DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Homework Assignment No. 04:

HW No. 04: Linear Discriminant Analysis

submitted to:

Professor Joseph Picone
ECE 8527: Introduction to Pattern Recognition and Machine Learning
Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

February 9th, 2023

prepared by:

Gavin Koma
Email: gavintkoma@temple.edu

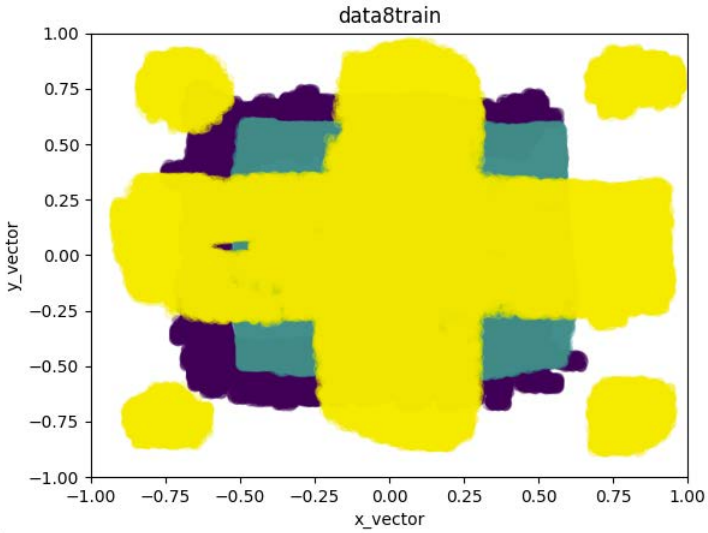
A. TASK 1

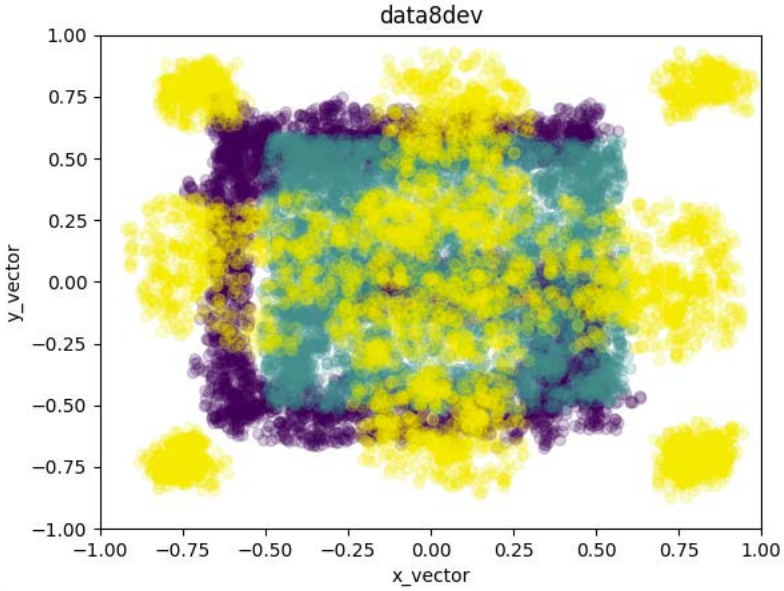
The first of three tasks for this homework assignment was to plot all sets of data for visualization. The instructions state that all axes should have the same limits; however, different datasets have notably different limits for their data. Therefore, I have elected to use the same axes for data that comes from the same data set, but individual datasets will have their own axes to best visualize their datapoints. The first of these datasets was from dataset 8 and includes three subfolders: dev, eval, and train csv files.

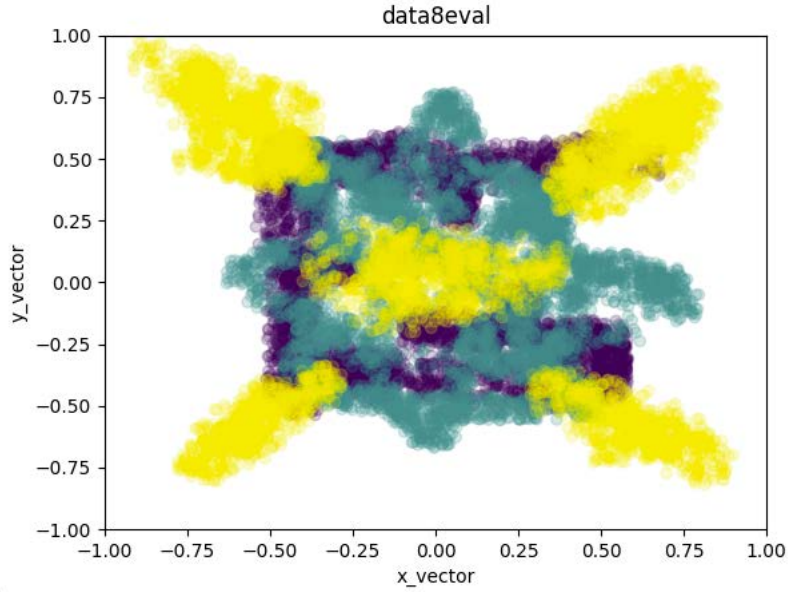
All graphs from the same dataset are placed onto one single page to prevent confusion during comparison and any mismatching that may occur.

