Lab_9_Dimensionality_Reduction

October 11, 2021

1 LAB 9 : Dimensionality Reduction

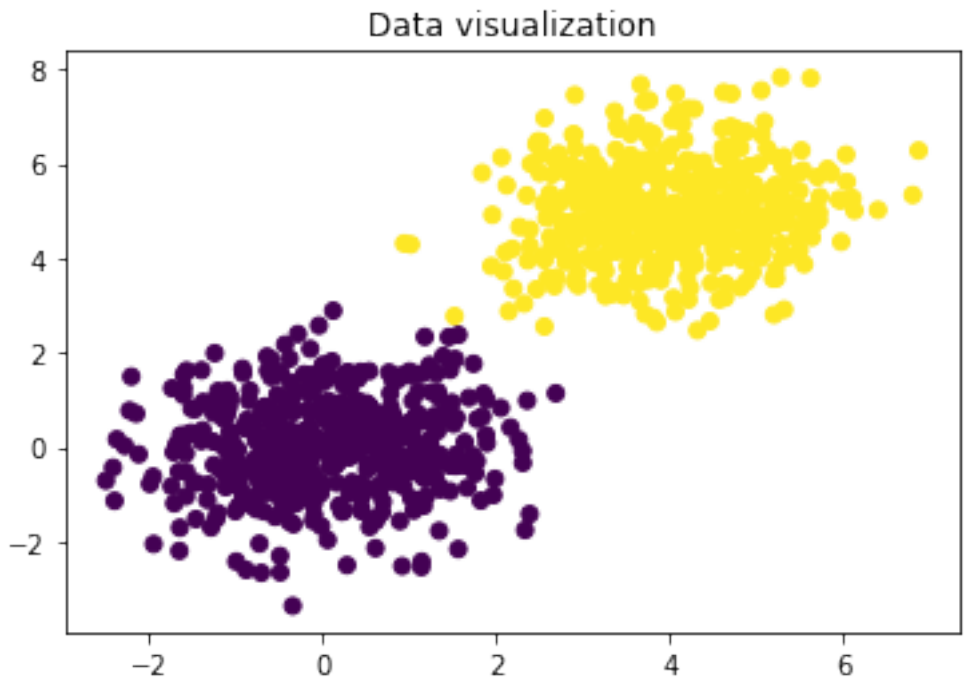
1. Principal Component Analysis (PCA)
2. Linear Discriminant Analysis (LDA)

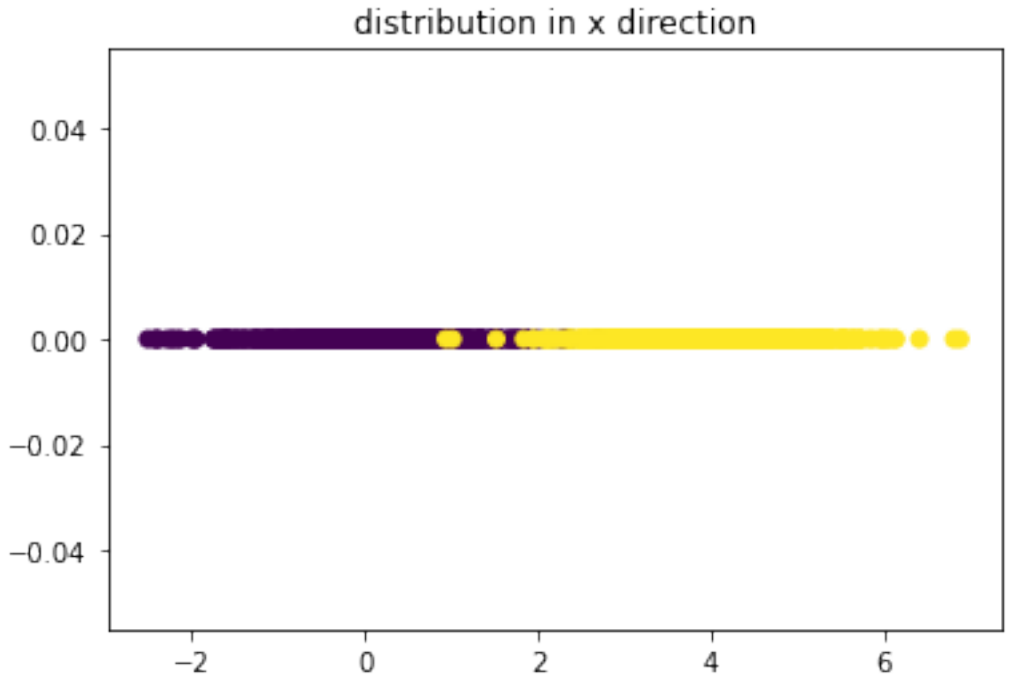
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

