pyCLAMs (CLassifiability Analysis Metrics)

An integrated toolkit for classifiability analysis

API demostration of pyCLAMs.py

load the library

In [1]:
```
import warnings
warnings.filterwarnings("ignore")

%run pyCLAMs.py
```

A list of all supported metrics

```
['classification.ACC',
 'classification.Kappa',
 'classification.F1_Score',
 'classification.Jaccard',
 'classification.Precision',
 'classification.Recall',
 'classification.CrossEntropy',
 'classification.AP',
 'classification.Brier',
 'classification.ROC_AUC',
 'classification.PR_AUC',
 'classification.BER',
 'correlation.IG',
 'correlation.IG.max',
 'correlation.r',
 'correlation.r.p',
 'correlation.r.max',
 'correlation.r.p.min',
 'correlation.rho',
 'correlation.rho.p',
 'correlation.rho.max',
 'correlation.rho.p.min',
 'correlation.tau',
 'correlation.tau.p',
 'correlation.tau.max',
 'correlation.tau.p.min',
 'test.ES',
 'test.ES.max',
 'test.ANOVA',
 'test.ANOVA.min',
 'test.ANOVA.min.log10',
 'test.ANOVA.F',
 'test.ANOVA.F.max',
 'test.MANOVA',
 'test.MANOVA.log10',
 'test.MANOVA.F',
 'test.MWW',
 'test.MWW.min',
 'test.MWW.min.log10',
 'test.MWW.U',
 'test.MWW.U.min',
 'test.KS',
 'test.KS.min',
 'test.KS.min.log10',
 'test.KS.D',
 'test.KS.D.max',
 'overlapping.F1.mean',
 'overlapping.F1.sd',
 'overlapping.F1v.mean',
 'overlapping.F1v.sd',
```

```
'overlapping.F2.mean',
'overlapping.F2.sd',
'overlapping.F3.mean',
'overlapping.F3.sd',
'overlapping.F4.mean',
'overlapping.F4.sd',
'neighborhood.N1',
'neighborhood.N2.mean',
'neighborhood.N2.sd',
'neighborhood.N3.mean',
'neighborhood.N3.sd',
'neighborhood.N4.mean',
'neighborhood.N4.sd',
'neighborhood.T1.mean',
'neighborhood.T1.sd',
'neighborhood.LSC',
'linearity.L1.mean',
'linearity.L1.sd']
```

In [2]:
```
# Total metric number
len(metrics_keys())
```

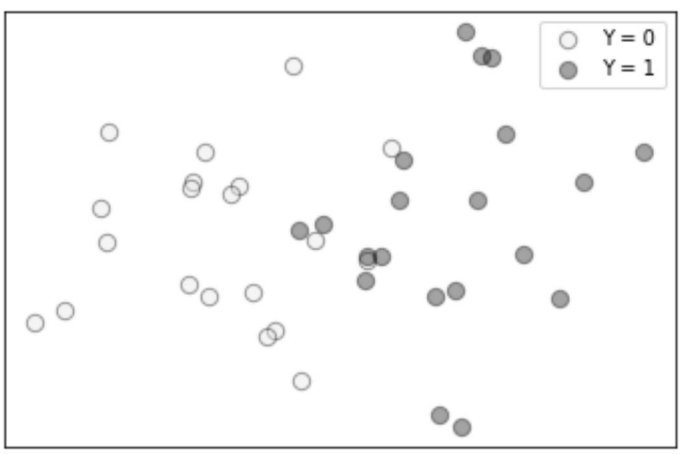
Out[2]: 68

The following demostrate main API functions

mvg

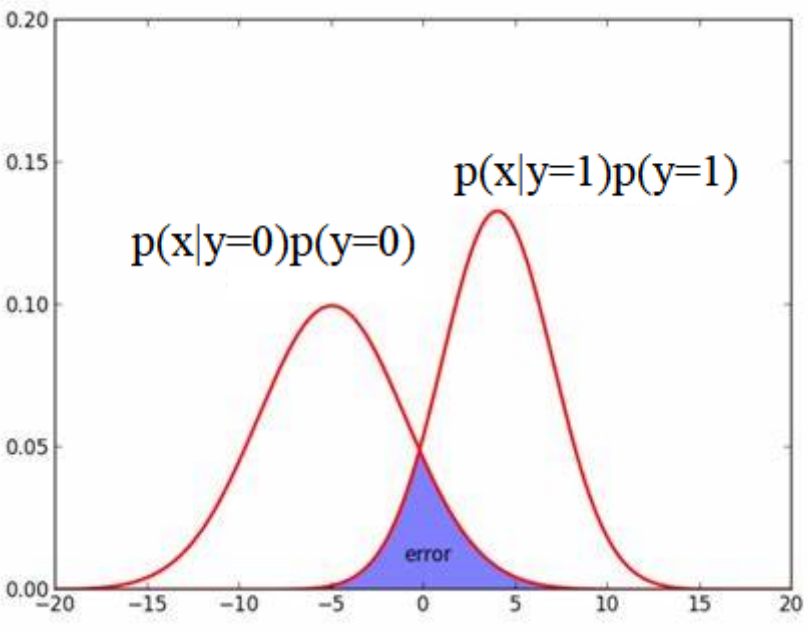
generates a 2D dataset with a specified between-class distance

In [4]:
```
X, y = mvg(md = 2)
X.shape, y.shape
plotComponents2D(X,y,labels = set(y))
```



BER

Bayes Error Rate.

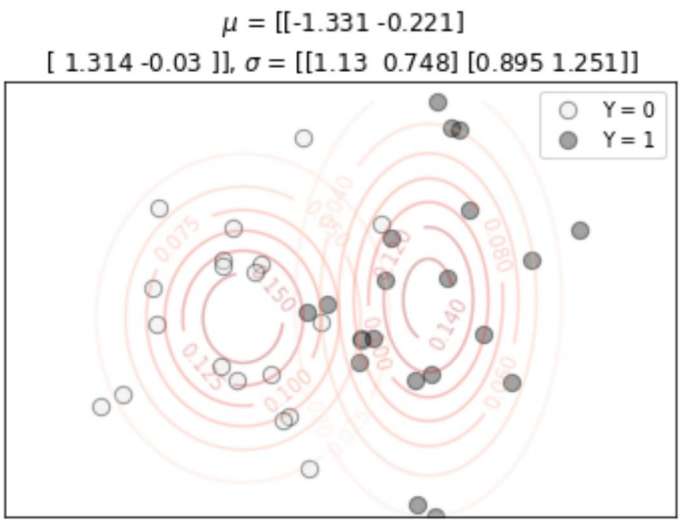


$$BER = 1 - E(\max_j P(Y = j|X)$$

It is the lowest possible test error rate in classification which is produced by the Bayes classifier. It is analogous to the irreducible error rate. Because of noise (inherently stochastic), the error incurred by the oracle prediction model from the true distribution p(x,y) is the Bayes error.

The Bayes optimal decision boundary will correspond to the point where two densities are equal.