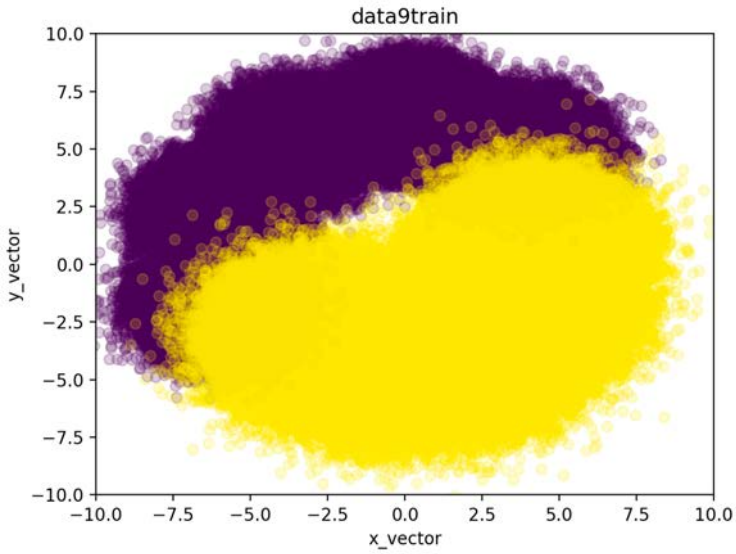
The code that was utilized to create these graphs worked by using a for-loop that created a dictionary of datasets. The titles were populated for each graph by using another dictionary and iterating through that via indexing of that dictionary.

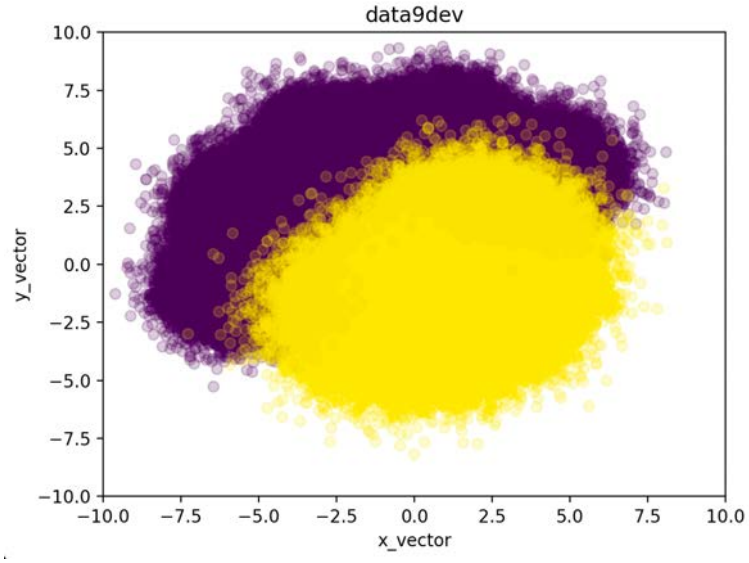
```
#%% start with modules
import pandas as pd
import glob
import os
import matplotlib.pyplot as plt
import matplotlib

os.chdir(r'/Users/gavinkoma/Desktop/pattern_rec/homework4/data')
print(os.getcwd())

files = glob.glob('*.csv') #make a list of all the csv files

d_all = {} #init dict
for file in files: #loop through .csv names
    #make a dict for all of them and add them to use later
    #there are no headers for any of this stuff, dont forget
    d_all["{0}".format(file)] = pd.read_csv(file,header=None)

name_dict = {
    0: "data10train",#data10train
    1: "data9train", #data9train
    2: "data8train", #data8train
    3: "data9eval", #data9eval
    4: "data8dev",#data8dev
    5: "data10dev",#data10dev
    6: "data10val",#data10eval
    7: "data8eval", #data8eval
    8: "data9dev" #data9dev
}

custom_xlim = (0, 100)
custom_ylim = (-100, 100)

#%% plots

data_val = list(d_all.values())

for i,kval in enumerate(data_val):
    plt.figure()
    #print(kval[1],kval[2])
    plt.scatter(kval[1],kval[2],c=kval[0],alpha=0.2)
    plt.xlim(-2,2)
    plt.ylim(-2,2)
    plt.title(name_dict[i])
    plt.ylabel("y_vector")
    plt.xlabel("x_vector")
    plt.savefig(str(name_dict[i]))
    plt.show()
```

