CSL2050 PATTERN RECOGNITION AND MACHINE LEARNING

LAB REPORT-04

NAME:	SAMEER SHARMA
ROLL NUMBER:	B21CS066
LAB TITLE:	GAUSSIAN BAYES CLASSIFIER

Q1)

Part 1) and Part 2)

Basic preprocessing on the iris dataset is done in this part. I have implemented the Gaussian Bayes classifier class from scratch. In the init function we are taking the type of the classifier as parameter to construct the classifier which we use at later time while training the dataset in choosing the covariance matrix.

Clf_type	Remarks
1	$\sum_i = \sigma^2 I$ (*I* stands for identity matrix)
2	$\sum_i = \sum$ covariance for all classes are identical but arbitrary
3	$\sum_i$ = actual covariance

Train function

It takes two arguments- X_train,y_train. It stores the class means, class probabilities by using the helper functions coded earlier in the class. Then, on the basis of value of clf_type, covariance matrix is calculated as follows:

1. Calculated variance of only feature of the dataset and multiplied it by identity matrix of compatible size $n \times n$ where n is number of features.
2. Calculated all $n \times n$ covariances for a single class and stored it into class_cov.
3. Calculated the covariance matrix for each class separately and stored them into a three dimensional matrix.

Predict function

It takes a single sample feature of $n \times 1$ size. Then, it uses prior probability of this feature given each class assuming the data to be normally distributed by using the below formula

$$N(x;\mu,\sigma^2) = \frac{1}{(2\pi)^{d/2}\left|\Sigma\right|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^t \Sigma^{-1}(x-\mu)\right]$$

Then posterior probability of each class given the feature is calculated using the below formula

$$posterior = \frac{likelihood \times prior}{evidence}$$

Remember we have calculated the prior probabilities, class mean and covariance matrices for classes previously while training the classifier.

Test function

This function takes a two dimensional X_test array on which we have to predict the class and y_test its corresponding class labels so that we can calculate accuracy for the classifier. This function calls predict function multiple times for each sample in X_test in stores in pred array. Then, using the pred and y_test arrays, we can calculate accuracy for the model. In the last, function returns the pred array and accuracy of the model on X_test.

