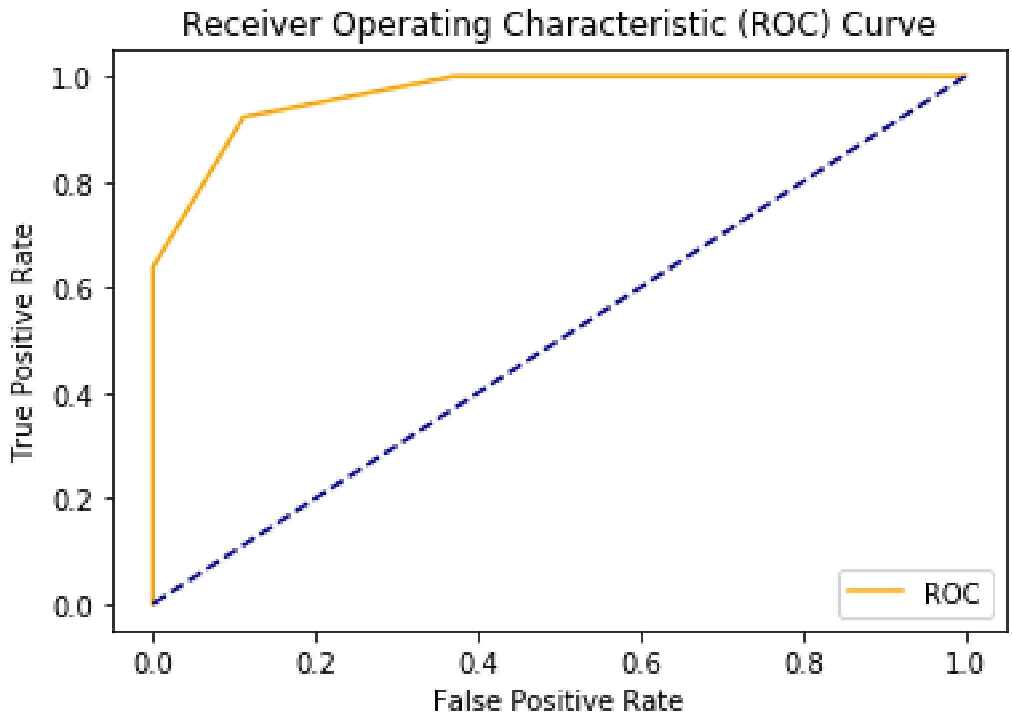
```
compared with test B: 10.00
data_set is True
AUC1: 0.97
optimal_threshold1: 0.84
[2.         1.         0.84006066 0.59846994 0.        ]
```



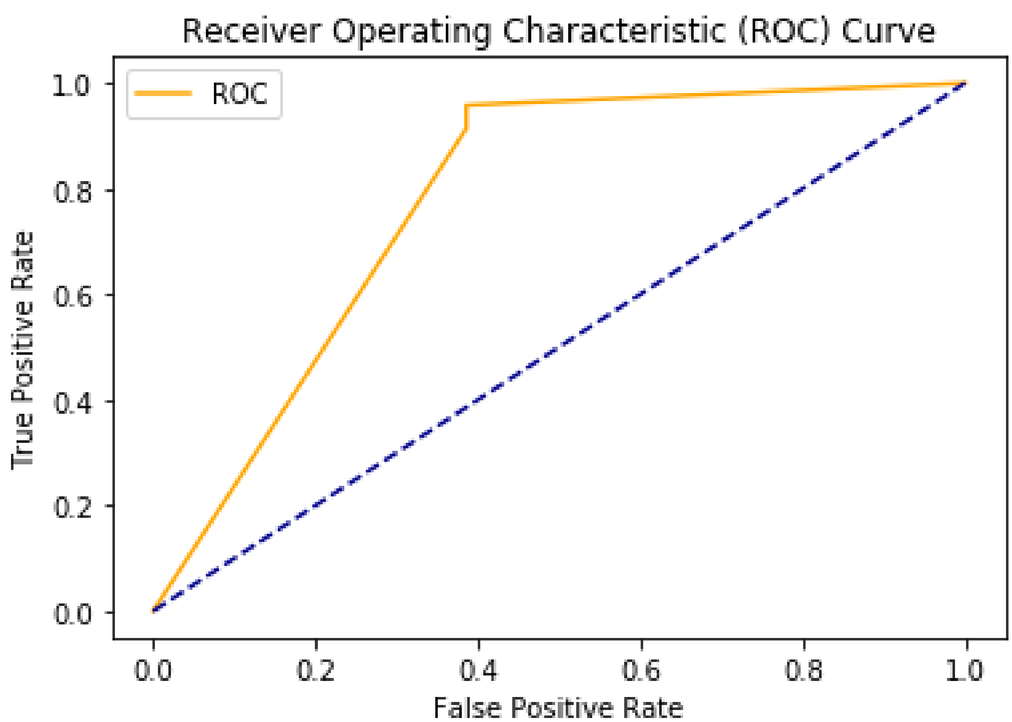

```
AUC2: 0.69
optimal_threshold2: 0.60
[2.         1.         0.84006066 0.59846994 0.        ]
```



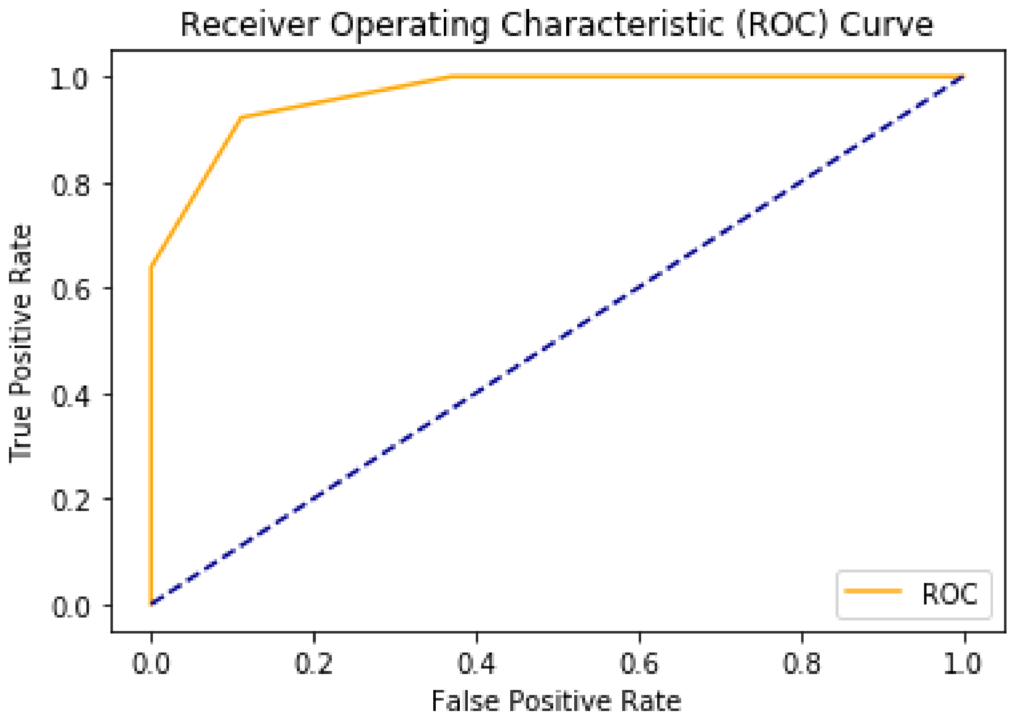

```
 The score cutoff 10.00 for Reference A community 0 with pplacer_stats adcl_log
compared with test C: 10.00
data_set is True
AUC1: 0.97
optimal_threshold1: 0.84
[2.         1.         0.83861265 0.6011308  0.        ]
```



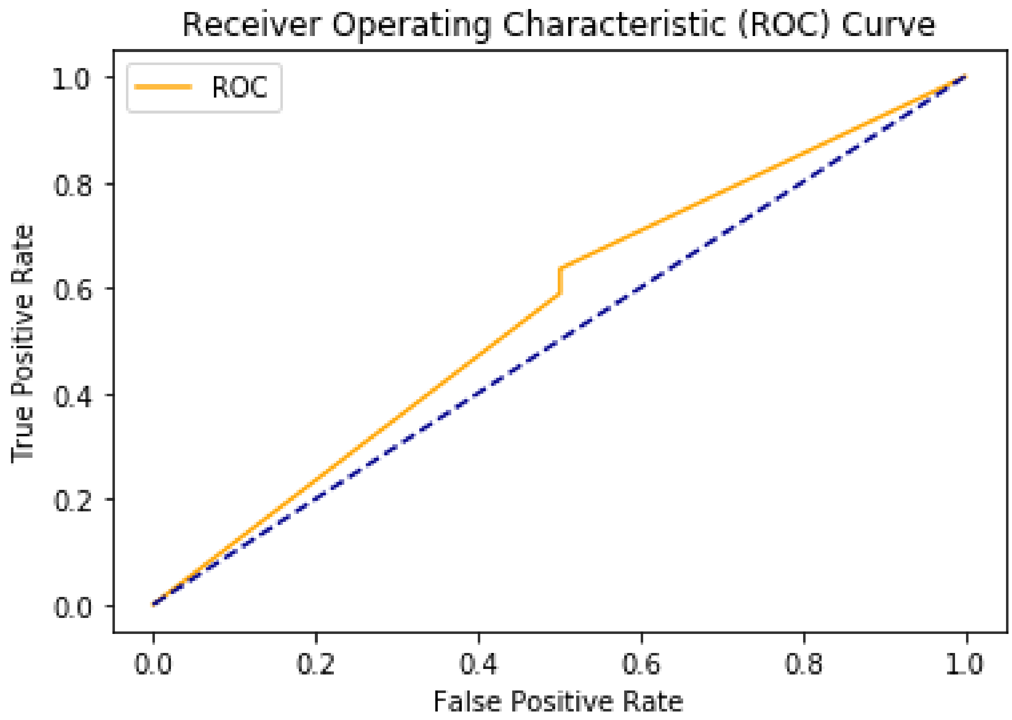

```
AUC2: 0.78
optimal_threshold2: 0.60
[2.         1.         0.83861265 0.6011308  0.        ]
```



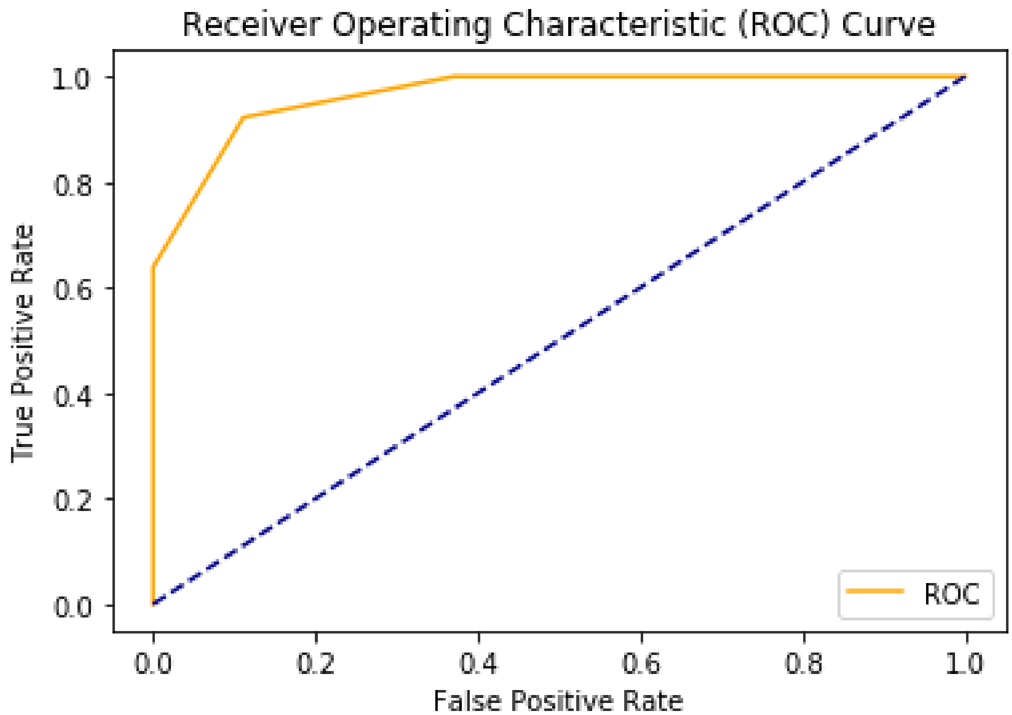
```
 The score cutoff 10.00 for Reference A community 0 with pplacer_stats adcl_log
compared with test D: 10.00
data_set is True
AUC1: 0.97
optimal_threshold1: 0.84
[2.         1.         0.83522539 0.58105212 0.        ]
```