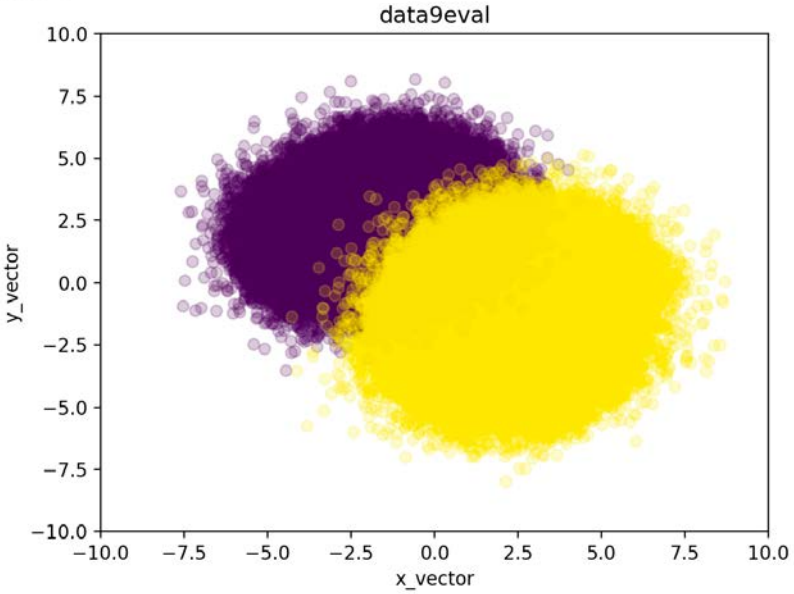

data8train
y_vector
x_vector
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

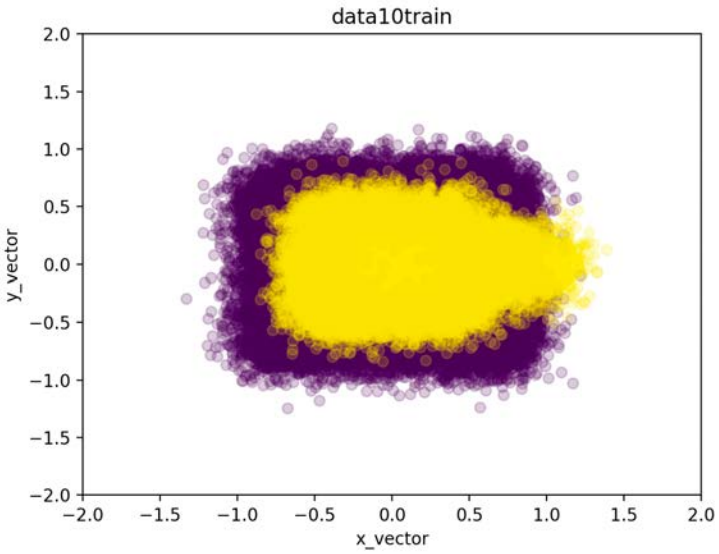

data8dev
y_vector
x_vector
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

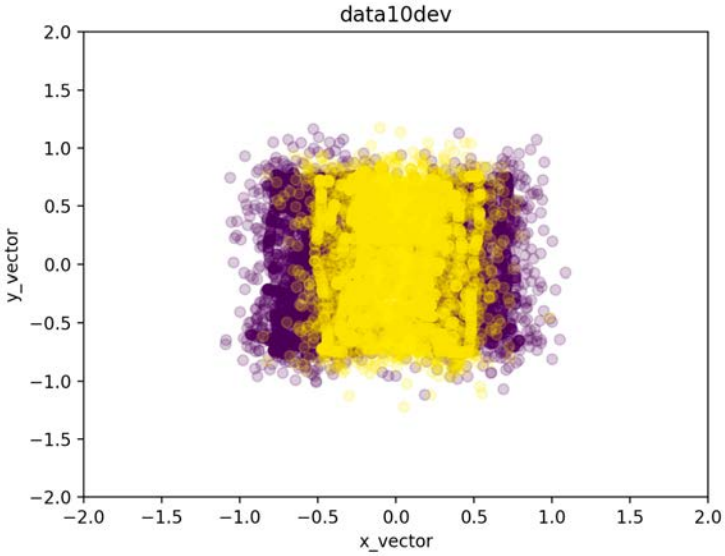

data8eval
y_vector
x_vector
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