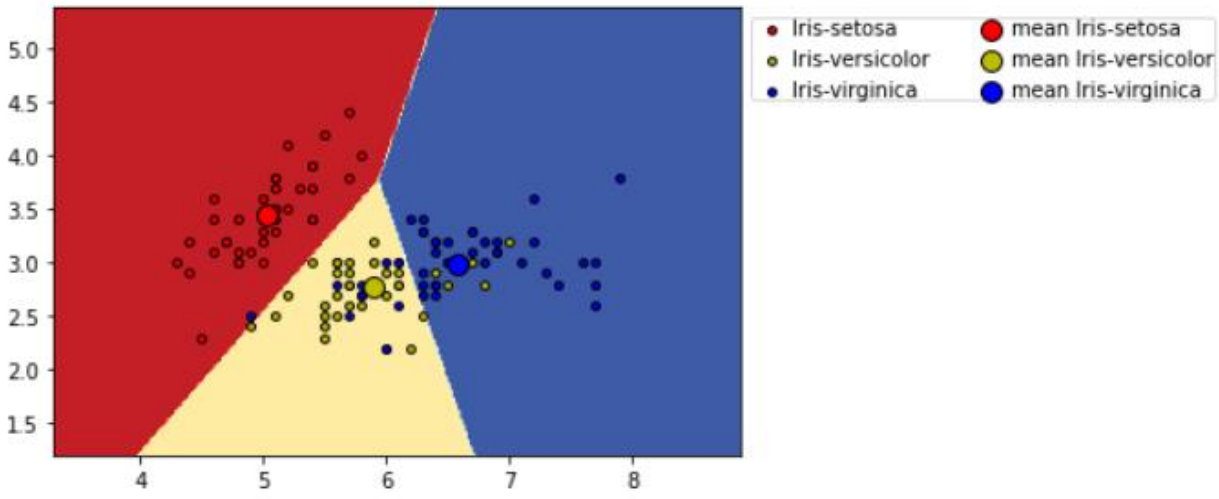
Plot decision boundary function

It takes X and y on which we have to plot the decision boundary. Remember X should have only two features. It trains a new classifier on X and y. Then, it creates a meshgrid of 2D datapoints in which x value ranges from min(X) to max(X) and y value ranges from min(y) to max(y). Then, using the test function we are predicting the class on these meshgird datapoints and stored the predictions in Z. Then, using the contourf function we plotted the decision boundary for the classifier. We also scattered the datapoints in X on the same plot.

Part 3)

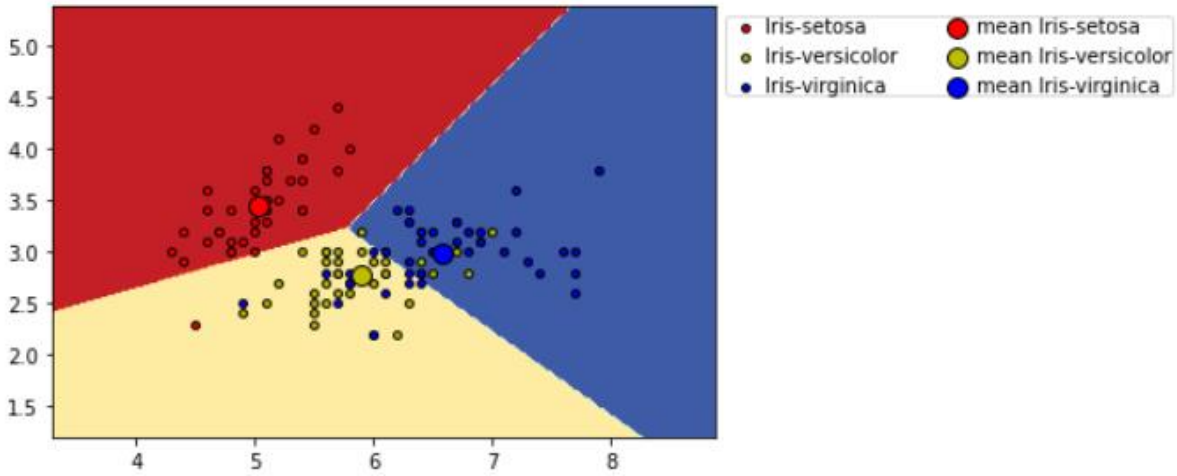
We plotted the decision boundary for the iris dataset by considering only sepal length and sepal width features for all the three variants as follows.

1. Clf_type=1



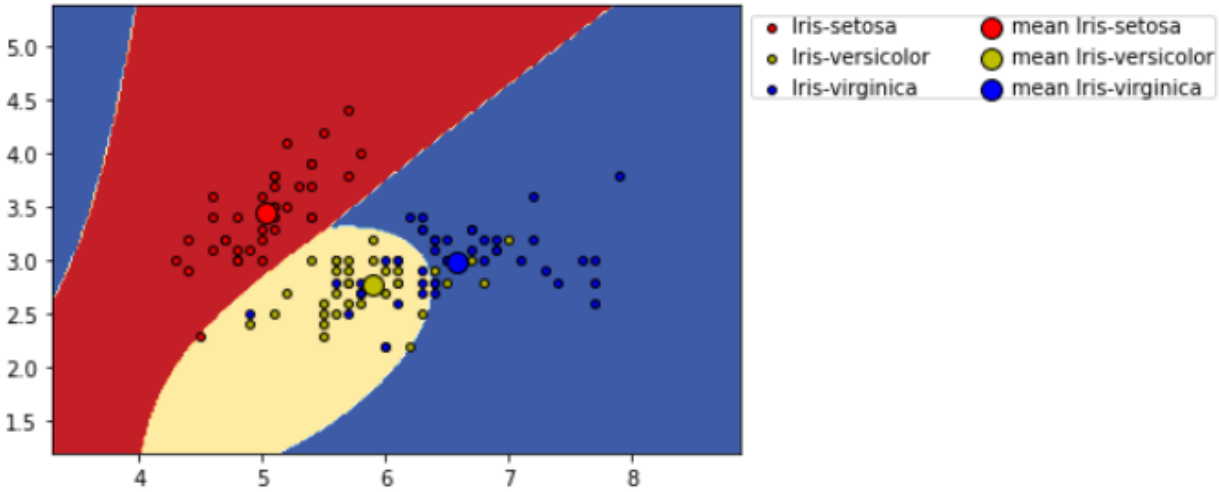
Here we can see that the decision boundary between two classes is a straight line passing from between the means of that two respective classes. Also, we can see that the decision boundary is nearly perpendicular to the line joining the mean points of that two respective classes.

