

LAB 9 : Dimensionality Reduction

1. Principal Component Analysis (PCA)
2. Linear Discriminant Analysis (LDA)

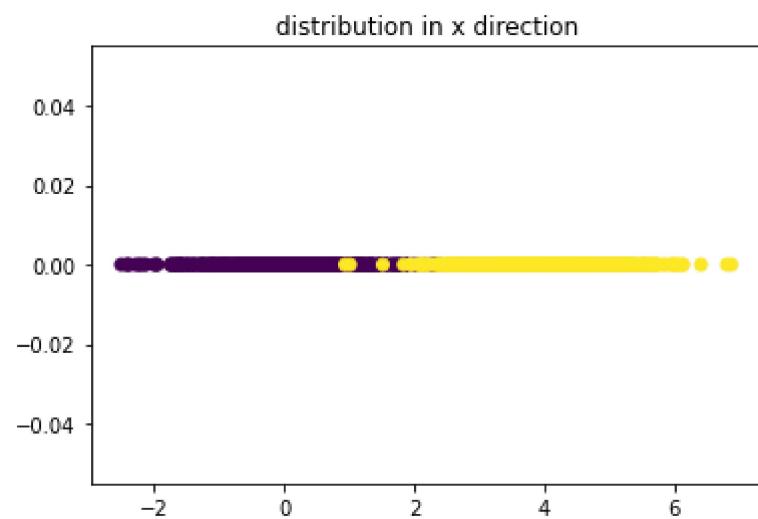
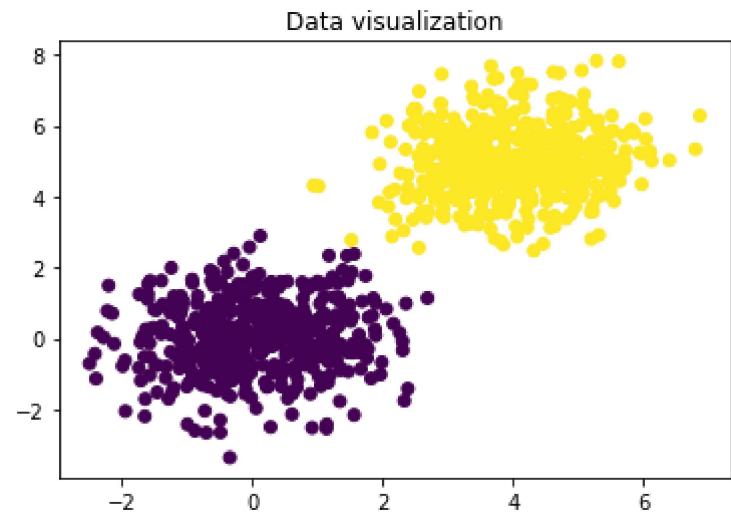
```
In [1]: import numpy as np  
import matplotlib.pyplot as plt
```

PCA

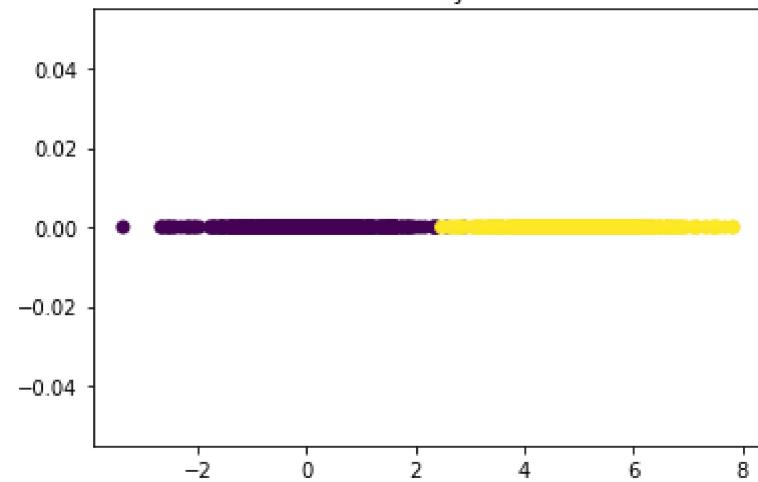
```
In [2]: mean1=np.array([0,0])
mean2=np.array([4,5])
var=np.array([[1,0.1],[0.1,1]])
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,500)
data2=np.random.multivariate_normal(mean2,var,500)
data=np.concatenate((data1,data2))
label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

plt.figure()
plt.scatter(data[:,0],data[:,1],c=label)
plt.title('Data visualization')
plt.figure()
plt.scatter(data[:,0],np.zeros(data.shape[0]),c=label)
plt.title('distribution in x direction')
plt.figure()
plt.scatter(data[:,1],np.zeros(data.shape[0]),c=label)
plt.title('distribution in y direction')
```

Out[2]: Text(0.5, 1.0, 'distribution in y direction')



distribution in y direction

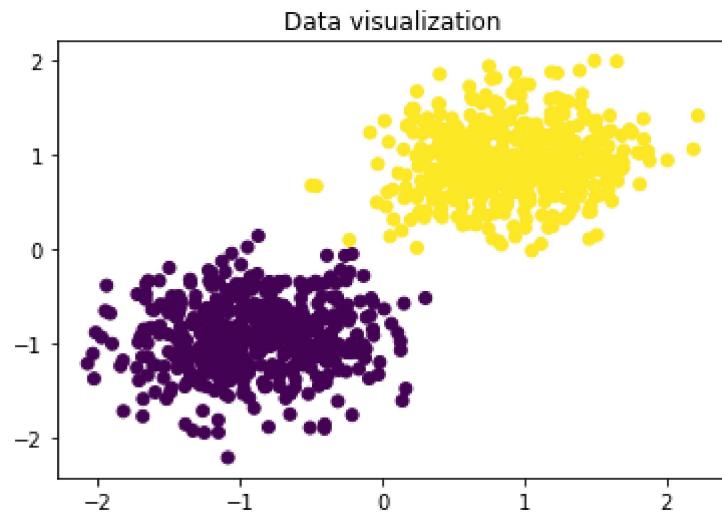


```
In [3]: # Data normalization
```

```
# Perform data normalization here using mean substraction and std division
data=(data-np.mean(data,0))/(np.std(data,0))

plt.figure()
plt.scatter(data[:,0],data[:,1],c=label)
plt.title('Data visualization')
```

```
Out[3]: Text(0.5, 1.0, 'Data visualization')
```



```
In [5]: print("This is the dimension of dataset : ", data.shape)
```

```
This is the dimension of dataset : (1000, 2)
```

