CS18M524 _Report

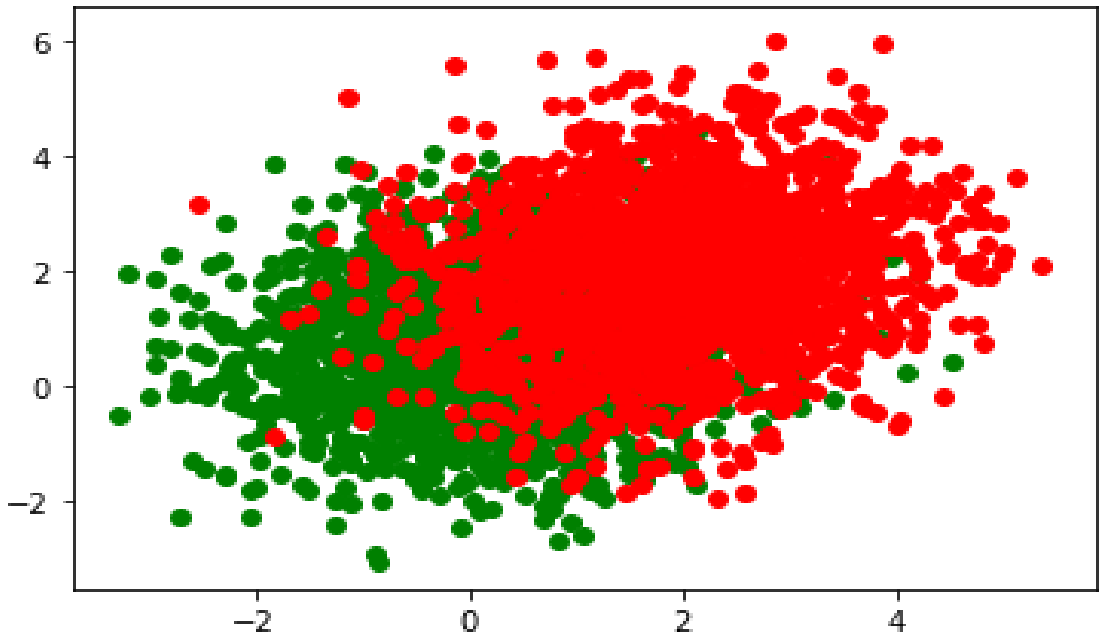
Please read the README file for running instructions

Question 1:

20 features were generated with a total of 2000 samples in a multivariate Gaussian distribution.