2. Clf_type=2



Here we can see that the decision boundary between two classes is a straight line passing from between the means of that two respective classes. However, we can see that the decision boundary is not perpendicular to the line joining the mean points of that two respective classes.

3. Clf_type=3



Here we are getting non linear decision boundary for the classifier.

Accuracies for the three models are as follows:

```
Accuracy for first model 0.9666666666666667
Accuracy for second model 0.8
Accuracy for third model 1.0
```

Clearly third model performed the best while the second model performed the worst.

Part 4)

Made the cross validation function to perform cross validation on the three models. In the function we are splitting the data into k splits. Then, we are picking each split one by one and training the classifier on all the splits excluding the selected one (k-1 splits) and testing the classifier on selected split and calculated the accuracy for each model.

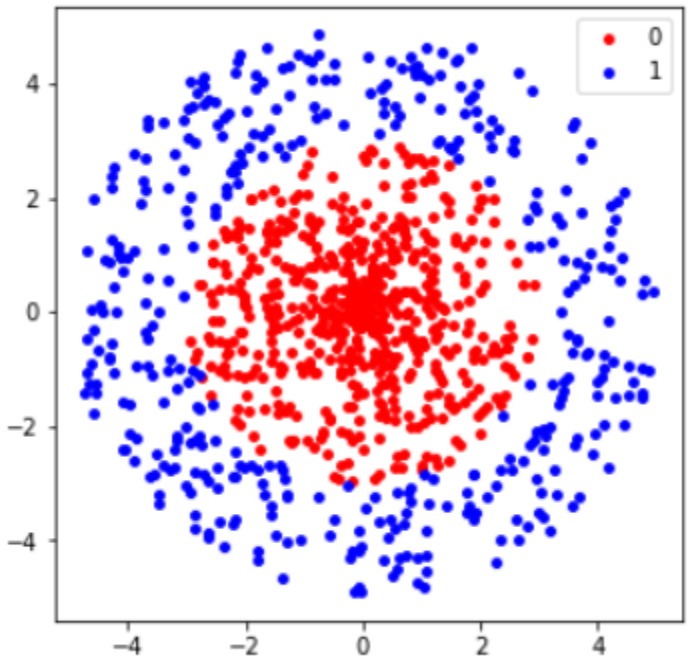
Accuracies of the models are as follows:

```
score array for  0 is [0.8666666666666667, 0.9333333333333333, 0.9333333333333333, 1.0, 0.9]
mean score 0.9266666666666667
variance 0.0019555555555555546
score array for  1 is [0.7, 0.9333333333333333, 0.9, 0.8333333333333334, 0.7333333333333333]
mean score 0.82
variance 0.008266666666666672
score array for  2 is [0.9666666666666667, 1.0, 1.0, 0.9333333333333333, 0.9333333333333333]
mean score 0.9666666666666668
variance 0.0008888888888888885
```

Clearly the third model is here more generalizable since it has much less variance compared to other ones. It is also intuitive for it to be the most generalizable since it considers separate covariance matrix for each class. The least generalizable model here is the second model where we are considering identical arbitrary covariance matrices for each and every class.