In [6]: # PCA

```
# covariance matrix
cov=data.T @ data

# using singular value decomposition
u,s,v=np.linalg.svd(cov)

trans_data=data @ u ## Write your code here

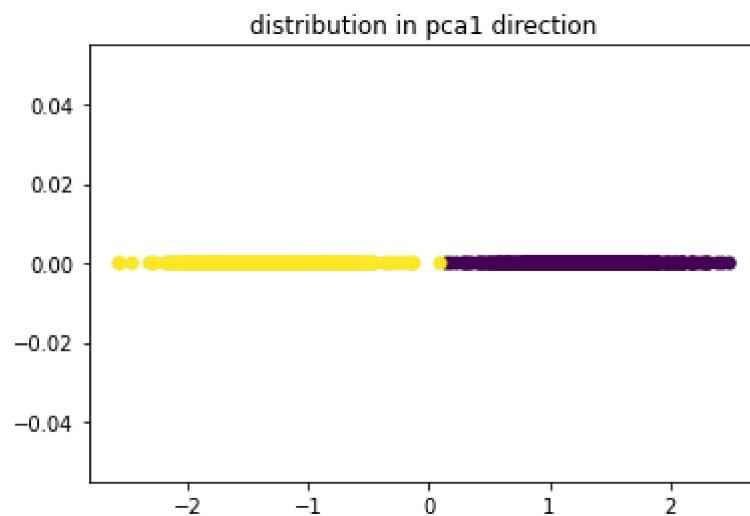
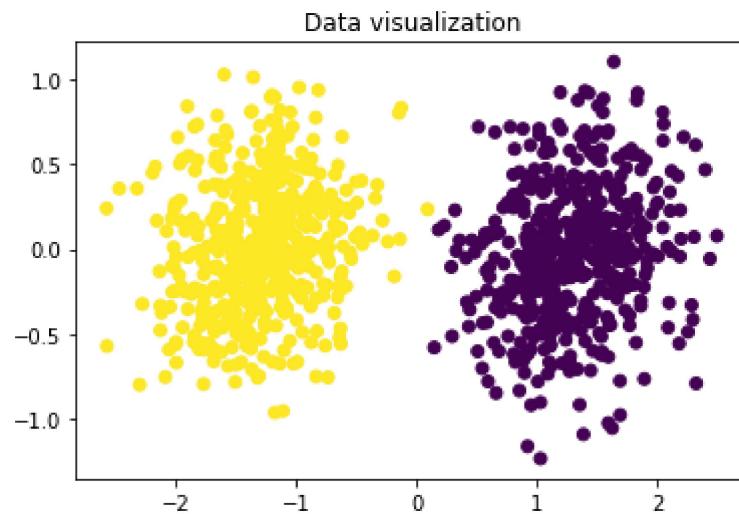
var_pca1=np.var(trans_data[:,0])
var_pca2=np.var(trans_data[:,1])

print('variance along pca1 direction=',var_pca1)
print('variance along pca2 direction=',var_pca2)

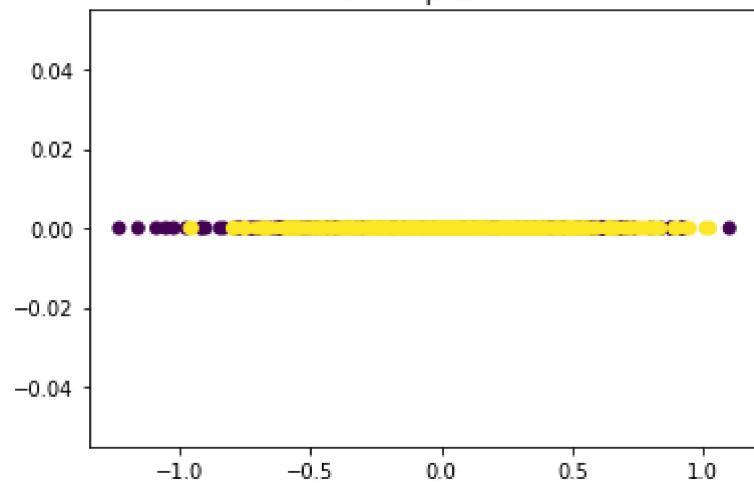
plt.figure()
plt.scatter(trans_data[:,0],trans_data[:,1],c=label)
plt.title('Data visualization')
plt.figure()
plt.scatter(trans_data[:,0],np.zeros(data.shape[0]),c=label)
plt.title('distribution in pca1 direction')
plt.figure()
plt.scatter(trans_data[:,1],np.zeros(data.shape[0]),c=label)
plt.title('distribution in pca2 direction')
```

```
variance along pca1 direction= 1.8477663843459717  
variance along pca2 direction= 0.15223361565402693
```

Out[6]: Text(0.5, 1.0, 'distribution in pca2 direction')