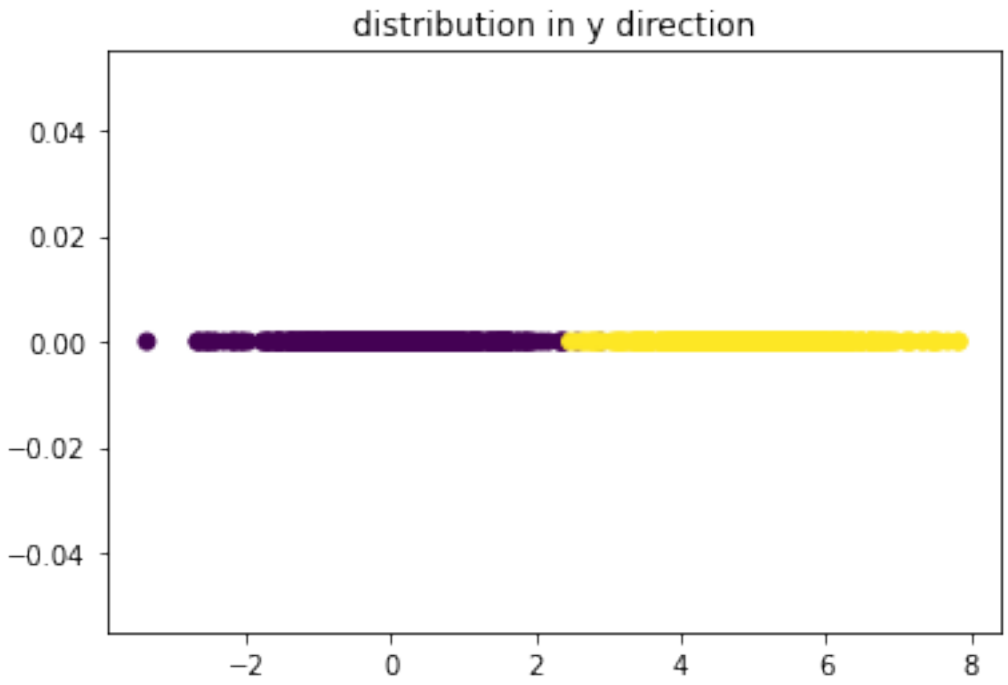
2 PCA

[ ]:
```
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')
```

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

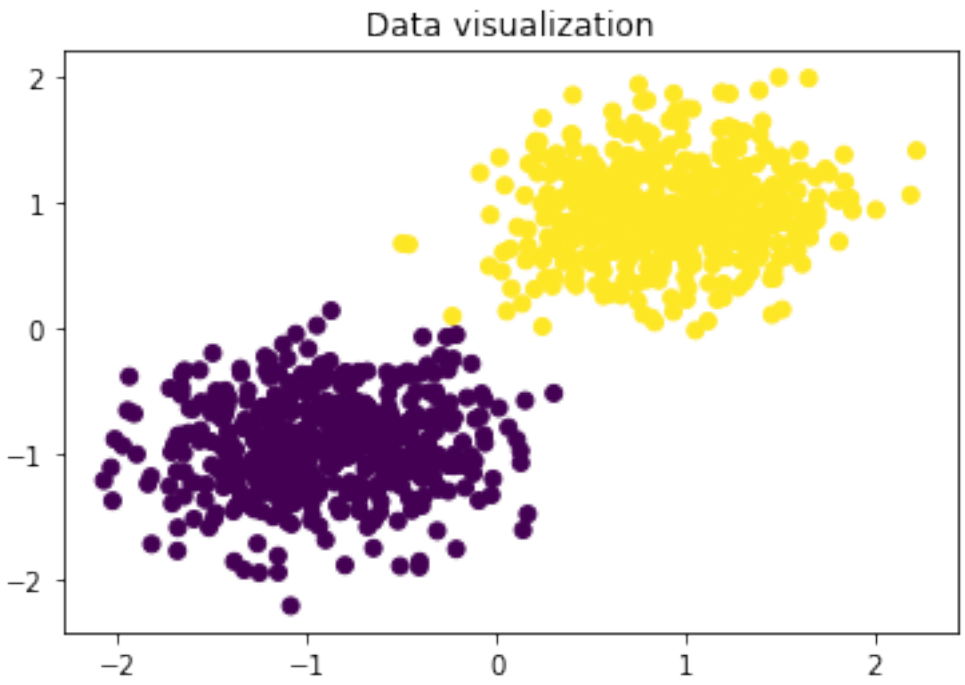

Data visualization
8
6
4
2
0
−2
−2
0
2
4
6


distribution in x direction
0.04
0.02
0.00
−0.02
−0.04
−2
0
2
4
6



[ ]:
```
# Data normalization

# Perform data normalization here using mean substraction and std division
## Write your code here

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