In [5]:
```
ber, _ = BER(X,y, show = True)
```

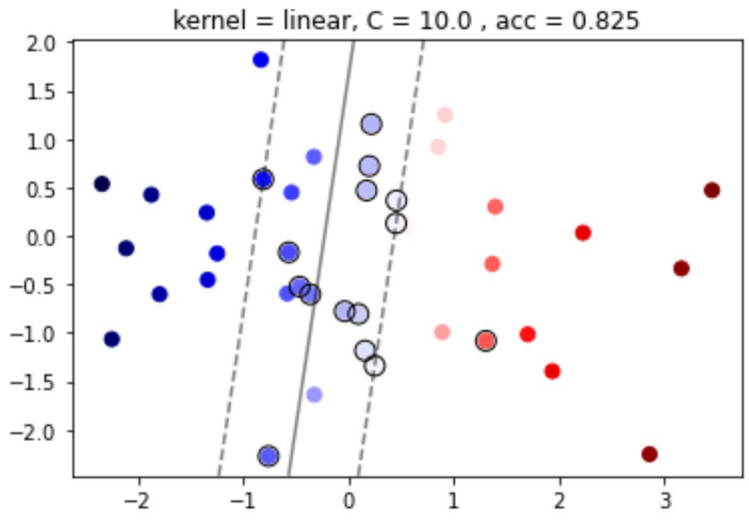


CLF

Classification Accuracy by a n-fold CV SVM

In [4]:
```
= CLF(X, y, verbose = True, show = True)
```

```
Best parameters set found by GridSearchCV:
{'C': 10.0, 'kernel': 'linear'}
Grid scores on cv set:
0.5303 (+/- 0.04285) for {'C': 1e-20, 'kernel': 'linear'}
0.5303 (+/- 0.04285) for {'C': 3.1622776601683796e-17, 'kernel': 'linear'}
0.5303 (+/- 0.04285) for {'C': 1e-13, 'kernel': 'linear'}
0.5303 (+/- 0.04285) for {'C': 3.1622776601683795e-10, 'kernel': 'linear'}
0.5303 (+/- 0.04285) for {'C': 1e-06, 'kernel': 'linear'}
0.5303 (+/- 0.04285) for {'C': 0.0031622776601683794, 'kernel': 'linear'}
0.77879 (+/- 0.18562) for {'C': 10.0, 'kernel': 'linear'}

Detailed classification report:

#### Training Set ####
              precision    recall  f1-score   support

           0       0.85      0.73      0.79        15
           1       0.79      0.88      0.83        17

    accuracy                           0.81        32
   macro avg       0.82      0.81      0.81        32
weighted avg       0.82      0.81      0.81        32


#### Test Set ####
              precision    recall  f1-score   support

           0       1.00      0.80      0.89         5
           1       0.75      1.00      0.86         3

    accuracy                           0.88         8
   macro avg       0.88      0.90      0.87         8
weighted avg       0.91      0.88      0.88         8

#### All Set ####
              precision    recall  f1-score   support

           0       0.88      0.75      0.81        20
           1       0.78      0.90      0.84        20

    accuracy                           0.82        40
   macro avg       0.83      0.82      0.82        40
weighted avg       0.83      0.82      0.82        40
```



IG

Information Gain. Output the IG between each feature Xi and y.

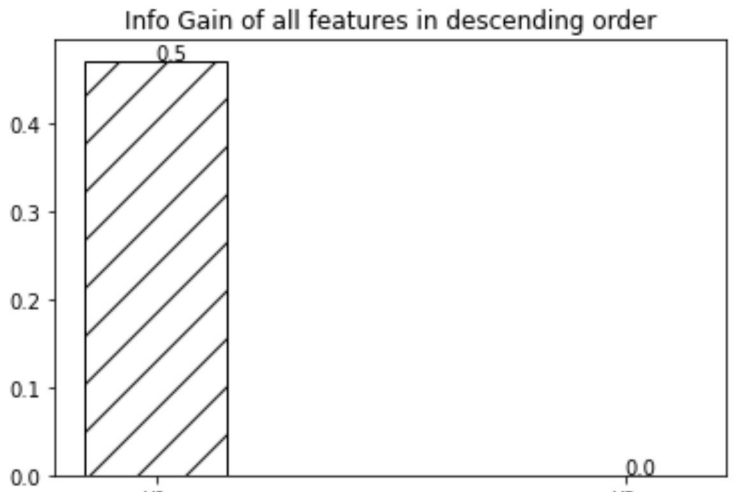
Information gain has been used in decision tree. For a specific feature, Information gain (IG) measures how much “information” a feature gives us about the class.

$$IG(Y|X) = H(Y) - H(Y|X)$$

In information theory, IG answers "if we transmit Y, how many bits can be saved if both sender and receiver know X?" Or "how much information of Y is implied in X?"

Attribute/feature X with a high IG is a good split on Y.

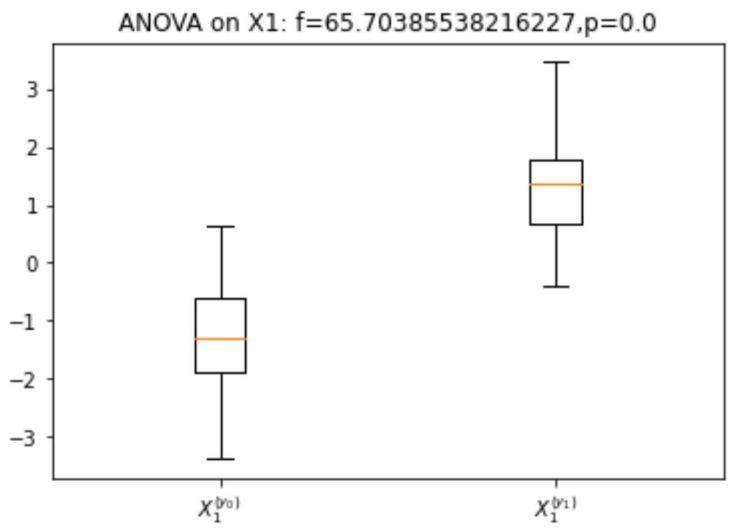
In [20]:
```
ig, _ = IG(X,y,show = True)
```



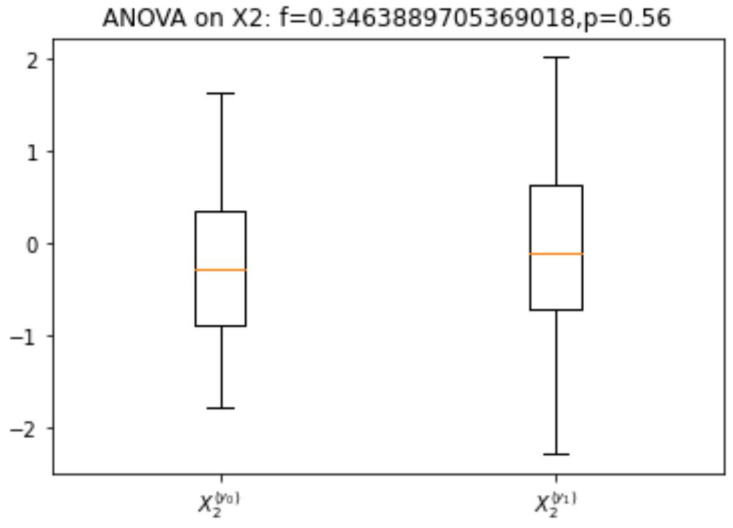
ANOVA

Perform ANOVA test on each feature Xi.

In [68]:
```
p, F, _ = ANOVA(X,y, verbose = True, show = True)
```




```
ANOVA on X1: f=65.70385538216227,p=0.0
```




```
ANOVA on X2: f=0.3463889705369018,p=0.56
```

MANOVA

Perform MANOVA test on the first N features.

In [7]:
```
%run pyCLAMs.py
p, F, _ = MANOVA(X,y, verbose = True)
```

```
endog: ['X1', 'X2']
exog: ['Intercept', 'y']
```

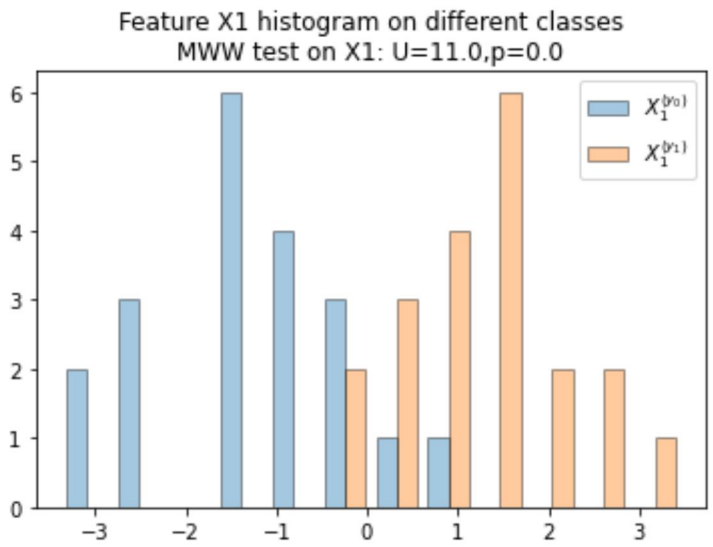
```
                Multivariate linear model
------------------------------------------------------------

------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.7222 2.0000 37.0000  7.1148 0.0024
         Pillai's trace 0.2778 2.0000 37.0000  7.1148 0.0024
 Hotelling-Lawley trace 0.3846 2.0000 37.0000  7.1148 0.0024
    Roy's greatest root 0.3846 2.0000 37.0000  7.1148 0.0024
------------------------------------------------------------

------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.5141 2.0000 37.0000 17.4885 0.0000
         Pillai's trace 0.4859 2.0000 37.0000 17.4885 0.0000
 Hotelling-Lawley trace 0.9453 2.0000 37.0000 17.4885 0.0000
    Roy's greatest root 0.9453 2.0000 37.0000 17.4885 0.0000
------------------------------------------------------------
```

