HW1-Programming

Graded

Student

Brian Bertness

View or edit group

Total Points

25 / 25 pts

Autograder Score

18.0 / 18.0

Passed Tests

test_C1Fit (test_MyDiscriminant.TestMyDiscriminant) (2/2)
test_C1Predict (test_MyDiscriminant.TestMyDiscriminant) (2/2)
test_C2Fit (test_MyDiscriminant.TestMyDiscriminant) (2/2)
test_C2Predict (test_MyDiscriminant.TestMyDiscriminant) (2/2)
test_C3Fit (test_MyDiscriminant.TestMyDiscriminant) (2/2)
test_C3Predict (test_MyDiscriminant.TestMyDiscriminant) (2/2)
test_computeCov (test_MyDiscriminant.TestMyDiscriminant) (2/2)
test_computeMean (test_MyDiscriminant.TestMyDiscriminant) (1/1)
test_computePrecision (test_MyDiscriminant.TestMyDiscriminant) (1/1)
test_computePrior (test_MyDiscriminant.TestMyDiscriminant) (1/1)
test_computeRecall (test_MyDiscriminant.TestMyDiscriminant) (1/1)

Question 2

Code Sanity — **7** / 7 pts

✔ **– 0 pts** Correct

Autograder Results

test_C1Fit (test_MyDiscriminant.TestMyDiscriminant) (2/2)

This test the fit function of classifier C1
Your S for both classes are:

```
[[[ 1.43781459  1.24407538  1.4050195   1.25014251  0.44606905
    0.21774186  1.12986533  2.42769858]
  [ 1.24407538  3.84355238  4.36296535  1.22208113  1.48580304
    0.51661422  0.50935232  6.4949891 ]
  [ 1.4050195   4.36296535  8.23386906  2.11785909  2.73801346
    1.56375799 -0.84281537  7.80134805]
  [ 1.25014251  1.22208113  2.11785909  3.99903851 -0.21275367
    1.08232519  1.01143298  3.2204665 ]
  [ 0.44606905  1.48580304  2.73801346 -0.21275367  2.81247403
   -0.12402908  0.86774363  2.13936869]
  [ 0.21774186  0.51661422  1.56375799  1.08232519 -0.12402908
    0.95087368 -0.46473474  1.86517572]
  [ 1.12986533  0.50935232 -0.84281537  1.01143298  0.86774363
   -0.46473474  3.96668692  1.63874881]
  [ 2.42769858  6.4949891   7.80134805  3.2204665   2.13936869
    1.86517572  1.63874881 14.54667084]]

 [[ 1.30424291  1.17675824  0.18340592  0.87089916  1.15054609
    0.7423137   1.07919291  1.84996001]
  [ 1.17675824  3.13097865 -0.16135333  0.43245499  1.20970531
    0.72685675  0.27247502  2.92221783]
  [ 0.18340592 -0.16135333  6.72650694  2.59256962  1.84112898
    1.50964579  2.03543038  6.93512008]
  [ 0.87089916  0.43245499  2.59256962  3.93849718  1.01394707
    1.98598543  2.27406539  4.73865443]
  [ 1.15054609  1.20970531  1.84112898  1.01394707  2.87137815
    0.83385036  2.25316578  3.38395507]
  [ 0.7423137   0.72685675  1.50964579  1.98598543  0.83385036
    2.37610777  1.18698751  2.33902631]
  [ 1.07919291  0.27247502  2.03543038  2.27406539  2.25316578
    1.18698751  6.44746773  2.55969108]
  [ 1.84996001  2.92221783  6.93512008  4.73865443  3.38395507
    2.33902631  2.55969108 19.39182337]]]
```

GT shared_S are:

```
[[[ 1.43781459  1.24407538  1.4050195   1.25014251  0.44606905
    0.21774186  1.12986533  2.42769858]
  [ 1.24407538  3.84355238  4.36296535  1.22208113  1.48580304
    0.51661422  0.50935232  6.4949891 ]
  [ 1.4050195   4.36296535  8.23386906  2.11785909  2.73801346
    1.56375799 -0.84281537  7.80134805]
  [ 1.25014251  1.22208113  2.11785909  3.99903851 -0.21275367
    1.08232519  1.01143298  3.2204665 ]
  [ 0.44606905  1.48580304  2.73801346 -0.21275367  2.81247403
   -0.12402908  0.86774363  2.13936869]
  [ 0.21774186  0.51661422  1.56375799  1.08232519 -0.12402908
    0.95087368 -0.46473474  1.86517572]
  [ 1.12986533  0.50935232 -0.84281537  1.01143298  0.86774363
   -0.46473474  3.96668692  1.63874881]
  [ 2.42769858  6.4949891   7.80134805  3.2204665   2.13936869
    1.86517572  1.63874881 14.54667084]]
```

```
[[ 1.30424291  1.17675824  0.18340592  0.87089916  1.15054609
   0.7423137   1.07919291  1.84996001]
 [ 1.17675824  3.13097865 -0.16135333  0.43245499  1.20970531
   0.72685675  0.27247502  2.92221783]
 [ 0.18340592 -0.16135333  6.72650694  2.59256962  1.84112898
   1.50964579  2.03543038  6.93512008]
 [ 0.87089916  0.43245499  2.59256962  3.93849718  1.01394707
   1.98598543  2.27406539  4.73865443]
 [ 1.15054609  1.20970531  1.84112898  1.01394707  2.87137815
   0.83385036  2.25316578  3.38395507]
 [ 0.7423137   0.72685675  1.50964579  1.98598543  0.83385036
   2.37610777  1.18698751  2.33902631]
 [ 1.07919291  0.27247502  2.03543038  2.27406539  2.25316578
   1.18698751  6.44746773  2.55969108]
 [ 1.84996001  2.92221783  6.93512008  4.73865443  3.38395507
   2.33902631  2.55969108 19.39182337]]]
```

test_C1Predict (test_MyDiscriminant.TestMyDiscriminant) (2/2)