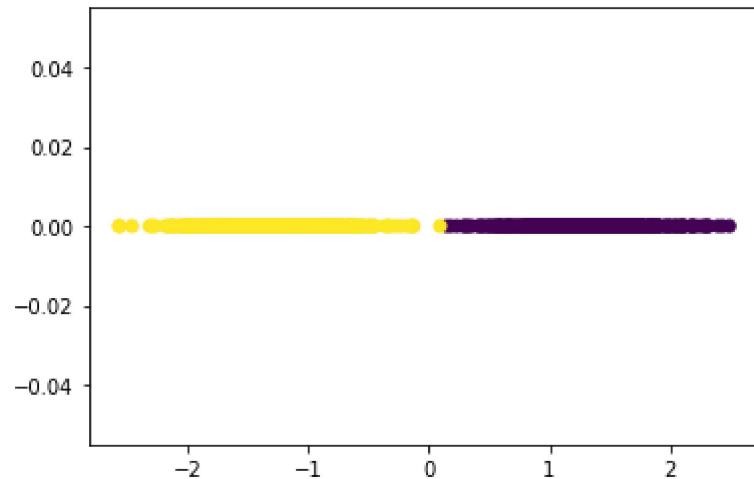
distribution in pca2 direction



```
In [7]: class pca:  
    # Constructor  
    def __init__(self, name='reg', data=None, retain_dim=None):  
        self.name = name # Create an instance variable  
        self.data=data  
        self.retain_dim=retain_dim if retain_dim is not None else self.ret_dim(self.data)  
        # compute pca transform value  
    def pca_comp(self,data):  
        data=self.pre_process(data)  
        cov= data.T @ data## Write your code here  
        u,_,_=np.linalg.svd(cov) # singular value decomposition  
        u_req= u[:, :self.retain_dim]## Write your code here  
        trans_data= data @ u_req## Write your code here  
        return trans_data,u_req  
        # compute the required retain dimension  
    def ret_dim(self,data):  
        data=self.pre_process(data)  
        cov=data.T @ data  
        _,s,_=np.linalg.svd(cov)  
        ind=(np.where((np.cumsum(s)/np.sum(s))>0.9))[0][0] ## Write your code here  
        return ind+1  
    def pre_process(self,data):  
        data1=(data-np.mean(data, axis=0))  
  
        data=data1/(np.std(data1, axis=0)+10**(-30)) # avoid divide by zero  
        return data
```

```
In [8]: # pca transformation  
PCA=pca(data=data)  
trans_data,trans_mat=PCA.pca_comp(data)  
plt.scatter(trans_data,np.zeros(trans_data.shape),c=label)
```

```
Out[8]: <matplotlib.collections.PathCollection at 0x7ff404525eb0>
```



```
In [9]: #classification using pca
#use k-nearest neighbour classifier after dimensionality reduction

from sklearn.neighbors import KNeighborsClassifier
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(trans_data, label)

print('KNN Training accuracy =',knn.score(trans_data,label)*100)

# test data
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,50)
data2=np.random.multivariate_normal(mean2,var,50)
data=np.concatenate((data1,data2))
tst_label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

print('KNN Testing accuracy =',knn.score(PCA.pre_process(data) @ trans_mat,tst_label)*100)
```

```
KNN Training accuracy = 99.9
KNN Testing accuracy = 100.0
```

PCA on MNIST

```
In [10]: %pip install idx2numpy
```

```
Requirement already satisfied: idx2numpy in /home/nirmit/.local/lib/python3.8/site-packages (1.2.3)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from idx2numpy) (1.14.0)
Requirement already satisfied: numpy in /home/nirmit/.local/lib/python3.8/site-packages (from idx2numpy) (1.2
2.3)
Note: you may need to restart the kernel to use updated packages.
```

In [17]: # MNIST data

```
file1='t10k-images-idx3-ubyte' ## Change the path accordingly
file2='t10k-labels-idx1-ubyte' ## Change the path accordingly

import idx2numpy

Images= idx2numpy.convert_from_file(file1)
labels= idx2numpy.convert_from_file(file2)

cl=[1,5]

# for class 1

id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[:50]
Im_1=Images[id1,:,:]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[:50]
Im_5=Images[id5,:,:]
lab_5=labels[id5]

plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])

#print(Im_5.shape)

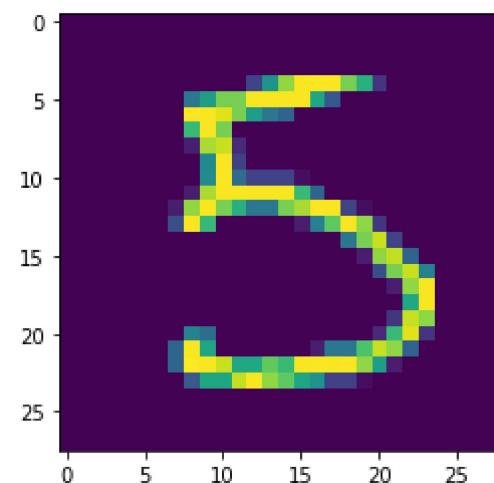
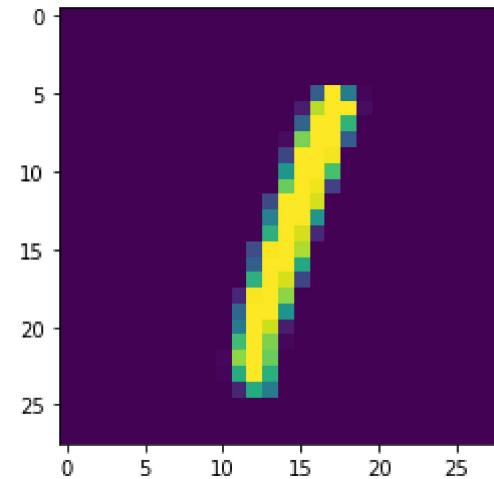
data=np.concatenate((Im_1,Im_5))
data=np.reshape(data,(data.shape[0],data.shape[1]*data.shape[2]))
print(data.shape)
G_lab=np.concatenate((lab_1,lab_5))
print(G_lab.shape)

data = data.astype('float32')

data /= 255
```

(100, 784)