MWW

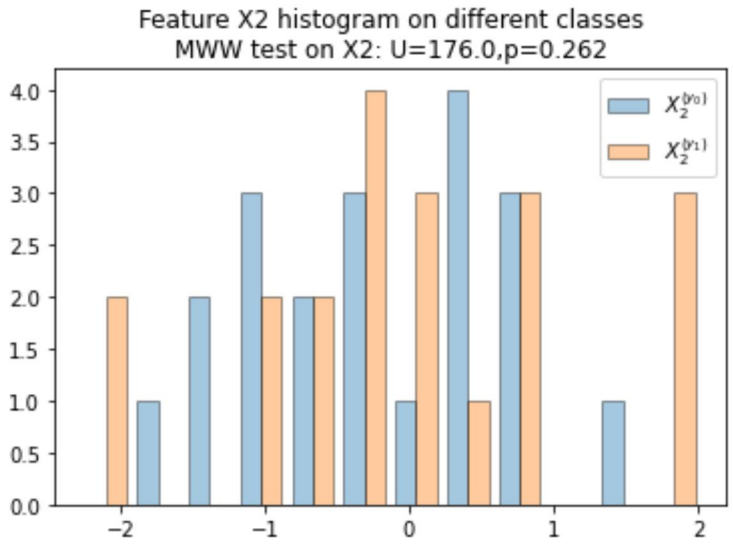
Mann–Whitney U test (also called the Mann–Whitney–Wilcoxon (MWW), Wilcoxon rank-sum test, or Wilcoxon–Mann–Whitney test) is a nonparametric test. This test can be used to determine whether two independent （独立、非配对） samples were selected from populations having the same distribution.

Other non-parametric tests are not suitable for classifiability analysis. For example, chi-square test is for nominal data. Signed rank sum test (符号秩和检验) is for paired (相关\配对样本) samples.

In [67]:
```
p, U, _ = MWW(X,y, verbose = True, show = True)
```




```
MWW test on X1: U=11.0,p=0.0
```




```
MWW test on X2: U=176.0,p=0.262
```

cohen_d

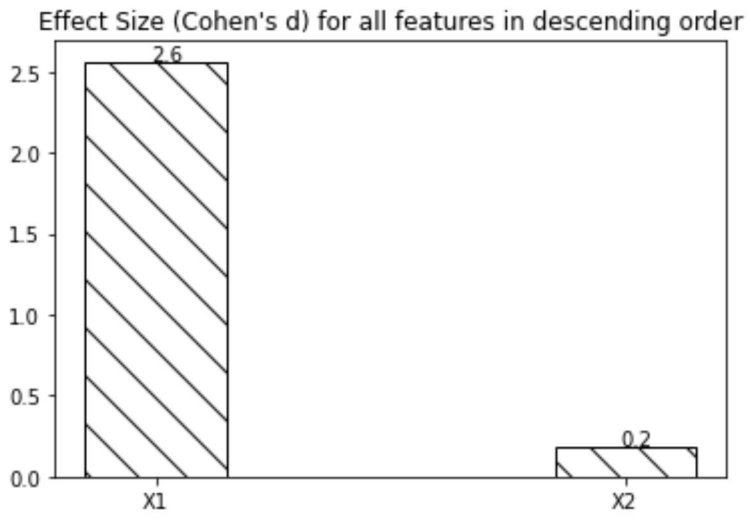
Cohen's d effect size. Use the pooled standard deviation internally.

$$Cohen_d = \frac{\mu_2 - \mu_1}{\sigma} = \frac{\mu_2 - \mu_1}{SD_{pooled}}$$

The Pooled Standard Deviation is

$$SD_{pooled} = \sqrt{\frac{(n_1 - 1)SD_1^2 + (n_2 - 1)SD_2^2}{n_1 \ + \ n_2 - 2}}$$

In [63]:
```
es, _ = cohen_d(X, y, show = True)
```



Pearson r, Spearman rho, Kendall tau

Besides these three correlation coefficients, IG/MI can also be seen as a measure of correlation.

In [11]:
```
correlate(X,y, verbose = True)
```

```
#### Correlation between X1 and y ####

Pearson r: 0.697, p-value: 0.0
Spearman rho: 0.749, p-value: 0.0
Kendall's tau: 0.619, p-value: 0.0

#### Correlation between X2 and y ####

Pearson r: -0.109, p-value: 0.505
Spearman rho: -0.117, p-value: 0.472
Kendall's tau: -0.097, p-value: 0.465
```

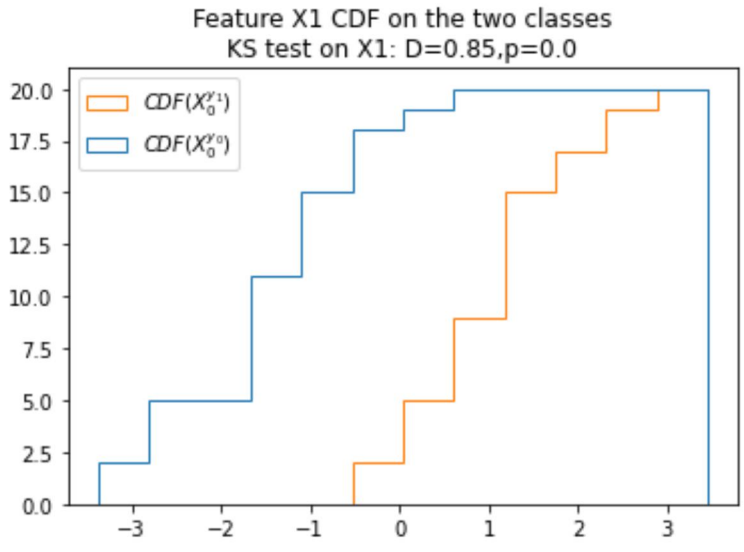
Out[11]:
```
({'correlation.r': [0.6967628846832323, -0.10859700961844498],
  'correlation.r.p': [5.914416500301044e-07, 0.50475584822651],
  'correlation.r.max': 0.6967628846832323,
  'correlation.r.p.min': 5.914416500301044e-07,
  'correlation.rho': [0.7493461815558833, -0.1169499820925367],
  'correlation.rho.p': [2.651197662237798e-08, 0.47234180506958745],
  'correlation.rho.max': 0.7493461815558833,
  'correlation.rho.p.min': 2.651197662237798e-08,
  'correlation.tau': [0.6194393660441094, -0.09667550799532344],
  'correlation.tau.p': [2.87343466130695e-06, 0.4651748036184682],
  'correlation.tau.max': 0.6194393660441094,
  'correlation.tau.p.min': 2.87343466130695e-06},
 "\n\n#### Correlation between X1 and y ####\n\nPearson r: 0.697, p-value: 0.0\nSpearman rho: 0.749, p-value: 0.0\nKe
ndall's tau: 0.619, p-value: 0.0\n\n#### Correlation between X2 and y ####\n\nPearson r: -0.109, p-value: 0.505\nSpea
rman rho: -0.117, p-value: 0.472\nKendall's tau: -0.097, p-value: 0.465")
```

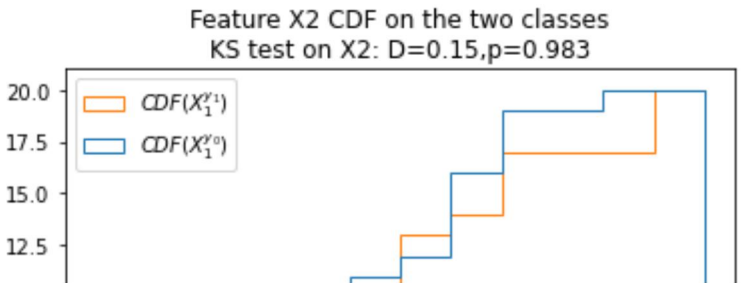
KS

Two-sample Kolmogorov–Smirnov test.

The two-sample KS test is one of the most useful and general nonparametric methods for comparing two samples, as it is sensitive to differences in both location and shape of the empirical cumulative distribution functions of the two samples. The Kolmogorov–Smirnov test can serve as a goodness of fit test. KS test is suitable for continous numeric values while chi-square test is for nominal values.

