Project 1

CMSC 828C

Bayesian and KNN Classifier with PCA/LDA Reduction

Sneha Nayak (UID: 115586284)

November 11, 2019

Introduction

This project involves the comparison between the optimal Bayesian classifier and K Nearest Neighbors Classifier.

The data set that we have used for this project is the Fashion MNIST dataset.

We first deduce the Bayesian classifier's accuracy in the following steps:

1. without any dimensionality reduction
2. then subsequently find its accuracy with Principal Component analysis
3. followed by Linear Discriminant Analysis
4. We compare these accuracies and study them.

This is done in **Chapter 2**.

Then we repeat the above mentioned four steps for K nearest neighbors in **Chapter 3**. Here we play around with different values of K to get different accuracies and compare them.

The different experiments carried out are mentioned in **Chapter 4**.

Then we go on to study and compare the accuracies achieved between Bayes and Knn classifier in **Chapter 5**.

Chapter 1

Dataset

The dataset we make use of here is the Fashion MNIST dataset. This set is divided into training and test data as below:

```
Xtrain_m, Ytrain = load_mnist('data/fashion', kind='train')
Xtest_m, Ytest = load_mnist('data/fashion', kind='t10k')
```

The training set comprises of 60,000 grayscale images belonging to 10 classes, whereas the test set comprises of 10,00 gray scaled images. These images are of size 28x28, which gives us a total of 784 pixel. Each of these 784 pixels are considered to be a feature. Each pixel can take on integer values in the range [0, 255]. Xtrain_m now is a matrix which has 60,000 rows and 784 columns, whereas Xtest_m is a matrix of 10,000 rows and 784 columns. Ytrain has 60,000 rows which consists of labels, labelled between 0 to 9, with each row element representing the labels of each of the 60,000 observations in Xtrain_m. Ytest works the same way, but with respect to Xtest_m.

We zero center this data, since this proves to give us better results when we perform PCA and LDA.

```
Xtrain_m = Xtrain_m - np.mean(Xtrain_m)
Xtest_m = Xtest_m - np.mean(Xtest_m)
```

Chapter 2

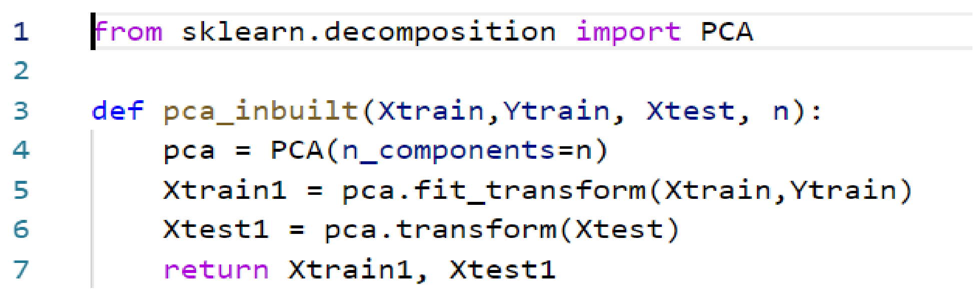
Dimensionality Reduction

Principal Component Analysis

The code for the implementation of PCA is in pca.py.

Here we use scikit.learn inbuilt library to implement PCA. After testing multiple times, the ideal number of principal components was chosen to be 85, since this gives the most accuracy in case of both Bayesian as well as Nearest Neighbor Classifier.

The implementation as below.



```python
from sklearn.decomposition import PCA

def pca_inbuilt(Xtrain,Ytrain, Xtest, n):
    pca = PCA(n_components=n)
    Xtrain1 = pca.fit_transform(Xtrain,Ytrain)
    Xtest1 = pca.transform(Xtest)
    return Xtrain1, Xtest1
```

Linear Discriminant Analysis

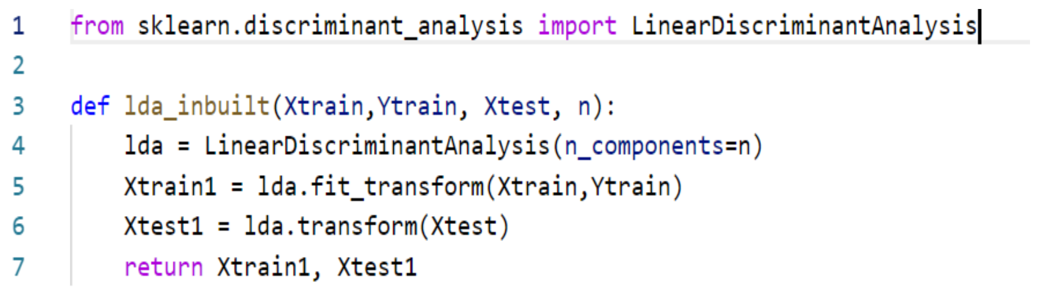
The code for the implementation of LDA is in lda.py.