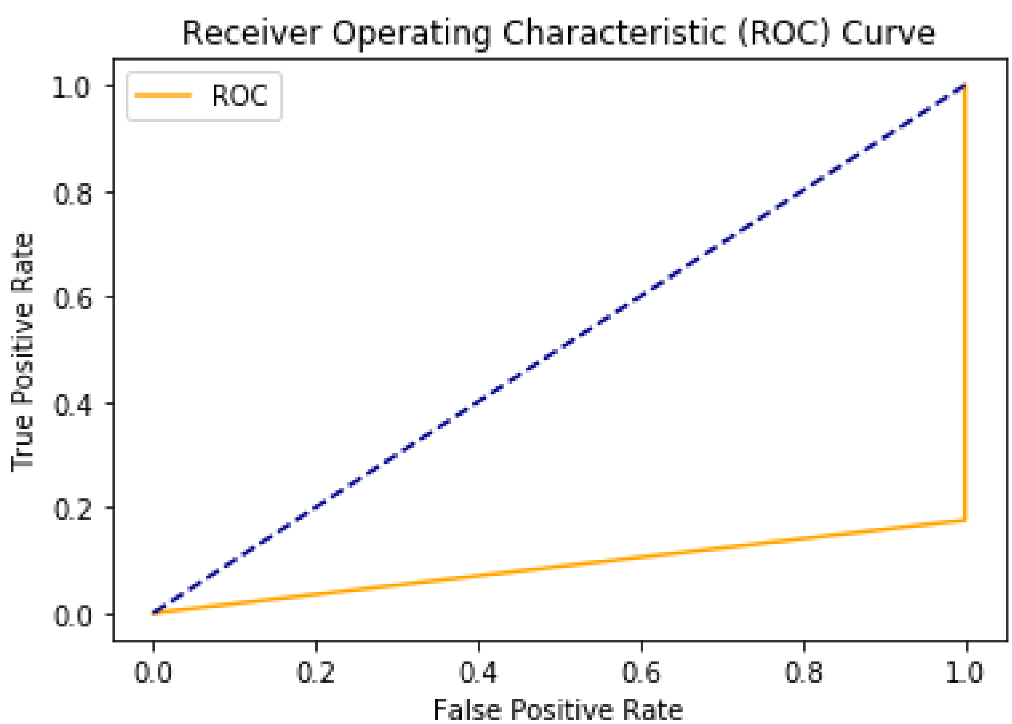

```
AUC2: 0.56
optimal_threshold2: 0.84
[2.         1.         0.83522539 0.        ]
```



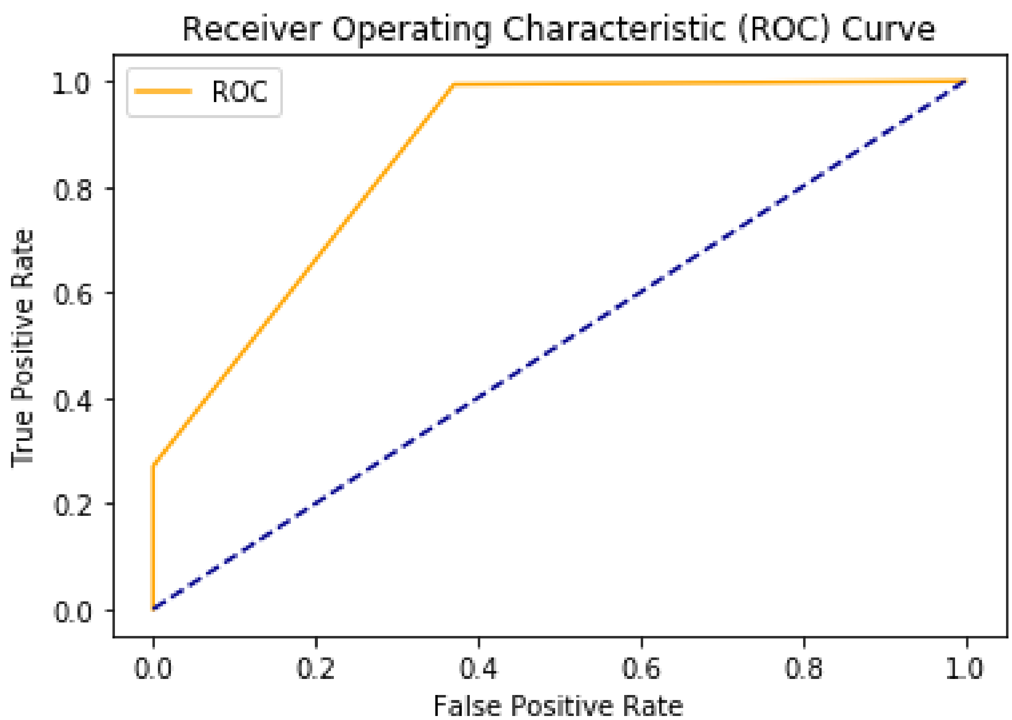

```
 The score cutoff 10.00 for Reference A community 0 with pplacer_stats adcl_log
compared with test E: 10.00
data_set is True
AUC1: 0.97
optimal_threshold1: 0.86
[2.         1.         0.85596766 0.60895695 0.        ]
```



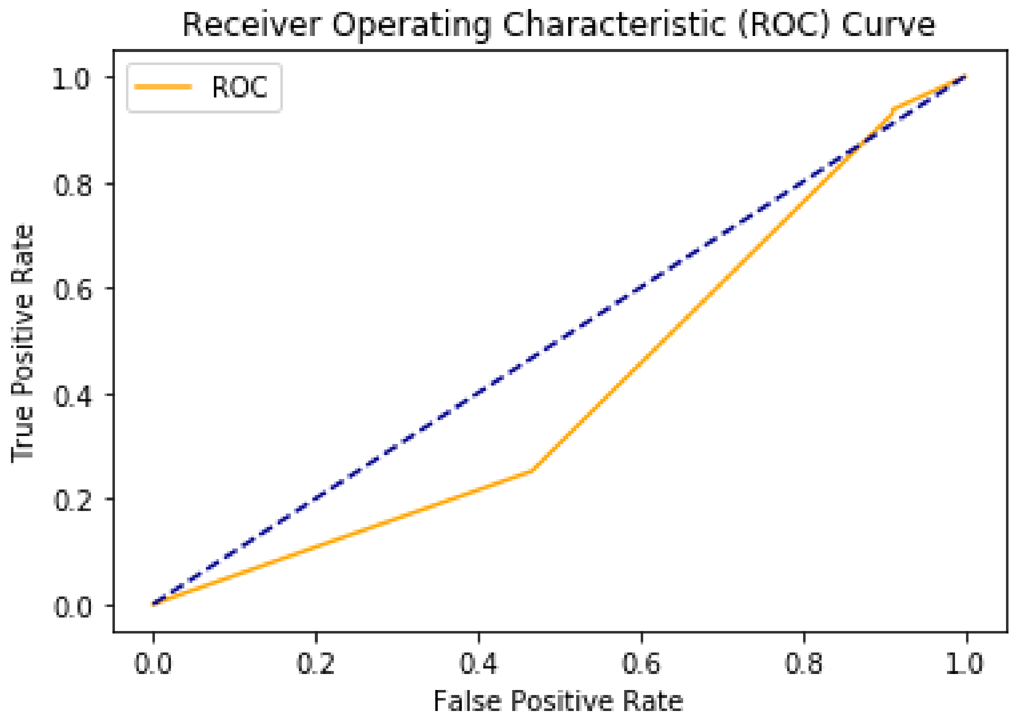

```
AUC2: 0.09
optimal_threshold2: 2.00
[2.         1.         0.85596766 0.        ]
```



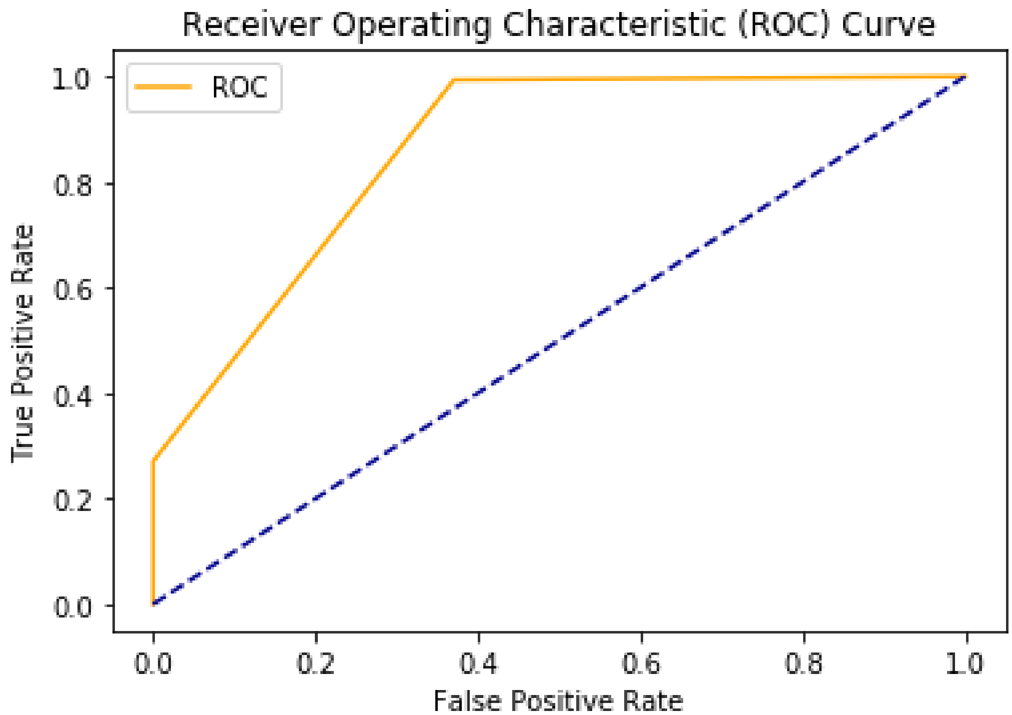
The score cutoff 10.00 for Reference A community 0 with pplacer_stats edpl compared with test B: 10.00
data_set is True
AUC1: 0.86
optimal_threshold1: 0.90
[2.   1.   0.9018491 0.   ]



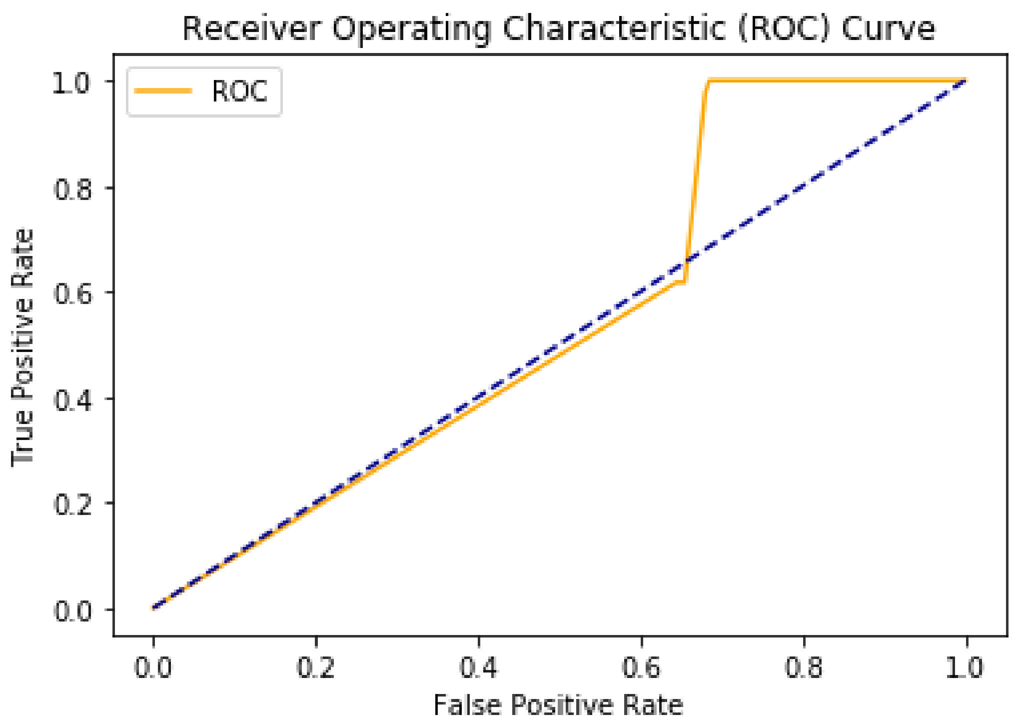
AUC2: 0.41
optimal_threshold2: 0.20
[2.   1.   0.9018491 0.2   0.   ]