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

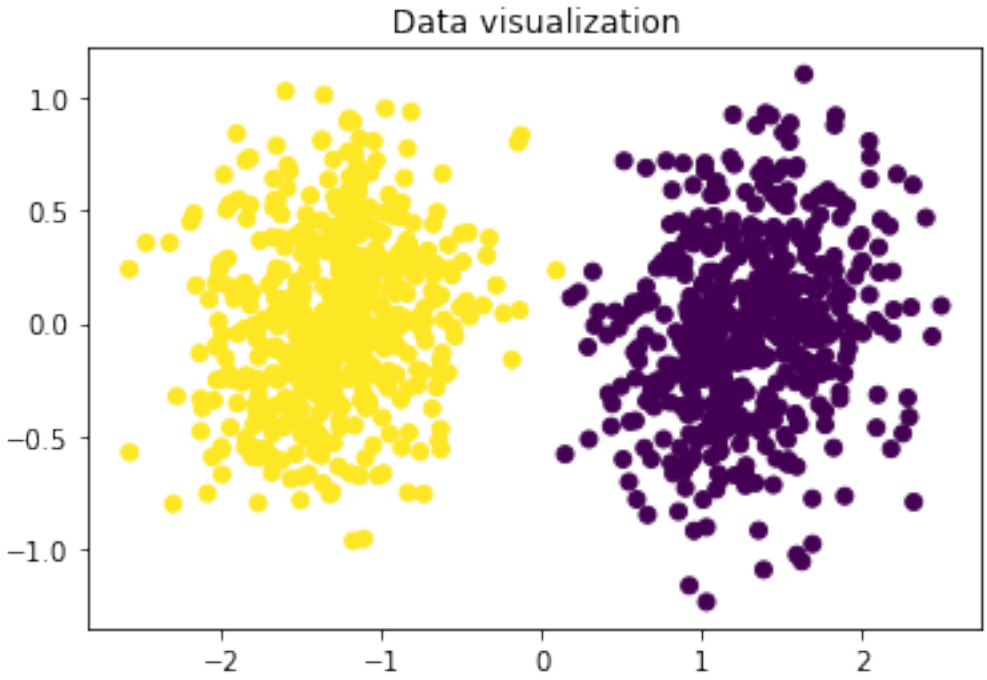


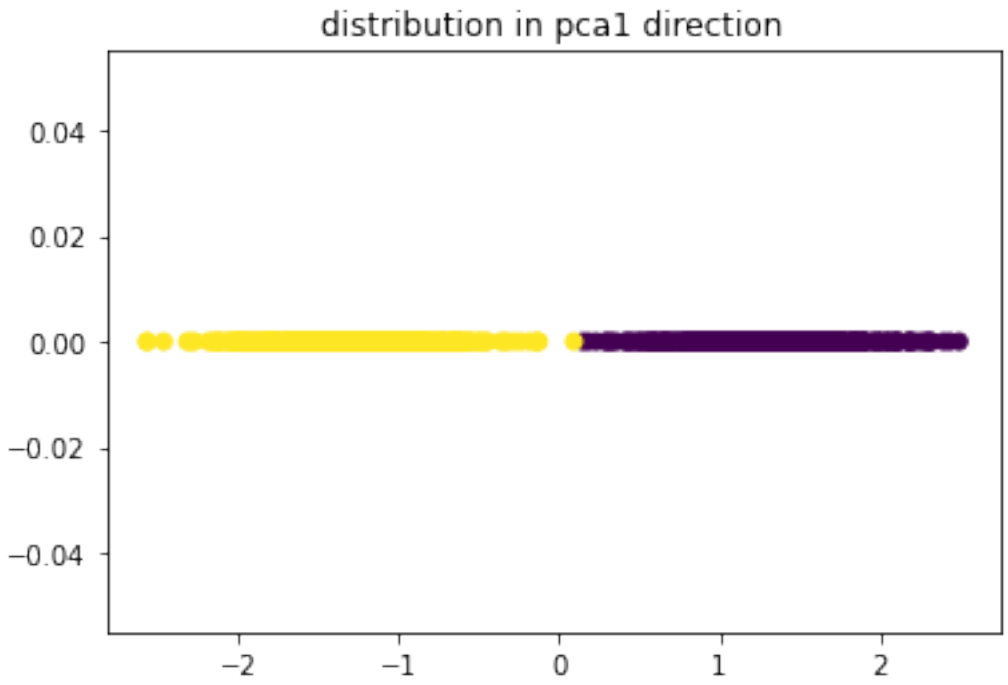
[ ]:
```
# PCA

# coverance matrix
cov=data.T @ data


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

trans_data= ## 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()
```

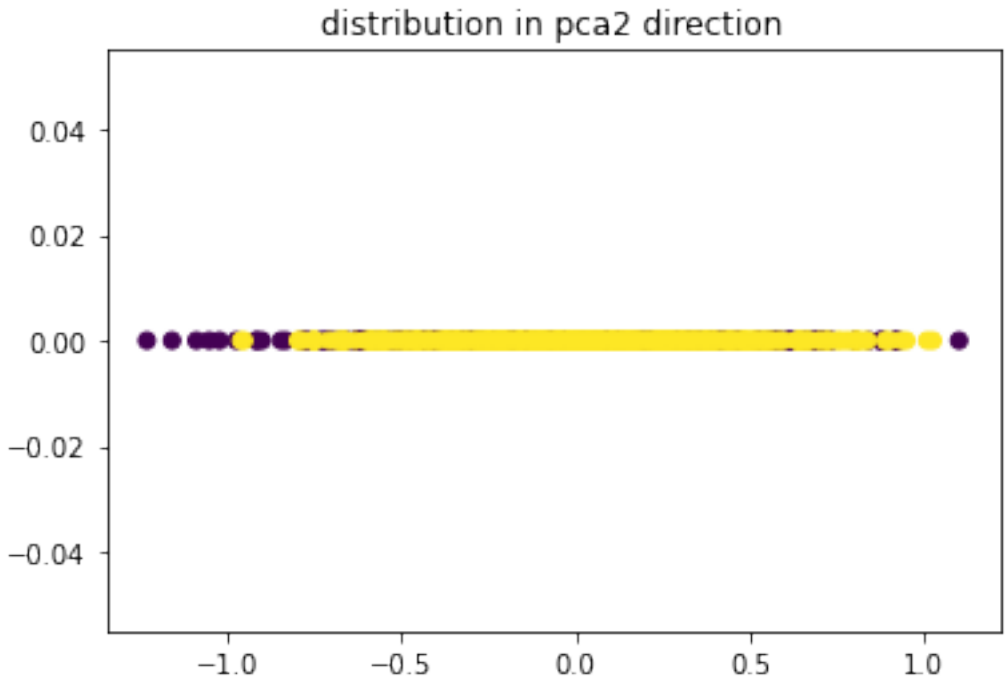
```
plt.scatter(trans_data[:,1],np.zeros(data.shape[0]),c=label)
plt.title('distribution in pca2 direction')
```