Class X = green

Class Y = red



For this a random numpy mean and covariance were generated. The other mean was obtained from the first mean by doing mean_1 = mean_0 +2* dist_mean(features). With these 2 means and a non spherical covariance , 2 classes were generated with 2000 features each. The covariance matrix was printed as follows.

```
Covariance matrix ::::::
 [[4.59026914e-01 4.44150576e-01 8.17359995e-01 7.43829130e-01
  1.08129384e-01 8.88961278e-01 1.26908775e-01 3.56997245e-02
  6.41255657e-01 8.12831409e-01 7.38054834e-01 4.28894197e-01
  2.87038022e-02 5.51175091e-01 5.03461873e-02 4.64341992e-01
  1.48744815e-01 8.77808779e-01 4.94967622e-01 3.13456989e-01]
 [9.14891290e-01 9.85384236e-02 8.16524926e-01 8.10468057e-02
  9.42600737e-01 6.30303375e-01 9.88021652e-01 3.34427534e-01
  7.65403320e-01 3.46704864e-01 4.33609717e-01 2.83327581e-01
  7.21652867e-01 2.15677305e-01 3.05907766e-01 3.97959544e-01
  9.38727328e-01 2.16581887e-01 7.81976676e-01 2.44283390e-01]
 [1.50055819e-01 6.62578472e-01 2.63653055e-01 8.63920813e-01
  5.35933627e-01 3.24076456e-01 1.98348860e-01 3.12539179e-01
  1.13988358e-01 7.68455027e-01 1.32370320e-01 6.23462935e-01
  6.19461109e-01 7.32792865e-01 4.27939682e-01 1.75612840e-01
```

```
 8.83538470e-01 8.00561884e-01 1.54940132e-01 7.57972929e-01]
[6.87522535e-01 3.66694677e-01 4.08354238e-02 9.12027820e-01
 2.47233555e-01 2.60410042e-01 8.67288654e-01 1.99804551e-03
 9.16122355e-02 6.48512935e-02 4.07627384e-01 1.10392072e-01
 6.14256139e-01 4.73479195e-01 3.59119093e-02 1.67266087e-01
 6.45446520e-01 4.38225867e-01 6.34475248e-01 1.93273155e-01]
[3.08534533e-01 3.81057377e-01 5.82789956e-01 7.12750617e-01
 3.24969029e-01 7.97983671e-01 2.08720916e-01 3.39472472e-01
 2.83174857e-02 8.67968402e-01 3.92692126e-02 3.75301183e-01
 7.05966736e-01 5.78321988e-01 6.76473327e-01 8.11635603e-01
 7.21459344e-01 5.73783860e-01 7.49668011e-01 9.43769916e-01]
[7.88897728e-01 8.32750139e-01 5.92247460e-01 4.31977487e-01
 8.01828596e-01 7.70177454e-01 6.73491117e-01 5.21196535e-01
 7.31393564e-01 4.61012023e-01 3.08561181e-01 2.07479104e-02
 1.66851892e-01 3.63906355e-01 1.52877293e-01 1.03017962e-01
 1.39890763e-01 1.83456605e-01 7.39897465e-01 6.84607079e-01]
[2.94796814e-01 8.23154369e-01 2.63413719e-01 9.83757911e-01
 5.68362498e-02 1.88595180e-02 8.35439353e-02 7.64619025e-01
 8.16439376e-01 5.89058844e-01 5.13466924e-01 6.88178824e-01
 2.99601215e-01 5.86075562e-04 3.60114975e-01 2.82690029e-01
 8.76131336e-01 7.29491596e-01 9.69336693e-01 7.37692351e-01]
[1.42888924e-01 9.03634676e-01 8.66594822e-01 6.12255163e-01
 6.38343813e-01 9.89229680e-01 5.42950409e-01 4.86942826e-01
 9.99450358e-01 3.66315280e-02 5.96842271e-02 7.37908041e-01
 2.98731842e-02 3.93182135e-01 9.95526268e-01 6.84139291e-01
 9.63355917e-01 9.06490307e-01 9.37823524e-01 2.74715551e-01]
[3.41105597e-01 7.35984161e-01 6.26059863e-01 2.61043951e-01
 9.51210439e-01 2.46242384e-01 3.04372010e-01 4.78223204e-01
 4.62925233e-01 7.42842909e-01 3.00132113e-01 3.03229129e-02
 2.59380351e-01 5.76017164e-01 4.36224372e-01 2.91170423e-01
 3.18479569e-01 8.33911311e-01 4.48173739e-01 5.92535540e-01]
[9.98559631e-01 8.78726153e-01 4.24484079e-01 7.25840414e-01
 9.91450259e-01 7.70124539e-01 8.29416470e-01 2.59936599e-01