(100,)



```
In [18]: print('Initial data dimension=' ,data.shape[1])
PCA=pca(data=data)

trans_data,trans_mat=PCA.pca_comp(data)
print('Retained dimesion after PCA=' ,trans_mat.shape[1])
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(trans_data, G_lab)

print('KNN Training accuracy =' ,knn.score(trans_data,G_lab)*100)

## testing
## data preparation
id_1=np.where(labels==c1[0])
id1=id_1[0]
id1=id1[100:150]
Im_1=Images[id1,:,:]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==c1[1])
id5=id_5[0]
id5=id5[100:150]
Im_5=Images[id5,:,:]
lab_5=labels[id5]

plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])

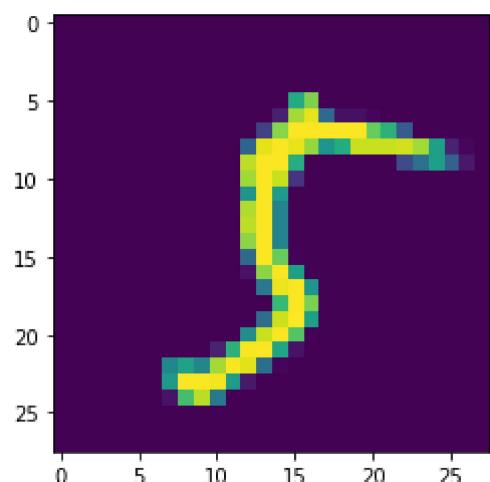
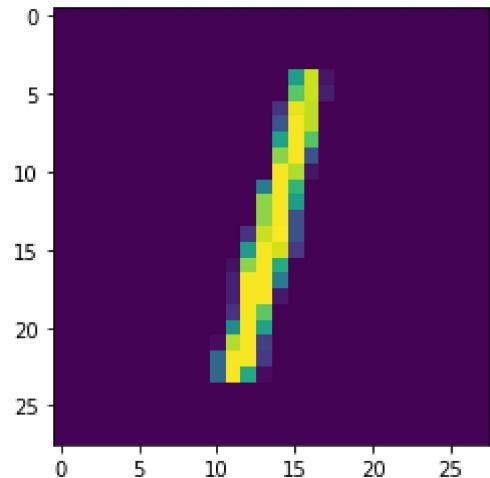
print(Im_5.shape)

data_tst=np.concatenate((Im_1,Im_5))
data_tst=np.reshape(data_tst,(data_tst.shape[0],data_tst.shape[1]*data_tst.shape[2]))

tst_lab=np.concatenate((lab_1,lab_5))

# final testing
print('KNN Testing accuracy =' ,knn.score(PCA.pre_process(data_tst) @ trans_mat,tst_lab)*100)
```

Initial data dimension= 784
Retained dimesion after PCA= 34
KNN Training accuracy = 96.0
(50, 28, 28)
KNN Testing accuracy = 98.0



Perform PCA on MNIST and Classify taking the data with any 3 Classes

In [19]: *## Write your code here*

```
c1=[1,5, 9]

## training data preparation
# for class 1
id_1=np.where(labels==c1[0])
id1=id_1[0]
id1=id1[:50]
Im_1=Images[id1,:,:]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==c1[1])
id5=id_5[0]
id5=id5[:50]
Im_5=Images[id5,:,:]
lab_5=labels[id5]

# for class 9
id_9=np.where(labels==c1[2])
id9=id_9[0]
id9=id9[:50]
Im_9=Images[id9,:,:]
lab_9=labels[id9]

plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])
plt.figure()
plt.imshow(Im_9[1,:,:])
plt.figure()

## Data Preprocessing
data=np.concatenate((Im_1,Im_5,Im_9))
data=np.reshape(data,(data.shape[0],data.shape[1]*data.shape[2]))
print(data.shape)
G_lab=np.concatenate((lab_1,lab_5, lab_9))
print(G_lab.shape)
data = data.astype('float32')
```

```
data /= 255

print('Initial data dimension=',data.shape[1])
PCA=pca(data=data)

trans_data,trans_mat=PCA.pca_comp(data)
print('Retained dimesion after PCA=',trans_mat.shape[1])

k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(trans_data, G_lab)

print('KNN Training accuracy =',knn.score(trans_data,G_lab)*100)

## testing data preparation
## for class 1
id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[100:150]
Im_1=Images[id1,:,:]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[100:150]
Im_5=Images[id5,:,:]
lab_5=labels[id5]

# for class 9
id_9=np.where(labels==cl[2])
id9=id_9[0]
id9=id9[100:150]
Im_9=Images[id9,:,:]
lab_9=labels[id9]

plt.imshow(Im_1[1,:,:])
```

```
plt.figure()
plt.imshow(Im_5[1,:,:])
plt.figure()
plt.imshow(Im_9[1,:,:])
plt.figure()
print(Im_5.shape)

data_tst=np.concatenate((Im_1,Im_5, Im_9))
data_tst=np.reshape(data_tst,(data_tst.shape[0],data_tst.shape[1]*data_tst.shape[2]))

tst_lab=np.concatenate((lab_1,lab_5,lab_9))

# final testing
print('KNN Testing accuracy =',knn.score(PCA.pre_process(data_tst) @ trans_mat,tst_lab)*100)
```

(150, 784)

(150,)

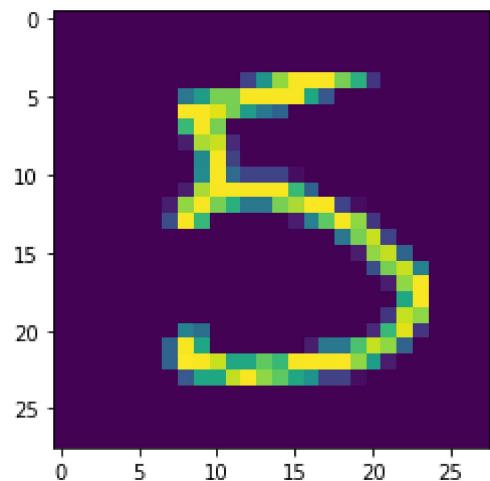
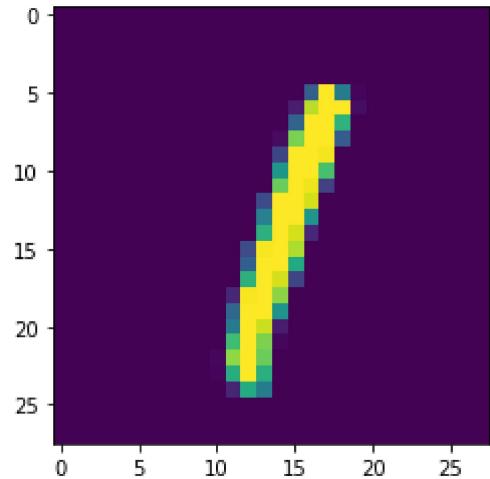
Initial data dimension= 784