In [29]:
```
%run pyCLAMs.py
_ = KS(X,y, show = True)
```





ECoL

In [19]:
```
setup_ECoL()
ECoL_metrics(X,y)
```

```
R[write to console]: Installing packages into 'C:/Users/eleve/Documents/R/win-library/4.0'
(as 'lib' is unspecified)
```

Out[19]:

```
({'overlapping.F1.mean': 0.43314808182421,
  'overlapping.F1.sd': 0.44717164103713286,
  'overlapping.F1v.mean': 0.7375,
  'overlapping.F1v.sd': 0.017677669529663705,
  'overlapping.F2.mean': 0.85,
  'overlapping.F2.sd': 0.3067021677297899,
  'overlapping.F3.mean': 0.1879633741493019,
  'overlapping.F3.sd': 0.1285131939612029,
  'overlapping.F4.mean': nan,
  'overlapping.F4.sd': 0.12632804648326365,
  'neighborhood.N1': 0.10158342033965749,
  'neighborhood.N2.mean': 0.3076923076923077,
  'neighborhood.N2.sd': 0.4645257966832582,
  'neighborhood.N3.mean': 0.2206044028263522,
  'neighborhood.N3.sd': 0.09814244416132344,
  'neighborhood.N4.mean': 0.3,
  'neighborhood.N4.sd': 0.4640954808922571,
  'neighborhood.T1.mean': 0.025809032196912635,
  'neighborhood.T1.sd': 0.16010397875326465,
  'neighborhood.LSC': 0.05,
  'linearity.L1.mean': 0.05,
  'linearity.L1.sd': 1.0},
 'overlapping.F1.mean\t0.43314808182421\noverlapping.F1.sd\t0.44717164103713286\noverlapping.F1v.mean\t0.7375\noverlapping.F1v.sd\t0.017677669529663705\noverlapping.F2.mean\t0.85\noverlapping.F2.sd\t0.3067021677297899\noverlapping.F3.mean\t0.1879633741493019\noverlapping.F3.sd\t0.1285131939612029\noverlapping.F4.mean\tnan\noverlapping.F4.sd\t0.12632804648326365\nneighborhood.N1\t0.10158342033965749\nneighborhood.N2.mean\t0.3076923076923077\nneighborhood.N2.sd\t0.4645257966832582\nneighborhood.N3.mean\t0.2206044028263522\nneighborhood.N3.sd\t0.09814244416132344\nneighborhood.N4.mean\t0.3\nneighborhood.N4.sd\t0.4640954808922571\nneighborhood.T1.mean\t0.025809032196912635\nneighborhood.T1.sd\t0.16010397875326465\nneighborhood.LSC\t0.05\nlinearity.L1.mean\t0.05\nlinearity.L1.sd\t1.0\n')
```


get_metrics()

Return a dictionary of all metrics

In [20]:
```
get_metrics(X,y)
```

Out[20]:

```
{'classification.ACC': 0.85,
 'classification.Kappa': 0.7,
 'classification.F1_Score': 0.8571428571428572,
 'classification.Jaccard': 0.75,
 'classification.Precision': 0.8181818181818182,
 'classification.Recall': 0.9,
 'classification.CrossEntropy': 0.3422633425508984,
 'classification.AP': 0.9303007460805449,
 'classification.Brier': 0.10452925434208045,
 'classification.ROC_AUC': 0.9424999999999999,
 'classification.PR_AUC': 0.9275878884766691,
 'classification.BER': 0.034293154304636864,
 'correlation.IG': [0.3507056221177285, 0.0],
 'correlation.IG.max': 0.3507056221177285,
 'correlation.r': [0.6967628846832323, -0.10859700961844498],
 'correlation.r.p': [5.914416500301044e-07, 0.50475584822651],
 'correlation.r.max': 0.6967628846832323,
 'correlation.r.p.min': 5.914416500301044e-07,
 'correlation.rho': [0.7493461815558833, -0.1169499820925367],
 'correlation.rho.p': [2.651197662237798e-08, 0.47234180506958745],
 'correlation.rho.max': 0.7493461815558833,
 'correlation.rho.p.min': 2.651197662237798e-08,
 'correlation.tau': [0.6194393660441094, -0.09667550799532344],
 'correlation.tau.p': [2.87343466130695e-06, 0.4651748036184682],
 'correlation.tau.max': 0.6194393660441094,
 'correlation.tau.p.min': 2.87343466130695e-06,
 'test.ES': [1.8935424716086386, 0.21295398432800342],
 'test.ES.max': 1.8935424716086386,
 'test.param.ANOVA': [5.91441650030106e-07, 0.5047558482265093],
 'test.param.ANOVA.min': 5.91441650030106e-07,
 'test.param.ANOVA.F': [35.85503091785752, 0.45349399441171484],
 'test.param.ANOVA.F.max': 35.85503091785752,
 'test.param.MANOVA': 5e-06,
 'test.param.MANOVA.F': 17.488523,
 'test.nonparam.MWW': [1.5345504609285516e-06, 0.23674031388610284],
 'test.nonparam.MWW.min': 1.5345504609285516e-06,
 'test.nonparam.MWW.U': [27.0, 173.0],
 'test.nonparam.MWW.U.min': 27.0,
 'test.nonparam.KS': [9.54696510144592e-06, 0.8319696107963263],
```

```
'test.nonparam.KS.min': 9.54696510144592e-06,
'test.nonparam.KS.D': [0.75, 0.2],
'test.nonparam.KS.D.max': 0.75,
'overlapping.F1.mean': 0.43314808182421,
'overlapping.F1.sd': 0.44717164103713286,
'overlapping.F1v.mean': 0.7375,
'overlapping.F1v.sd': 0.017677669529663705,
'overlapping.F2.mean': 0.85,
'overlapping.F2.sd': 0.3067021677297899,
'overlapping.F3.mean': 0.1879633741493019,
'overlapping.F3.sd': 0.1285131939612029,
'overlapping.F4.mean': nan,
'overlapping.F4.sd': 0.11214176651743686,
'neighborhood.N1': 0.09386208880307799,
'neighborhood.N2.mean': 0.3076923076923077,
'neighborhood.N2.sd': 0.4645257966832582,
'neighborhood.N3.mean': 0.2206044028263522,
'neighborhood.N3.sd': 0.09814244416132344,
'neighborhood.N4.mean': 0.3,
'neighborhood.N4.sd': 0.4640954808922571,
'neighborhood.T1.mean': 0.029568649534784025,
'neighborhood.T1.sd': 0.16135653548142317,
'neighborhood.LSC': 0.05,
'linearity.L1.mean': 0.05,
'linearity.L1.sd': 1.0}
```