Here we use scikit.learn inbuilt library to implement LDA

It is seen that for values of dimensions less than 9, the accuracy begins to detriment in both the cases.

Values above 9 tend to give errors with the inbuilt LDA library since the value of dimension should be min (features, classes-1).

The implementation as below.



```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

def lda_inbuilt(Xtrain,Ytrain, Xtest, n):
    lda = LinearDiscriminantAnalysis(n_components=n)
    Xtrain1 = lda.fit_transform(Xtrain,Ytrain)
    Xtest1 = lda.transform(Xtest)
    return Xtrain1, Xtest1
```

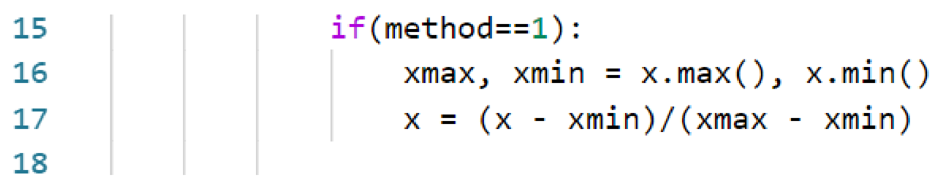
Chapter 3

Bayes Classifier

For Bayes classifier we need to find the Maximum Likelihood estimations of the mean and covariances.

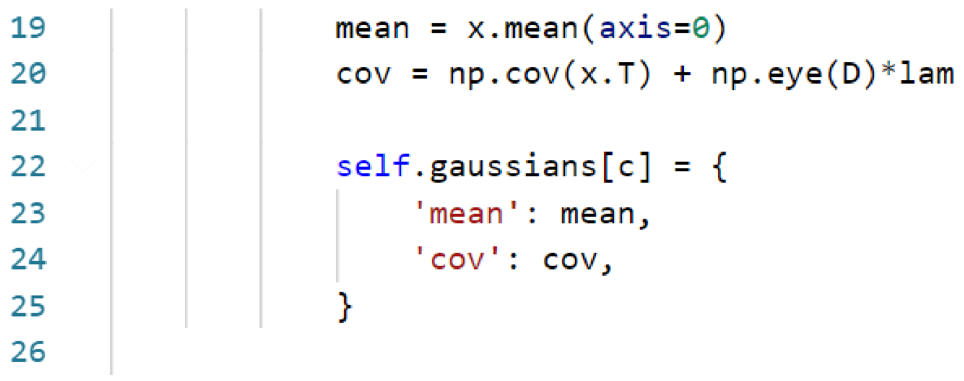
To get better results for the Bayesian classifier, we first normalize the data that belong to the same class.

Below is a code snipped of it.



```
if(method==1):
    xmax, xmin = x.max(), x.min()
    x = (x - xmin)/(xmax - xmin)

```

Since we assume a gaussian distribution of the data, it is found that the MLE of mean is just the sample mean and the MLE of covariance is just the sample covariances. Hence, we find the sample mean and covariance with respect to data belonging to the same class. The respective mean and covariances are calculated. For the covariance, we add a scaling factor ‘lam’ and multiply with the identity matrix in order to avoid any singularities and therefore, to regularize the data. After testing with different scalar values of ‘lam’, it was tested that a value of 0.01989 gives the highest accuracy.

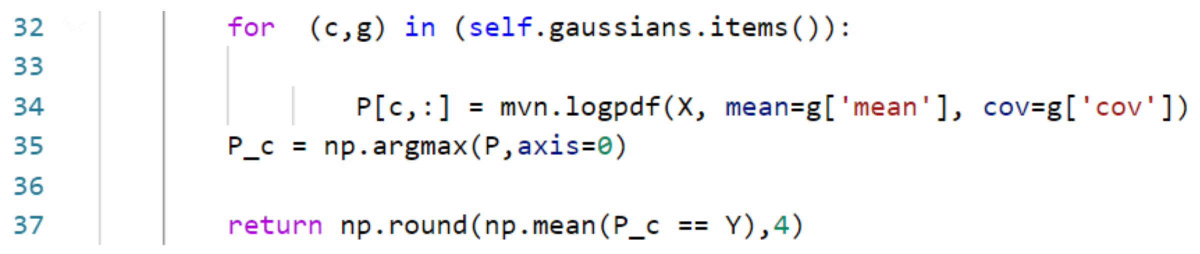


```
mean = x.mean(axis=0)
cov = np.cov(x.T) + np.eye(D)*lam

self.gaussians[c] = {
    'mean': mean,
    'cov': cov,
}