 8.78890555e-01 1.60054670e-01 5.04850988e-01 7.95508234e-01
 3.38075130e-02 9.88652517e-01 6.48372287e-01 5.28001326e-02
 8.38181077e-01 1.36531533e-01 7.42533244e-01 5.79204505e-01]
[1.89861577e-01 8.96301296e-01 3.09161254e-01 9.11462221e-01
 1.06512655e-01 5.88505217e-01 8.56981072e-01 4.25406519e-01
 4.71922500e-01 3.16000338e-01 7.28817549e-01 9.70050890e-01
 3.00170349e-01 3.01207269e-02 9.93855159e-01 8.92070966e-01
 5.12137534e-01 2.51662700e-02 9.44873233e-01 8.39985634e-02]
[1.83314067e-01 3.72381013e-01 1.27178640e-01 5.91855211e-01
 4.41478223e-01 7.70640604e-01 5.64446557e-01 6.76145665e-01
 2.12311973e-01 1.04171934e-01 6.47711463e-01 9.54873841e-01
 3.39920401e-01 9.08231181e-01 2.79429812e-01 8.10594714e-01
 1.24967258e-01 9.16330984e-01 9.76501926e-01 1.20706376e-01]
[1.45495156e-01 6.39502050e-01 5.08745853e-01 6.35190400e-01
 1.33971017e-02 6.05721068e-01 2.34653281e-02 5.19002753e-01
 2.61684979e-01 7.13692115e-01 8.86956835e-01 5.47327368e-01
 7.31840490e-01 5.16410856e-01 7.95047035e-01 5.35400712e-01
 3.07932163e-01 5.03610606e-02 8.06093866e-01 9.65187668e-01]
[1.32017170e-01 4.49549819e-01 5.47571398e-01 8.94840446e-01
 7.42460093e-01 6.71237239e-01 8.86316761e-01 8.09933833e-01
 4.02322472e-02 5.90532833e-01 9.56663535e-01 4.75514763e-01
```

```
  5.33846495e-01 6.45201089e-01 7.24144715e-01 6.23743270e-01
  7.31304256e-02 9.65252438e-01 1.72410363e-01 5.11738865e-02]
 [5.23631860e-01 3.47760892e-01 9.91381517e-01 2.75694578e-01
  8.38997340e-01 3.05667415e-01 6.94773146e-01 2.89650874e-01
  7.98221017e-01 7.00402365e-01 1.76010274e-01 9.15463021e-01
  5.65096224e-01 1.96408508e-01 4.05932307e-01 5.38380997e-03
  6.89143739e-02 9.15321322e-01 3.07055013e-01 7.61473270e-01]
 [4.70866219e-02 7.38643799e-01 5.95919510e-01 6.77846474e-02
  4.79377240e-01 2.98248282e-01 5.29872445e-01 2.53533483e-01
  6.61604447e-01 8.15388123e-01 8.42052331e-01 8.92163466e-01
  5.91413697e-01 7.22715402e-01 2.71030904e-01 8.56426355e-02
  7.53081161e-01 1.30055624e-01 7.94971813e-01 3.07970588e-01]
 [2.06040679e-01 5.40115481e-01 9.65022482e-01 6.98611940e-01
  7.26367105e-02 1.14820549e-01 1.03803638e-03 6.91459615e-01
  3.37649241e-01 3.26257037e-02 5.19637746e-01 6.05258314e-02
  8.85035306e-01 3.42507791e-01 8.86788035e-01 3.08809501e-01
  6.36076364e-01 2.65013212e-01 4.49715513e-01 8.81738070e-02]
 [9.80322534e-02 3.55974469e-01 9.07475237e-02 5.14987169e-01
  1.37003791e-01 9.79625543e-01 5.60548577e-01 1.47100403e-01
  7.70670034e-01 2.45223388e-01 4.03719666e-01 9.44930564e-01
  8.41828238e-01 2.33698630e-01 1.44691242e-01 2.51300887e-01
  7.78727990e-01 6.40453396e-01 7.31619568e-01 3.88406743e-01]
 [6.90472185e-01 1.23596967e-01 1.34335591e-01 2.69867059e-01
  4.59252024e-01 6.38636013e-01 2.15949374e-01 6.71837480e-01
  4.91122233e-01 8.68793443e-01 4.32732651e-01 3.68386505e-01
  9.79757450e-01 9.22913400e-02 9.73582190e-01 6.80849871e-01
  1.54123717e-01 7.25163177e-01 9.49952941e-01 5.81810438e-01]
 [2.79274029e-01 6.92847436e-01 4.67093355e-01 9.96003693e-01
  8.17839857e-01 8.40792332e-01 5.82155550e-01 4.98466261e-01
  4.86071790e-02 8.43664647e-01 7.50331007e-01 4.48137384e-01
  5.99664387e-01 1.40952492e-01 4.13095308e-01 3.25501588e-01
  3.91189967e-01 3.47264672e-01 8.84483098e-01 6.53461603e-01]]
```