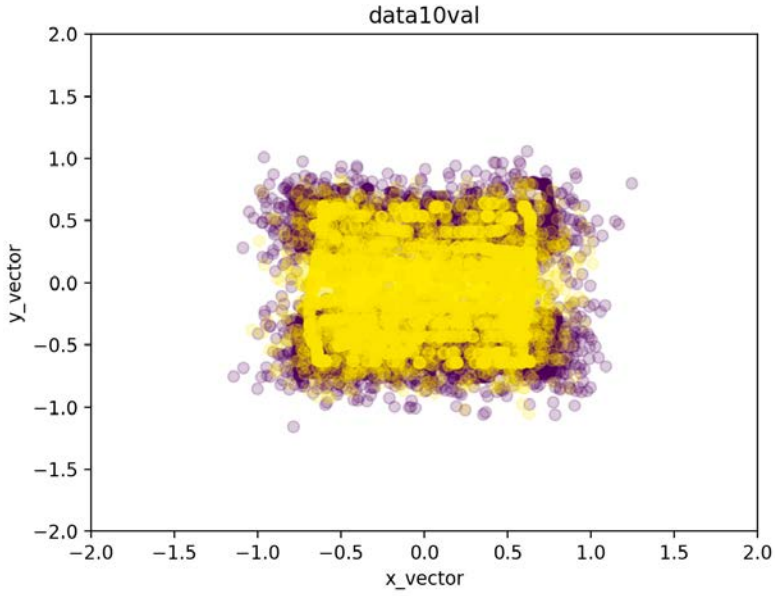

data9train
y_vector
x_vector
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


data9dev
y_vector
x_vector
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


data9eval
y_vector
x_vector
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


data10train
y_vector
x_vector


data10dev
y_vector
x_vector


data10val
y_vector
x_vector

B. TASK 2

The goal for Task 2 was to complete the table below. For ease of reading, I have altered the values that I have calculated myself to be highlighted. Performing these calculations was relatively easy as we have performed a Quadratic Discriminant Analysis (QDA) in a past assignment and Linear Discriminant Analysis (LDA) is quite easy to implement. Conducting the Class-Independent Principle Component Analysis (PCA), however, took a bit more time to complete and implement successfully.

DS	System	Training Data	Train	Dev	Eval
#08	KNN	/train	23.48	26.62	64.18
	RNF	/train	29.23	29.45	59.77
	CI-PCA	/train	36.07	36.85	31.77
	QDA	/train	49.92	49.78	46.33
	LDA	/train	36.10	36.08	41.34
	CI-PCA	/train + /dev	36.07	35.98	33.01
	QDA	/train + /dev	49.93	49.92	46.33
	LDA	/train + /dev	36.10	36.08	41.34
#09	KNN	/train	2.11	3.81	16.63
	RNF	/train	2.06	3.82	18.32
	CI-PCA	/train	96.34	27.98	97.47
	QDA	/train	96.87	97.13	93.04
	LDA	/train	96.20	91.44	98.03
	CI-PCA	/train + /dev	94.75	91.82	04.13
	QDA	/train + /dev	97.15	96.05	96.68
	LDA	/train + /dev	96.20	91.44	98.03
#10	KNN	/train	7.63	38.83	33.44
	RNF	/train	2.15	39.74	33.28
	CI-PCA	/train	53.01	52.70	48.92
	QDA	/train	83.92	55.24	65.89
	LDA	/train	53.01	55.34	48.93
	CI-PCA	/train + /dev	52.82	50.74	50.92
	QDA	/train + /dev	85.03	82.47	65.30
	LDA	/train + /dev	53.01	55.34	48.93

In order to complete this task, code was recycled for each dataset. The only thing that would change was the value of the variables or the dataset that was read in. For the data that says /train + /dev, both dev and train files were concatenated together, and the model was fit based off of the new aggregated dataset. Code has been commented out and included on the next few pages beginning with class-independent principal component analysis, quadratic discriminant analysis, and linear discriminant analysis.

To begin, I changed the current working directory and use the glob module to read in all data files that have the same number in it. For this set, I read in all data files that are from dataset 8 and added them to a dictionary.

```
os.chdir(r'/Users/gavinkoma/Desktop/pattern_rec/homework4/data')
print(os.getcwd())

files = glob.glob('*8*.csv') #make a list of all the csv files

d_all = {} #init dict
for file in files: #loop through .csv names
    #make a dict for all of them and add them to use later
    #there are no headers for any of this stuff, dont forget
    d_all["{0}".format(file)] = pd.read_csv(file,header=None)
```

I then changed the dictionary items into separate pandas dataframes and included column names to allow myself to better understand which variables I am working with at any given point.

```
df_train = pd.DataFrame(d_all['data8train.csv'])
df_train.columns = ['labels','vec1','vec2']

df_eval = pd.DataFrame(d_all['data8eval.csv'])
df_eval.columns = ['labels','vec1','vec2']

x_train = df_train[:][['vec1','vec2']]
y_train = df_train[:]['labels']

x_test = df_eval[:][['vec1','vec2']]
y_test = df_eval[:]['labels']

df_dev = pd.DataFrame(d_all['data8dev.csv'])
df_dev.columns = ['labels','vec1','vec2']
x_dev = df_dev[:][['vec1','vec2']]
y_dev = df_dev[:]['labels']
```