```
variance along pca1 direction= 1.8477663843459722
variance along pca2 direction= 0.15223361565402702
```

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





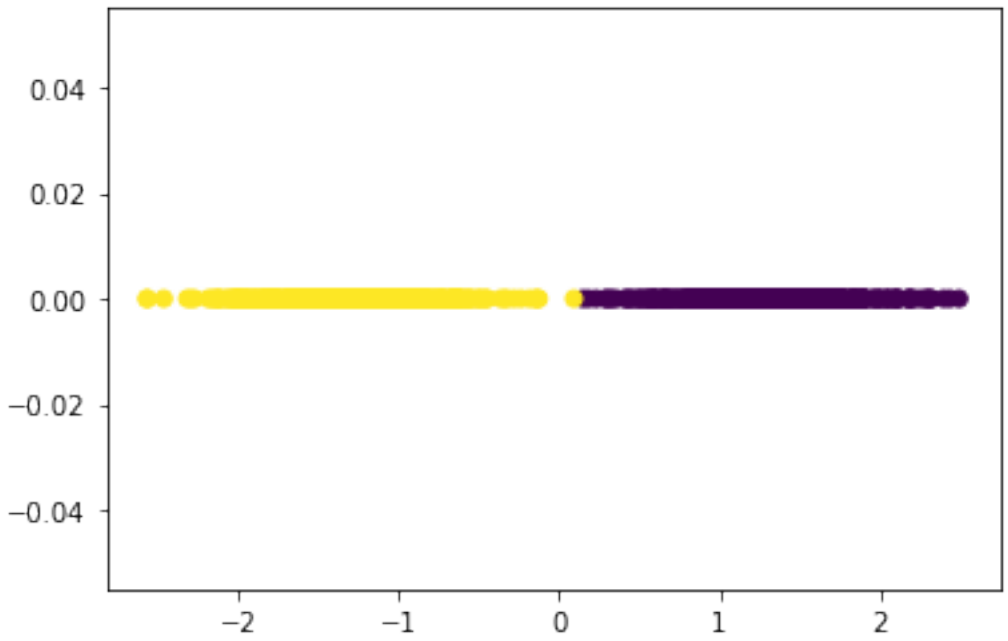


[ ]:
```
class pca:
  # Constructor
```

```
  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= ## Write your code here
    u,_,_=np.linalg.svd(cov) # singular value decomposition
    u_req= ## Write your code here
    trans_data= ## 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= ## 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
```

[ ]:
```
# 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)
```

[ ]:
```
<matplotlib.collections.PathCollection at 0x7ff5b79a5e10>
```



[ ]:
```
#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
```

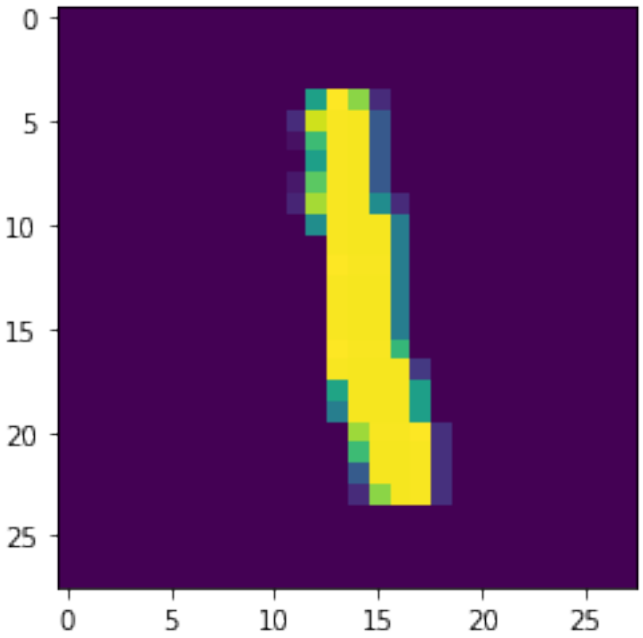
2.1 PCA on MNIST

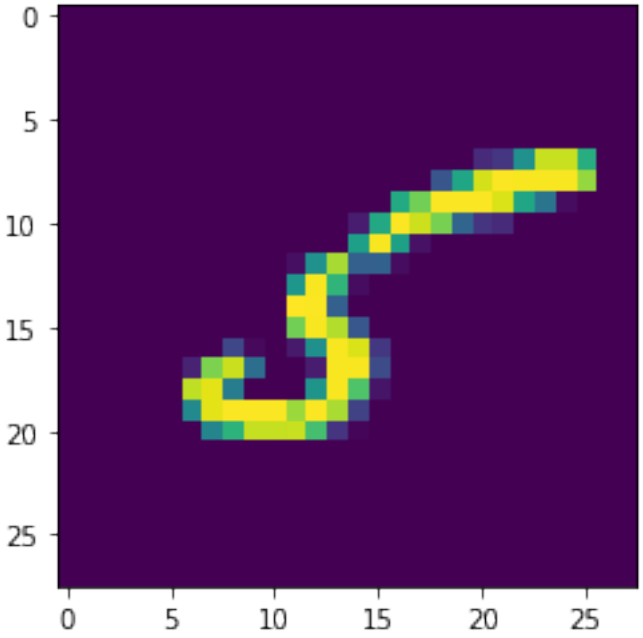
[ ]:
```
!pip install idx2numpy
```

[ ]:
```
# MNIST data