Thereafter the generated samples were split into test-training samples with a 3:7 split. This was done using train_test_split imported from sklearn.model_selection. The Xtest dataset was called DS1 as required.

Question 2:

Linear Classification

A linear classifier was learnt for DS1. For this SGDClassifier was imported from sklearn.linear_model. To this classifier we fit the Xtrain and ytrain datasets too train the classifier.

The learnt co-efficients are :

```
Coefficients learnt for Linear Classifier::::::::::::::
 [[-0.22307286 -0.25645516 -0.13414231  0.78127622  0.00585741  0.02423993
  -0.92928414 -0.40588433 -0.20257899  0.03534471 -0.23915428 -0.00122009
   0.46546661 -0.81770583 -0.51603534  0.02309113  0.04995469  0.28394915
  -0.41174267  0.33099656]]
```

Then we test the classifier on the training as well as test sets to obtain the following results:

Results of Linear Classification on the training set are ::::::::::::::::

	precision	recall	f1-score	support
0	0.59	0.02	0.05	702
1	0.50	0.98	0.66	698
accuracy			0.50	1400
macro avg	0.54	0.50	0.35	1400
weighted avg	0.54	0.50	0.35	1400

Results of Linear Classification on the test are ::::::::::::::::

	precision	recall	f1-score	support
0	0.64	0.02	0.05	298
1	0.51	0.99	0.67	302
accuracy			0.51	600
macro avg	0.57	0.51	0.36	600
weighted avg	0.57	0.51	0.36	600

Question 3:

K-Neareset Neighbours

For DS1, use k-NN to learn a classifier. Repeat the experiment for different values of k and report the perf ormance for each value. Technically this is not a linear classifier, but we want you to appreciate how pow erful linear classifiers can be. • Do you do better than regression on indicator variables or worse? • Are the re particular values of k which perform better? • Report the best fit accuracy, precision, recall and f-measu re achieved by this classifier.

A K-NN classifier was learnt for DS1. For this KNeighborsClassifier was imported from sklearn.neighbors.

For K = 5 the results are as follows:-

Results of K-NN on the training set are ::::::::::::::::

	precision	recall	f1-score	support
0	0.67	0.69	0.68	702
1	0.68	0.66	0.67	698
accuracy			0.68	1400
macro avg	0.68	0.67	0.67	1400
weighted avg	0.68	0.68	0.67	1400

Results of K-NN the test are ::::::::::::::::

	precision	recall	f1-score	support

0	0.48	0.51	0.50	298
1	0.49	0.46	0.48	302
accuracy			0.49	600
macro avg	0.49	0.49	0.49	600
weighted avg	0.49	0.49	0.49	600

For K = 50 results are as follows:

Results of K-NN on the training set are ::::::::::::::::

	precision	recall	f1-score	support
0	0.52	0.65	0.58	702
1	0.53	0.40	0.46	698
accuracy			0.53	1400
macro avg	0.53	0.53	0.52	1400
weighted avg	0.53	0.53	0.52	1400

Results of K-NN the test are ::::::::::::::::

	precision	recall	f1-score	support
0	0.50	0.62	0.55	298
1	0.50	0.38	0.43	302
accuracy			0.50	600
macro avg	0.50	0.50	0.49	600
weighted avg	0.50	0.50	0.49	600

For K= 500 results are as follows:-

Results of K-NN on the training set are ::::::::::::::::

	precision	recall	f1-score	support
0	0.51	0.68	0.58	702
1	0.52	0.35	0.42	698
accuracy			0.51	1400
macro avg	0.51	0.51	0.50	1400
weighted avg	0.51	0.51	0.50	1400

Results of K-NN the test are ::::::::::::::::

	precision	recall	f1-score	support
0	0.52	0.70	0.60	298
1	0.55	0.36	0.44	302
accuracy			0.53	600

```
   macro avg       0.53      0.53      0.52       600
weighted avg       0.53      0.53      0.51       600
```

For K= 1000 results are as follows:-

```
Results of K-NN on the training set are ::::::::::::::::

              precision    recall  f1-score   support

           0       0.52      0.73      0.60       702
           1       0.53      0.31      0.39       698

    accuracy                           0.52      1400
   macro avg       0.52      0.52      0.50      1400
weighted avg       0.52      0.52      0.50      1400

Results of K-NN the test are ::::::::::::::::

              precision    recall  f1-score   support

           0       0.51      0.71      0.59       298
           1       0.53      0.31      0.39       302

    accuracy                           0.51       600
   macro avg       0.52      0.51      0.49       600
weighted avg       0.52      0.51      0.49       600
```

- The accuracy on test data is slightly better than linear classification as k increases upto a certain value
- The accuracy on test data increases as k increases upto a certain value (between 500 and 1000). After that the accuracy is almost constant or decreases marginally (by .01 or .02)

Question 4:

The Communities and Crime (CandC) Data Set was downloaded and the data (with missing values) is present in communities.data and description is present in communities.names in the same folder.

The missing values were filled by the use of Imputer from from sklearn.preprocessing. To do this first the '?' or missing values were replaced by numpy.NaN.

Column 4 contained community name which was a string. Imputer does not work on strings , hence this column was converted to numeric to apply the imputer.

The dataframe after imputation with the mean of every column was stored in imputed_DF which was then stored back in df.The community names column was deleted from the imputed dataframe. This does not matter as the first 5 columns are non predictive.

The dataset with the missing values substituted by mean of columns was stored in CandC_no_missing_Values.xlsx present in the same folder.