get_json

In [21]:
```
get_json(X,y)
```

Out[21]:

```
'{"classification.ACC": 0.825, "classification.Kappa": 0.65, "classification.F1_Score": 0.8372093023255814, "classification.Jaccard": 0.72, "classification.Precision": 0.782608695652174, "classification.Recall": 0.9, "classification.CrossEntropy": 0.3422633425508984, "classification.AP": 0.9303007460805449, "classification.Brier": 0.10452925434208045, "classification.ROC_AUC": 0.9424999999999999, "classification.PR_AUC": 0.9275878884766691, "classification.BER": 0.03251621242868674, "correlation.IG": [0.3507056221177285, 0.0], "correlation.IG.max": 0.3507056221177285, "correlation.r": [0.6967628846832323, -0.10859700961844498], "correlation.r.p": [5.914416500301044e-07, 0.50475584822651], "correlation.r.max": 0.6967628846832323, "correlation.r.p.min": 5.914416500301044e-07, "correlation.rho": [0.7493461815558833, -0.1169499820925367], "correlation.rho.p": [2.651197662237798e-08, 0.47234180506958745], "correlation.rho.max": 0.7493461815558833, "correlation.rho.p.min": 2.651197662237798e-08, "correlation.tau": [0.6194393660441094, -0.09667550799532344], "correlation.tau.p": [2.87343466130695e-06, 0.4651748036184682], "correlation.tau.max": 0.6194393660441094, "correlation.tau.p.min": 2.87343466130695e-06, "test.ES": [1.8935424716086386, 0.21295398432800342], "test.ES.max": 1.8935424716086386, "test.param.ANOVA": [5.91441650030106e-07, 0.5047558482265093], "test.param.ANOVA.min": 5.91441650030106e-07, "test.param.ANOVA.F": [35.85503091785752, 0.45349399441171484], "test.param.ANOVA.F.max": 35.85503091785752, "test.param.MANOVA": 5e-06, "test.param.MANOVA.F": 17.488523, "test.nonparam.MWW": [1.5345504609285516e-06, 0.23674031388610284], "test.nonparam.MWW.min": 1.5345504609285516e-06, "test.nonparam.MWW.U": [27.0, 173.0], "test.nonparam.MWW.U.min": 27.0, "test.nonparam.KS": [9.54696510144592e-06, 0.8319696107963263], "test.nonparam.KS.min": 9.54696510144592e-06, "test.nonparam.KS.D": [0.75, 0.2], "test.nonparam.KS.D.max": 0.75, "overlapping.F1.mean": 0.43314808182421, "overlapping.F1.sd": 0.44717164103713286, "overlapping.F1v.mean": 0.7375, "overlapping.F1v.sd": 0.017677669529663705, "overlapping.F2.mean": 0.85, "overlapping.F2.sd": 0.3067021677297899, "overlapping.F3.mean": 0.1879633741493019, "overlapping.F3.sd": 0.1285131939612029, "overlapping.F4.mean": NaN, "overlapping.F4.sd": 0.10591480568337752, "neighborhood.N1": 0.11251368015090392, "neighborhood.N2.mean": 0.3076923076923077, "neighborhood.N2.sd": 0.4645257966832582, "neighborhood.N3.mean": 0.2206044028263522, "neighborhood.N3.sd": 0.09814244416132344, "neighborhood.N4.mean": 0.3, "neighborhood.N4.sd": 0.4640954808922571, "neighborhood.T1.mean": 0.14948130469674084, "neighborhood.T1.sd": 0.356041022210029, "neighborhood.LSC": 0.05, "linearity.L1.mean": 0.05, "linearity.L1.sd": 1.0}'
```

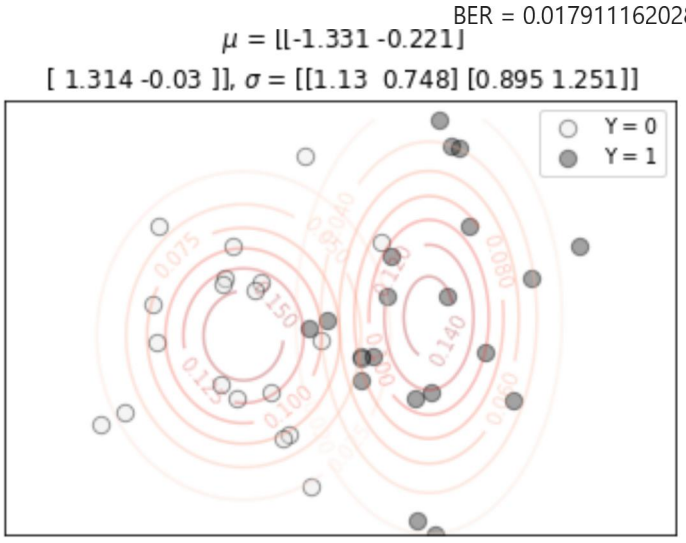

get_html

Return a piece of HTML to be embedded in web applications.

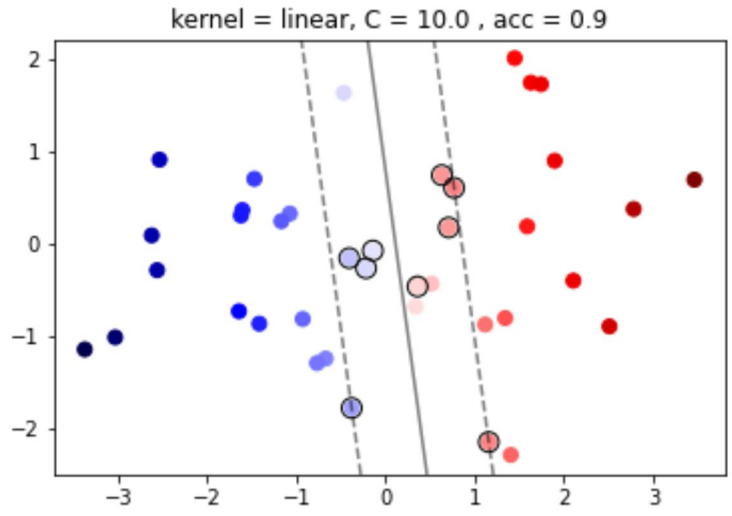
In [66]:
```
from IPython.display import display, HTML
%run pyCLAMs.py
display(HTML(get_html(X,y)))
```

Metric/Statistic

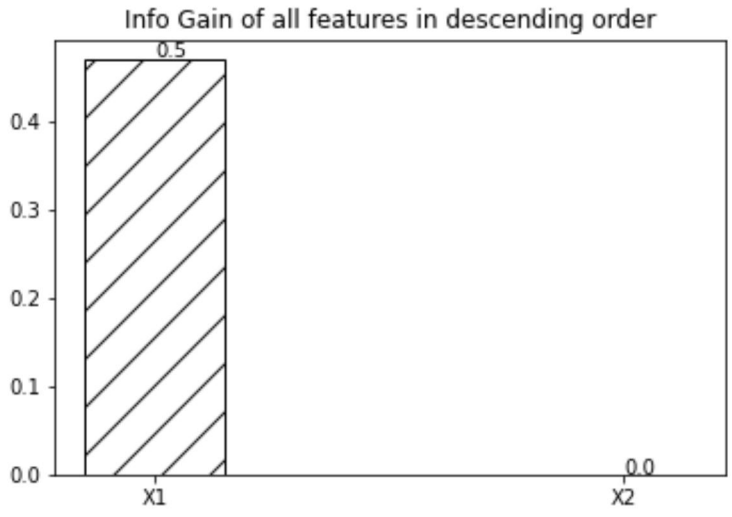
BER = 0.01791116202814358



{'classification.ACC': 0.9, 'classification.Kappa': 0.8, 'classification.F1_Score': 0.9, 'classification.Jaccard': 0.8181818181818182, 'classification.Precision': 0.9, 'classification.Recall': 0.9, 'classification.CrossEntropy': 0.6913953396035699, 'classification.AP': 0.9761858076563958, 'classification.Brier': 0.24912408209070022, 'classification.ROC_AUC': 0.9750000000000001, 'classification.PR_AUC': 0.9755804105585788}