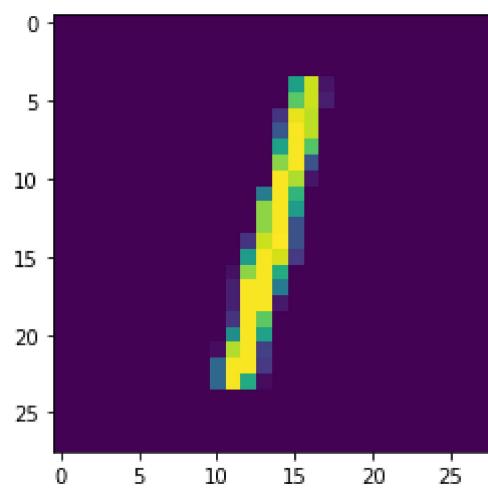
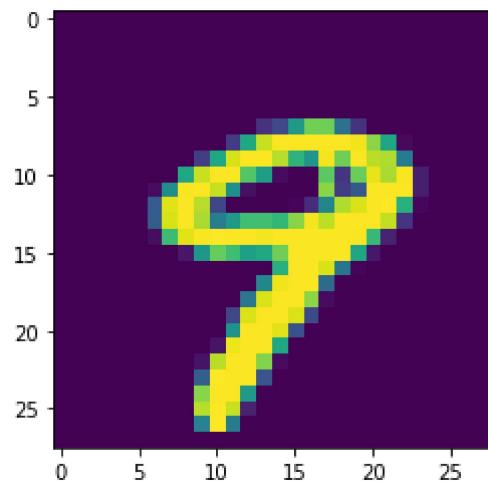
Retained dimesion after PCA= 50

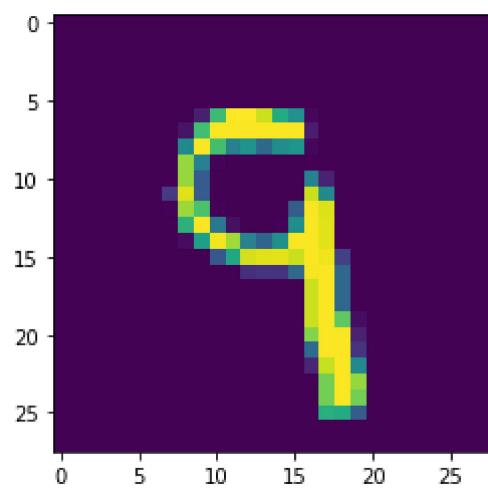
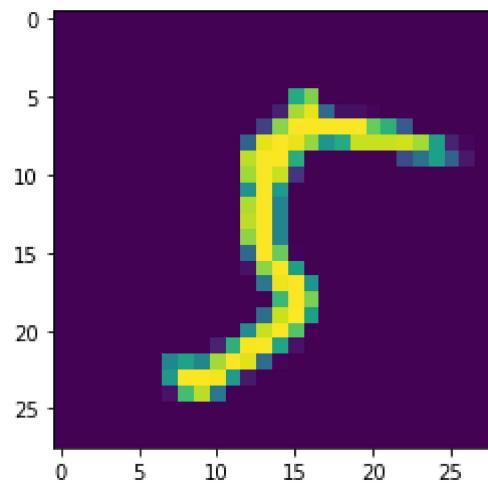
KNN Training accuracy = 94.66666666666667

(50, 28, 28)