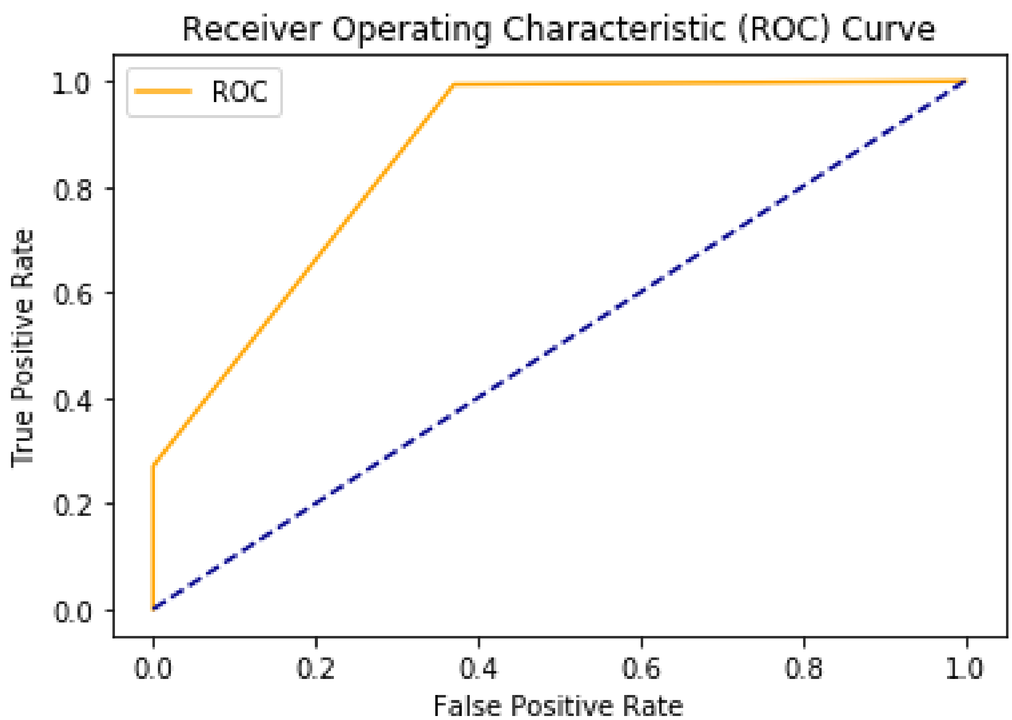

```
 The score cutoff 10.00 for Reference A community 0 with pplacer_stats edpl
compared with test C: 10.00
data_set is True
AUC1: 0.86
optimal_threshold1: 0.88
[2.         1.         0.88408218 0.        ]
```



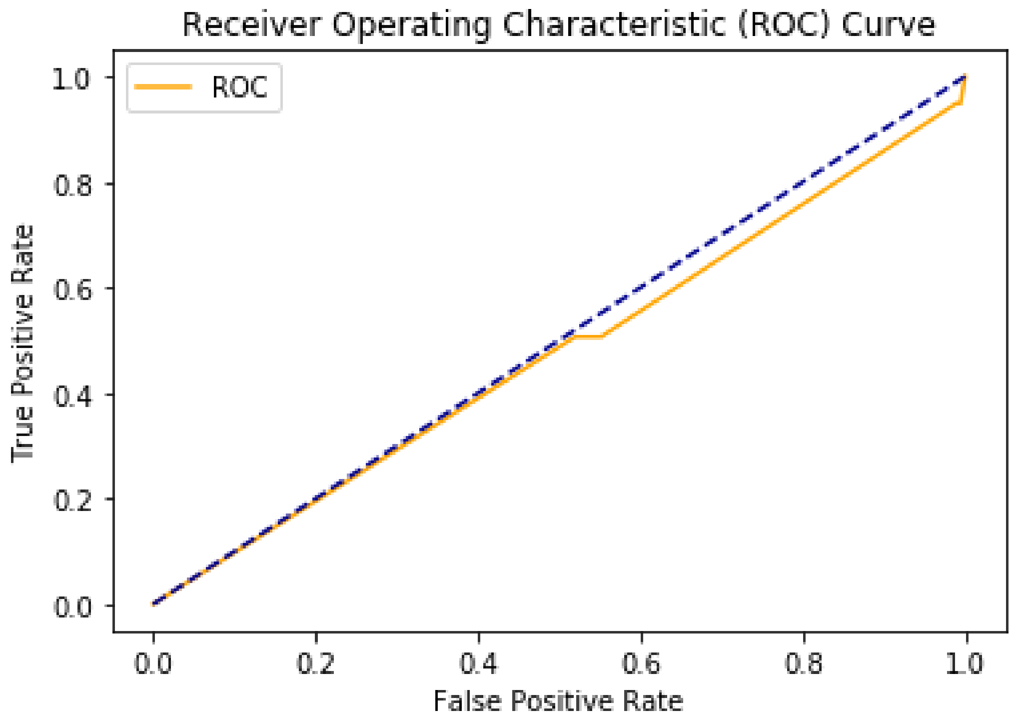

```
AUC2: 0.55
optimal_threshold2: 0.10
[2.         1.         0.9        0.88408218 0.1        0.        ]
```



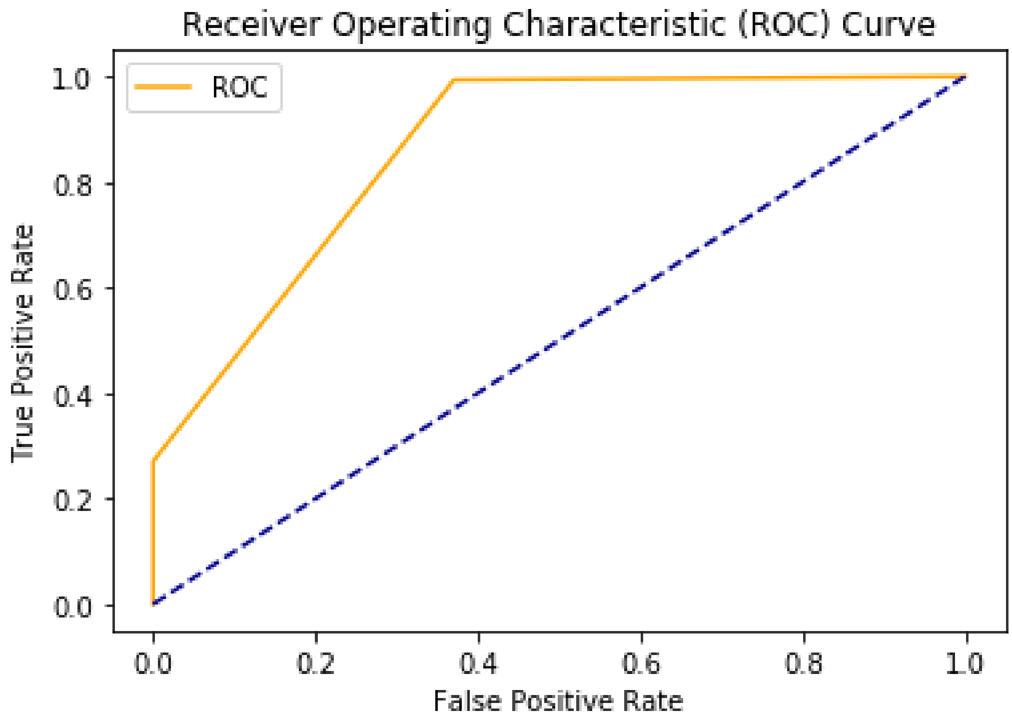
```
 The score cutoff 10.00 for Reference A community 0 with pplacer_stats edpl
compared with test D: 10.00
data_set is True
AUC1: 0.86
optimal_threshold1: 0.87
[2.         1.         0.87262189 0.        ]
```



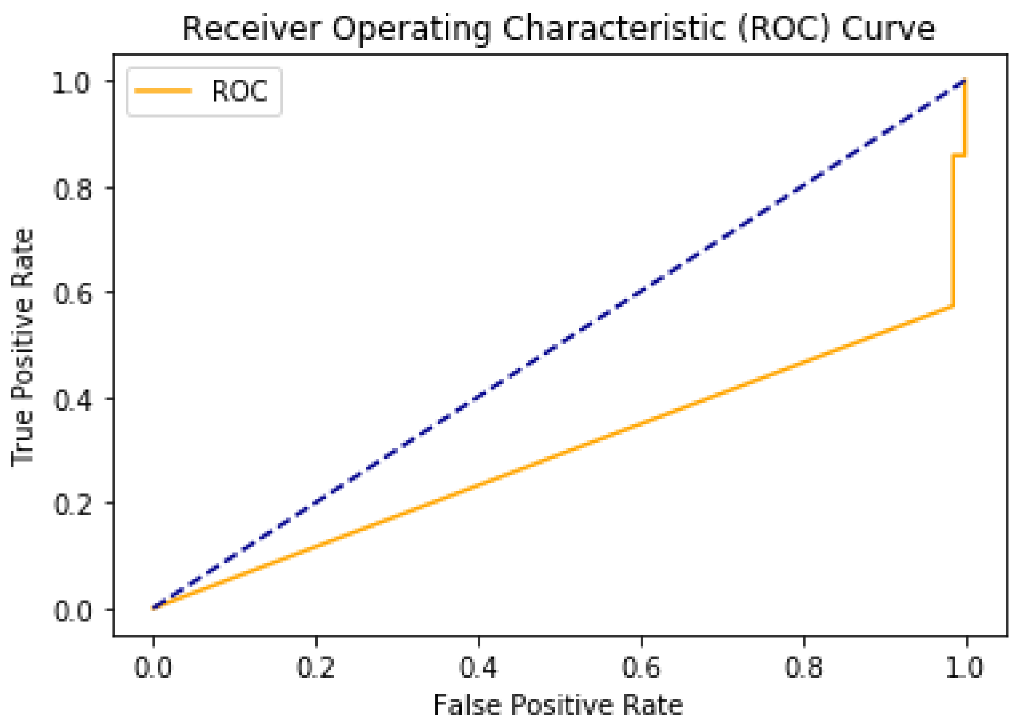

```
AUC2: 0.48
optimal_threshold2: 2.00
[2.         1.         0.9        0.87262189 0.1        0.        ]
```




```
 The score cutoff 10.00 for Reference A community 0 with pplacer_stats edpl
compared with test E: 10.00
data_set is True
AUC1: 0.86
optimal_threshold1: 0.89
[2.         1.         0.98710937 0.89088363 0.        ]
```

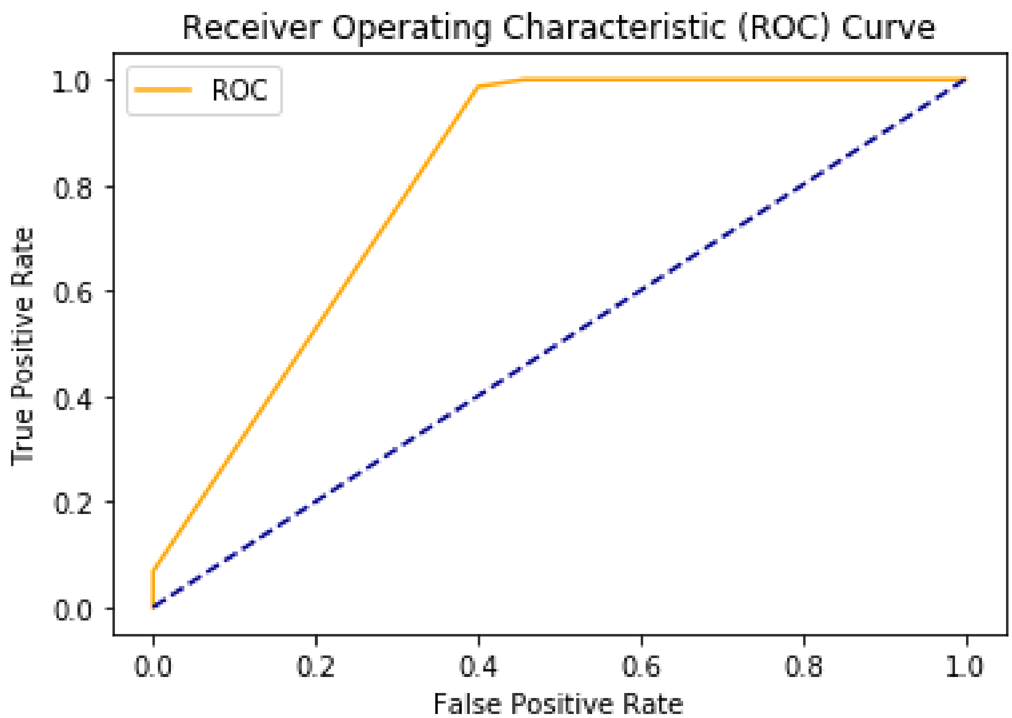



```
AUC2: 0.29
optimal_threshold2: 2.00
[2.         1.         0.89088363 0.7        0.1        0.        ]
```