If the assignment requested that I utilize both /train *and* /dev for the training set, then another step was included that permitted me to concatenate both the dev and train dataframes together. When needed, the variables x_dev_train and y_dev_train were supplied to the fit of the model as the overall training data.

```
df_train = pd.DataFrame(d_all['data9train.csv'])
df_train.columns = ['labels','vec1','vec2']
df_dev = pd.DataFrame(d_all['data9dev.csv'])
df_dev.columns = ['labels','vec1','vec2']
df_dev_train = [df_train,df_dev]
df_dev_train = pd.concat(df_dev_train)
x_dev_train = df_dev_train[['vec1','vec2']]
y_dev_train = df_dev_train['labels']
```

These three steps were repeated for every dataset and for every type of analysis we perform. This code was not altered in anyway except for the numeric value of the dataset which ensured that I was always working with the proper data. I had initially tried to work through this code in a more dynamic manner utilizing for loops and keeping all values in the dictionary, but I quickly lost track of which variables I was using and what they represented and found myself making things more confusing than need be. Although this may be considered “the long way” to complete this assignment, it is what worked best for me.

Class-Independent Principal Component Analysis (CI-PCA)

The class independent principal component analysis was incredibly easy to implement. To begin, we needed to call on the StandardScalar() function and then pre-process our data such it would fit the Standard scale. From here, we apply the PCA function for analysis and fit the logistic regression to whatever training set we are using. To conclude, we predict the test result.

```
sc=StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

pca = PCA(n_components=2)

x_train = pca.fit_transform(x_train)
x_test = pca.transform(x_test)

explained_variance = pca.explained_variance_ratio_

classifier = LogisticRegression(random_state=0)
classifier.fit(x_train,y_train)

y_pred_train = classifier.predict(x_train)
print("train accuracy:\n",metrics.accuracy_score(y_train,y_pred_train))

y_pred_dev = classifier.predict(x_dev)
print("dev accuracy:\n",metrics.accuracy_score(y_dev,y_pred_dev))

y_pred_test = classifier.predict(x_test)
print("test accuracy:\n",metrics.accuracy_score(y_test,y_pred_test))
```

Quadratic Discriminant Analysis (QDA)

Performing a QDA on the given data required very few steps. We imported the function and then fit the model based on the training data. We then supply the model with any data that we want to predict the labels for.

```
# run qda
qda = QuadraticDiscriminantAnalysis()
model = qda.fit(x_train,y_train)
print(model.priors_)
print(model.means_)

pred_train = model.predict(x_train)
print("score for the training data:\n", metrics.accuracy_score(y_train, pred_train))
#look at prediction
pred = model.predict(x_test)
print(np.unique(pred,return_counts=True))
print(confusion_matrix(pred,y_test))
print("the score for the test data:",classification_report(y_test, pred, digits=4))
```

Linear Discriminant Analysis (LDA)

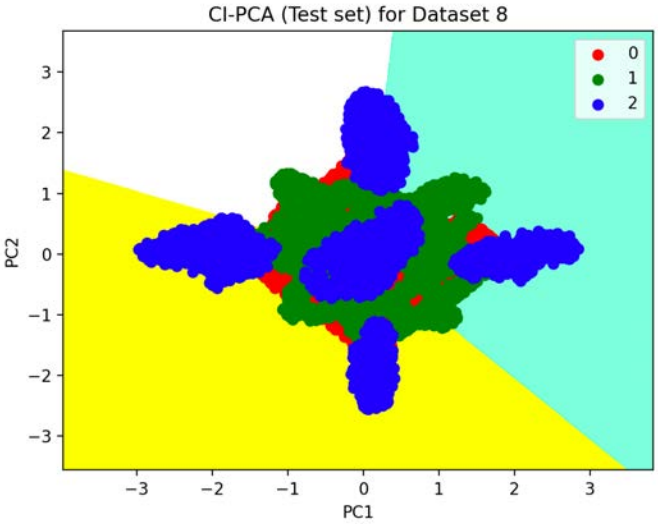
An LDA analysis was equally as easy to implement as the previous two analyses. Similar to QDA, we call on the LDA function and fit the model based on whatever training data that we are using. From here, we have to implement a method of cross-validation to evaluate the efficiency of our model. The most commonly implemented method is the repeated stratified k-fold cross validation. This method repeats the cross-validation procedure some number of times and will report the mean result across all folds and all runs. For my LDA, I implemented 10 folds and 3 repeats with a specified random_state of 1.