KNN Testing accuracy = 90.66666666666666







<Figure size 432x288 with 0 Axes>

LDA

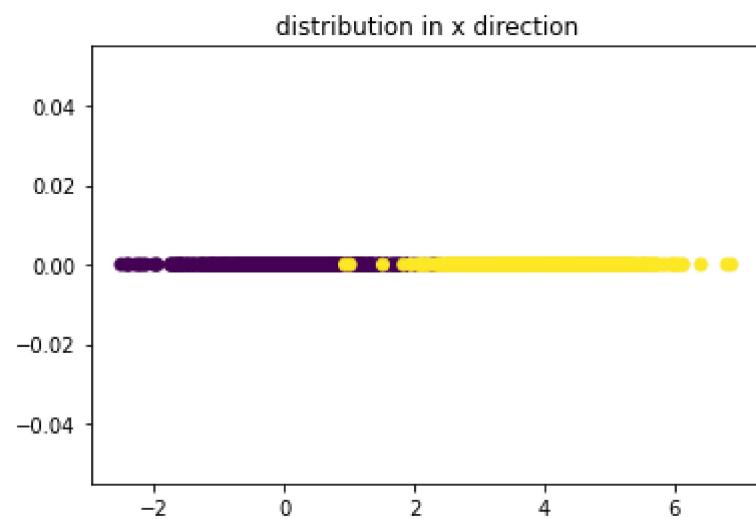
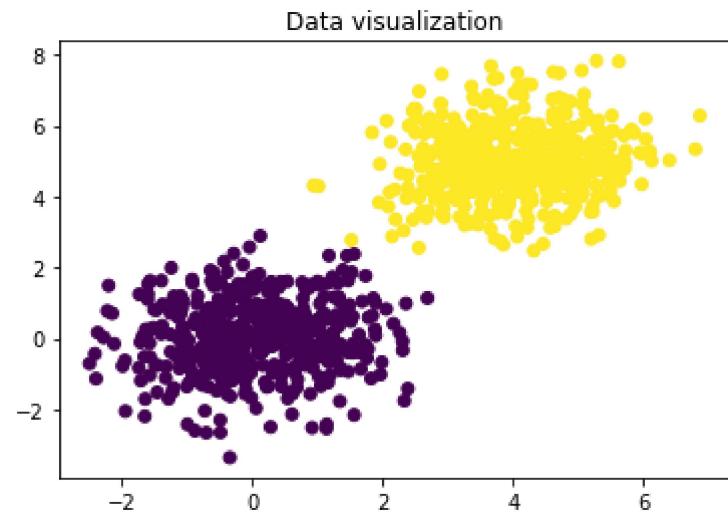
```
In [21]: import numpy as np
import matplotlib.pyplot as plt

# data generation

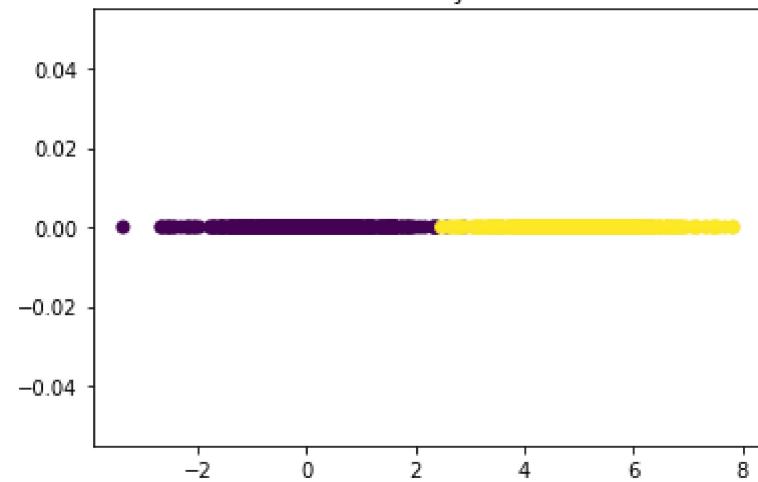
mean1=np.array([0,0])
mean2=np.array([4,5])
var=np.array([[1,0.1],[0.1,1]])
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,500)
data2=np.random.multivariate_normal(mean2,var,500)
data=np.concatenate((data1,data2))
label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

plt.figure()
plt.scatter(data[:,0],data[:,1],c=label)
plt.title('Data visualization')
plt.figure()
plt.scatter(data[:,0],np.zeros(data.shape[0]),c=label)
plt.title('distribution in x direction')
plt.figure()
plt.scatter(data[:,1],np.zeros(data.shape[0]),c=label)
plt.title('distribution in y direction')
```

Out[21]: Text(0.5, 1.0, 'distribution in y direction')



distribution in y direction



```
In [22]: # perform 2-class and m-class LDA
def LDA(data,label):
    id={}
    data_l={}
    mean_l={}
    cov_l={}
    S_w=np.zeros((data.shape[1],data.shape[1]))

    cls=np.unique(label)
    for i in cls:
        id[i]=np.where(label==i)[0]
        data_l[i]=data[id[i],:]
        mean_l[i]=np.mean(data_l[i],axis=0)
        cov_l[i]=(np.matmul((data_l[i]-mean_l[i]).T , (data_l[i]-mean_l[i])))/(data_l[i].shape[0]-1)## Write your code here
    r code here
    S_w=S_w+cov_l[i]

    S_w=S_w/len(data_l)

    if len(data_l)==2:
        S_b= np.matmul((mean_l[1]-mean_l[0]).T , (mean_l[1]-mean_l[0]))## Write your code here
        w= np.matmul(np.linalg.pinv(S_w) , (mean_l[1]-mean_l[0]).T)## Write your code here

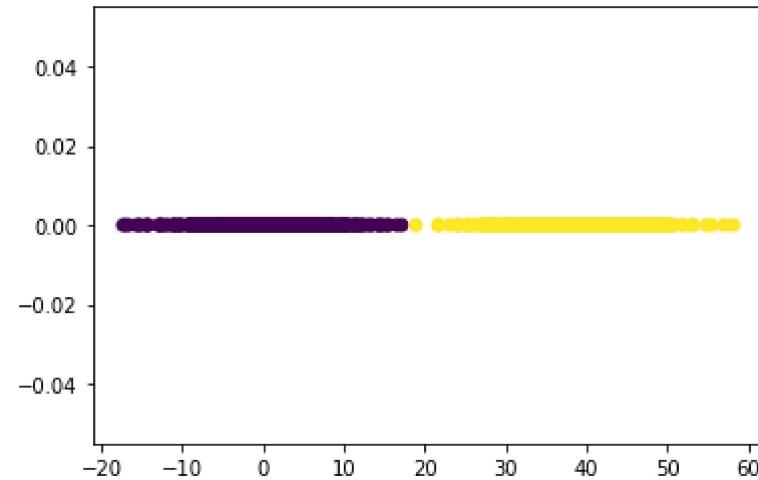
    else:
        S_t=np.cov(data,rowvar=False)
        S_b= S_t-S_w ## Write your code here
        u,_,_= np.linalg.svd(np.matmul(np.linalg.pinv(S_w) , S_b))## Write your code here
        w=u[:,len(data_l)-1]

return w
```

```
In [23]: # after LDA projection
```

```
w=LDA(data,label)
plt.figure()
plt.scatter(data @ w,np.zeros(data.shape[0]),c=label)
```

```
Out[23]: <matplotlib.collections.PathCollection at 0x7ff3fc24e4f0>
```



```
In [24]: # Classification using :DA
# Use k-nearest neighbour classifier (Scikit Learn) after dimensionality reduction

## Write your code here
from sklearn.neighbors import KNeighborsClassifier

LDA_data= np.matmul(data , w[:,np.newaxis])
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(LDA_data, label)

print('Training accuracy =',knn.score(LDA_data,label)*100)

# testing data generation
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,50)
data2=np.random.multivariate_normal(mean2,var,50)
data_tst=np.concatenate((data1,data2))
tst_label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

print('Testing accuracy =',knn.score(np.matmul(data_tst, w[:,np.newaxis]),tst_label))*100)
```