This test the predictions of C1.
Your prediction are:

```
[2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 1. 2. 1.
 2. 2. 2. 2. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 2. 2. 1. 2. 2. 2.
 1. 2. 2. 2. 2. 2. 1. 2. 2. 2. 2. 1. 2. 2. 1. 1. 2. 2. 1. 2. 2. 2. 2. 2.
 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 1. 2. 2. 1. 2. 2. 2. 2.
 2. 2. 2. 2.]
```

GT is:

```
[2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 1. 2. 1.
 2. 2. 2. 2. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 2. 2. 1. 2. 2. 2.
 1. 2. 2. 2. 2. 2. 1. 2. 2. 2. 2. 1. 2. 2. 1. 1. 2. 2. 1. 2. 2. 2. 2. 2.
 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 1. 2. 2. 1. 2. 2. 2. 2.
 2. 2. 2. 2.]
```

test_C2Fit (test_MyDiscriminant.TestMyDiscriminant) (2/2)

This test the fit function of classifier C2
Your shared_S are:

```
[[ 2.71394249  2.8980981   2.03005823  2.32788275  2.44556456  1.94408372
   2.11467397  3.87525675]
 [ 2.8980981   5.4016963   2.98814342  2.33615901  3.1401789   2.33936364
   1.62228674  6.22657007]
 [ 2.03005823  2.98814342  8.63209716  3.8712958   3.69177331  2.96198898
   2.29155721  9.07916957]
 [ 2.32788275  2.33615901  3.8712958   5.21078844  2.1280328   3.02302387
   2.8862445   6.06305947]
 [ 2.44556456  3.1401789   3.69177331  2.1280328   4.44793178  2.02884713
   2.94510795  5.02238615]
 [ 1.94408372  2.33936364  2.96198898  3.02302387  2.02884713  3.25654972
   1.71047814  4.01132185]
 [ 2.11467397  1.62228674  2.29155721  2.8862445   2.94510795  1.71047814
   6.38519461  3.66022826]
 [ 3.87525675  6.22657007  9.07916957  6.06305947  5.02238615  4.01132185
   3.66022826 20.18824015]]
```

GT shared_S are:

```
[[ 2.71394249  2.8980981   2.03005823  2.32788275  2.44556456  1.94408372
   2.11467397  3.87525675]
 [ 2.8980981   5.4016963   2.98814342  2.33615901  3.1401789   2.33936364
   1.62228674  6.22657007]
 [ 2.03005823  2.98814342  8.63209716  3.8712958   3.69177331  2.96198898
   2.29155721  9.07916957]
 [ 2.32788275  2.33615901  3.8712958   5.21078844  2.1280328   3.02302387
   2.8862445   6.06305947]
 [ 2.44556456  3.1401789   3.69177331  2.1280328   4.44793178  2.02884713
   2.94510795  5.02238615]
 [ 1.94408372  2.33936364  2.96198898  3.02302387  2.02884713  3.25654972
   1.71047814  4.01132185]
 [ 2.11467397  1.62228674  2.29155721  2.8862445   2.94510795  1.71047814
   6.38519461  3.66022826]
 [ 3.87525675  6.22657007  9.07916957  6.06305947  5.02238615  4.01132185
   3.66022826 20.18824015]]
```

test_C2Predict (test_MyDiscriminant.TestMyDiscriminant) (2/2)

This test the predictions of C2.
Your prediction are:
[2. 2. 2. 1. 2. 1. 1. 2. 1. 1. 2. 1. 2. 1. 2. 2. 1. 2. 2. 2. 2. 1. 2. 1.
 2. 2. 2. 2. 1. 1. 1. 2. 2. 2. 1. 1. 2. 2. 1. 2. 2. 2. 2. 2. 1. 2. 2. 2.
 1. 2. 2. 2. 2. 2. 1. 2. 2. 2. 1. 1. 2. 1. 2. 2. 2. 1. 1. 2. 1. 2. 2. 2.
 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 1. 2. 2. 1. 2. 2. 2. 2.
 2. 2. 2. 2.]
GT is:
[2. 2. 2. 1. 2. 1. 1. 2. 1. 1. 2. 1. 2. 1. 2. 2. 1. 2. 2. 2. 2. 1. 2. 1.
 2. 2. 2. 2. 1. 1. 1. 2. 2. 2. 1. 1. 2. 2. 1. 2. 2. 2. 2. 2. 1. 2. 2. 2.
 1. 2. 2. 2. 2. 2. 1. 2. 2. 2. 1. 1. 2. 1. 2. 2. 2. 1. 1. 2. 1. 2. 2. 2.
 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 1. 2. 2. 1. 2. 2. 2. 2.
 2. 2. 2. 2.]