file1='/content/train-images.idx3-ubyte' ## Change the path accordingly
file2='/content/train-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,)
```

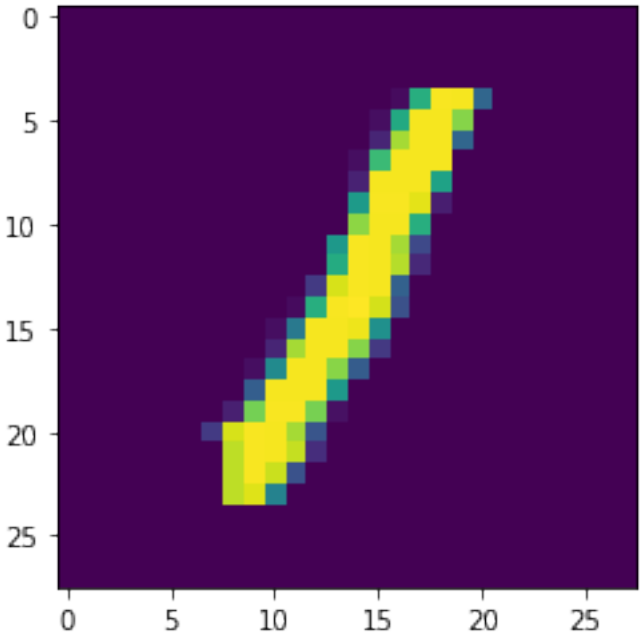


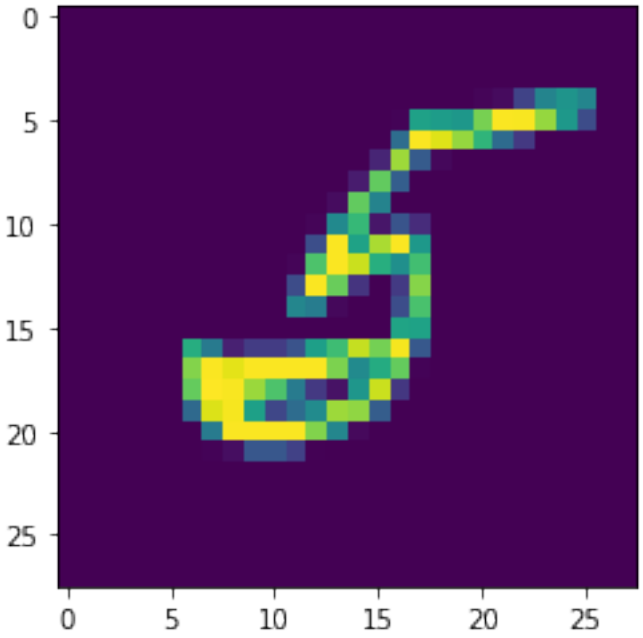


[ ]:
```
print('Initial data dimension=',data.shape[1])
PCA=pca(data=data)
```

```
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==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]