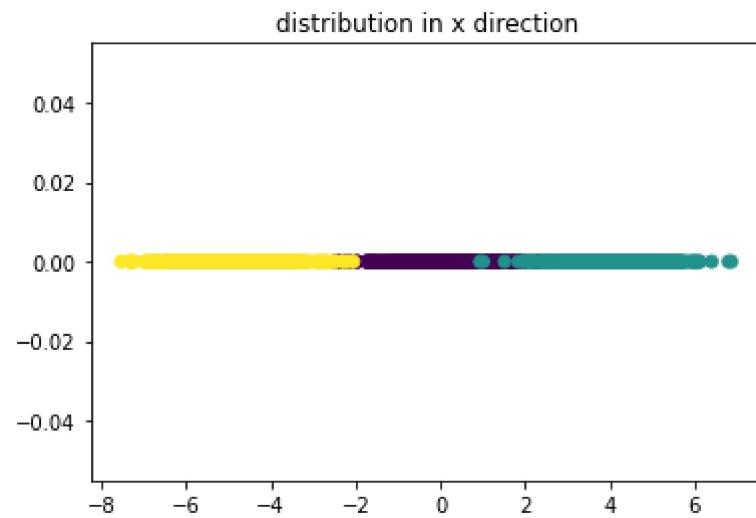
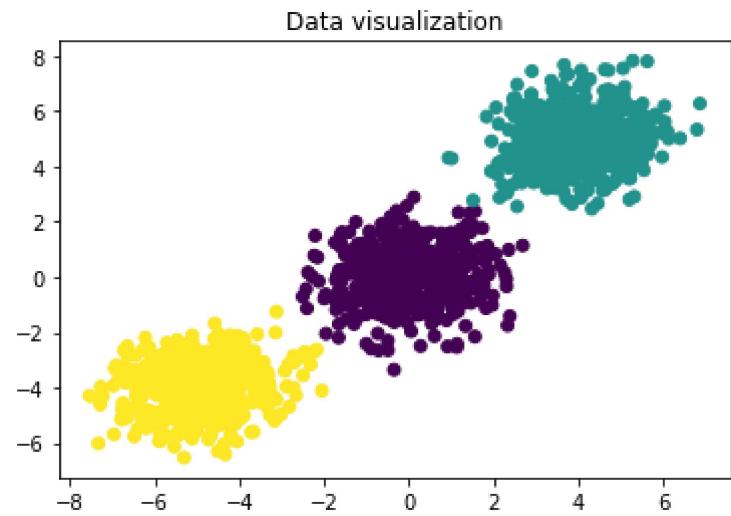
```
Training accuracy = 100.0
Testing accuracy = 100.0
```

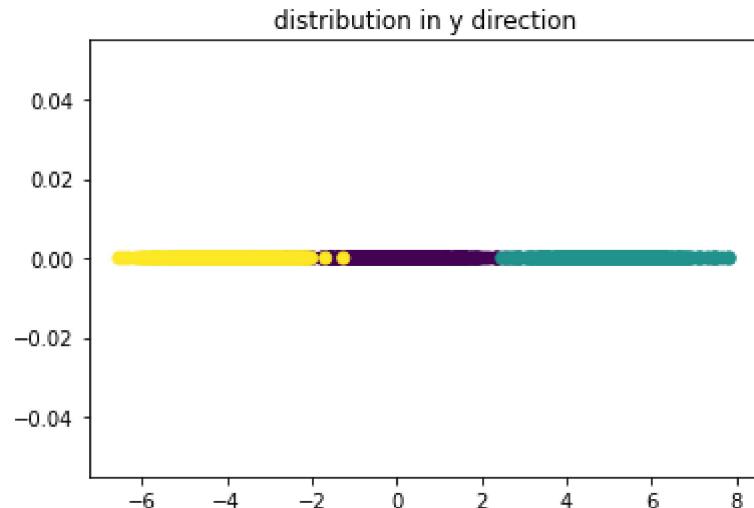
LDA Multiclass

```
In [25]: mean1=np.array([0,0])
mean2=np.array([4,5])
mean3=np.array([-5,-4])
var=np.array([[1,0.1],[0.1,1]])
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,500)
data2=np.random.multivariate_normal(mean2,var,500)
data3=np.random.multivariate_normal(mean3,var,500)
data=np.concatenate((data1,data2,data3))
label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0]),np.ones(data3.shape[0])+1))

plt.figure()
plt.scatter(data[:,0],data[:,1],c=label)
plt.title('Data visualization')
plt.figure()
plt.scatter(data[:,0],np.zeros(data.shape[0]),c=label)
plt.title('distribution in x direction')
plt.figure()
plt.scatter(data[:,1],np.zeros(data.shape[0]),c=label)
plt.title('distribution in y direction')
```

Out[25]: Text(0.5, 1.0, 'distribution in y direction')