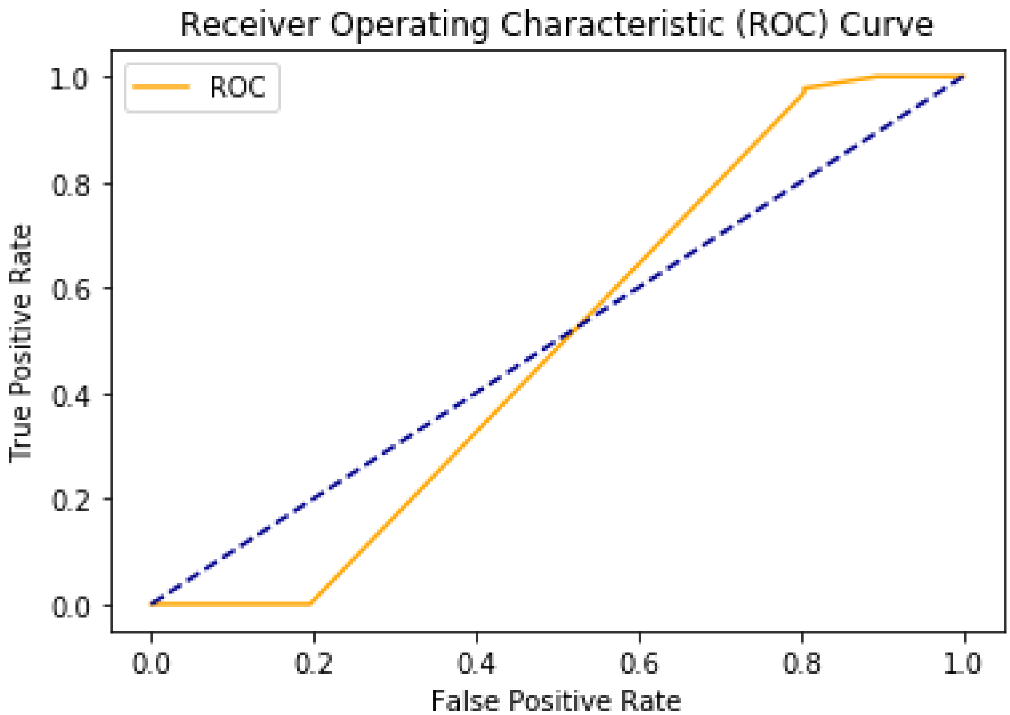


[177]:
```
plot_roc_curve_microbiome_test2(pplacer_ref_list =␣
↪['A'],pplacer_stats_list=['_adcl_log'],community_list=['0'],cutoff_list=['-4.
↪00'], test_data_list=['B','C','D','E'],testOption=True, scoreOption=False)
```

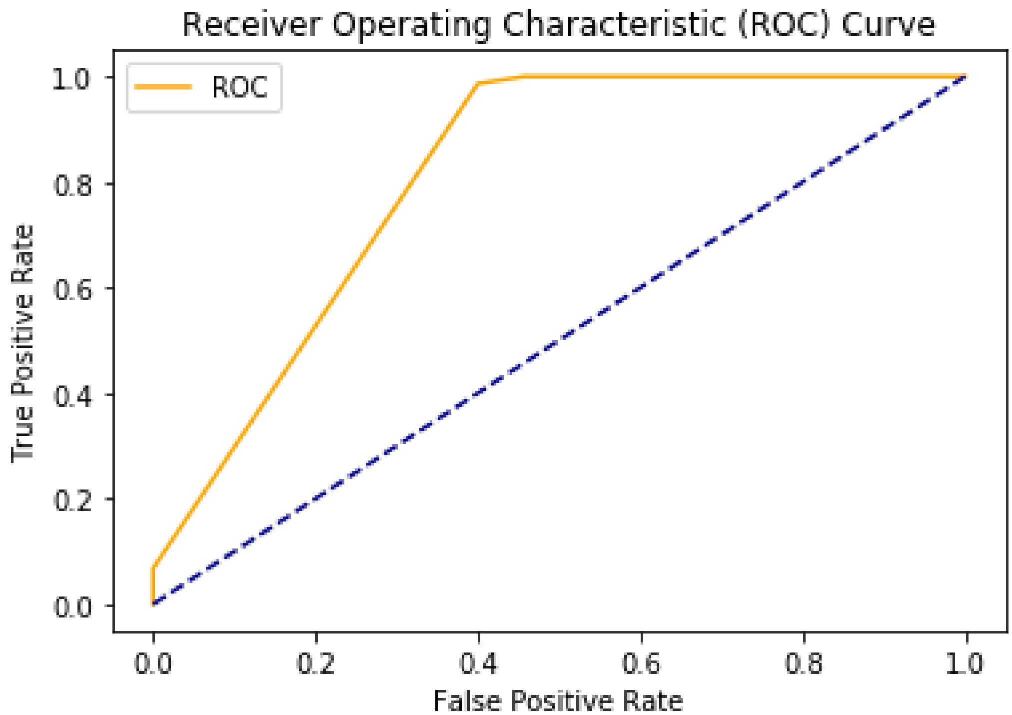
```
 The pplacer_stats_cutoff -4.00 for Reference A community 0 pplacer_stats
adcl_log compared with test B: -4.00
data_set is True
AUC1: 0.81
optimal_threshold1: 0.92
[2.         1.         0.92196452 0.75654762 0.        ]
```



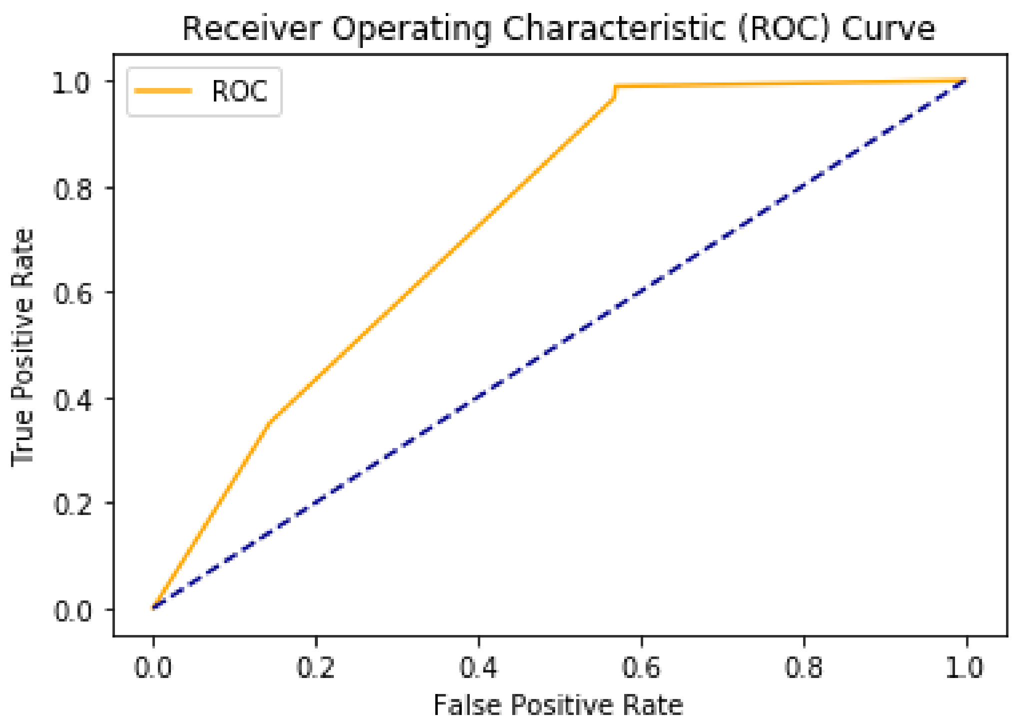

```
AUC2: 0.49
optimal_threshold2: 0.90
[2.         1.         0.92196452 0.9        0.75654762 0.        ]
```



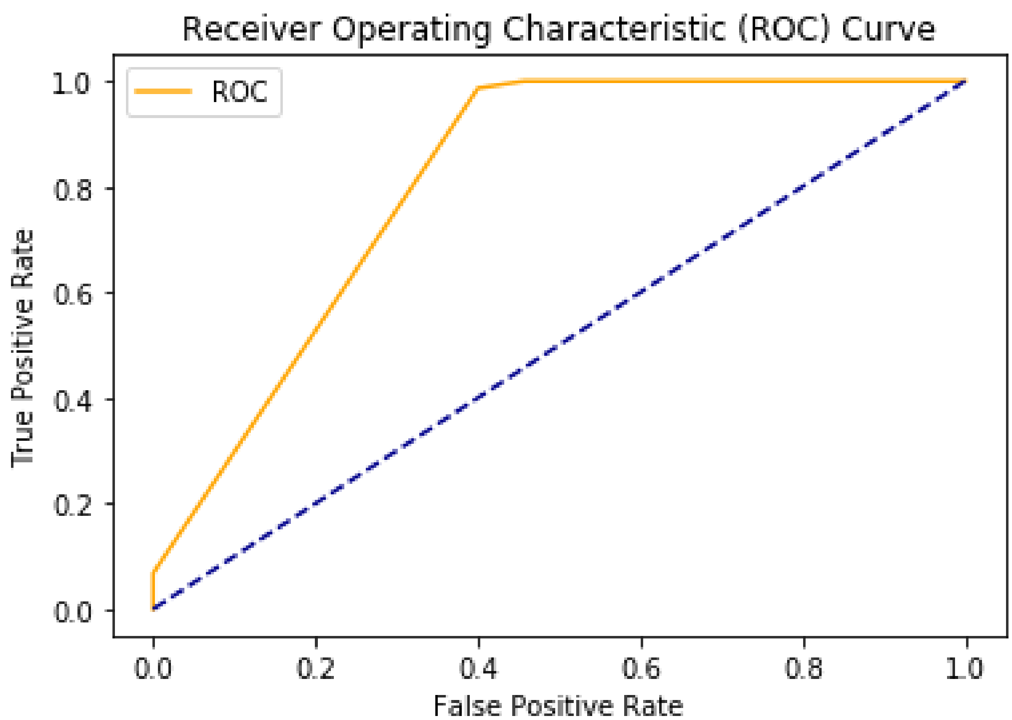

```
 The pplacer_stats_cutoff -4.00 for Reference A community 0 pplacer_stats
adcl_log compared with test C: -4.00
data_set is True
AUC1: 0.81
optimal_threshold1: 0.92
[2.         1.         0.92433893 0.6797619  0.        ]
```



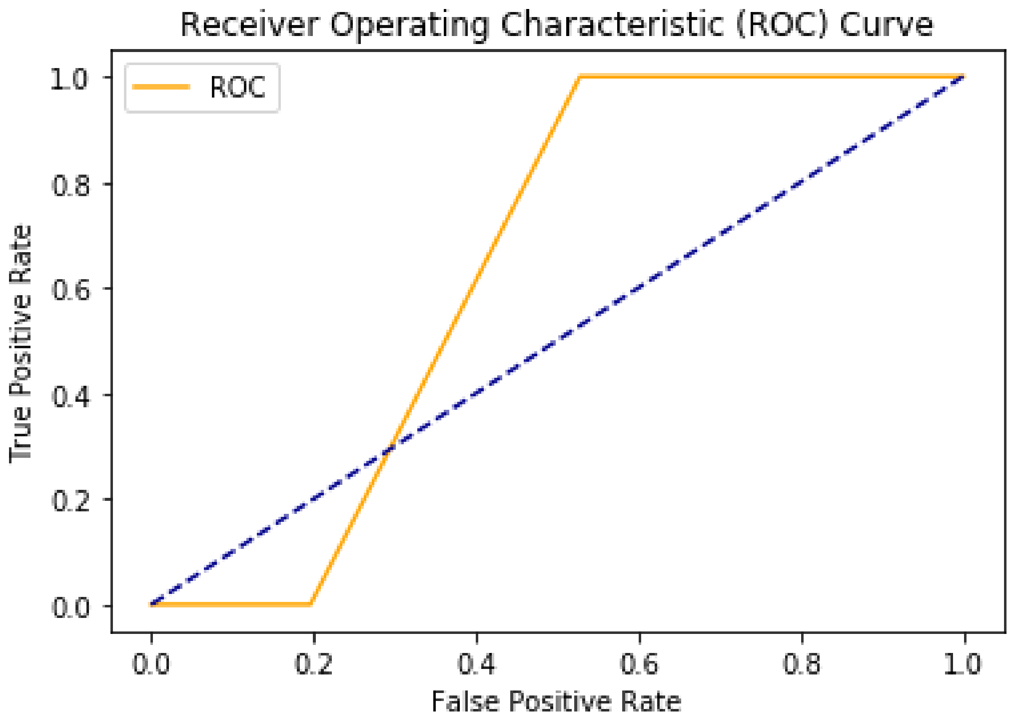

```
AUC2: 0.73
optimal_threshold2: 0.68
[2.         1.         0.92433893 0.6797619  0.        ]
```