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= 36
KNN Training accuracy = 96.0
(50, 28, 28)
KNN Testing accuracy = 97.0
```





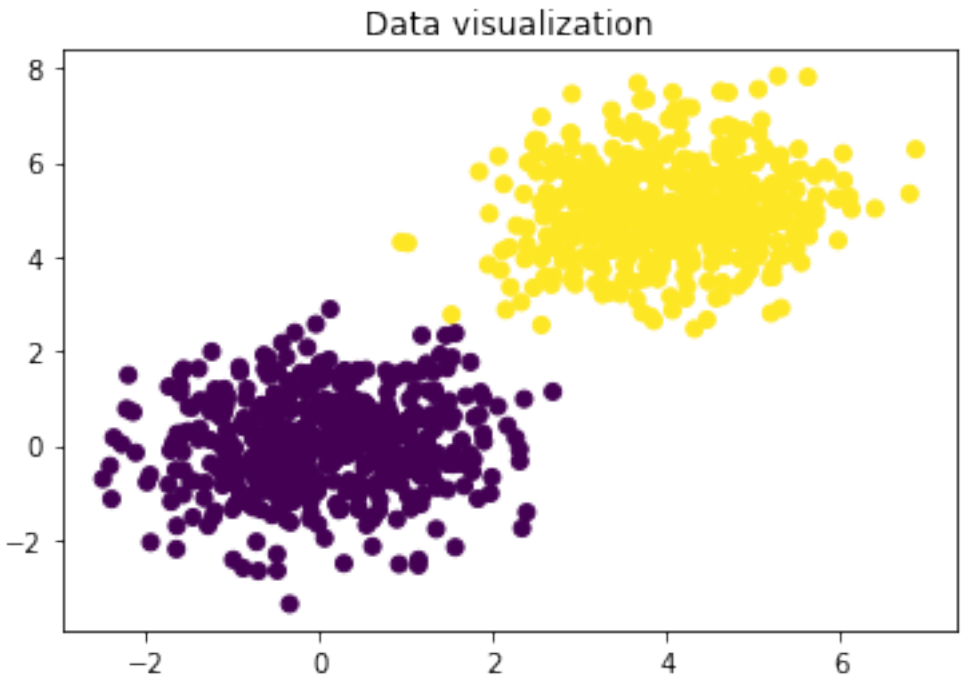
Perform PCA on MNIST and Classify taking the data with any 3 Classes

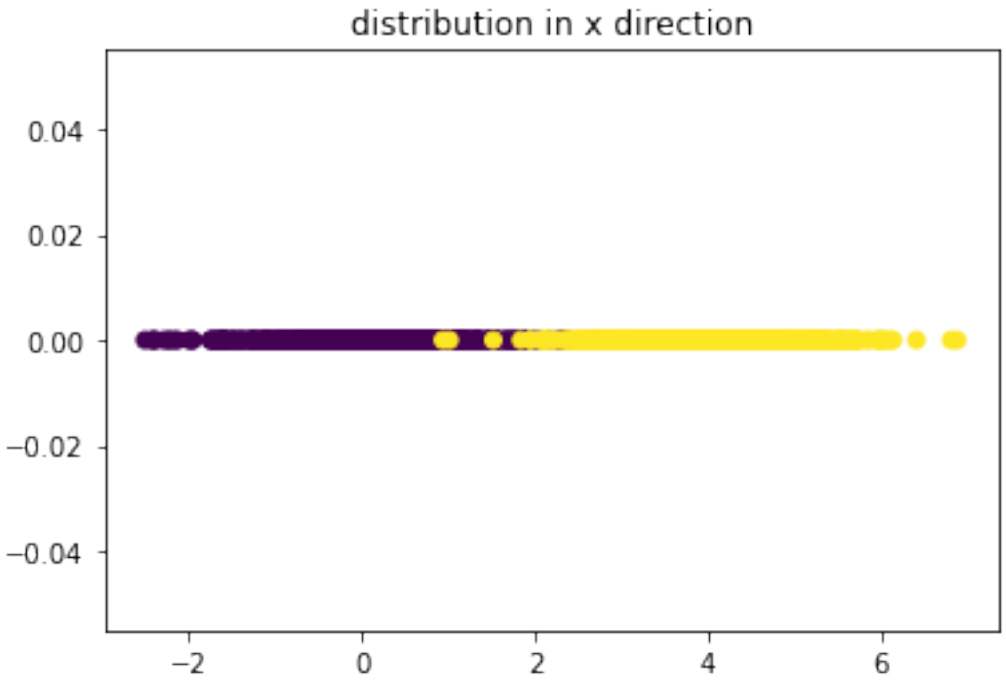
[ ]:
```
## Write your code here
```

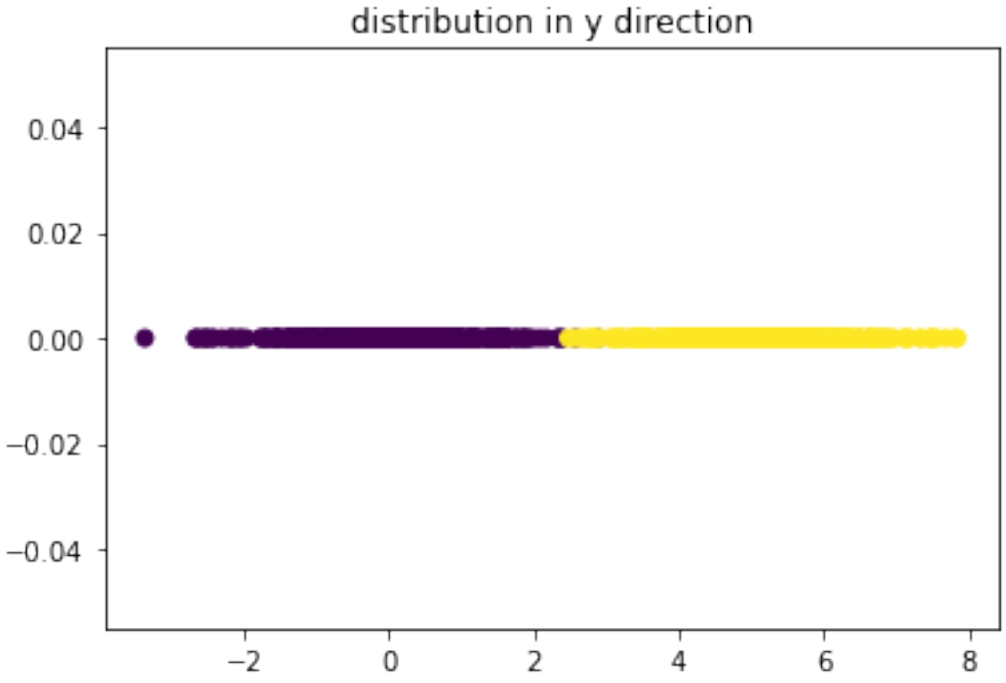
3 LDA

[ ]:
```
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')
```

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



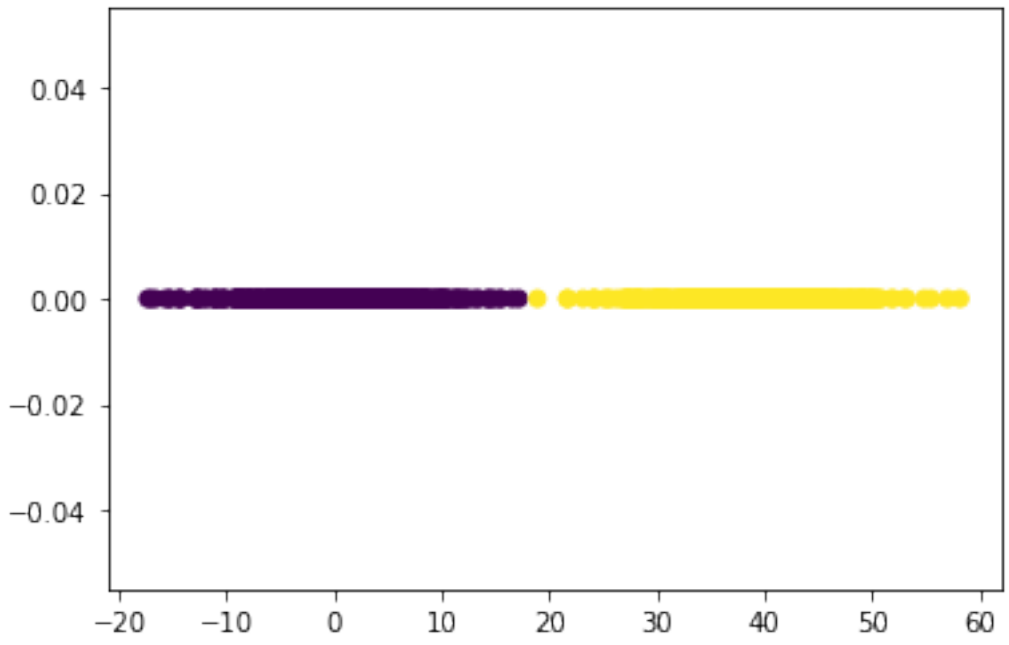

distribution in x direction
0.04
0.02
0.00
−0.02
−0.04
−2
0
2
4
6


distribution in y direction
0.04
0.02
0.00
−0.02
−0.04
−2
0
2
4
6
8

[ ]:
```
# 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]= ## Write your code here
    S_w=S_w+cov_l[i]