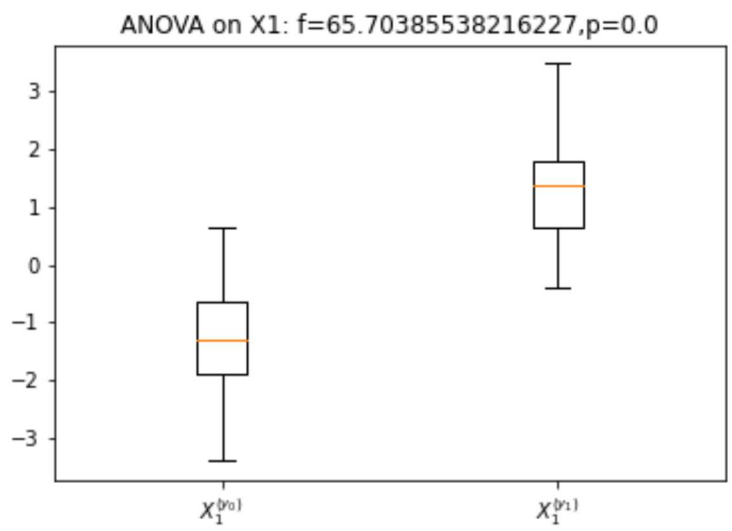


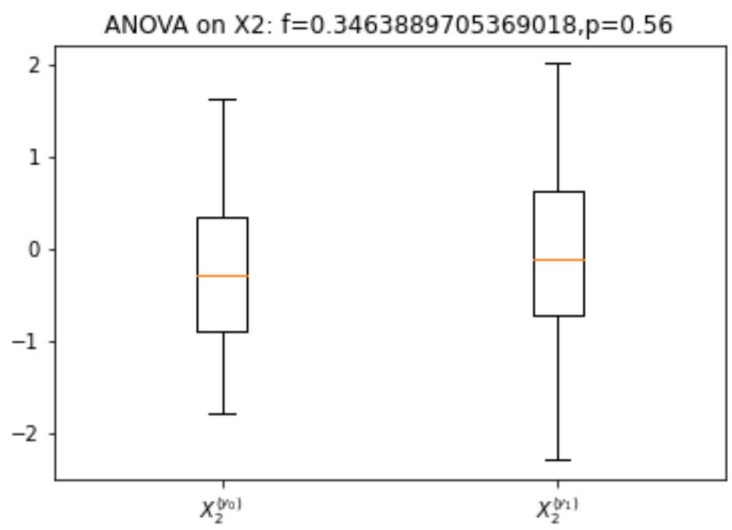

```
Best parameters set found by GridSearchCV:
{'C': 10.0, 'kernel': 'linear'}
Grid scores on cv set:
0.56364 (+/- 0.05143) for {'C': 1e-20, 'kernel': 'linear'}
0.56364 (+/- 0.05143) for {'C': 3.1622776601683796e-17, 'kernel': 'linear'}
0.56364 (+/- 0.05143) for {'C': 1e-13, 'kernel': 'linear'}
0.56364 (+/- 0.05143) for {'C': 3.1622776601683795e-10, 'kernel': 'linear'}
0.56364 (+/- 0.05143) for {'C': 1e-06, 'kernel': 'linear'}
0.56364 (+/- 0.05143) for {'C': 0.0031622776601683794, 'kernel': 'linear'}
0.87273 (+/- 0.10285) for {'C': 10.0, 'kernel': 'linear'}

Detailed classification report:

                                #### Training Set ####
              precision    recall  f1-score   support

           0       0.89      0.89      0.89        18
           1       0.86      0.86      0.86        14

    accuracy                           0.88        32
   macro avg       0.87      0.87      0.87        32
weighted avg       0.88      0.88      0.88        32


                                    #### Test Set ####
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         6

    accuracy                           1.00         8
   macro avg       1.00      1.00      1.00         8
weighted avg       1.00      1.00      1.00         8


                                     #### All Set ####
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        20
           1       0.90      0.90      0.90        20

    accuracy                           0.90        40
   macro avg       0.90      0.90      0.90        40
weighted avg       0.90      0.90      0.90        40

                                classification.ACC      0.9
                                classification.Kappa    0.8
                                classification.F1_Score 0.9
                      classification.Jaccard  0.8181818181818182
                             classification.Precision        0.9
                                classification.Recall   0.9
          classification.CrossEntropy     0.6913953396035699
                  classification.AP       0.9761858076563958
                 classification.Brier    0.24912408209070022
                  classification.ROC_AUC  0.9750000000000001
                  classification.PR_AUC   0.9755804105585788

                                                IG = [0.47042702 0. ]
```




```
#### Correlation between X1 and y ####

        Pearson r: 0.796, p-value: 0.0
    Spearman rho: 0.819, p-value: 0.0
   Kendall's tau: 0.677, p-value: 0.0

#### Correlation between X2 and y ####

       Pearson r: 0.095, p-value: 0.56
   Spearman rho: 0.104, p-value: 0.523
   Kendall's tau: 0.086, p-value: 0.516
```

ANOVA p[8.279433135569151e-10, 0.5596466995878596]





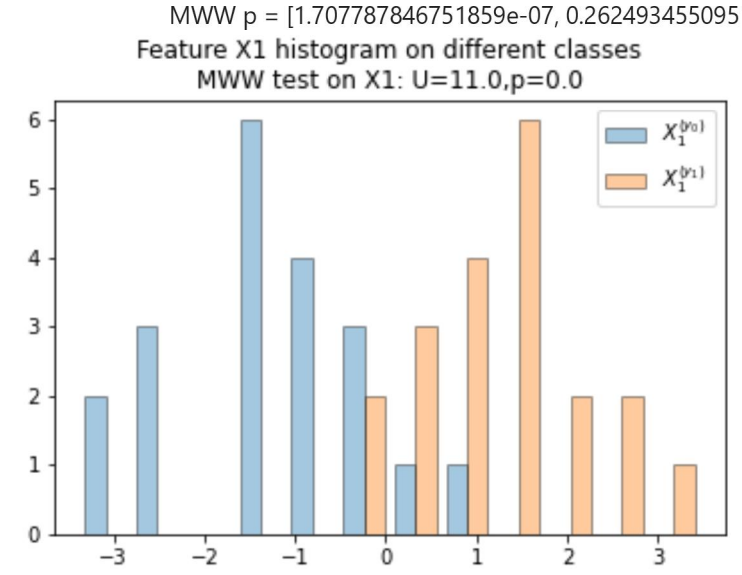
MANOVA p = 2.220446049250313e-16
endog: ['X1', 'X2']
exog: ['Intercept', 'y']

Multivariate linear model

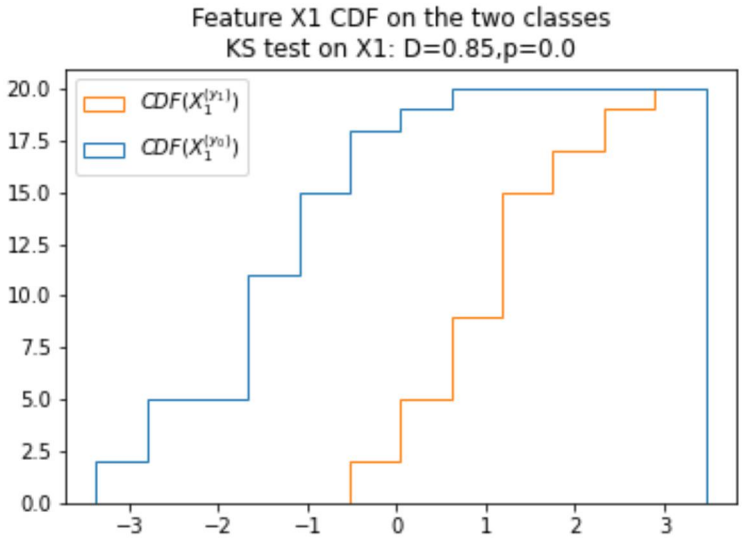
Intercept	Value	Num DF	Den DF	F Value	Pr > F
Wilks' lambda	0.5331	2.0000	37.0000	16.2040	0.0000
Pillai's trace	0.4669	2.0000	37.0000	16.2040	0.0000
Hotelling-Lawley trace	0.8759	2.0000	37.0000	16.2040	0.0000
Roy's greatest root	0.8759	2.0000	37.0000	16.2040	0.0000

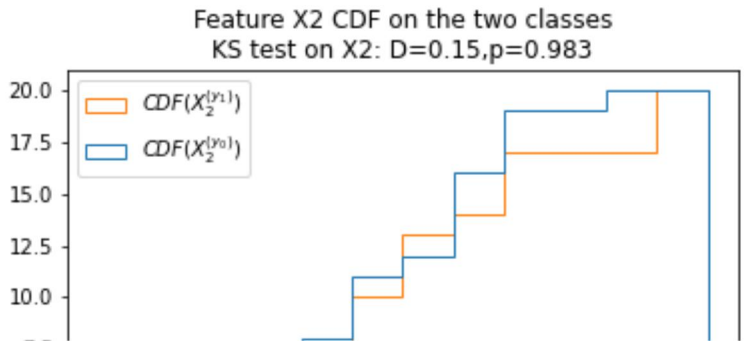
y	Value	Num DF	Den DF	F Value	Pr > F
Wilks' lambda	0.3650	2.0000	37.0000	32.1805	0.0000
Pillai's trace	0.6350	2.0000	37.0000	32.1805	0.0000
Hotelling-Lawley trace	1.7395	2.0000	37.0000	32.1805	0.0000
Roy's greatest root	1.7395	2.0000	37.0000	32.1805	0.0000

MWW p = [1.707787846751859e-07, 0.26249345509518074]



K-S p = [1.4334782434603487e-07, 0.9831368772656193]





Metric Consistency Test

1. Analyze how the metrics change with datasets / their consistency

Generate a series of sample datasets with different class distances, and calculate metrics on these datasets.