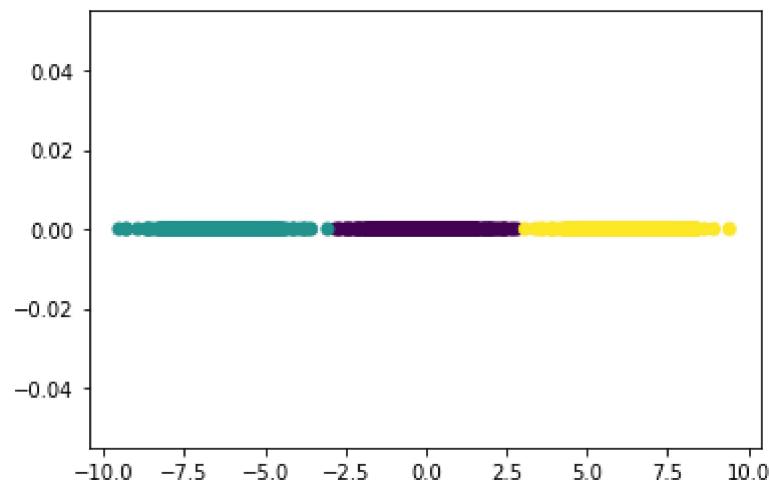




```
In [26]: # after projection
w=LDA(data,label)
print(w.shape)
plt.figure()
plt.scatter(data @ w[:,0],np.zeros(data.shape[0]),c=label) # by performing 1D projection
```

(2, 2)

```
Out[26]: <matplotlib.collections.PathCollection at 0x7ff418805820>
```



```
In [27]: # Testing (using KNN)
# Use k-nearest neighbour classifier (Scikit Learn) after dimensionality reduction

## Write your code here
LDA_data= np.matmul(data , w)
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(LDA_data, label)

print('Training accuracy =',knn.score(LDA_data,label)*100)

# test data
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,50)
data2=np.random.multivariate_normal(mean2,var,50)
data3=np.random.multivariate_normal(mean3,var,50)
data_tst=np.concatenate((data1,data2,data3))
tst_label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0]),np.ones(data2.shape[0])+1))

print('Testing accuracy =',knn.score(np.matmul(data_tst, w),tst_label)*100)
```

Training accuracy = 99.9333333333332

Testing accuracy = 100.0

Perform LDA on MNIST and Classify using the data of any 3 classes

```
In [37]: ## Write your code here
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

## data preparation
## for class 1
id_1=np.where(labels==cl[0])
id1=id_1[0]
id1=id1[50:150]
Im_1=Images[id1,:,:]
lab_1=labels[id1]

# for class 5
id_5=np.where(labels==cl[1])
id5=id_5[0]
id5=id5[50:150]
Im_5=Images[id5,:,:]
lab_5=labels[id5]

# for class 9
id_9=np.where(labels==cl[2])
id9=id_9[0]
id9=id9[50:150]
Im_9=Images[id9,:,:]
lab_9=labels[id9]

plt.imshow(Im_1[1,:,:])
plt.figure()
plt.imshow(Im_5[1,:,:])
plt.figure()
plt.imshow(Im_9[1,:,:])
plt.figure()
print(Im_5.shape)

data=np.concatenate((Im_1,Im_5, Im_9))
data=np.reshape(data,(data.shape[0],data.shape[1]*data.shape[2]))
print(data.shape)
G_lab=np.concatenate((lab_1,lab_5, lab_9))
```

```
print(G_lab.shape)
data = data.astype('float32')
data /= 255

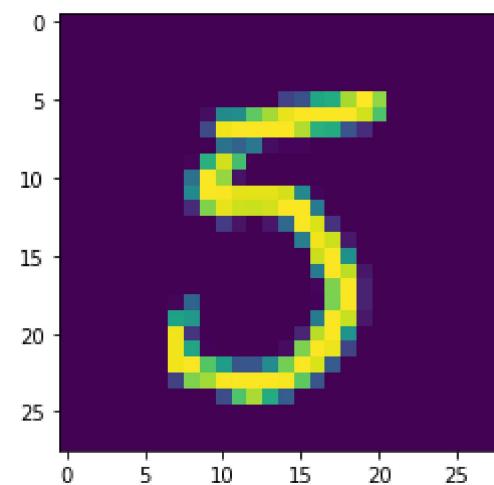
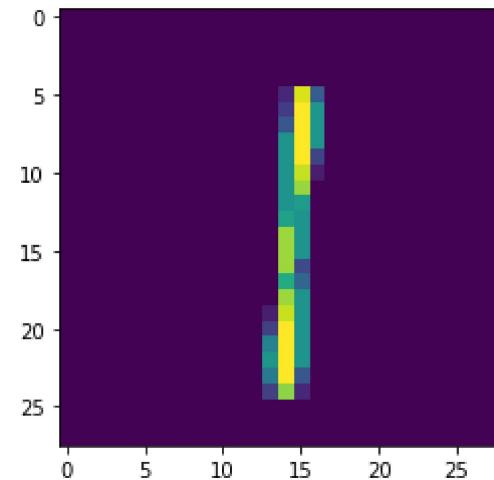
X_train, X_test, y_train, y_test = train_test_split(data, G_lab, test_size=0.2, random_state=0)

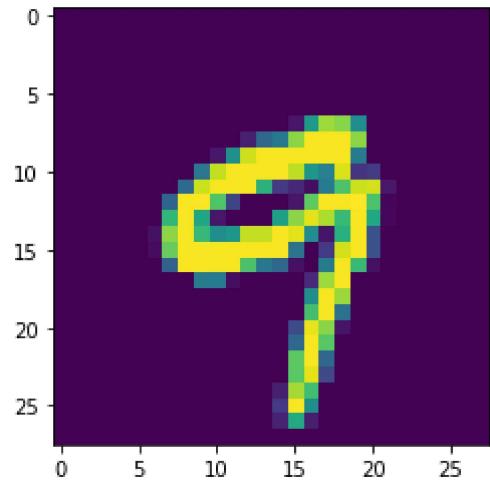
lda = LDA(n_components=1)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
```

(100, 28, 28)

(300, 784)

(300,)





<Figure size 432x288 with 0 Axes>

```
In [38]: k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
print('Training accuracy = ',knn.score(X_train, y_train)*100)

y_pred = knn.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix :-\n", cm)
print('Accuracy : ' + str(accuracy_score(y_test, y_pred)*100))
```

Training accuracy = 100.0

Confusion Matrix :-

```
[[20  0  0]
 [ 2 12  6]
 [ 2  5 13]]
```

Accuracy : 75.0