  S_w=S_w/len(data_l)

  if len(data_l)==2:
    S_b= ## Write your code here
    w= ## Write your code here

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

  return w
```

[ ]:
```
# after LDA projection

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

[ ]:
```
<matplotlib.collections.PathCollection at 0x7ff59edab2d0>
```



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

## Write your code here
```

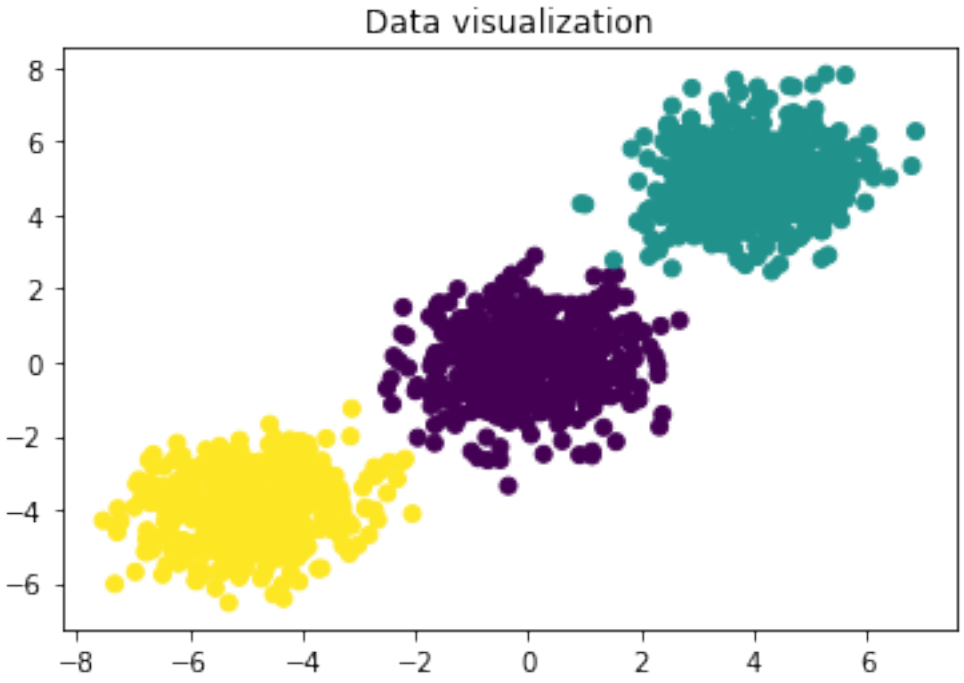
```
KNN Training accuracy = 100.0
KNN Testing accuracy = 100.0
```

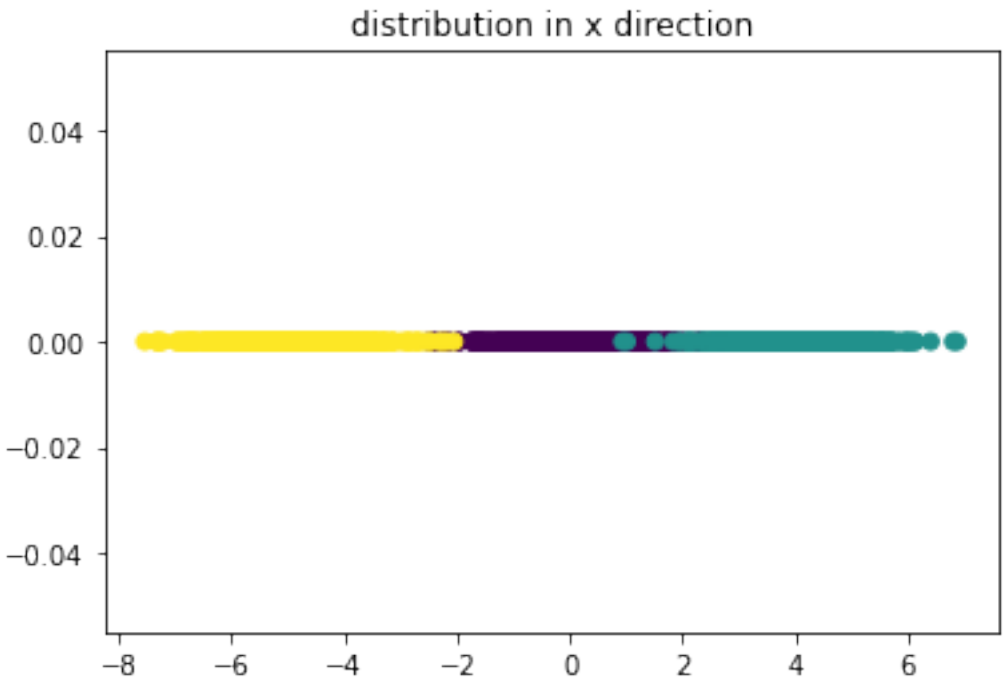
3.1 LDA Multiclass

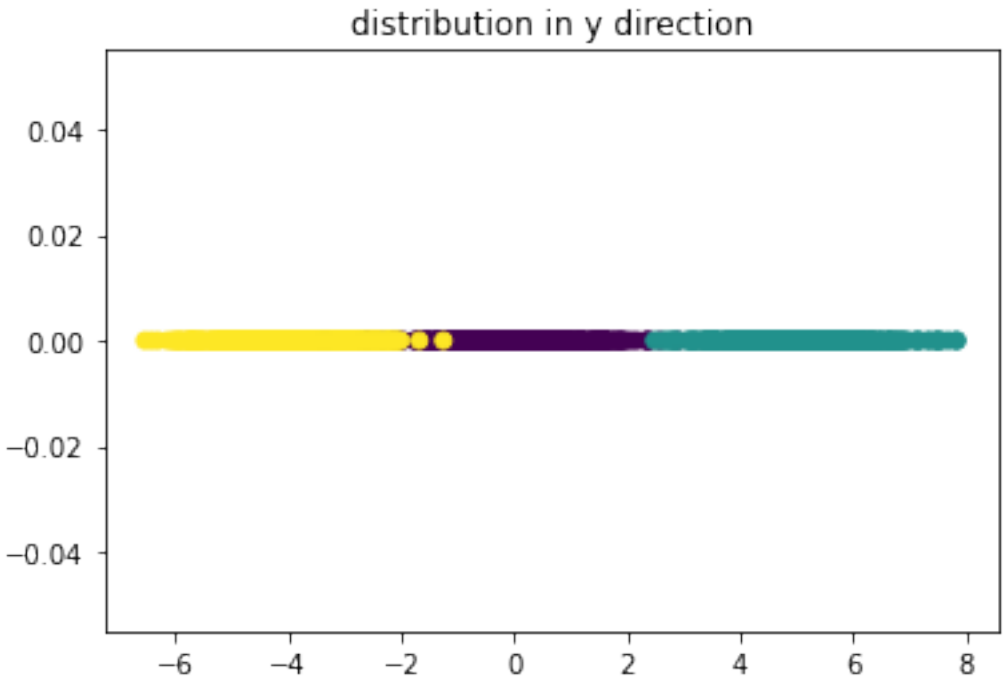
[ ]:
```
mean1=np.array([0,0])
mean2=np.array([4,5])
mean3=np.array([-5,-4])
var=np.array([[1,0.1],[0.1,1]])
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,500)