test_C3Fit (test_MyDiscriminant.TestMyDiscriminant) (2/2)

This test the fit function of classifier C3, you need to cast the non-diagonal entries to 0
also don't forget to weighted combine S1/S2 to get the shared_S
Your shared_S are:

```
[[ 2.71394249  0.          0.          0.          0.          0.
   0.          0.        ]
 [ 0.          5.4016963   0.          0.          0.          0.
   0.          0.        ]
 [ 0.          0.          8.63209716  0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.          5.21078844  0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.          4.44793178  0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          3.25654972
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   6.38519461  0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.         20.18824015]]
```

GT shared_S are:

```
[[ 2.71394249  0.          0.          0.          0.          0.
   0.          0.        ]
 [ 0.          5.4016963   0.          0.          0.          0.
   0.          0.        ]
 [ 0.          0.          8.63209716  0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.          5.21078844  0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.          4.44793178  0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          3.25654972
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   6.38519461  0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.         20.18824015]]
```

test_C3Predict (test_MyDiscriminant.TestMyDiscriminant) (2/2)

This test the predictions of C3.
Your prediction are:

```
[1. 2. 1. 1. 2. 1. 1. 2. 1. 1. 2. 1. 2. 1. 1. 2. 1. 2. 2. 2. 1. 1. 2. 1.
 2. 2. 2. 2. 1. 1. 1. 2. 2. 1. 1. 1. 2. 2. 1. 2. 2. 2. 2. 2. 1. 2. 2. 2.
 1. 2. 2. 2. 1. 2. 1. 2. 2. 2. 1. 1. 2. 1. 2. 1. 1. 1. 1. 2. 1. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 1. 2. 2. 1. 1. 2. 2. 2.
 2. 2. 2. 2.]
```

GT is:

```
[1. 2. 1. 1. 2. 1. 1. 2. 1. 1. 2. 1. 2. 1. 1. 2. 1. 2. 2. 2. 1. 1. 2. 1.
 2. 2. 2. 2. 1. 1. 1. 2. 2. 1. 1. 1. 2. 2. 1. 2. 2. 2. 2. 2. 1. 2. 2. 2.
 1. 2. 2. 2. 1. 2. 1. 2. 2. 2. 1. 1. 2. 1. 2. 1. 1. 1. 1. 2. 1. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 1. 2. 2. 1. 1. 2. 2. 2.
 2. 2. 2. 2.]
```