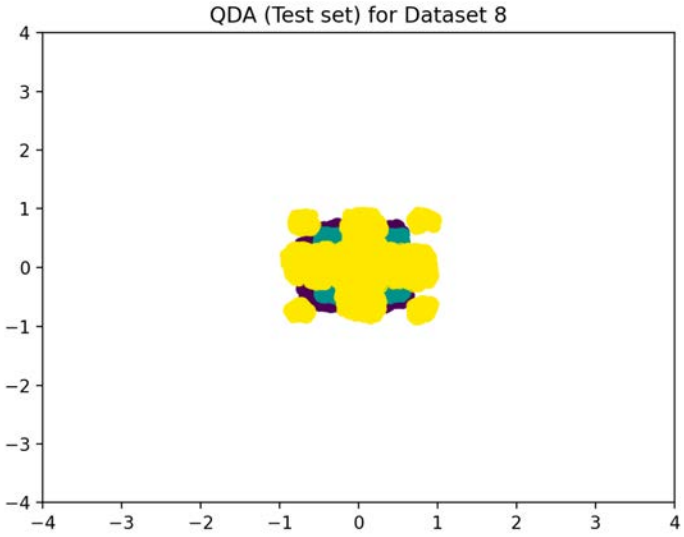
```
#itll be important to not forget that LDA only has one line as a boundary
#and its gotta be linear!

model = LinearDiscriminantAnalysis()
model.fit(x_train,y_train)

#use model and make prediction
#we need to eval also

df_dev = pd.DataFrame(d_all['data8dev.csv'])
df_dev.columns = ['labels','vec1','vec2']
x_dev = df_dev[:][['vec1','vec2']]
y_dev = df_dev[:]['labels']

cv = RepeatedStratifiedKFold(n_splits=10,n_repeats=3,random_state=1)


scores_train = cross_val_score(model, x_train,y_train,scoring='accuracy',
                        cv=cv,n_jobs=1)

scores_dev=cross_val_score(model, x_dev,y_dev,scoring='accuracy',
                        cv=cv,n_jobs=1)

scores_eval=cross_val_score(model, x_test,y_test,scoring='accuracy',
                        cv=cv,n_jobs=1)

print("train score: ",np.mean(scores_train))
print("dev score: ",np.mean(scores_dev))
print("test score: ",np.mean(scores_eval))
```

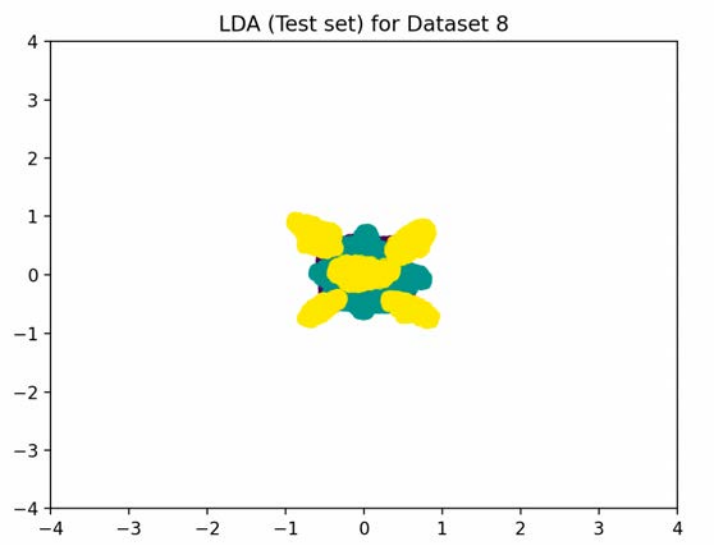
All values that were output from this portion of the assignment were included in the table on page 5 of this report.