data2=np.random.multivariate_normal(mean2,var,500)
data3=np.random.multivariate_normal(mean3,var,500)
data=np.concatenate((data1,data2,data3))
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')
```

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





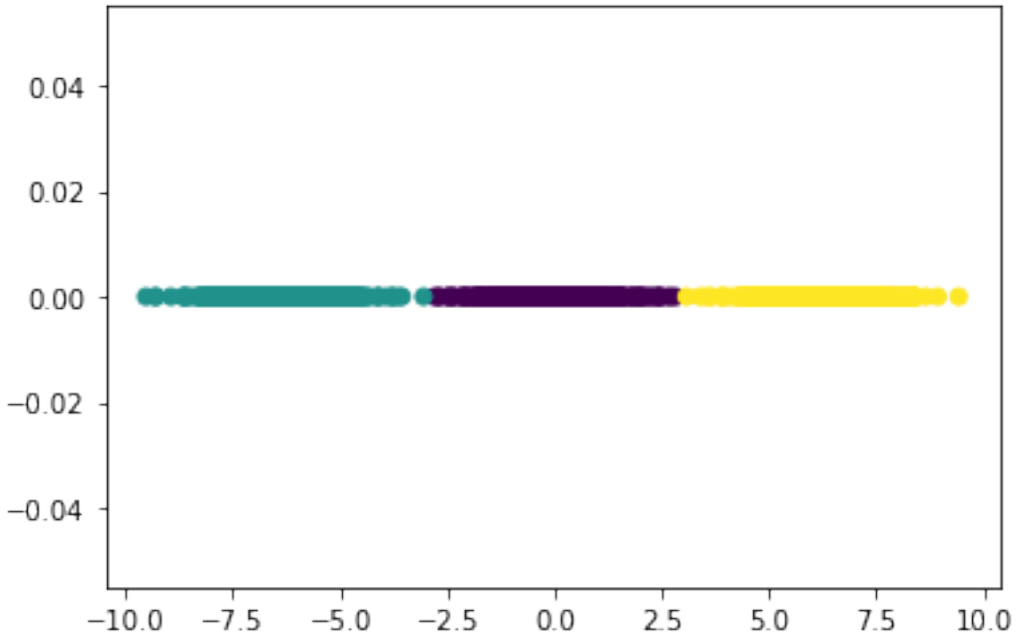


[ ]:
```
# after projection
w=LDA(data,label)
```

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

```
(2, 2)
```

[ ]: <matplotlib.collections.PathCollection at 0x7ff59f0c2190>



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

## Write your code here
```

```
KNN Training accuracy = 99.93333333333332
KNN Testing accuracy = 100.0
```

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

[ ]:
```
## Write your code here
```