test_computeCov (test_MyDiscriminant.TestMyDiscriminant) (2/2)

```
This test the computeCov() helper functions.
We randomly generate 5 sets with 10*8 shape to find the COV
-----
Your solusion at Test 0 is:
[[ 0.0889507  -0.01647154 -0.02153902 -0.011146    0.01614989  0.04350251
   0.02469651  0.0287701 ]
 [-0.01647154  0.06973672  0.0388712  -0.01590982  0.00395175 -0.05082096
  -0.01641289  0.0018101 ]
 [-0.02153902  0.0388712   0.06625452 -0.01803218  0.01763471  0.00027809
  -0.01957927  0.00916325]
 [-0.011146   -0.01590982 -0.01803218  0.02375462 -0.00240452 -0.00639558
   0.00577777 -0.01034688]
 [ 0.01614989  0.00395175  0.01763471 -0.00240452  0.07941694 -0.0106533
   0.07544718  0.05285736]
 [ 0.04350251 -0.05082096  0.00027809 -0.00639558 -0.0106533   0.10160804
  -0.02614318  0.00096413]
 [ 0.02469651 -0.01641289 -0.01957927  0.00577777  0.07544718 -0.02614318
   0.11150464  0.06342069]
 [ 0.0287701   0.0018101   0.00916325 -0.01034688  0.05285736  0.00096413
   0.06342069  0.06656478]]
GT is:
[[ 0.0889507  -0.01647154 -0.02153902 -0.011146    0.01614989  0.04350251
   0.02469651  0.0287701 ]
 [-0.01647154  0.06973672  0.0388712  -0.01590982  0.00395175 -0.05082096
  -0.01641289  0.0018101 ]
 [-0.02153902  0.0388712   0.06625452 -0.01803218  0.01763471  0.00027809
  -0.01957927  0.00916325]
 [-0.011146   -0.01590982 -0.01803218  0.02375462 -0.00240452 -0.00639558
   0.00577777 -0.01034688]
 [ 0.01614989  0.00395175  0.01763471 -0.00240452  0.07941694 -0.0106533
   0.07544718  0.05285736]
 [ 0.04350251 -0.05082096  0.00027809 -0.00639558 -0.0106533   0.10160804
  -0.02614318  0.00096413]
 [ 0.02469651 -0.01641289 -0.01957927  0.00577777  0.07544718 -0.02614318
   0.11150464  0.06342069]
 [ 0.0287701   0.0018101   0.00916325 -0.01034688  0.05285736  0.00096413
   0.06342069  0.06656478]]
-----
Your solusion at Test 1 is:
[[ 0.05479457  0.00776728  0.02819913  0.04859511  0.0486294   0.01835938
  -0.01093299  0.04941813]
 [ 0.00776728  0.07361011  0.00340686  0.04364852  0.00536263  0.02299631
   0.01113939  0.03204267]
 [ 0.02819913  0.00340686  0.08482313  0.06943254  0.01998049  0.08094043
  -0.01020302  0.03810356]
 [ 0.04859511  0.04364852  0.06943254  0.10562777  0.04529594  0.0845132
  -0.0202048   0.07262209]
 [ 0.0486294   0.00536263  0.01998049  0.04529594  0.05543721  0.01407108
  -0.02652481  0.04746691]
 [ 0.01835938  0.02299631  0.08094043  0.0845132   0.01407108  0.10330048
  -0.0115905   0.05731749]
 [-0.01093299  0.01113939 -0.01020302 -0.0202048  -0.02652481 -0.0115905
```

```
  0.09447296 -0.04330112]
 [ 0.04941813  0.03204267  0.03810356  0.07262209  0.04746691  0.05731749
  -0.04330112  0.1105482 ]]
GT is:
[[ 0.05479457  0.00776728  0.02819913  0.04859511  0.0486294   0.01835938
  -0.01093299  0.04941813]
 [ 0.00776728  0.07361011  0.00340686  0.04364852  0.00536263  0.0229631
   0.01113939  0.03204267]
 [ 0.02819913  0.00340686  0.08482313  0.06943254  0.01998049  0.08094043
  -0.01020302  0.03810356]
 [ 0.04859511  0.04364852  0.06943254  0.10562777  0.04529594  0.0845132
  -0.0202048   0.07262209]
 [ 0.0486294   0.00536263  0.01998049  0.04529594  0.05543721  0.01407108
  -0.02652481  0.04746691]
 [ 0.01835938  0.0229631   0.08094043  0.0845132   0.01407108  0.10330048
  -0.0115905   0.05731749]
 [-0.01093299  0.01113939 -0.01020302 -0.0202048  -0.02652481 -0.0115905
   0.09447296 -0.04330112]
 [ 0.04941813  0.03204267  0.03810356  0.07262209  0.04746691  0.05731749
  -0.04330112  0.1105482 ]]
-----
Your solusion at Test 2 is:
[[ 0.08505974  0.03450228 -0.01193273 -0.00524796  0.01120509 -0.01875967
   0.00673483  0.01588703]
 [ 0.03450228  0.10518799  0.01518292 -0.04231919  0.04174065 -0.02661213
   0.010678    0.01386302]
 [-0.01193273  0.01518292  0.08547138 -0.04860232 -0.00703112  0.00141279
   0.0332502   0.03646121]
 [-0.00524796 -0.04231919 -0.04860232  0.08071985 -0.0218343   0.04525702
  -0.03339081 -0.03566894]
 [ 0.01120509  0.04174065 -0.00703112 -0.0218343   0.06660513 -0.02430969
   0.02139677 -0.01292649]
 [-0.01875967 -0.02661213  0.00141279  0.04525702 -0.02430969  0.04555685
  -0.01760783 -0.01280827]
 [ 0.00673483  0.010678    0.0332502  -0.03339081  0.02139677 -0.01760783
   0.05182447  0.03130241]