```

We then go on to predict the labels of the test data and subsequently find the accuracy. We do so by summing up the log likelihoods and prior probabilities. But since the prior probabilities of all the classes are the exact same (0.1), we don’t have to consider it. So, it just comes down to finding the largest log likelihood for each observation for each class.

$$\mathbf{P(\omega) = argmax\ (logP(x|\omega))}$$

We use mvn.logpdf function to compute log(P(x|ω))

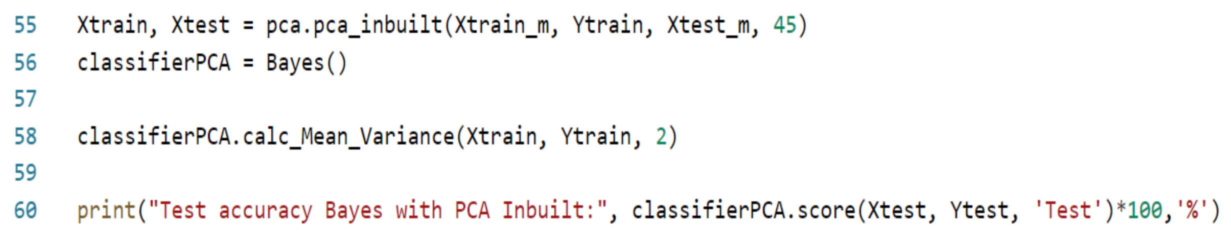


```
for  (c,g) in (self.gaussians.items()):

    P[c,:] = mvn.logpdf(X, mean=g['mean'], cov=g['cov'])
P_c = np.argmax(P,axis=0)

return np.round(np.mean(P_c == Y),4)
```

We then compute the accuracy and find it to be **84.05%**.

We then move on to carry out dimensionality reduction by deducing 45 Principal components and then projecting the data onto these 45 principal components. We find that the accuracy is found out to be **79.45%**.

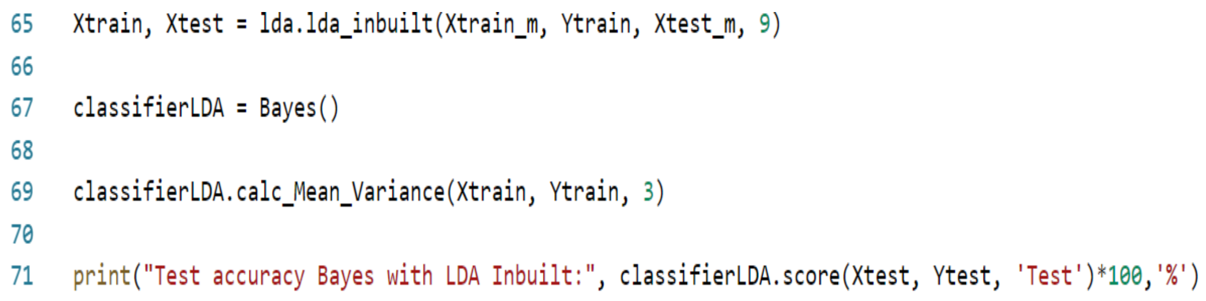
The code for this is as below:



```
Xtrain, Xtest = pca.pca_inbuilt(Xtrain_m, Ytrain, Xtest_m, 45)
classifierPCA = Bayes()

classifierPCA.calc_Mean_Variance(Xtrain, Ytrain, 2)

print("Test accuracy Bayes with PCA Inbuilt:", classifierPCA.score(Xtest, Ytest, 'Test')*100,'%')
```

We then carry out dimensionality reduction using LDA where again we project using 9 components. This gives us an accuracy of **82.11%**.

The code for this is as below:



```
Xtrain, Xtest = lda.lda_inbuilt(Xtrain_m, Ytrain, Xtest_m, 9)

classifierLDA = Bayes()

classifierLDA.calc_Mean_Variance(Xtrain, Ytrain, 3)

print("Test accuracy Bayes with LDA Inbuilt:", classifierLDA.score(Xtest, Ytest, 'Test')*100,'%')
```

Analysis

We see that normalizing the data about each class for the training data gives us better results in the case of no reduction. However, when we try to normalize the data in the case of PCA and LDA, it gives us accuracies of 32.19% and 73.78% respectively as opposed to 74.85% and 82.11% without any normalization.