Part 5)

In this part, the synthetic data is generated by using np.random.uniform function and then assigned the class to them by taking their distance from origin into account.

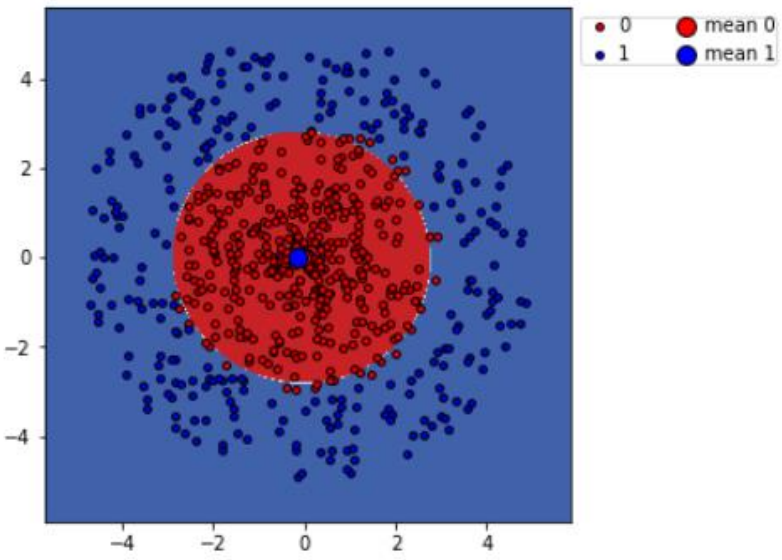


As one can see that the two classes are not linearly seperable.

Now, we trained third variant of the Gaussian classifier on this dataset and tested its accuracy

```
accuracy of classifier is 0.95
```

And the decision boundary is as follows

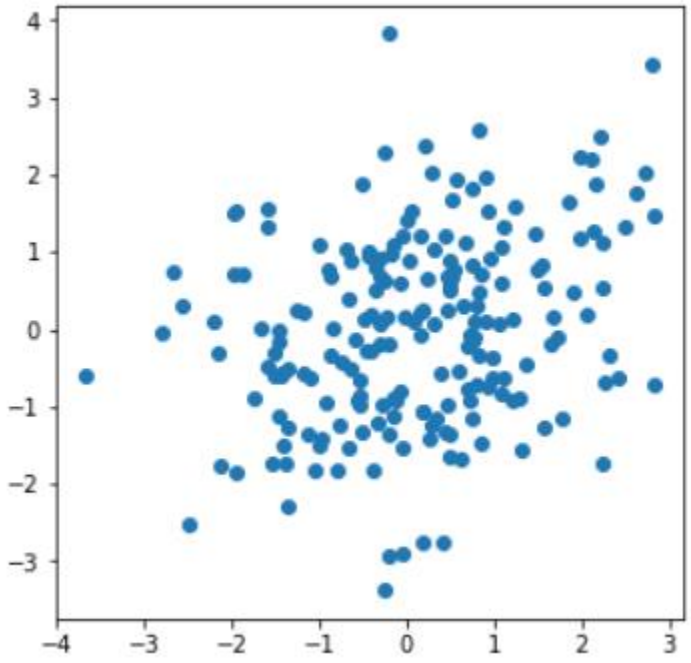


Clearly one can see that the model is nicely fitting on the datapoints and the decision boundary is nicely aligning with the actual boundary between the datapoints of two classes as we can see in the previous plot.