 [ 0.01588703  0.01386302  0.03646121 -0.03566894 -0.01292649 -0.01280827
   0.03130241  0.0574954 ]]
GT is:
[[ 0.08505974  0.03450228 -0.01193273 -0.00524796  0.01120509 -0.01875967
   0.00673483  0.01588703]
 [ 0.03450228  0.10518799  0.01518292 -0.04231919  0.04174065 -0.02661213
   0.010678    0.01386302]
 [-0.01193273  0.01518292  0.08547138 -0.04860232 -0.00703112  0.00141279
   0.0332502   0.03646121]
 [-0.00524796 -0.04231919 -0.04860232  0.08071985 -0.0218343   0.04525702
  -0.03339081 -0.03566894]
 [ 0.01120509  0.04174065 -0.00703112 -0.0218343   0.06660513 -0.02430969
   0.02139677 -0.01292649]
 [-0.01875967 -0.02661213  0.00141279  0.04525702 -0.02430969  0.04555685
  -0.01760783 -0.01280827]
 [ 0.00673483  0.010678    0.0332502  -0.03339081  0.02139677 -0.01760783
```

```
  0.05182447  0.03130241]
 [ 0.01588703  0.01386302  0.03646121 -0.03566894 -0.01292649 -0.01280827
  0.03130241  0.0574954 ]]
-----
Your solusion at Test 3 is:
[[ 0.09163497  0.01249203 -0.03393975  0.04103219 -0.01706239  0.00742286
  0.01981043  0.00538194]
 [ 0.01249203  0.04492758  0.00270599  0.0366516   0.01555546  0.00406492
  0.03895344  0.00778529]
 [-0.03393975  0.00270599  0.06417279  0.01686394  0.00415029  0.01892078
  0.0253209  -0.00051454]
 [ 0.04103219  0.0366516   0.01686394  0.0814071  -0.01178271  0.04871216
  0.05183951  0.00425153]
 [-0.01706239  0.01555546  0.00415029 -0.01178271  0.0371799  -0.01403554
  0.02177715 -0.00679106]
 [ 0.00742286  0.00406492  0.01892078  0.04871216 -0.01403554  0.05654842
  0.01760016 -0.03119823]
 [ 0.01981043  0.03895344  0.0253209   0.05183951  0.02177715  0.01760016
  0.08528434  0.00604487]
 [ 0.00538194  0.00778529 -0.00051454  0.00425153 -0.00679106 -0.03119823
  0.00604487  0.11629381]]
GT is:
[[ 0.09163497  0.01249203 -0.03393975  0.04103219 -0.01706239  0.00742286
  0.01981043  0.00538194]
 [ 0.01249203  0.04492758  0.00270599  0.0366516   0.01555546  0.00406492
  0.03895344  0.00778529]
 [-0.03393975  0.00270599  0.06417279  0.01686394  0.00415029  0.01892078
  0.0253209  -0.00051454]
 [ 0.04103219  0.0366516   0.01686394  0.0814071  -0.01178271  0.04871216
  0.05183951  0.00425153]
 [-0.01706239  0.01555546  0.00415029 -0.01178271  0.0371799  -0.01403554
  0.02177715 -0.00679106]
 [ 0.00742286  0.00406492  0.01892078  0.04871216 -0.01403554  0.05654842
  0.01760016 -0.03119823]
 [ 0.01981043  0.03895344  0.0253209   0.05183951  0.02177715  0.01760016
  0.08528434  0.00604487]
 [ 0.00538194  0.00778529 -0.00051454  0.00425153 -0.00679106 -0.03119823
  0.00604487  0.11629381]]
-----
Your solusion at Test 4 is:
[[ 0.12369371  0.07515975  0.03984408  0.03115606  0.00595641 -0.00662781
 -0.00615451 -0.03015835]
 [ 0.07515975  0.09499617  0.0115346   0.02843702  0.0120597   0.02199165
  0.01655307 -0.02350239]
 [ 0.03984408  0.0115346   0.09885199 -0.01907784  0.02493639 -0.00816587
 -0.01845778 -0.01257176]
 [ 0.03115606  0.02843702 -0.01907784  0.04881931 -0.02619879  0.00927612
  0.01999505  0.01941759]
 [ 0.00595641  0.0120597   0.02493639 -0.02619879  0.04557326  0.00662006
 -0.02384613 -0.00081745]
 [-0.00662781  0.02199165 -0.00816587  0.00927612  0.00662006  0.04495194
 -0.00035861 -0.00062773]
```

```
 [-0.00615451  0.01655307 -0.01845778  0.01999505 -0.02384613 -0.00035861
   0.05843185  0.02168865]
 [-0.03015835 -0.02350239 -0.01257176  0.01941759 -0.00081745 -0.00062773
   0.02168865  0.07056546]]
GT is:
[[ 0.12369371  0.07515975  0.03984408  0.03115606  0.00595641 -0.00662781
  -0.00615451 -0.03015835]
 [ 0.07515975  0.09499617  0.0115346   0.02843702  0.0120597   0.02199165
   0.01655307 -0.02350239]
 [ 0.03984408  0.0115346   0.09885199 -0.01907784  0.02493639 -0.00816587
  -0.01845778 -0.01257176]
 [ 0.03115606  0.02843702 -0.01907784  0.04881931 -0.02619879  0.00927612
   0.01999505  0.01941759]
 [ 0.00595641  0.0120597   0.02493639 -0.02619879  0.04557326  0.00662006
  -0.02384613 -0.00081745]
 [-0.00662781  0.02199165 -0.00816587  0.00927612  0.00662006  0.04495194
  -0.00035861 -0.00062773]
 [-0.00615451  0.01655307 -0.01845778  0.01999505 -0.02384613 -0.00035861
   0.05843185  0.02168865]
 [-0.03015835 -0.02350239 -0.01257176  0.01941759 -0.00081745 -0.00062773
   0.02168865  0.07056546]]
```