In [46]:

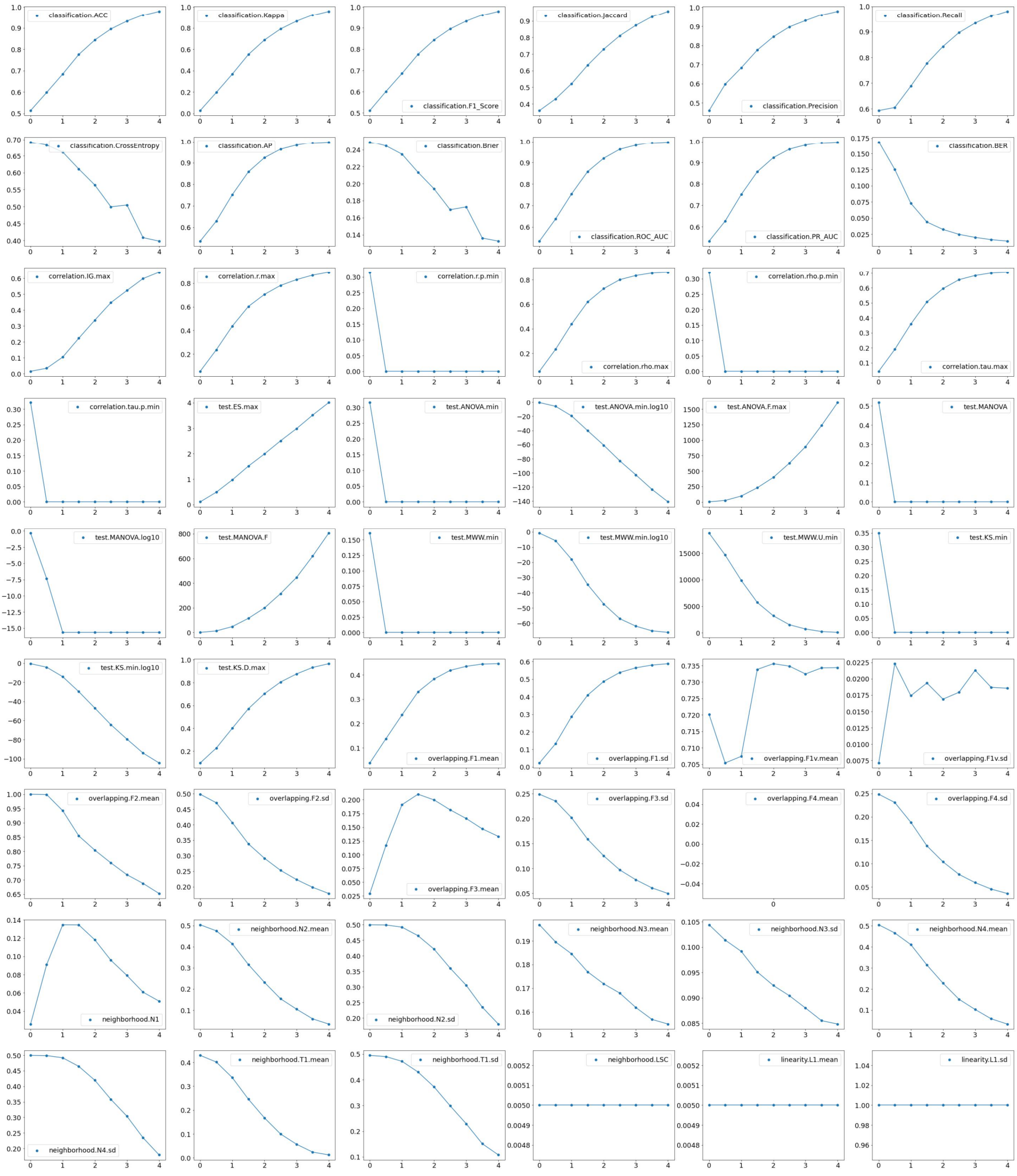
```
%run pyCLAMs.py
dcts = simulate(np.linspace(0, 3, 4), repeat = 1, nobs = 50)
visualize_dcts(dcts)
```

```
100%|██████████| 4/4 [00:09<00:00,  2.31s/it]
```

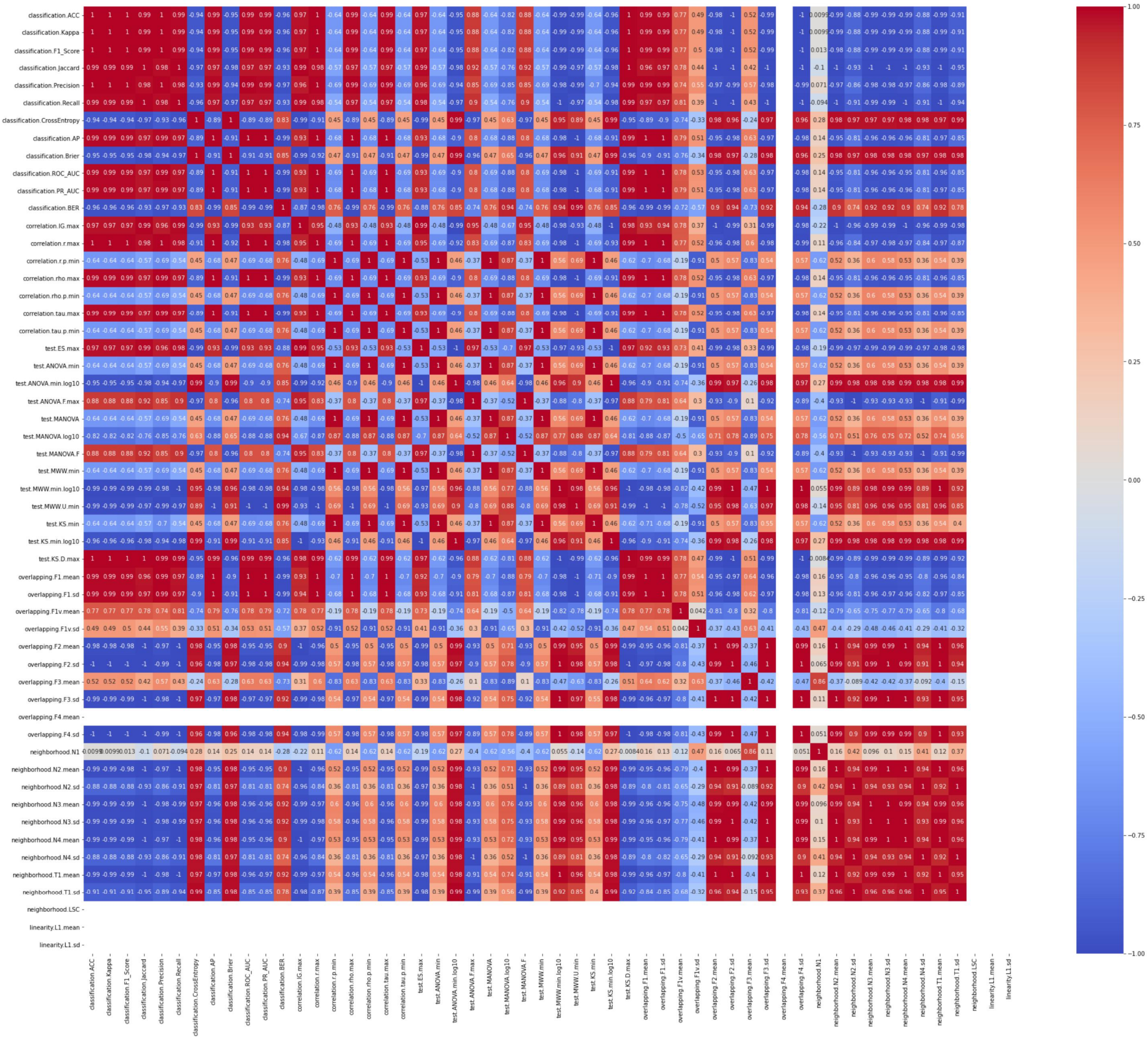
Repeat N times to get the averaged curves against different md values.