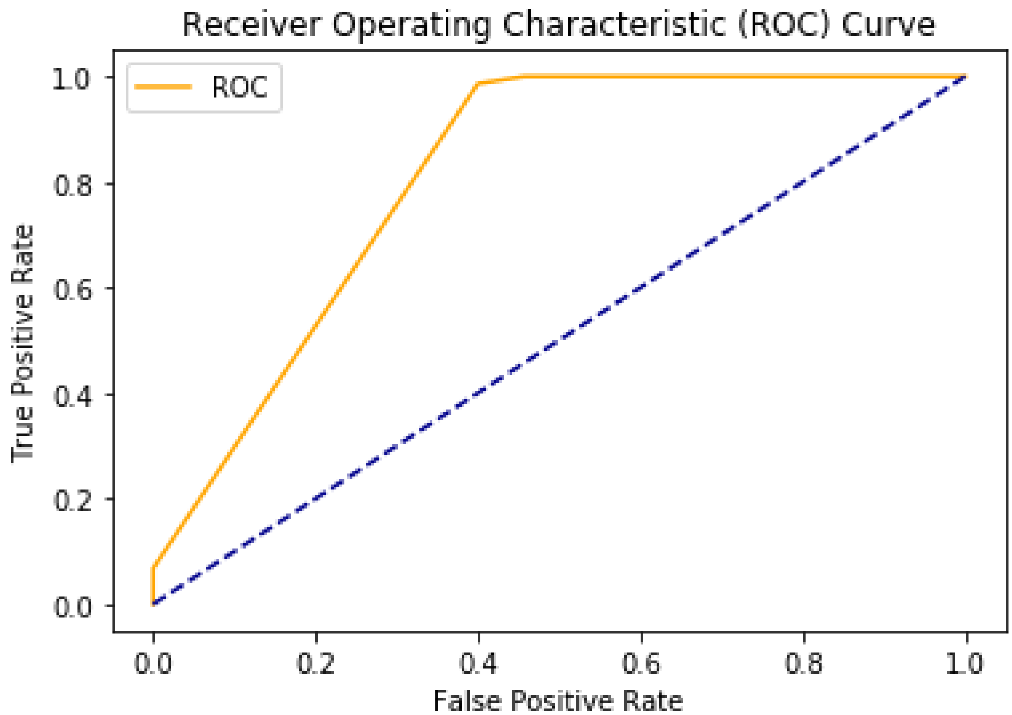
 The pplacer_stats_cutoff -4.00 for Reference A community 0 pplacer_stats
adcl_log compared with test D: -4.00
data_set is True
AUC1: 0.81
optimal_threshold1: 0.92
[2.   1.   0.91710262 0.5984632 0.   ]



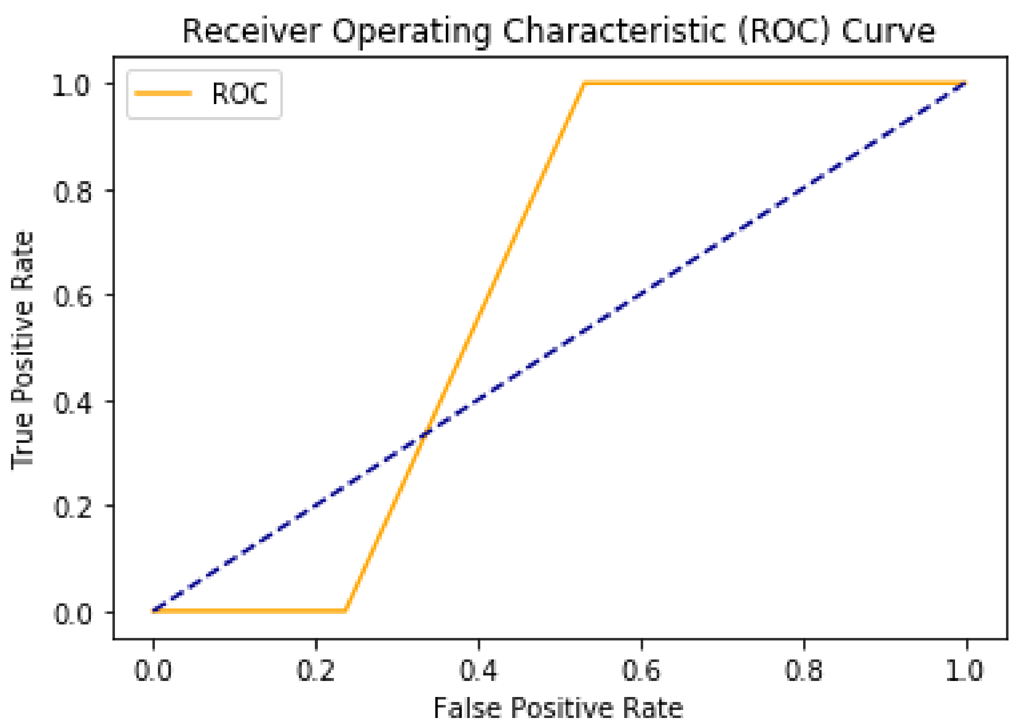
AUC2: 0.64
optimal_threshold2: 0.92
[2.   1.   0.91710262 0.5984632 0.   ]




```
 The pplacer_stats_cutoff -4.00 for Reference A community 0 pplacer_stats
adcl_log compared with test E: -4.00
data_set is True
AUC1: 0.81
optimal_threshold1: 0.93
[2.         1.         0.93096139 0.69437908 0.        ]
```

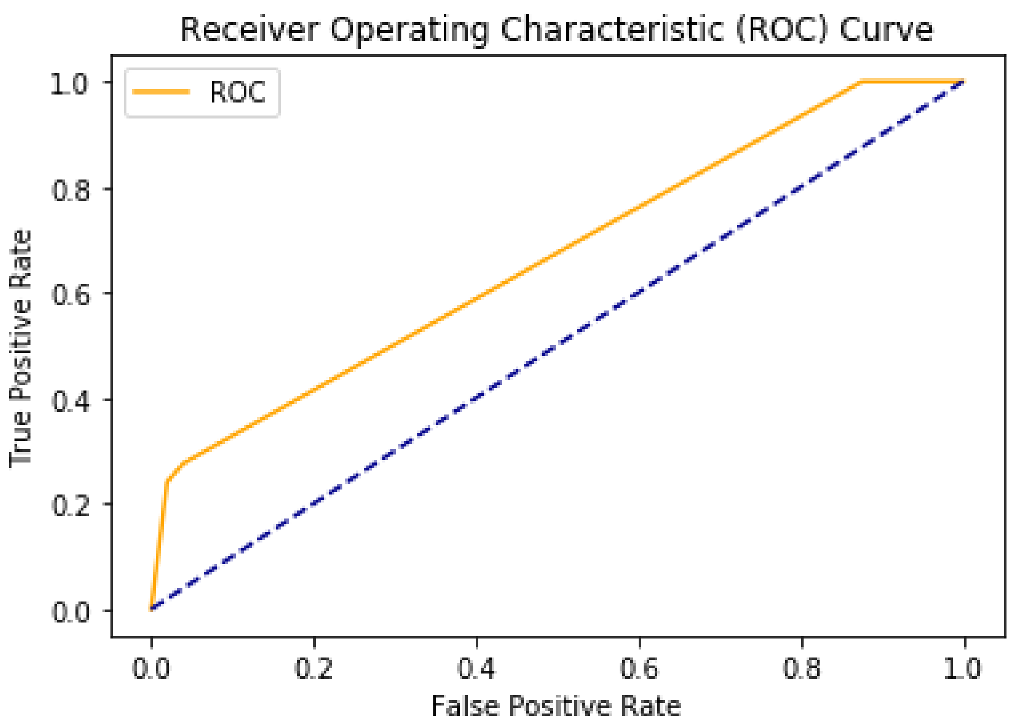



```
AUC2: 0.62
optimal_threshold2: 0.93
[2.         1.         0.93096139 0.69437908]
```

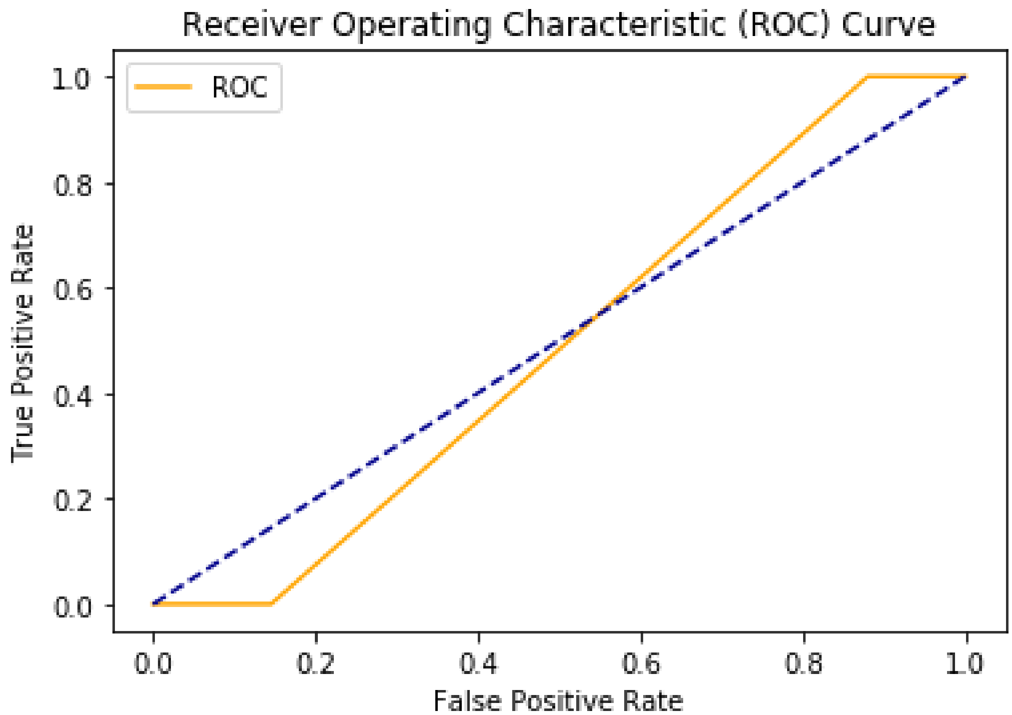


[136]:
```
plot_roc_curve_microbiome_test2(pplacer_ref_list =␣
↪['A'],pplacer_stats_list=['_adcl_log'],community_list=['0'],cutoff_list=['25%'],␣
↪test_data_list=['B','C'],testOption=True, scoreOption=False)
```

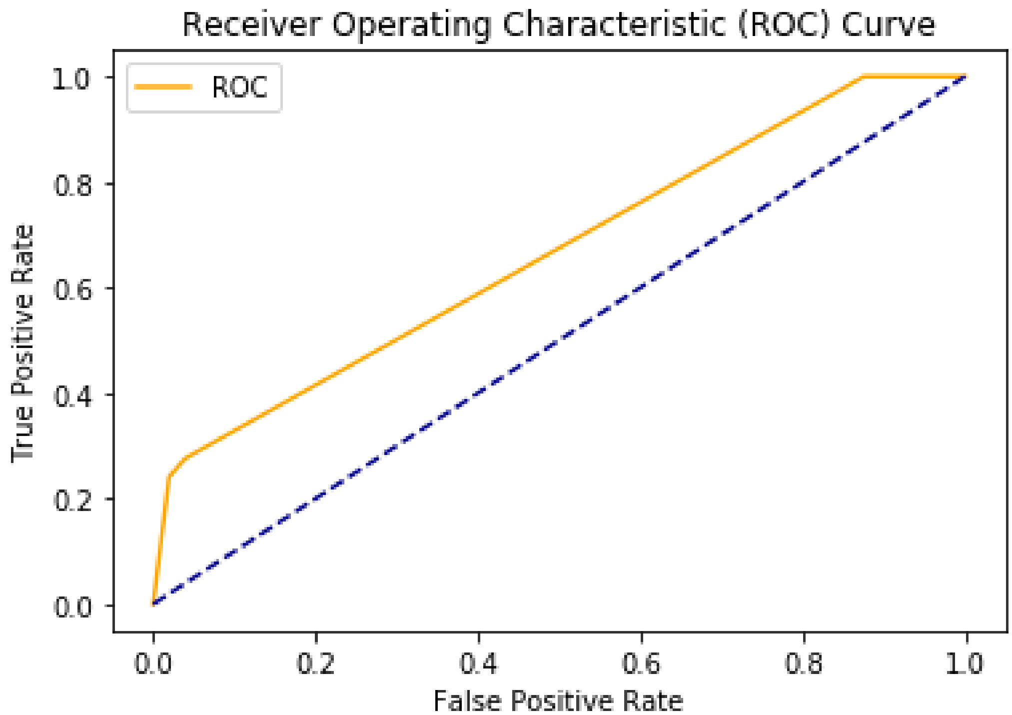
```
 The pplacer_stats_cutoff 25% for Reference A community 0 pplacer_stats adcl_log
compared with test B: -5.22
data_set is True
AUC1: 0.67
optimal_threshold1: 0.46
```