test_computeMean (test_MyDiscriminant.TestMyDiscriminant) (1/1)

This test the computeMean() helper functions.
We randomly generate 5 sets with 10*8 shape to find the mean

Your solusion at Test 0 is:
[0.48620993 0.60142359 0.42328614 0.5208569 0.62093252 0.48997793
 0.47616414 0.48229321]
GT is:
[0.48620993 0.60142359 0.42328614 0.5208569 0.62093252 0.48997793
 0.47616414 0.48229321]

Your solusion at Test 1 is:
[0.47078338 0.65417328 0.61337443 0.42501799 0.50691927 0.36362749
 0.40788188 0.5367783]
GT is:
[0.47078338 0.65417328 0.61337443 0.42501799 0.50691927 0.36362749
 0.40788188 0.5367783]

Your solusion at Test 2 is:
[0.50027491 0.58800114 0.29322758 0.51637001 0.54116711 0.4892752
 0.41076239 0.50394513]
GT is:
[0.50027491 0.58800114 0.29322758 0.51637001 0.54116711 0.4892752
 0.41076239 0.50394513]

Your solusion at Test 3 is:
[0.53886073 0.60022392 0.63459956 0.50831958 0.63470775 0.33840958
 0.56117373 0.43171471]
GT is:
[0.53886073 0.60022392 0.63459956 0.50831958 0.63470775 0.33840958
 0.56117373 0.43171471]