- Mean is not a good choice for imputation although it is easy and fast because of the following reasons:

a. Doesn't factor the correlations between features. It only works on the column level.

b. Will give poor results on encoded categorical features .

c. Not very accurate.

d. Doesn't account for the uncertainty in the imputations.

- **Instead we can use K-NN for imputation.** This will five much better accuracy than mean/median or most frequent value imputation but it is computationally intensive and more sensitive to training data.

In this assignment I have used column mean for imputation since it is easy and fast.

Question 5:.

The CandC dataset with no missing values was taken in a dataframe and it was split into training and test sets with ratio 8:2.

Five different training and test sets were obtained like this. These training and test sets are available as .csv files in the same folder. Eg CandC_train_num1.csv , CandC_test_num1.csv and so on.

Each of the 5 training data was fitted into Linear Regression.model by importing LinearRegression from from sklearn.linear_model. And then the corresponding classifier was used to predict the test set classification.

The results are as follows:-

```
_____________________Linear Regression_____________________________________

//////////All test and train sets have been stored in .csv files in the sa
me folder////////////////////

For set 1 rss1 =  8.376488646756119

For set 2 rss2 =  8.348527718781192

For set 3 rss3 =  8.303116505892948

For set 4 rss4 =  6.974667827444956

For set 5 rss5 =  7.98050868686426

Average rss of 5 sets :::::::: 7.996661877147895
```

Question 6:

The CandC dataset with no missing values was taken in a dataframe and it was split into training and test sets with ratio 8:2.

Five different training and test sets were obtained like this. These training and test sets are available as .csv files in the same folder. Eg CandC_train_num1.csv , CandC_test_num1.csv and so on.

Each of the 5 training data was fitted into Ridge Regression.model by importing Ridge from sklearn.linear_model. And then the corresponding classifier was used to predict the test set classification.

This was done for 3 values of alpha to show variations. The results are as follows:-

_____Answer to Question 6_____

_____Ridge Regression_____

//////////All Coefficients have been stored in .csv files in the same folder//////////////////

////////////// alpha1 = 1e+55 /////////////////

for alpha = 1e+55 set 1 rss1 = 21.43068144342135

for alpha = 1e+55 set 2 rss2 = 23.72699997291693

for alpha = 1e+55 set 3 rss3 = 24.346638022464397

for alpha = 1e+55 set 4 rss4 = 21.721169899077253

for alpha = 1e+55 set 5 rss5 = 22.79412525470465

for alpha = 1e+55 Average rss :::::: 22.803922918516914

////////////// alpha2 = 0 /////////////////

for alpha = 0 set 1 rss1 = 44.642300000000006

for alpha = 0 set 2 rss2 = 8.348527718778808

for alpha = 0 set 3 rss3 = 8.303116505885578

for alpha = 0 set 4 rss4 = 6.97466782744305

for alpha = 0 set 5 rss5 = 7.980508686865085
for alpha = 0 Average rss ::::::::: 15.249824147794504

////////////// alpha3 = -1e-55/////////////////

```
for alpha = -1e-55 set 1 rss1 =  8.376488646757338

for alpha = -1e-55 set 2 rss2 =  8.348527718778808

for alpha = -1e-55 set 3 rss3 =  8.303116505885578

for alpha = -1e-55 set 4 rss4 =  6.97466782744305

for alpha = -1e-55 set 5 rss5 =  7.980508686865085

for alpha = -1e-55 Average rss :::::::  7.996661877145972
```

- Alpha = -1e-55 gives the best fit
- Yes it is possible to do feature selection with the data that we obtained here. For this we have to SelectFromModel ridge reression with some penalty i.e L2 penalty in ridge Regression case. This can be done using **`from sklearn.feature_selection import SelectFromModel`**
  After this we can put a scaled feature set Xtrain.
  Then we do `SelectFromModel(clf)` and fit the scaled data. Then we obtain the features retained by ridge using sel.get_support(). This will give us a subset of all selected features.

Question 7:

Download dataset DS2 from here. Perform 2-class Logistic Regression on it. Report per-class precision, recall and f-measure on the test data. Now perform L1-regularized Logistic Regression on the same dataset and report similar performance results. [Optional for MATLAB Users: Use l1 logreg code provided by Boyds Group (http://www. stanford.edu/~boyd/l1_logreg/)]

The dataset DS2 was downloaded and stored as DS2-test.csv and DS2-train.csv. First 96 column of the training data as well as test data were assumed to be features and the last column was assumed to be the classification result.