Q2)

Part 1)

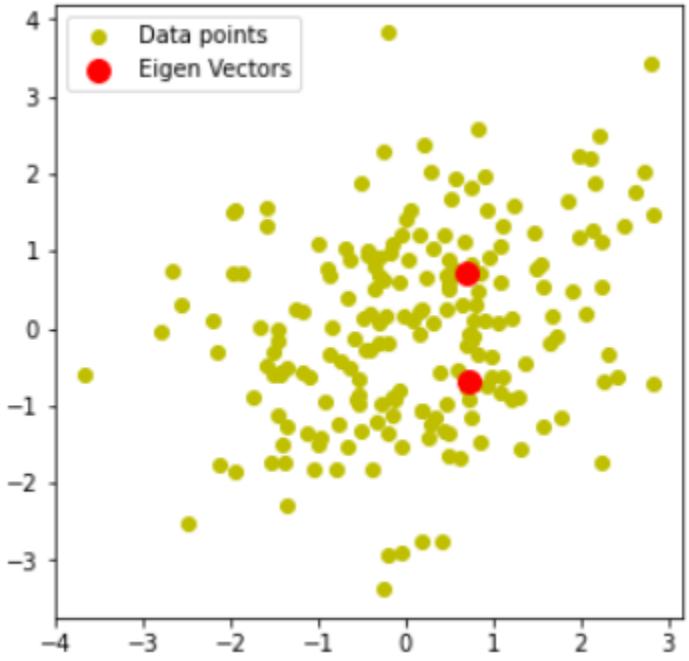
First generated the synthetic dataset by using the np.random.multivariate_normal function and the mean and covariance matrix provided in the function



Then made a function to calculate covariance between two column vectors X and Y by using the formula

$$\frac{\sum(x_i - \bar{X})(y_i - \bar{Y})}{N}$$

Then calculated the covariance matrix of the dataset by calculating covariance for each pair of columns. Calculated covariance matrix is nearly identical to the real one. In the last, eigen values and vectors for this matrix is calculated using inbuilt functions and plotted them along with the datapoints.



Part 2)

Transformed the dataset by the following operation

$$Y = \Sigma_s^{-1/2} X$$

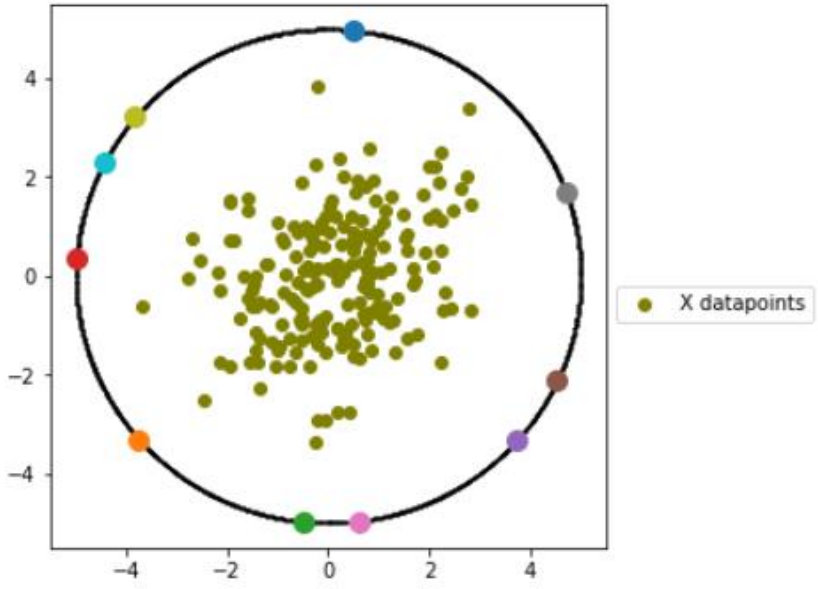
And then calculated the covariance matrix for it which is nearly equal to identity

```
array([[1.00000000e+00, 2.76718905e-16],
       [2.76718905e-16, 1.00000000e+00]])
```