Your solusion at Test 4 is:
[0.38529714 0.50343767 0.37877246 0.41150032 0.2833117 0.54969377
 0.5030993 0.50614784]
GT is:
[0.38529714 0.50343767 0.37877246 0.41150032 0.2833117 0.54969377
 0.5030993 0.50614784]

test_computePrecision (test_MyDiscriminant.TestMyDiscriminant) (1/1)

this test the compute_precision() helper function
we randomly generate 5 sets with 100*1 shape predictions and ytest to find the precision
GT Precision: 0.7014925373134329; Your Precision: 0.7014925373134329
GT Precision: 0.6307692307692307; Your Precision: 0.6307692307692307
GT Precision: 0.8309859154929577; Your Precision: 0.8309859154929577
GT Precision: 0.625; Your Precision: 0.625
GT Precision: 0.7671232876712328; Your Precision: 0.7671232876712328

test_computePrior (test_MyDiscriminant.TestMyDiscriminant) (1/1)

This test the computePrior() helper functions.
We randomly generate 5 sets with 10*1 shape to find the prior
-----
The labels are:
[2. 2. 2. 2. 2. 2. 2. 2. 1. 2.]
Your solusion at Test 0 is:
[0.1 0.9]
GT is:
[0.1 0.9]

The labels are:
[1. 2. 2. 2. 1. 1. 2. 1. 2. 2.]
Your solusion at Test 1 is:
[0.4 0.6]
GT is:
[0.4 0.6]

The labels are:
[1. 2. 2. 2. 2. 1. 1. 2. 2. 1.]
Your solusion at Test 2 is:
[0.4 0.6]
GT is:
[0.4 0.6]

The labels are:
[2. 2. 1. 2. 1. 2. 2. 2. 2. 2.]
Your solusion at Test 3 is:
[0.2 0.8]
GT is:
[0.2 0.8]

The labels are:
[2. 2. 2. 2. 1. 2. 1. 2. 2. 1.]
Your solusion at Test 4 is:
[0.3 0.7]
GT is:
[0.3 0.7]

test_computeRecall (test_MyDiscriminant.TestMyDiscriminant) (1/1)

this test the compute_recall() helper function
we randomly generate 5 sets with 100*1 shape predictions and ytest to find the recall
GT Recall: 0.6575342465753424; Your Recall: 0.6575342465753424
GT Recall: 0.7088607594936709; Your Recall: 0.7088607594936709
GT Recall: 0.640625; Your Recall: 0.640625
GT Recall: 0.6764705882352942; Your Recall: 0.6764705882352942
GT Recall: 0.726027397260274; Your Recall: 0.726027397260274

Submitted Files

MyDiscriminant.py Download

```
import numpy as np
# some tips
# |S| is the determinant of S in the discriminant functions, try np.linalg.det()
# you can also directly get the inverse of a matrix by np.linalg.inv()


# ------------------------------------ You are going to implement 3 classifiers and corresponding helper functions -------------------
# ------------------------------------ Three classifiers start from here ------------------------------------------------------------
class GaussianDiscriminantBase:
    def __init__(self) -> None:
        pass

    def calculate_metrics(self, ytest, predictions):
        precision = compute_precision(ytest, predictions)
        recall = compute_recall(ytest, predictions)
        return precision, recall

class GaussianDiscriminant_C1(GaussianDiscriminantBase):
    # classifier initialization
    # input:
    #   k: number of classes (2 for this assignment)
    #   d: number of features; feature dimensions (8 for this assignment)
    def __init__(self, k=2, d=8):
        self.m = np.zeros((k,d))  # m1 and m2, store in 2*8 matrices
        self.S = np.zeros((k,d,d))   # S1 and S2, store in 2*(8*8) matrices
        self.p = np.zeros(2)  # p1 and p2, store in dimension 2 vectors

    # compute the parameters for both classes based on the training data
    def fit(self, Xtrain, ytrain):
        # Step 1: Split the data into two parts based on the labels
        Xtrain1, Xtrain2 = splitData(Xtrain, ytrain)

        # Step 2: Compute the parameters for each class
        # m1, S1 for class1
        self.m[0,:] = computeMean(Xtrain1)
        self.S[0] = computeCov( Xtrain1 )

        # m2, S2 for class2
        self.m[1,:]  = computeMean(Xtrain2)
        self.S[1] = computeCov( Xtrain2 )

        # priors for both class
        self.p = computePrior(ytrain)
```

```python
    # predict the labels for test data
    # Input:
    # Xtest: n*d
    # Output:
    # Predictions: n (all entries will be either number 1 or 2 to denote the labels)
    def predict(self, Xtest):
        # placeholders to store the predictions
        # can be ignored, removed or replaced with any following implementations
        predictions = np.zeros(Xtest.shape[0])

        # for indexing predictions in the loop
        i = 0

        # must convert the means to 2d matrices and transpose to a column vector
        #  at that point I can use formula 5.20 in the book
        mean0 = np.zeros((1, 8))
        mean0[0] = self.m[0]
        mean0 = mean0.transpose()
        mean1 = np.zeros((1, 8))
        mean1[0] = self.m[1]
        mean1 = mean1.transpose()

        for x_observation in Xtest:
            # change the vector to a column vector so it will work with the book's formula
            observation = np.zeros( (1, 8) )
            observation[0] = x_observation
            observation = observation.transpose()   # we pass in a column vectors!

            probClassC1 = computeDisc(observation, mean0, self.S[0], self.p[0])
            probClassC2 = computeDisc(observation, mean1, self.S[1], self.p[1])

            if probClassC1[0][0] > probClassC2[0][0]:
                predictions[i] = 1
            else:
                predictions[i] = 2
            i += 1

        return np.array(predictions)




class GaussianDiscriminant_C2(GaussianDiscriminantBase):
    # classifier initialization
    # input:
    #   k: number of classes (2 for this assignment)
    #   d: number of features; feature dimensions (8 for this assignment)
    def __init__(self, k=2, d=8):
        self.m = np.zeros((k,d))  # m1 and m2, store in 2*8 matrices
        self.shared_S =np.zeros((d,d))  # the shared convariance S that will be used for both classes
        self.p = np.zeros(2)  # p1 and p2, store in dimension 2 vectors