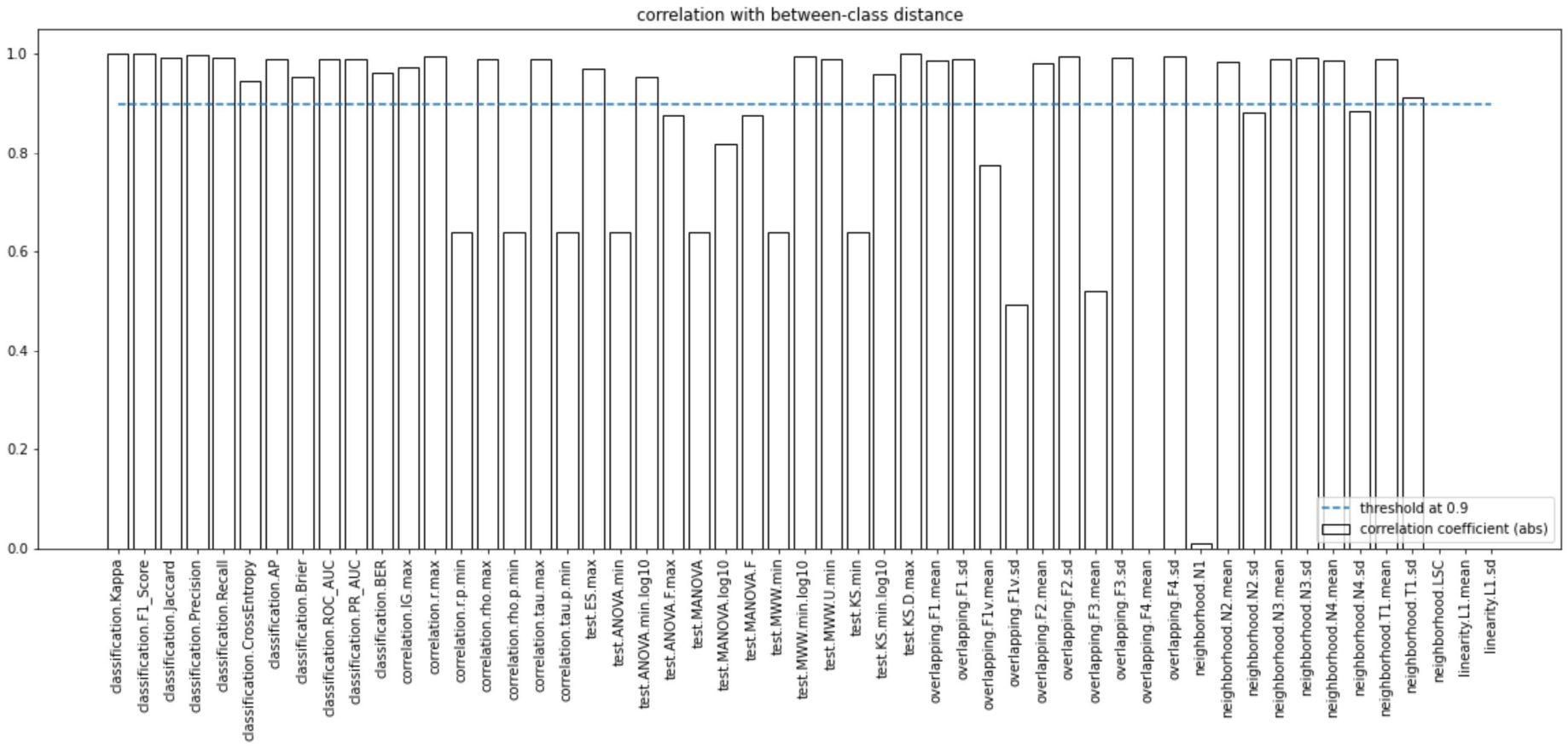
In [47]:
```
%run pyCLAMs.py
dcts = simulate(np.linspace(0, 4, 9), repeat = 50, nobs = 200)
visualize_dcts(dcts)
```

```
100%|██████████| 450/450 [22:52<00:00,  3.05s/it]
```



In [58]:
```
%run pyCLAMs.py
visualize_corr_matrix(dcts, 'coolwarm')
```






```
Metrics above the threshold:  ['classification.Kappa' 'classification.F1_Score' 'classification.Jaccard'
 'classification.Precision' 'classification.Recall'
 'classification.CrossEntropy' 'classification.AP' 'classification.Brier'
 'classification.ROC_AUC' 'classification.PR_AUC' 'classification.BER'
 'correlation.IG.max' 'correlation.r.max' 'correlation.rho.max'
 'correlation.tau.max' 'test.ES.max' 'test.ANOVA.min.log10'
 'test.MWW.min.log10' 'test.MWW.U.min' 'test.KS.min.log10' 'test.KS.D.max'
 'overlapping.F1.mean' 'overlapping.F1.sd' 'overlapping.F2.mean'
 'overlapping.F2.sd' 'overlapping.F3.sd' 'overlapping.F4.sd'
 'neighborhood.N2.mean' 'neighborhood.N3.mean' 'neighborhood.N3.sd'
 'neighborhood.N4.mean' 'neighborhood.T1.mean' 'neighborhood.T1.sd']
```

```
pca = extract_PC(dcts)
```

```
pca.explained_variance_ratio_, pca.components_[0]
```

In [ ]:

Appendix: Libraries and versions

In [53]:

```
import sklearn
import scipy
import statsmodels
import numpy
import pandas
import rpy2
import seaborn
import matplotlib

print("sklearn", sklearn.__version__, "\n",
      "scipy", scipy.__version__, "\n",
      "statsmodels", statsmodels.__version__,  "\n",
      "numpy", numpy.__version__, "\n",
      "pandas", pandas.__version__, "\n",
      "matplotlib", matplotlib.__version__, "\n",
      "seaborn", seaborn.__version__, "\n",
      "rpy2", rpy2.__version__)

print("ECoL (R package) 0.4.2")
```

```
sklearn 0.24.1
 scipy 1.6.2
 statsmodels 0.12.2
 numpy 1.19.2
 pandas 1.2.3
 matplotlib 3.3.4
 seaborn 0.11.1
 rpy2 3.4.4
ECoL (R package) 0.4.2
```

Appendix: py script used as a call interface for upper applications

```
from pyCLAMs import *
import sys
import json
import uuid
import os

# Three params passed from upper applications

def generate(d, n):

    X,y = mvg(nobs = n, md = d)

    # get the local file path
    fn = os.path.dirname(os.path.realpath(__file__)) + "/" + str(uuid.uuid4()) + ".csv"

    # save to csv file
    save_file(X,y, fn)

    return fn

def analyze(csv):

    # X,y = load_file(csv)
    # s = get_html(X,y)

    # store html result into a local html file
    fn = os.path.dirname(os.path.realpath(__file__)) + "/" + str(uuid.uuid4()) + ".html"
    with open(fn, 'w') as f:
        f.write(analyze_file(csv))

    return fn

if __name__ == "__main__":

    mode = sys.argv[1]

    # Mode 1: generate a sample data file
    if mode == 'generate':

        d = float(sys.argv[2]) # distance between means, respect to std, i.e. (mu2 - mu1) / std, or how many
stds is the difference.
        n = int(sys.argv[3]) # number of observations / samples

        fn = generate(d, n)
```

```
        print(fn)

    # Mode 2: analyze a local data file
    elif mode == 'analyze':

        csv = sys.argv[2]

        fn = analyze(csv)

        print(fn)

    # Mode 3: generate + analyze
    elif mode == 'generate_analyze':

        d = float(sys.argv[2]) # distance between means, respect to std, i.e. (mu2 - mu1) / std, or how many
stds is the difference.
        n = int(sys.argv[3]) # number of observations / samples

        csv = generate(d, n)
        fn = analyze(csv)

        ### Without the intermediate csv file
        # X,y = mvg(nobs = n, md = d)
        # fn = os.path.dirname(os.path.realpath(__file__)) + "/" + str(uuid.uuid4()) + ".html"
        # with open(fn, 'w') as f:
        #     f.write(get_html(X,y))
        print(fn)
```

Trouble Shooting

1. module 'rpy2.robjects.conversion' has no attribute 'py2rpy'

> pip insall rpy2==3.4.4 # use the correct version

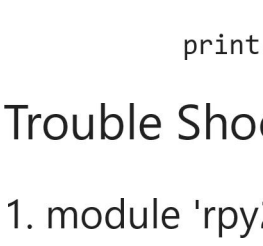
2. R_HOME must be set in the environment or Registry

```
Install R
Create R_HOME system variable
Add R_HOME\bin to the PATH, in order to execute R from python
Add R_HOME\bin\x64 to the PATH, in order to load R.dll
Install package tzlocal
May also need to reinstall rpy2
```

3. unable to initialize the JIT

```
Happens only on Windows server. Remains to research
```

Appendix: wCLAMs - A web GUI tool based on pyCLAMs



Appendix: Code Ocean Capsule (latest code)