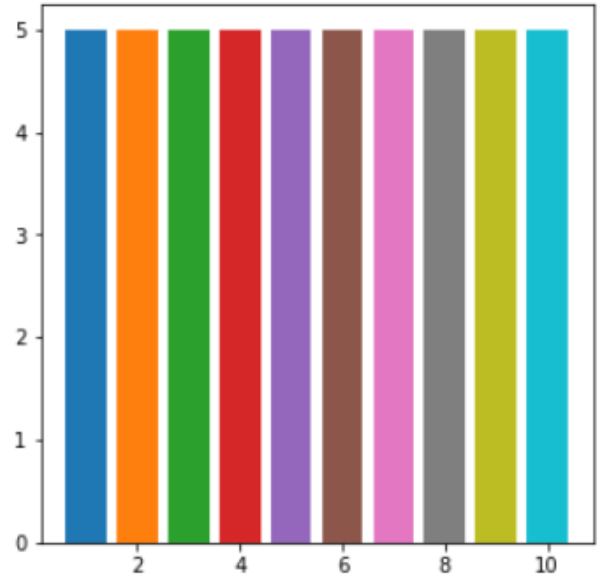
The purpose of this transformation is to make the two variables ('X1' and 'X2') independent (since their covariance is nearly zero)

Part 3)

Generated the random points on the circle $x^2 + y^2 = 25$ using the np.random.uniform function



Distance of each datapoint from $\mu = [0,0]$ are as follows

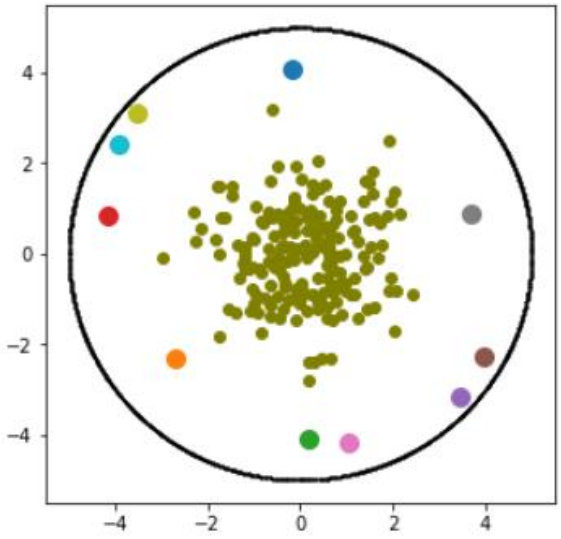


Clearly the Euclidian distance of each point from μ is nearly equal to 5

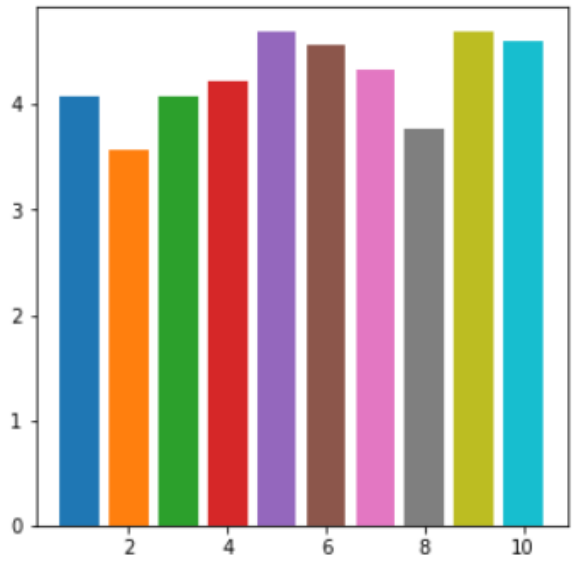
Part 4)

Now applied the same transformation to these ten datapoints that we did in the part 2. Its plot with datapoints of y is as follows

$$Q = \Sigma_s^{-1/2} P$$



And Euclidian distances are as follows



Clearly, each Euclidian distance after the transformation is less than that of before the transformation.