```

```python
    # compute the parameters for both classes based on the training data
    def fit(self, Xtrain, ytrain):
        # Step 1: Split the data into two parts based on the labels
        Xtrain1, Xtrain2 = splitData(Xtrain, ytrain)

        # Step 2: Compute the parameters for each class
        # m1
        self.m[0, :] = computeMean(Xtrain1)

        # m2
        self.m[1, :] = computeMean(Xtrain2)

        # compute the shared covariance
        self.shared_S = computeCov(Xtrain)  # compute the covariance on the whole data set!

        # priors for both class
        self.p = computePrior(ytrain)

        # Fill in your code here !!!!!!!!!!!!!!!!!!!!!!!!
        # Step 3: Compute the shared covariance matrix that is used for both class
        # shared_S is computed by finding a covariance matrix of all the data

    # predict the labels for test data
    # Input:
    # Xtest: n*d
    # Output:
    # Predictions: n (all entries will be either number 1 or 2 to denote the labels)
    def predict(self, Xtest):
        # placeholders to store the predictions
        # can be ignored, removed or replaced with any following implementations
        predictions = np.zeros(Xtest.shape[0])

        # for indexing predictions in the loop
        i = 0

        # must convert the means to 2d matrices and transpose to a column vector
        #  at that point I can use formula 5.20 in the book
        mean0 = np.zeros((1, 8))
        mean0[0] = self.m[0]
        mean0 = mean0.transpose()
        mean1 = np.zeros((1, 8))
        mean1[0] = self.m[1]
        mean1 = mean1.transpose()

        for x_observation in Xtest:
            # change the vector to a column vector so it will work with the book's formula
            observation = np.zeros( (1, 8) )
            observation[0] = x_observation
            observation = observation.transpose()   # we pass in a column vectors!

            probClassC1 = computeDisc(observation, mean0, self.shared_S, self.p[0])
            probClassC2 = computeDisc(observation, mean1, self.shared_S, self.p[1])
```

```python

            if probClassC1[0][0] > probClassC2[0][0]:
                predictions[i] = 1
            else:
                predictions[i] = 2
            i += 1

        return np.array(predictions)


class GaussianDiscriminant_C3(GaussianDiscriminantBase):
    # classifier initialization
    # input:
    #   k: number of classes (2 for this assignment)
    #   d: number of features; feature dimensions (8 for this assignment)
    def __init__(self, k=2, d=8):
        self.m = np.zeros((k,d))  # m1 and m2, store in 2*8 matrices
        self.shared_S = np.zeros((d,d))  # the shared convariance S that will be used for both classes
        self.p = np.zeros(2)  # p1 and p2, store in dimension 2 vectors

    # compute the parameters for both classes based on the training data
    def fit(self, Xtrain, ytrain):
        # Step 1: Split the data into two parts based on the labels
        Xtrain1, Xtrain2 = splitData(Xtrain, ytrain)

        # Step 2: Compute the parameters for each class
        # m1
        self.m[0, :] = computeMean(Xtrain1)

        # m2
        self.m[1, :] = computeMean(Xtrain2)

        # compute the shared covariance
        self.shared_S = np.diag(np.diag(computeCov(Xtrain)))

        # priors for both class
        self.p = computePrior(ytrain)

    # predict the labels for test data
    # Input:
    # Xtest: n*d
    # Output:
    # Predictions: n (all entries will be either number 1 or 2 to denote the labels)
    def predict(self, Xtest):
        # placeholders to store the predictions
        # can be ignored, removed or replaced with any following implementations
        predictions = np.zeros(Xtest.shape[0])

        # for indexing predictions in the loop
        i = 0

        # must convert the means to 2d matrices and transpose to a column vector
```

```
        #  at that point I can use formula 5.20 in the book
        mean0 = np.zeros((1, 8))
        mean0[0] = self.m[0]
        mean0 = mean0.transpose()
        mean1 = np.zeros((1, 8))
        mean1[0] = self.m[1]
        mean1 = mean1.transpose()

        for x_observation in Xtest:
            # change the vector to a column vector so it will work with the book's formula
            observation = np.zeros( (1, 8) )
            observation[0] = x_observation
            observation = observation.transpose()   # we pass in a column vectors!

            probClassC1 = computeDisc(observation, mean0, self.shared_S, self.p[0])
            probClassC2 = computeDisc(observation, mean1, self.shared_S, self.p[1])

            if probClassC1[0][0] > probClassC2[0][0]:
                predictions[i] = 1
            else:
                predictions[i] = 2
            i += 1

        return np.array(predictions)


# ------------------------------------ Helper Functions start from here -------------------------------------------------------------
# Input:
# features: n*d matrix (n is the number of samples, d is the number of dimensions of the feature)
# labels: n vector
# Output:
# features1: n1*d
# features2: n2*d
# n1+n2 = n, n1 is the number of class1, n2 is the number of samples from class 2
def splitData(features, labels):

    # defensive programming: make sure features and labels are of the same size.
    if (np.size(features, 0) != np.size(labels)):
        raise IndexError( "The number of rows of features and labels do not match.")

    # placeholders to store the separated features (feature1, feature2),
    # can be ignored, removed or replaced with any following implementations
    features1 = np.zeros([np.sum(labels == 1),features.shape[1]])  # array[ num of class y=1 values, number of features
    features2 = np.zeros([np.sum(labels == 2),features.shape[1]])  # array[ num of class y=2 values, number of features

    # need to know what index each feature array is at so I can copy from main feature array.
    # I could rely on the fact that the data has all of class 1 first and then all of class 2 but that seems a wonky
    #   and bug-ridden way to program.
```

```
    featuresIndex = np.array( [0,0] )

    # separate the features according to the corresponding labels, for example
    # if features = [[1,1],[2,2],[3,3],[4,4]] and labels = [1,1,1,2], the resulting feature1 and feature2 will be
    # feature1 = [[1,1],[2,2],[3,3]], feature2 = [[4,4]]
    for i in range(0, len(labels)):
        if labels[i] == 1:
            features1[ featuresIndex[0] ] = features[i]
            featuresIndex[0] += 1
        elif labels[i] == 2:
            features2[ featuresIndex[1] ] = features[i]
            featuresIndex[1] += 1
        else:
            raise ValueError("Class in data not in {1,2}.")

    return features1, features2


# compute the mean of input features
# input:
# features: n*d
# output: d
def computeMean(features):

    # placeholders to store the mean for one class
    # can be ignored, removed or replaced with any following implementations
    m = np.zeros(features.shape[1])

    # fill in the code here !!!!!!!!!!!!!!!!!!!!!!
    # try to explore np.mean() for convenience
    # decided to go a different route so I could practice with routines I will probably use for calculating variance
    #   and covariance

    m = (np.sum(features, axis = 0))

    # the number of columns is features[0] while the number of rows is len(features)
    m = np.divide( m, float( len(features) ) )

    return m



# wrapper function that calls np.cov() and computes the covariance
# input:
# features: n*d
# output: d*d
def computeCov(features):
    # placeholders to store the covariance matrix for one class
    # can be ignored, removed or replaced with any following implementations
    covMatrix = np.eye(features.shape[1])
```

```
    # try to explore np.cov() for convenience
    covMatrix = np.cov(features, rowvar = False)

    return covMatrix


# compute the priors of input features
# input:
# labels: n*1
# output: 2
def computePrior(labels):
    # placeholders to store the priors for both class
    # can be ignored, removed or replaced with any following implementations
    p = np.array([0.5,0.5])

    # p[0] contains prior for class 1
    # p[1] contains prior for class 2
    p[0] = np.count_nonzero(labels == 1.0) / float( len(labels ) )
    p[1] = np.count_nonzero(labels == 2.0) / float( len(labels ) )

    return p

# compute the discriminant function
# input:
# observation: d * 1
# means: d * 1
# covMatrix: d*d
# prior: scaler
def computeDisc(observation, means, covMatrix, prior):

    return (-.5) * np.log( np.linalg.det( covMatrix ) ) - 0.5 * ( observation.T @ np.linalg.inv(covMatrix ) @ observation - 2 * observation.T @ np.linalg.inv( covMatrix ) @ means +  means.T @ np.linalg.inv( covMatrix) @ means ) + np.log( prior )


# compute the precision
# input:
# ytest: the ground truth labels of the test data, n*1
# predictions: the predicted labels of the test data, n*1
# output:
# precision: a float with size 1
def compute_precision(ytest, predictions):
    precision = 0.0 # a place holder can be neglected

    # precision = countOf[true positive predictions] / countOf[positive predictions]
    # here we assume label==2 is the positive label
    truePositives = 0
    countOfPositives = 0
    for i in range( len (predictions) ):

        if ( predictions[i] == 2 ):
```

```
            countOfPositives += 1

            if ( ytest[i] == 2 ):
                truePositives += 1

    precision = truePositives / countOfPositives
    return precision

# compute the recall
# input:
# ytest: the ground truth labels of the test data, n*1
# predictions: the predicted labels of the test data, n*1
# output:
# recall: a float with size 1
def compute_recall(ytest, predictions):
    recall = 0.0 # a place holder can be neglected

    # recall = countOf[true positive predictions] / countOf[positive labels in ytest]
    # here we assume label==2 is the positive label
    truePositives = 0
    positiveLabelsInTest = 0
    for i in range( len (predictions) ):

        if ytest[i] == 2:
            positiveLabelsInTest += 1

            if ( predictions[i] == 2):
                truePositives += 1

    recall = truePositives / positiveLabelsInTest
    return recall
```