Hence, we normalize the data only for the case when there is no LDA or PCA applied.

It can also be seen that centering the data gives us better results in the case of PCA and LDA. However, for the case without any dimensionality reduction, the accuracy when the data is centered about the mean is 29.85% as opposed to 84.05% without centering the data.

We also see here that the accuracy is the highest when there is no dimensionality reduction. This is because when we use dimensionality reduction (both PCA and LDA) we are bringing down the dimensions from 784.

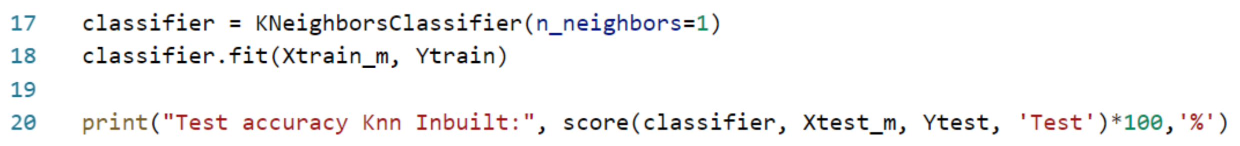
The code for this is found in main1.py

Chapter 4

K Nearest Neighbors

Inbuilt

Here for the nearest neighbor classifier, we use scikit-learn library's inbuilt function. We first specify the number of neighbors, which we take as 1. We then fit the training data to the model. We predict the labels on the test data and subsequently find the accuracy.

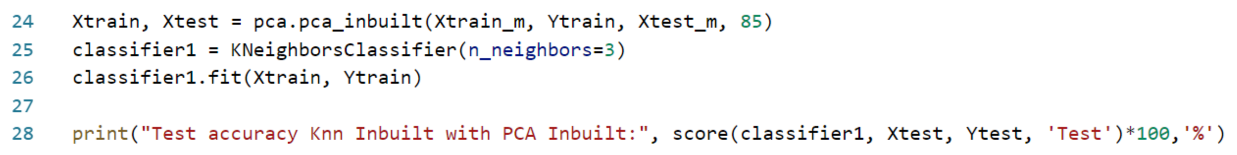


```python
classifier = KNeighborsClassifier(n_neighbors=1)
classifier.fit(Xtrain_m, Ytrain)

print("Test accuracy Knn Inbuilt:", score(classifier, Xtest_m, Ytest, 'Test')*100,'%')
```

Without any dimensionality reduction, the accuracy is **84.97%.**

We then move on to carry out dimensionality reduction by deducing 85 Principal components and then projecting the data onto these 85 principal components. We find that the accuracy is found out to be **84.92%.**

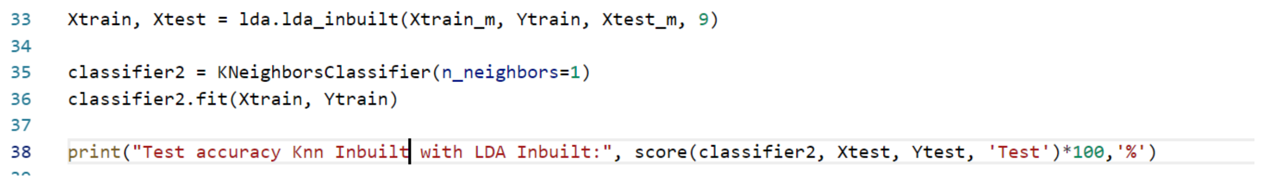
We use the code snippet as below:



```python
Xtrain, Xtest = pca.pca_inbuilt(Xtrain_m, Ytrain, Xtest_m, 85)
classifier1 = KNeighborsClassifier(n_neighbors=3)
classifier1.fit(Xtrain, Ytrain)

print("Test accuracy Knn Inbuilt with PCA Inbuilt:", score(classifier1, Xtest, Ytest, 'Test')*100,'%')
```

We then carry out dimensionality reduction using LDA where again we project using 9 components. This gives us an accuracy of **79.11%.**

We use the same snippet as below:

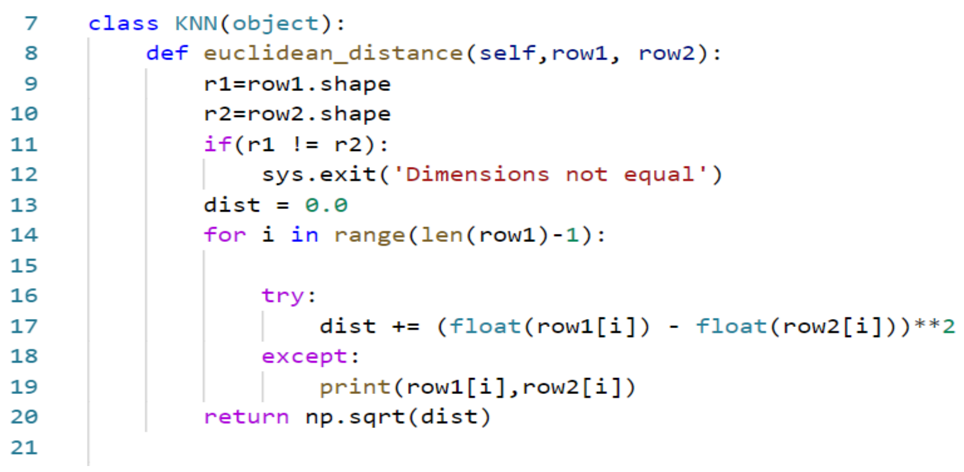


```python
Xtrain, Xtest = lda.lda_inbuilt(Xtrain_m, Ytrain, Xtest_m, 9)

classifier2 = KNeighborsClassifier(n_neighbors=1)
classifier2.fit(Xtrain, Ytrain)

print("Test accuracy Knn Inbuilt with LDA Inbuilt:", score(classifier2, Xtest, Ytest, 'Test')*100,'%')
```

The code for this is found in main2.py

Code from Scratch

Here the Euclidean distances between the training data and testing data are calculated.



```python
class KNN(object):
    def euclidean_distance(self,row1, row2):
        r1=row1.shape
        r2=row2.shape
        if(r1 != r2):
            sys.exit('Dimensions not equal')
        dist = 0.0
        for i in range(len(row1)-1):

            try:
                dist += (float(row1[i]) - float(row2[i]))**2
            except:
                print(row1[i],row2[i])
        return np.sqrt(dist)

```

The class training point with the least distance from the test point, is the class that the test point is classified to. For this 1 have taken k to be equal to 1. We have also taken the training data to be 5000 and test data to be 5000. Although the computation turns out to be expensive and time consuming as compared to the inbuilt function, the accuracy is found to be **80.33%.**

The same is repeated with PCA dimensionality reduction. This turned out to be computationally less expensive and less time consuming. The accuracy found here is **78.87%.**

The above is repeated with LDA dimensionality reduction. This turned out to be computationally less expensive and less time consuming as well. The accuracy found here is **75.95%.**

The code for this is found in main3.py

Analysis

We see that centering the data about each class for the training data doesn’t make much of a difference. Hence, we don’t do it in the case of KNN.

Chapter 5

Experiments

n: number of components
k: number of nearest neighbors

Classifier	Accuracy (in %)	
Bayes	84.05	
Bayes with PCA	n=85	79.45
	n=45	79.84
	n=9	74.85
Bayes with LDA	n=9	82.11
	n=8	80.589
	n=7	78.05
kNN Inbuilt	k=1	84.97
	k=3	85.41
	k=5	85.54
kNN Inbuilt With PCA, n=85	k=1	84.92
	k=3	85.59
	k=5	86.15
kNN Inbuilt With LDA, n=9	k=1	79.11
	k=3	81.4
	k=5	82.21
1NN (5000 test and train)	80.33	
1NN with PCA, n=85 (5000 test and train)	78.87	
1NN with LDA, n=9 (5000 test and train)	75.95	

Chapter 6

Conclusion

From the experiments it is found that the classifier with the highest accuracy is the nearest Neighbor classifier with PCA dimensionality reduction with 85 Principal components and k=5 neighbors with an accuracy of 86.15%. From the experiments it is also deduced that the poorest performing classifier is the 1 NN classifier with LDA reduction.

In the Bayesian classifier with PCA reduction, it can be seen that as the principal components reduce below 45, the accuracy starts to decrement as well.

The ideal value for the number of Principal Components is found to be 85 after rigorous testing.

It can also be noted that for the nearest Neighbor classifier, as k starts to take on values greater than 5, the accuracy starts to decrease as well. Hence, k=5 is found to be the optimal value for the number of nearest neighbors.