LogisticRegression was impoted from sklearn.linear_model. The training data was scaled using MinMaxScaler() from preprocessing package to help convergence of the data.

Then this data was fit into the model. And the trained model was used to classify the test set. The results are as follows:-

```
Results on the test are ::::::::::::::::

              precision    recall  f1-score   support

        -1.0       1.00      0.79      0.88        19
         1.0       0.83      1.00      0.91        20

    accuracy                           0.90        39
   macro avg       0.92      0.89      0.90        39
```

```
weighted avg       0.91      0.90      0.90        39
```

In the next part L1-regularized Logistic Regression was performed on the same dataset with different constants. All coefficients have been printed along with the results :-

```
_____________________________________________Logistic Regression with L1 regular
ization _____________________________________________

C: 10
Coefficient of each feature:
 [[-0.23533856 -7.39787113  4.09086989  0.         -0.33367451 -3.70380366
   0.          0.          5.61099142 -2.02689231  1.69423653 -0.9810526
   2.86470153  0.         -0.83220355  1.69209634 -2.42031825  2.40402247
   0.27343637  0.          0.77921769 -2.28589397  0.         -3.59044027
   0.          0.         -2.28909014  0.         -3.5392342   0.
  -2.98971908  6.22207385 -4.88516331  0.          0.         -7.27119182
   0.         -0.39043052  0.          0.         -3.14436543  0.
   0.         -9.03829266 -7.65278129  7.30075469 -1.49882774  0.
  -2.81580144  0.          4.99711298 -2.54505263  4.64609728  0.
   1.24073927  0.          2.4728722   0.          0.79787998  0.
   0.          8.04848006 -2.88666705  4.13787525  9.48902027  5.75504323
   0.          8.10402404  4.51834997  0.          1.32644575  1.74304264
   0.          3.04393739  0.         -0.66631351  4.35114168  0.
  -0.69821006  0.          4.93033227  0.         -6.00352393  0.
   0.         -0.93261171  0.         -5.80744897  0.         -6.28089061
   0.         -4.63502194 -5.47055312  0.         -0.31677395 -4.44739644]
]

Results on the test are ::::::::::::::::

              precision    recall  f1-score   support

        -1.0       0.95      0.95      0.95        19
         1.0       0.95      0.95      0.95        20

    accuracy                           0.95        39
   macro avg       0.95      0.95      0.95        39
weighted avg       0.95      0.95      0.95        39

C: 1
Coefficient of each feature:
 [[-2.37721516 -1.14487319  0.          0.         -0.61028864 -0.1014559
   0.          0.          0.          0.          0.44778084  0.
   0.          0.          0.          0.16521023  0.          1.27341472
   0.          0.          0.          0.         -0.83844586  0.
   0.          0.          0.          0.          0.          0.
   0.          0.54383213 -0.7263387   0.          0.          0.
   0.         -0.4695749   0.          0.         -1.49523137  0.
   0.         -2.16967142  0.          0.          0.34880286  0.
   0.          0.          1.55481274  0.          1.54474372  0.
   0.          0.          0.          0.          0.          0.
```

```
   0.          0.          0.          0.          4.58358191  1.989458
   0.02194217  0.          2.82107275  0.          0.97482423  0.
   0.          0.47418753  0.          0.          0.          0.
   0.          0.          0.          0.         -2.67723877 -0.69436208
  -0.44393066 -1.43085445  0.         -5.94253089  0.         -0.67679957
  -0.80463765 -4.28609146 -2.98701657  0.          0.          0.        ]
]

Results on the test are ::::::::::::::::

              precision    recall  f1-score   support

        -1.0       0.78      0.95      0.86        19
         1.0       0.94      0.75      0.83        20

    accuracy                           0.85        39
   macro avg       0.86      0.85      0.85        39
weighted avg       0.86      0.85      0.84        39

C: 0.1
Coefficient of each feature:
 [[ 0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.55447457  0.
   0.          0.          1.41922969  0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.         -0.16959098
   0.          0.          0.         -1.91994959  0.          0.
   0.         -2.24570342  0.          0.          0.          0.        ]
]

Results on the test are ::::::::::::::::

              precision    recall  f1-score   support

        -1.0       0.88      0.79      0.83        19
         1.0       0.82      0.90      0.86        20

    accuracy                           0.85        39
   macro avg       0.85      0.84      0.85        39
weighted avg       0.85      0.85      0.85        39
```