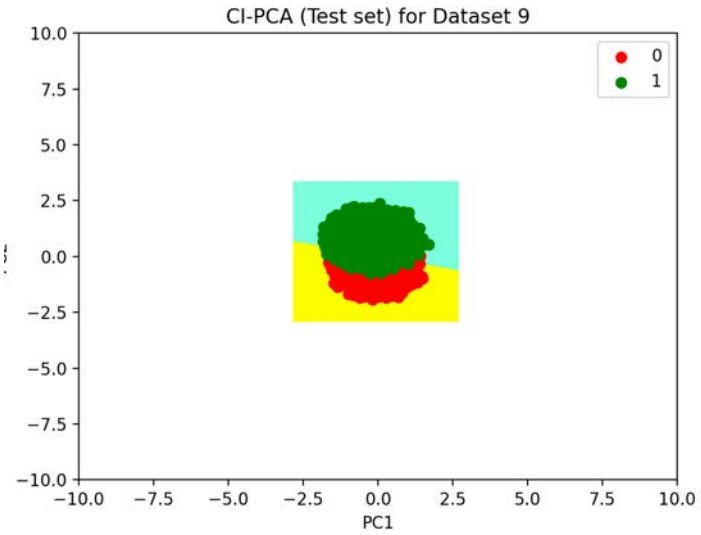
C. TASK 3

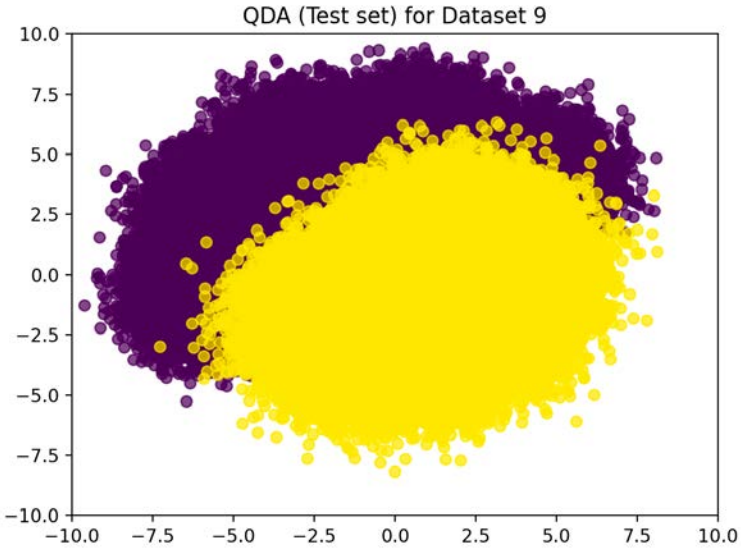
The final task of this assignment was to include images of the decision boundaries that were calculated by each of the three analyses that were implemented above. I have elected to include only the same datasets onto each page facilitate the comparison of these graphs. Therefore, CI-PCA, QDA, and LDA for dataset 8 will be on the next page; then dataset 9 will follow and the comparison will be concluded with dataset 10.