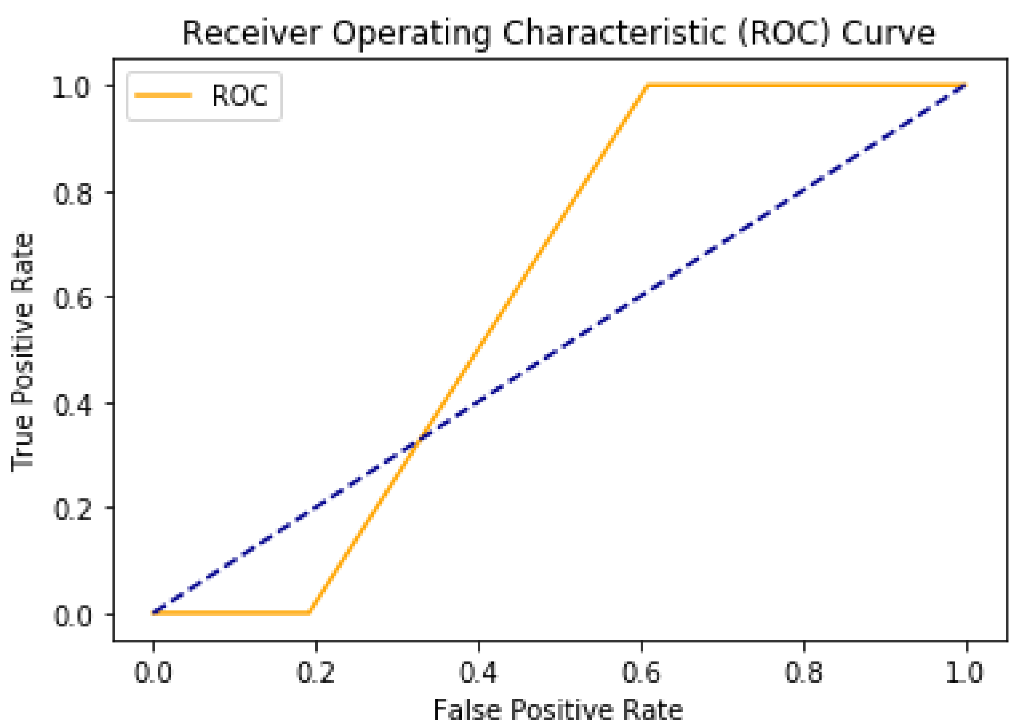

```
AUC2: 0.49
optimal_threshold2: 0.15
```



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: 0.42

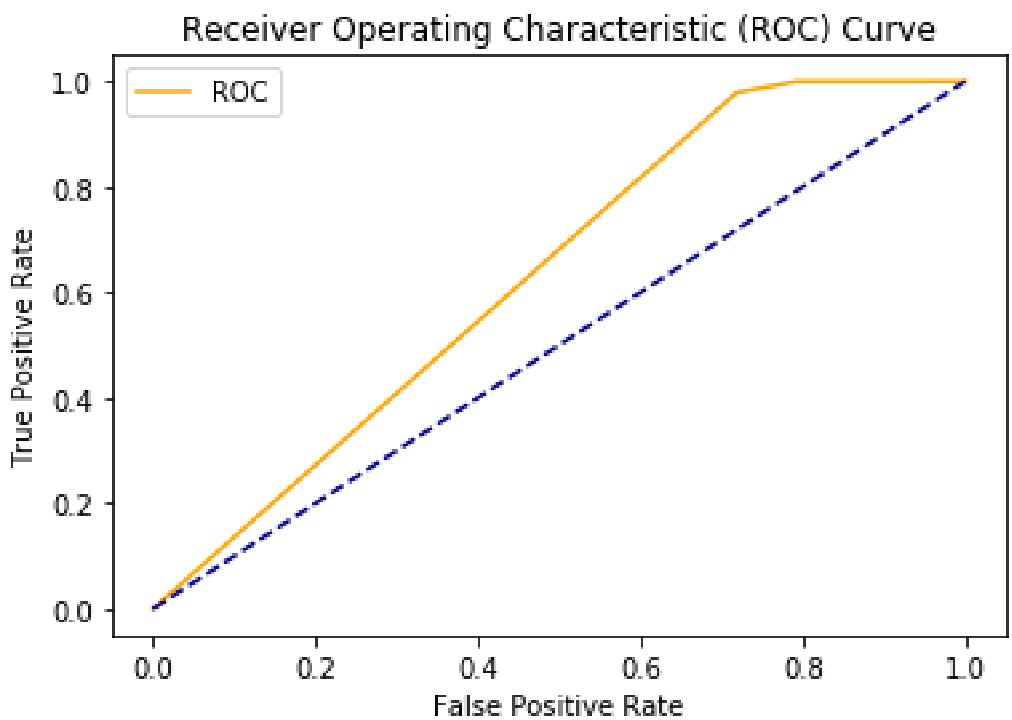



```
AUC2: 0.60
optimal_threshold2: 0.14
```

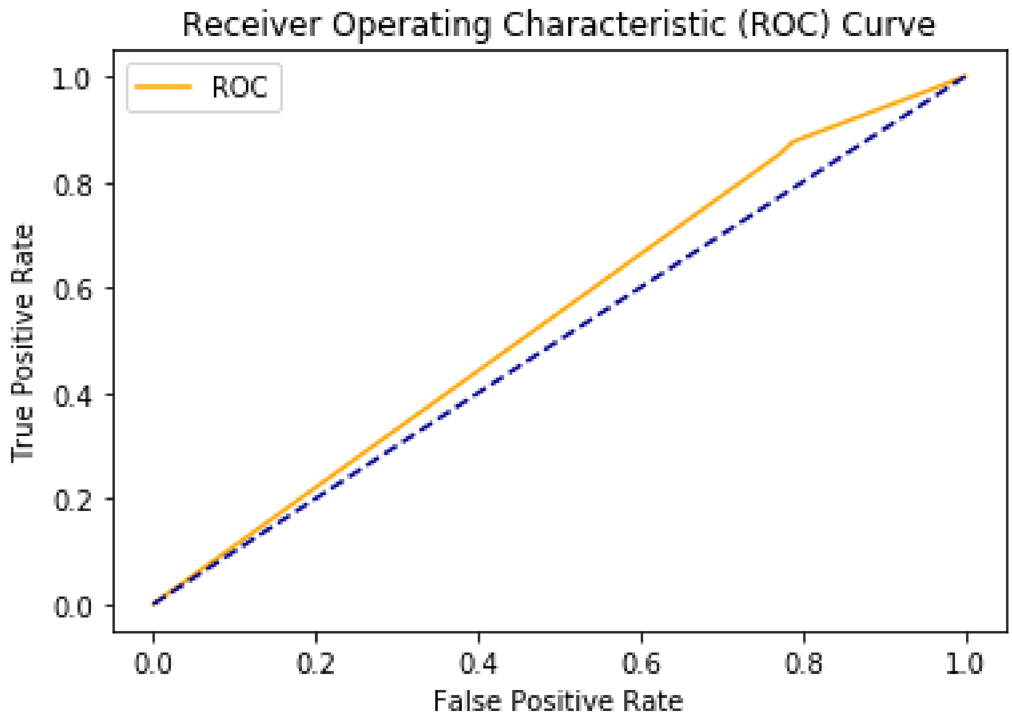


[128]:
```
plot_roc_curve_microbiome_test2(pplacer_ref_list =
    ['B'],pplacer_stats_list=['_adcl_log'],community_list=['0'],cutoff_list=['25%'],
    test_data_list=['A','C'],testOption=True, scoreOption=False)
```

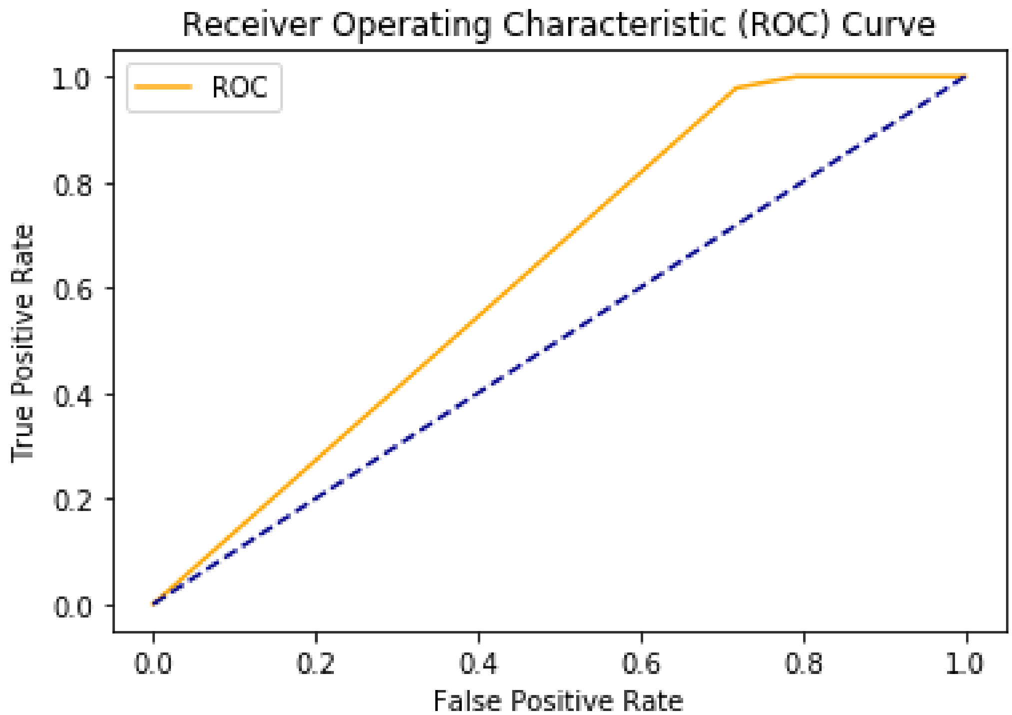
```
 The pplacer_stats_cutoff 25% for Reference B community 0 pplacer_stats adcl_log
compared with test A: -5.15
data_set is True
AUC1: 0.63
optimal_threshold1: 0.23
```



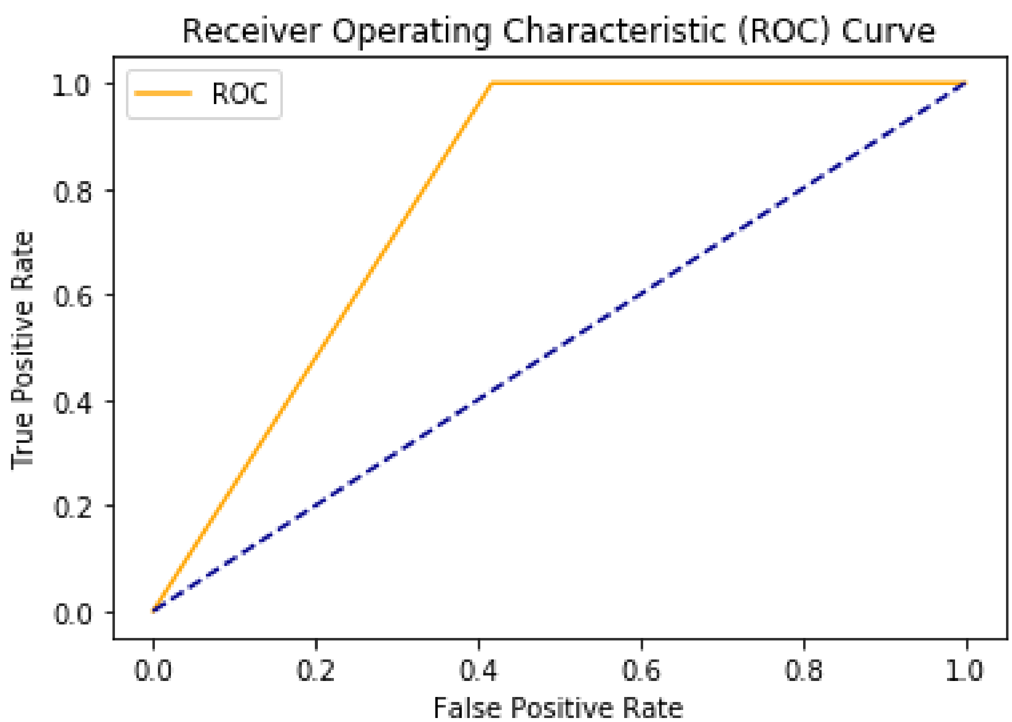

```
AUC2: 0.54
optimal_threshold2: 0.11
```