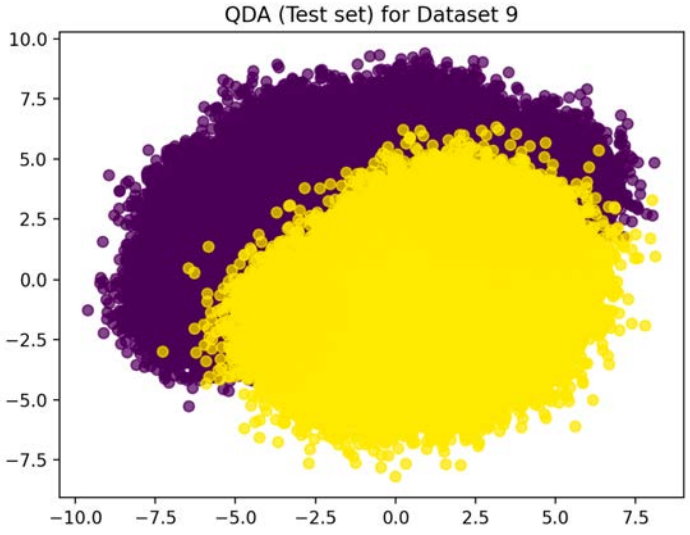

CI-PCA (Test set) for Dataset 8
0
1
2
PC1
PC2

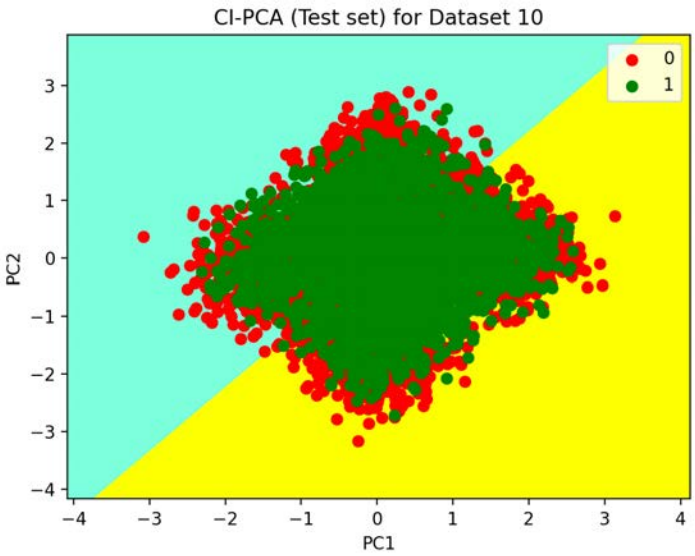

QDA (Test set) for Dataset 8

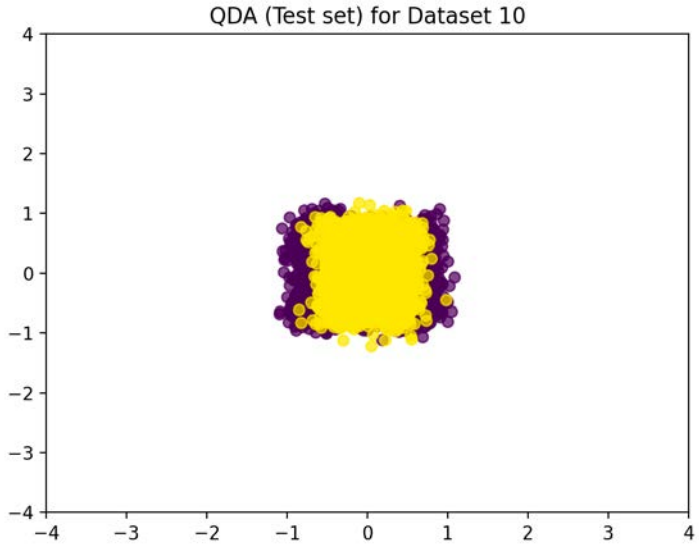

LDA (Test set) for Dataset 8

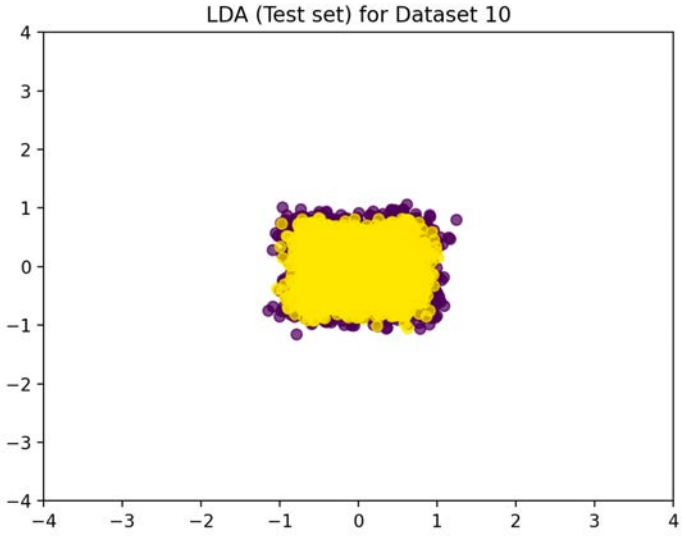

CI-PCA (Test set) for Dataset 9
0
1
PC1


QDA (Test set) for Dataset 9


QDA (Test set) for Dataset 9


CI-PCA (Test set) for Dataset 10
0
1
PC1
PC2


QDA (Test set) for Dataset 10


LDA (Test set) for Dataset 10

D. CONCLUSION

Unfortunately, I was only able to output the decision boundary for the class-independent principal component analysis. After many hours of trying and scouring different forums, I couldn't not figure out how to plot the decision boundaries for QDA nor LDA but I was able to obtain their accuracy scores and included them in the table found on Page 5.

The pooling of /train and /dev datasets occasionally helped with an increased accuracy score but also in other cases slightly hindered the accuracy score. When looking at the outputted results, we see that pooling the data for QDA with dataset 9, the accuracy score decreased from 97.13% (unpooled) to 91.82% (pooled). We also see that for dataset 10, the accuracy score *increased* from 52.24% (unpooled) to 82.47% (pooled) when performing a QDA. The general rule of thumb is that the more data for training, the better. However, it may be possible that if the data that is being pooled together varies too much, then the accuracy score will be affected poorly and *vice versa*.

The assumptions of the varying algorithms implemented are as follows:

Class-Independent PCA

1. No unique variance
2. Total variance is equal to common variance

Quadratic Discriminant Analysis

1. All observations are drawn from a normal distribution
2. Each class has its own covariance matrix

Linear Discriminant Analysis

1. All samples are drawn from a normal distribution
2. There is only one common covariance matrix

We know that if we are working with data that is of a higher dimension (consider dataset 8 with three classes), then we will have to estimate more parameters. If we have more parameters then we may also have an increasingly high variance which would result in poor results given a QDA analysis. LDA, however, allows us to estimate fewer parameters but is less malleable in terms of prediction accuracy. CI-PCA will allow us to work with a higher number of classes and create a set of data that is a linear combination of the original data.

Based on the data values obtained for the above table, it seems that there was no one algorithm that did notably better than any of the others. Each algorithm performed very well in some cases and poorly in other cases. I think that this furthers the point made by Dr. Picone where the choice of your algorithm is random and may perform well or may perform poorly. This degree of randomness makes it difficult to compare algorithms at a base level.