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: 0.23




```
AUC2: 0.79
optimal_threshold2: 0.23
```



[96]:
```
# print("the head for df is {}".format(df.head)+ " the columns of the df is {}".
 ↪format(df.columns))
#
```

[33]:
```
# df['A0'].describe(), df['B0'].describe(), df['C0'].describe(), df['D0'].
 ↪describe(),df['E0'].describe()
```

[34]:
```
# for community in ['A','B','C','D','E']:
#     for i in range(10):
#             print(df[community+str(i)].describe())
```

[35]:
```
df_0 = df
```

[36]:
```
# plot_pplacer('90')
```

[37]:
```
# plotScatter('B','0')
```

[38]:
```
# plotScatterRef('_adcl_log','0')
```

[39]:
```
# plot_pplacer('_adcl_log')
```

[40]:
```
df['A_adcl_log'].describe()
```

[40]:
```
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
```

[41]:
```
# plot_pplacer('0')
```

[42]:
```
df['A0'].describe()
```

[42]:
```
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
```

[43]:
```
df1 = df[(df['A0']>10)]
```

[44]:
```
df1['A0'].describe()
```

[44]:
```
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
```

[45]:
```
99/605
```

[45]:
```
0.16363636363636364
```

[46]:
```
df1['B0'].describe()
```

[46]:
```
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
```

[47]:
```
df2=df[['seqID','A0','B0', 'C0','D0','E0']].dropna()
```

[48]:
```
df3 = df2[(df2['A0']>10) & (df2['B0']>10)  & (df2['C0']>10)  & (df2['D0']>10) &␣
↪(df2['E0']>10)]
```

[49]:
```
# df2.describe(), df3.describe()
```

[50]:
```
df3
```

[50]:
```
                                          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
```

[51]:
```
2/605
```

[51]:
```
0.003305785123966942
```

[52]:
```
dfc90 = df[(df['A90']>10) & (df['B90']>10)  & (df['C90']>10)  & (df['D90']>10)␣
↪& (df['E90']>10)]
```

[53]:
```
dfc90['B0'].describe()
```

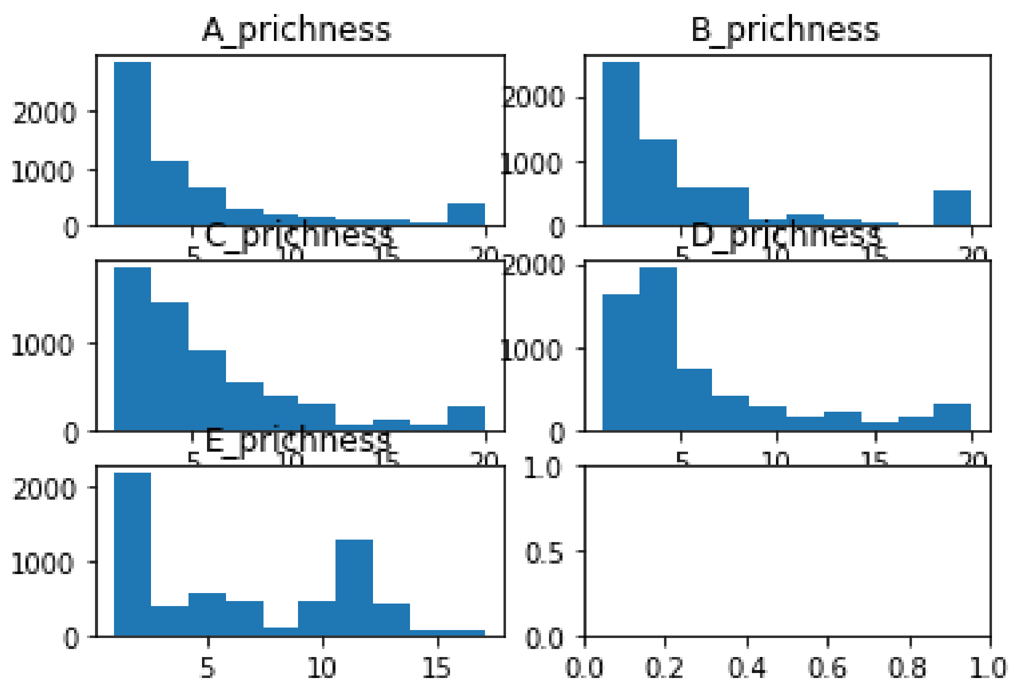
[53]:
```
count    0.0
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
```

```
75%       NaN
max       NaN
Name: B0, dtype: float64
```

[54]:
```
df[(df.community=='CC11CM0')]['C_adcl_log'].dropna().describe()
```

[54]:
```
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
```

[55]:
```
plot_pplacer('_prichness')
```



[56]:
```
df['A_prichness'].describe()
```

[56]:
```
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
```

[57]:
```
df[df.A0>10].A0.count()
```

[57]:
```
99
```

[58]:
```
df[df.A0>10].A0.count()/df.A0.count()
```

[58]:
```
0.16363636363636364
```

[59]:
```
# df.head()
```

[60]:
```
df.A_adcl.count()
```

[60]:
```
5974
```

[61]:
```
d={"a":1, "b":2}
```

[62]:
```
d
```

[62]:
```
{'a': 1, 'b': 2}
```

[63]:
```
dd = pd.Series(d, name='score')
```

[64]:
```
dd.index.name="community"
```

[65]:
```
dd.reset_index()
```

[65]:
```
  community  score
0         a      1
1         b      2
```

[66]:
```
"CC11CM"+str(0)
```

[66]:
```
'CC11CM0'
```

[67]:
```
c0=df['A0'][df['community']=='CC11CM0']
```

[68]:
```
per=c0[c0>10].count()/c0.count()
```

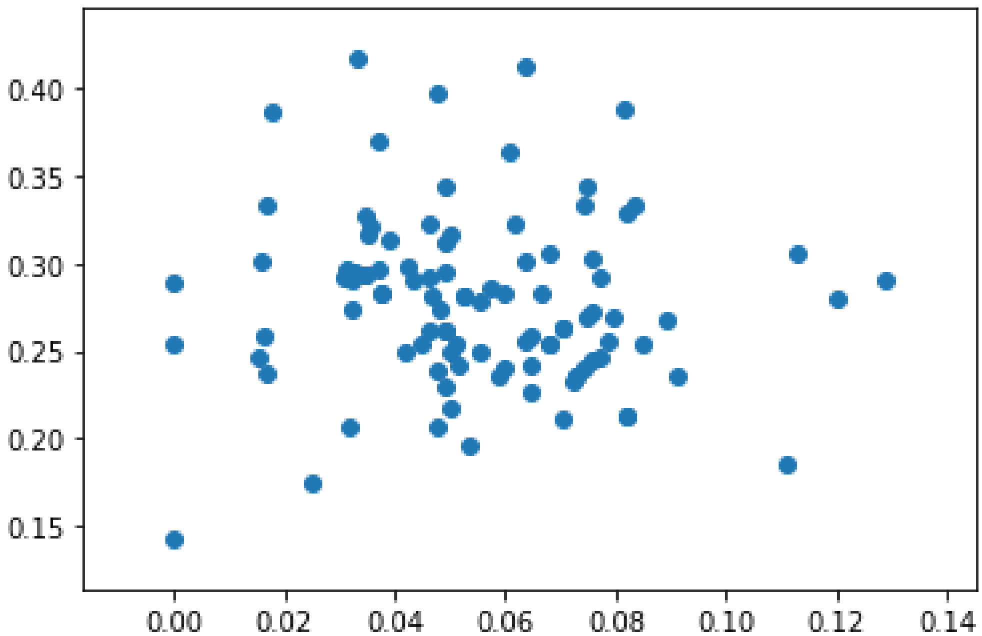
[69]:
```
def generateScore(stats, referenceID,scorecutoff,statscutoff):
    d1={}
    d2={}
    for i in range(100):
        values = df[referenceID+str(i)][df.community=='CC11CM'+str(i)]
        statsvalues = df[stats][df['community']=='CC11CM'+str(i)]
        d1['CC11CM'+str(i)] = values[values>scorecutoff].count()/values.count()
        d2['CC11CM'+str(i)] =statsvalues[statsvalues>statscutoff].count()/
 ↪statsvalues.count()
    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)
```

[70]:
```
dt=generateScore('A_adcl', 'A', 10, 0.001)
```

[71]:
```
plt.scatter(dt.A, dt.A_adcl)
```

[71]: <matplotlib.collections.PathCollection at 0x1efb8cd0>



[72]:
```
t=[]
for referenceID in ['A','B','C','D','E']:
    t.append(generateScore('A_adcl', referenceID, 10, 0.001))
```

[73]:
```
t[0].head()
```

[73]:

	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

[74]:
```
tt = pd.concat([t[0],t[1],t[2],t[3],t[4]], axis=1)
```

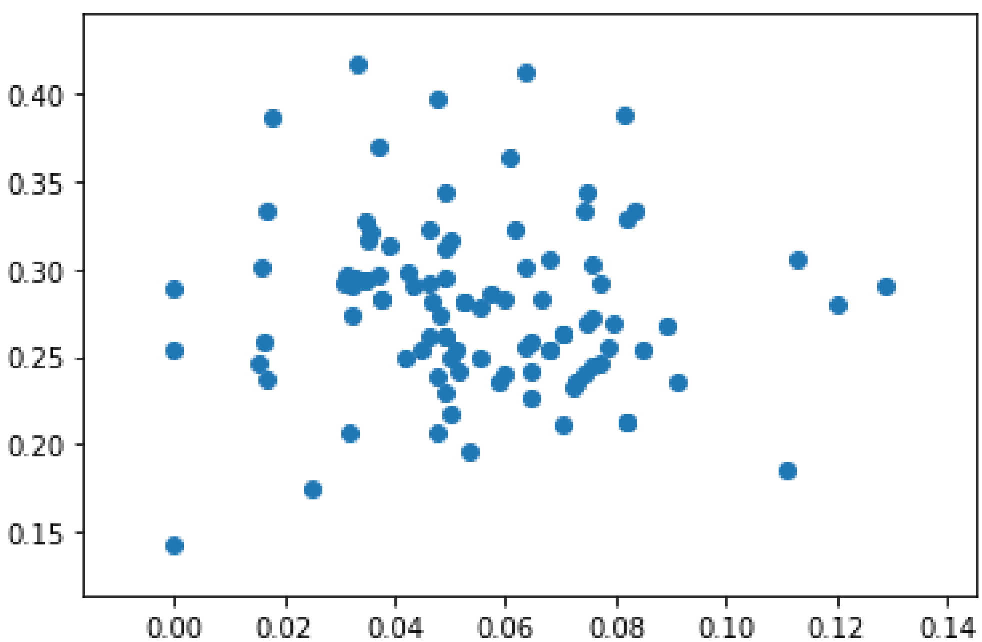
[75]:
```
def generateScoreu(stats, referenceID,scorecutoff,statscutoff):
    d1={}
    d2={}
    for i in range(100):
        values = df[referenceID+str(i)][df.community=='CC11CM'+str(i)]
```

```
        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.count()
    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)
```

[76]:
```
dtu=generateScoreu('_adcl', 'A', 10, 0.001)
```

[77]:
```
plt.scatter(dtu.A, dtu.A_adcl)
```

[77]: <matplotlib.collections.PathCollection at 0x1f294fd0>



[ ]:

[78]:
```
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]))
```

[ ]:

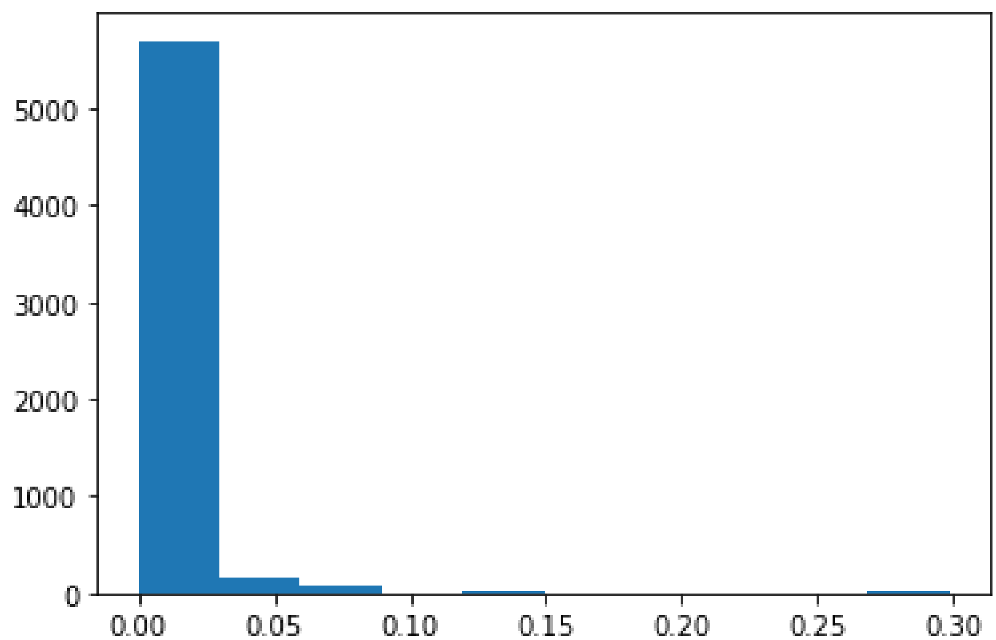
[79]:
```
ttt = pd.
↪concat([t[0],t[1],t[2],t[3],t[4],t[5],t[6],t[7],t[8],t[9],t[10],t[11],t[12],t[13],t[14],t[1
↪axis=1)
ttt=ttt.loc[:, ~ttt.columns.duplicated()]
```

[80]:
```
# ttt.describe()
```

[81]:
```
ttt.to_csv("community-based.csv")
```

[82]:
```
plt.hist(df.A_mindistl)
```

[82]:
```
(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>)
```



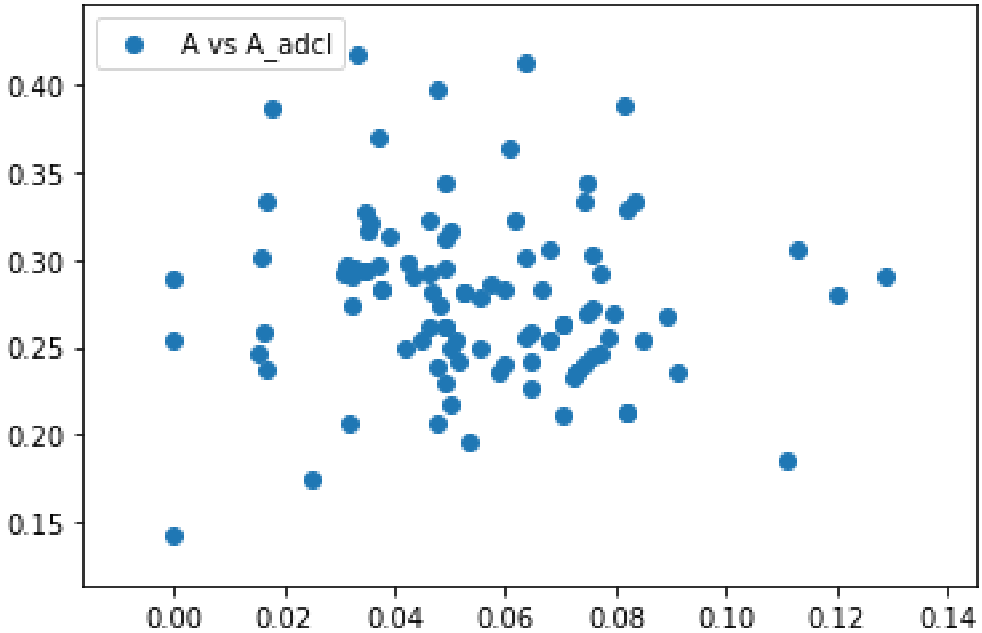
[83]:
```
dp =pd.read_csv("community-based.csv", index_col=0)
```

[84]:
```
dp.describe()
```

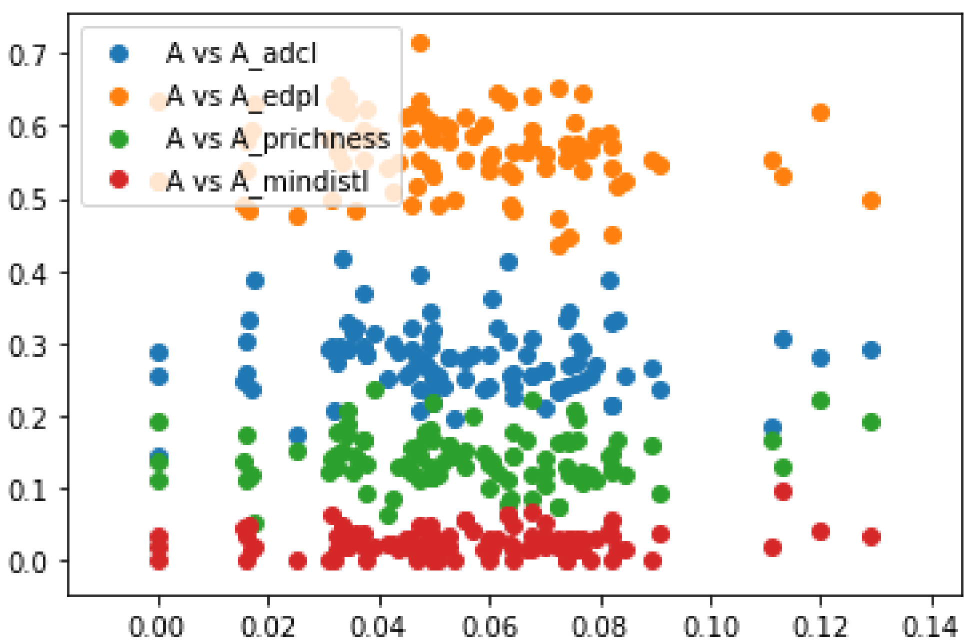
[84]:
```
                A      A_adcl           B      B_adcl           C      C_adcl  \
count  100.000000  100.000000  100.000000  100.000000  100.000000  100.000000
mean     0.055034    0.277586    0.116043    0.557102    0.318386    0.799584
std      0.024318    0.049060    0.032415    0.054352    0.048526    0.044362
```

```
min      0.000000    0.142857    0.051724    0.406780    0.216667    0.666667
25%      0.037736    0.246154    0.095238    0.516532    0.285119    0.773585
50%      0.052178    0.275986    0.112007    0.563333    0.315789    0.800000
75%      0.072530    0.301587    0.136310    0.600000    0.346392    0.830769
max      0.129032    0.416667    0.209677    0.682540    0.448980    0.888889

                D      D_adcl           E      E_adcl  ...  A_prichness  \
count  100.000000  100.000000  100.000000  100.000000  ...   100.000000
mean     0.647292    0.962263    0.229600    0.989559  ...     0.140234
std      0.051736    0.017520    0.045910    0.008343  ...     0.035764
min      0.491803    0.916667    0.097222    0.975000  ...     0.052632
25%      0.612455    0.949788    0.193768    0.983051  ...     0.118395
50%      0.649561    0.966667    0.229508    0.984615  ...     0.137147
75%      0.682738    0.978723    0.261943    1.000000  ...     0.164801
max      0.786885    1.000000    0.339623    1.000000  ...     0.235294

       B_prichness  C_prichness  D_prichness  E_prichness  A_mindistl  \
count   100.000000   100.000000   100.000000   100.000000  100.000000
mean      0.147809     0.142149     0.160060     0.304304    0.025369
std       0.041895     0.037576     0.034324     0.042562    0.018591
min       0.042553     0.056604     0.080645     0.190476    0.000000
25%       0.123077     0.120690     0.144585     0.274194    0.015873
50%       0.146257     0.140351     0.155048     0.306452    0.020221
75%       0.179410     0.157540     0.180082     0.338524    0.034044
max       0.240741     0.254545     0.244898     0.393443    0.096774

       B_mindistl  C_mindistl  D_mindistl  E_mindistl
count  100.000000  100.000000  100.000000  100.000000
mean     0.114361    0.097589    0.198869    0.290481
std      0.030794    0.030017    0.042808    0.047167
min      0.037037    0.033333    0.109091    0.156863
25%      0.095013    0.079132    0.172414    0.265789
50%      0.111111    0.096774    0.196400    0.298507
75%      0.134615    0.114754    0.229823    0.323077
max      0.187500    0.203390    0.333333    0.387755

[8 rows x 25 columns]
```

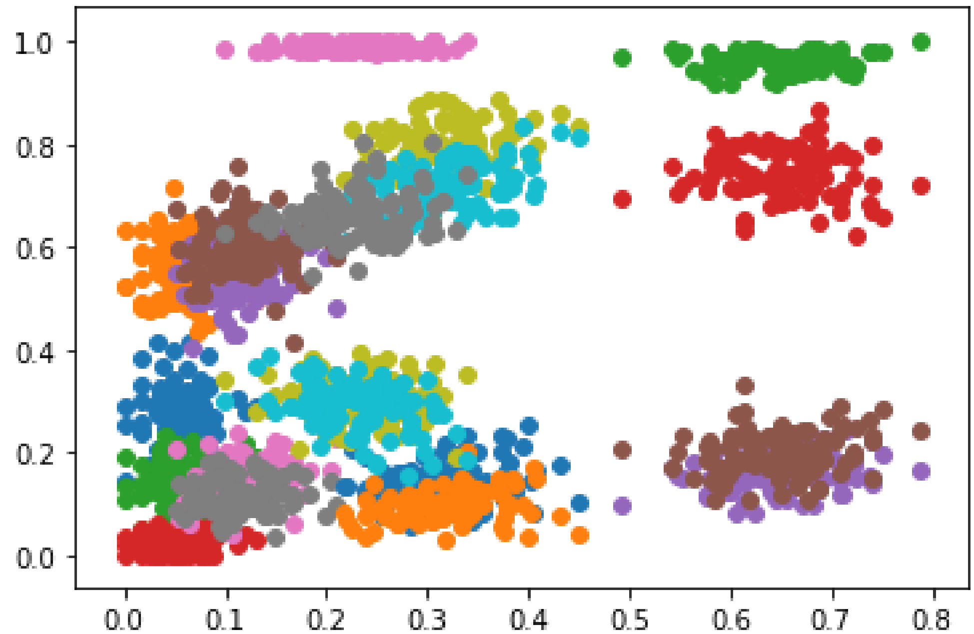
[85]:
```
for score in ['A','B','C','D','E'][0:1]:
    for stats in ['_adcl', '_edpl','_prichness','_mindistl'][0:1]:
        plt.scatter(dp[score], dp[score+stats], label=score + ' vs '+␣
↪score+stats)
        plt.legend(loc='upper left')
        plt.show
```



[86]:
```
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
```



[87]:
```
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
```



[88]:
```
dp.describe()
```

[88]:

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

	D	D_adcl	E	E_adcl	...	A_prichness	\
count	100.000000	100.000000	100.000000	100.000000	...	100.000000	
mean	0.647292	0.962263	0.229600	0.989559	...	0.140234	
std	0.051736	0.017520	0.045910	0.008343	...	0.035764	
min	0.491803	0.916667	0.097222	0.975000	...	0.052632	
25%	0.612455	0.949788	0.193768	0.983051	...	0.118395	

```
50%      0.649561    0.966667    0.229508    0.984615  ...     0.137147
75%      0.682738    0.978723    0.261943    1.000000  ...     0.164801
max      0.786885    1.000000    0.339623    1.000000  ...     0.235294

       B_prichness  C_prichness  D_prichness  E_prichness  A_mindistl  \
count   100.000000   100.000000   100.000000   100.000000  100.000000
mean      0.147809     0.142149     0.160060     0.304304    0.025369
std       0.041895     0.037576     0.034324     0.042562    0.018591
min       0.042553     0.056604     0.080645     0.190476    0.000000
25%       0.123077     0.120690     0.144585     0.274194    0.015873
50%       0.146257     0.140351     0.155048     0.306452    0.020221
75%       0.179410     0.157540     0.180082     0.338524    0.034044
max       0.240741     0.254545     0.244898     0.393443    0.096774

       B_mindistl  C_mindistl  D_mindistl  E_mindistl
count  100.000000  100.000000  100.000000  100.000000
mean     0.114361    0.097589    0.198869    0.290481
std      0.030794    0.030017    0.042808    0.047167
min      0.037037    0.033333    0.109091    0.156863
25%      0.095013    0.079132    0.172414    0.265789
50%      0.111111    0.096774    0.196400    0.298507
75%      0.134615    0.114754    0.229823    0.323077
max      0.187500    0.203390    0.333333    0.387755

[8 rows x 25 columns]
```

[ ]: