200010021

October 29, 2022

#LAB 9 : Dimensionality Reduction

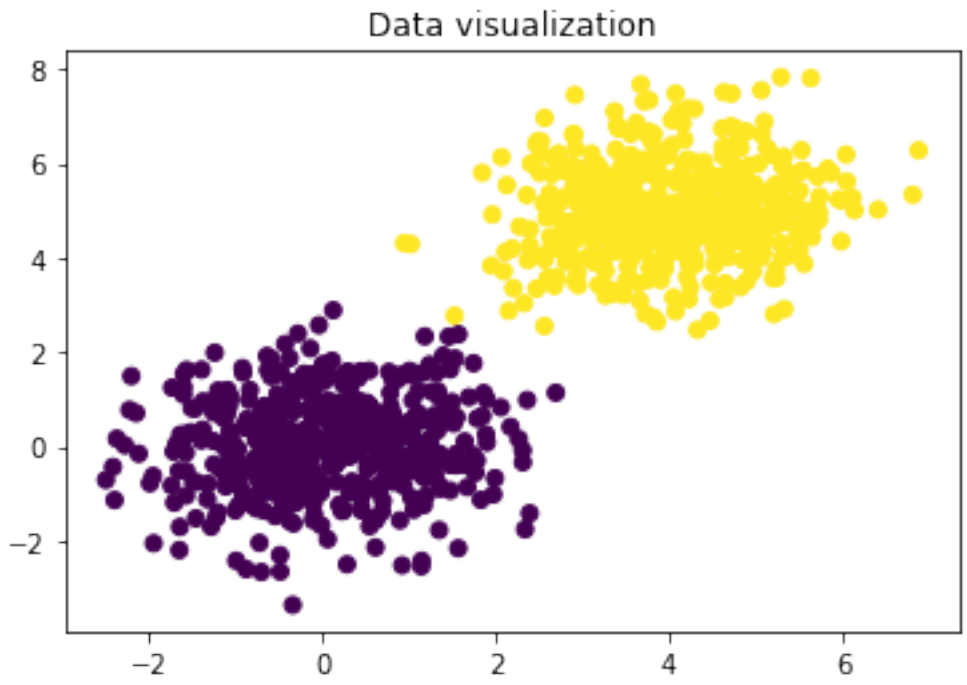
1. Principal Component Analysis (PCA)
2. Linear Discriminant Analysis (LDA)

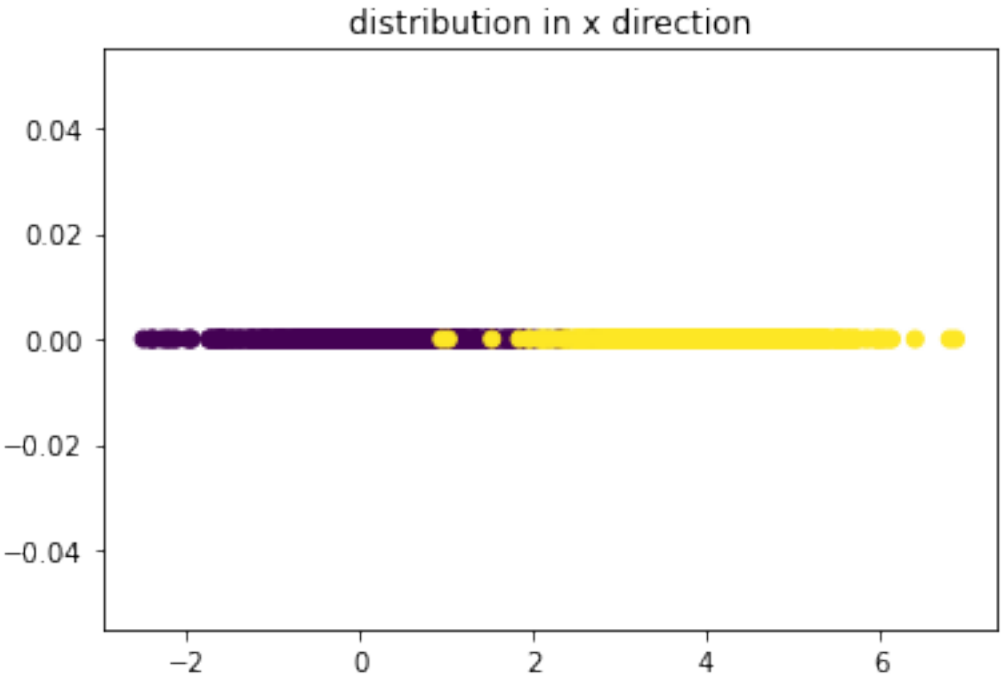
[202]:
```
import numpy as np
import matplotlib.pyplot as plt
```

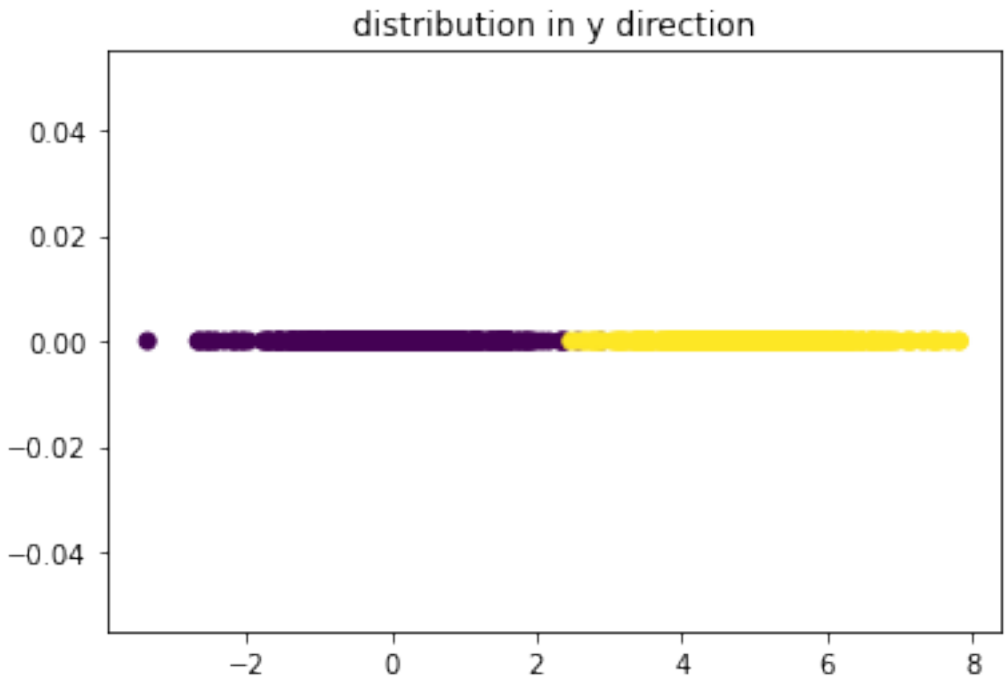
#PCA

[203]:
```
import numpy as np
import matplotlib.pyplot as plt

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')
```

[203]:
```
Text(0.5, 1.0, 'distribution in y direction')
```

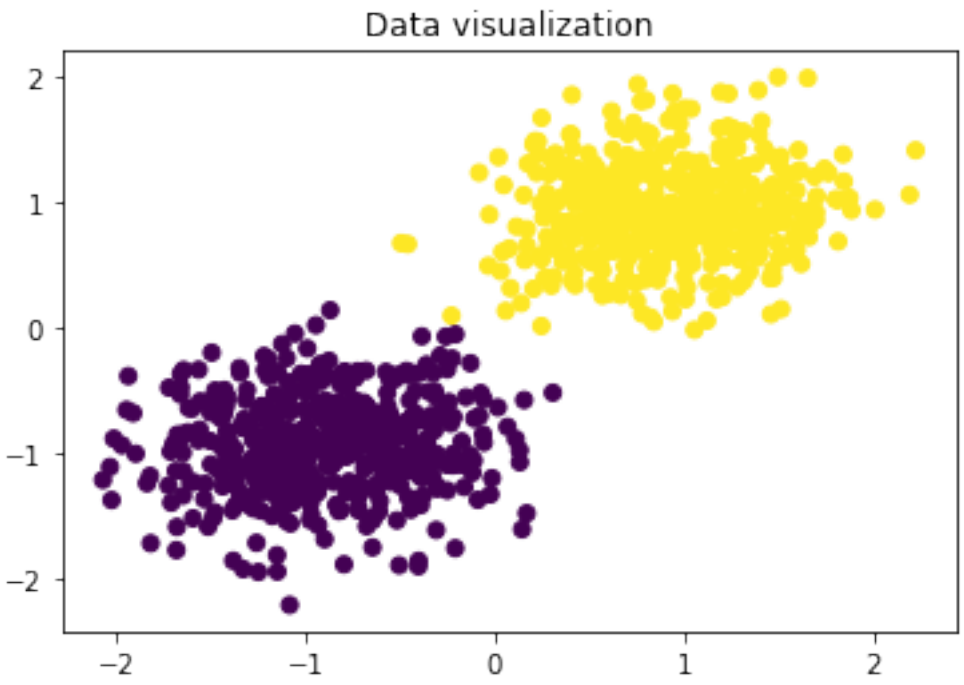

Data visualization
8
6
4
2
0
−2
−2
0
2
4
6


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



[204]:
```
#Data normalization

data1=(data-np.mean(data,axis=0))
data=data1/np.std(data1,axis=0)
plt.figure()
plt.scatter(data[:,0],data[:,1],c=label)
plt.title('Data visualization')
```

[204]:
```
Text(0.5, 1.0, 'Data visualization')
```

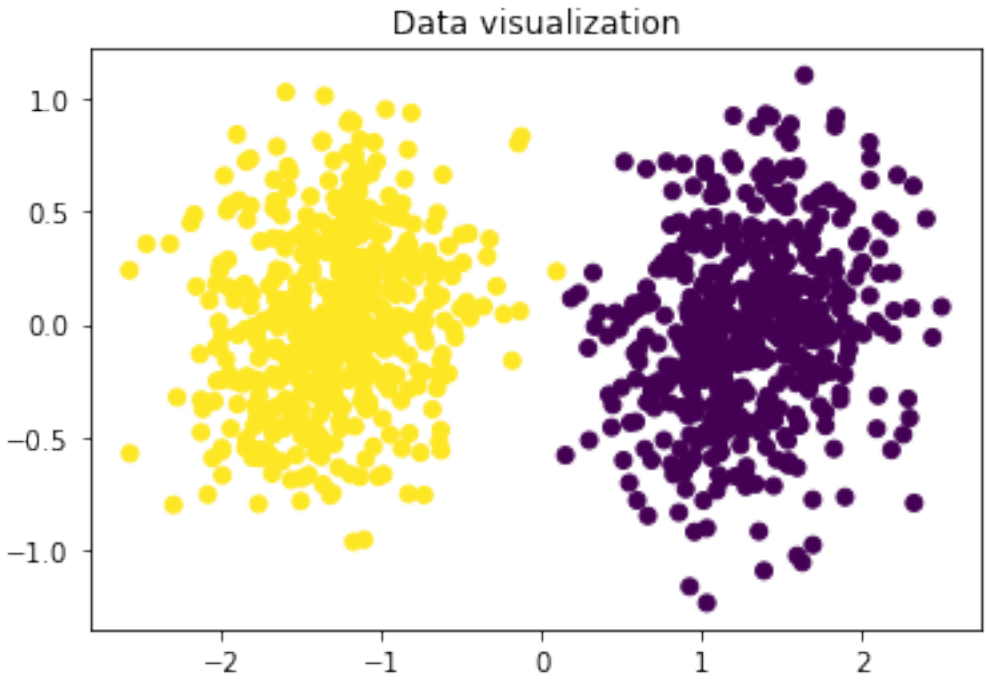


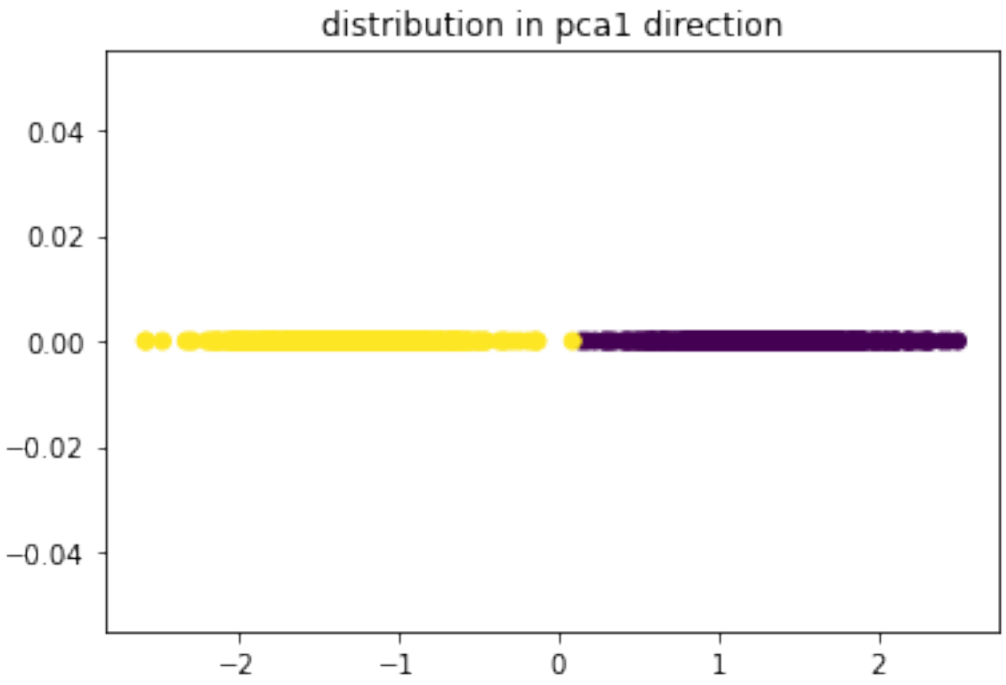
[205]:
```
# PCA

# coverance matrix
cov=data.T @ data


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

trans_data=data @ u

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')
```

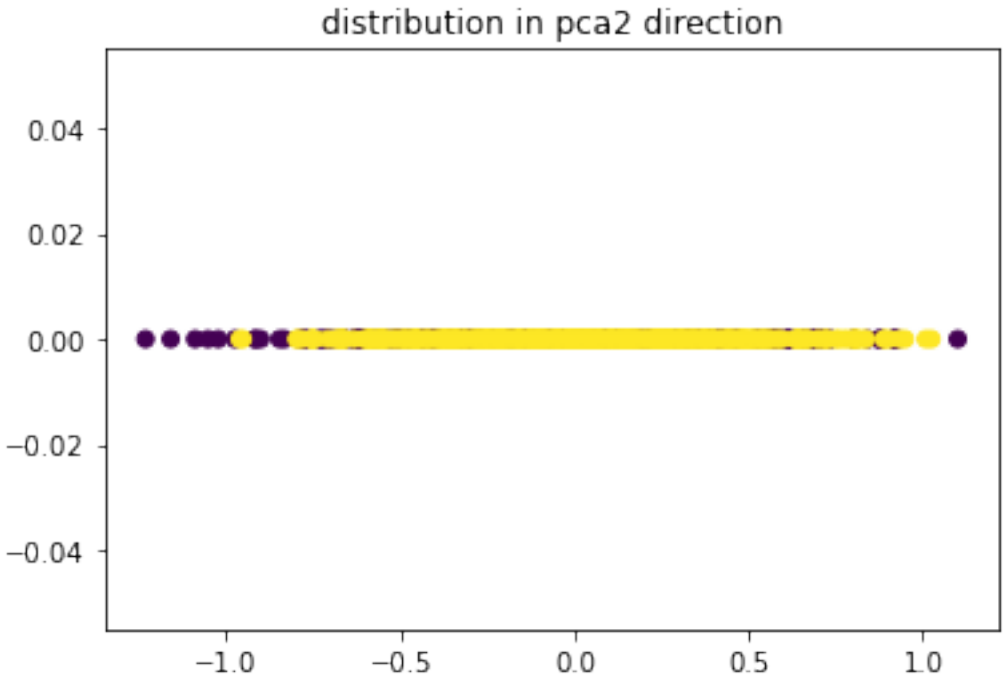
```
plt.figure()
plt.scatter(trans_data[:,1],np.zeros(data.shape[0]),c=label)
plt.title('distribution in pca2 direction')
```

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

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



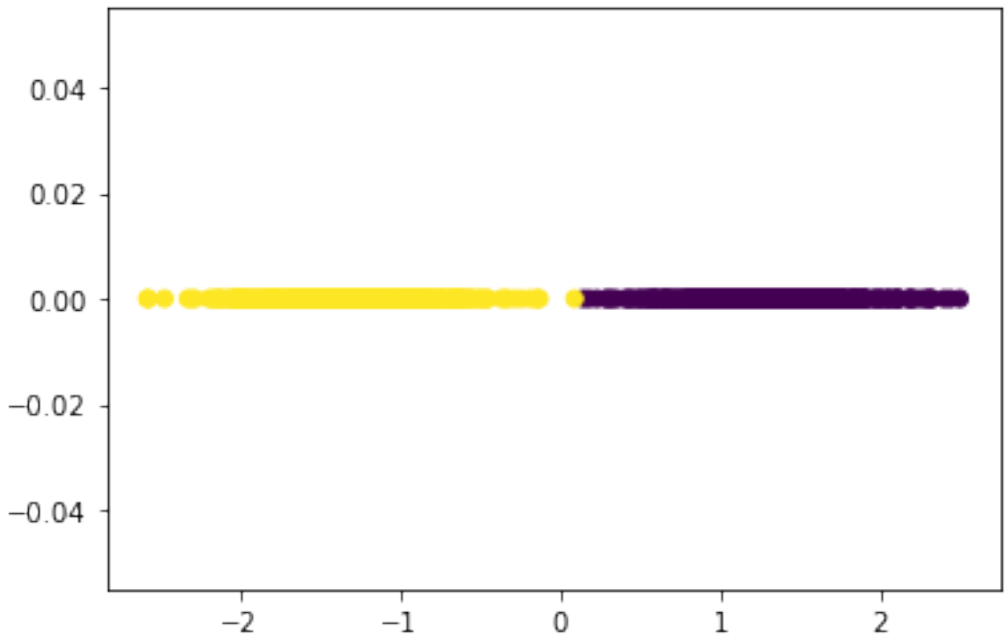

distribution in pca1 direction
0.04
0.02
0.00
−0.02
−0.04
−2
−1
0
1
2


distribution in pca2 direction
0.04
0.02
0.00
−0.02
−0.04
−1.0
−0.5
0.0
0.5
1.0

[206]:
```
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
    u,_,_=np.linalg.svd(cov) # singular value decomposition
    u_req=u[:,:self.retain_dim]
    trans_data=data @ u_req
    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] # can also take 99%
    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
```

[207]:
```
# 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)
```

[207]:
```
<matplotlib.collections.PathCollection at 0x7f7e17e79cc0>
```



[208]:
```
#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

[209]:
```
%pip install idx2numpy
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: idx2numpy in
/home/abhishekj/.local/lib/python3.10/site-packages (1.2.3)
Requirement already satisfied: numpy in /usr/lib64/python3.10/site-packages
(from idx2numpy) (1.22.0)
Requirement already satisfied: six in /usr/lib/python3.10/site-packages (from
idx2numpy) (1.16.0)
WARNING: You are using pip version 21.3.1; however, version 22.3 is
available.
You should consider upgrading via the '/bin/python -m pip install --upgrade pip'
command.
Note: you may need to restart the kernel to use updated packages.
```

[210]:
```
from tensorflow import keras

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

[211]:
```
print(x_train.shape)
print(y_train.shape)

print(x_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

[212]:
```
cls = [2,3,4]

indx2_train = np.where(y_train == 2)
indx3_train = np.where(y_train == 3)
indx4_train = np.where(y_train == 4)

img2_train  = x_train[indx2_train]
img3_train  = x_train[indx3_train]
img4_train  = x_train[indx4_train]

X_train = (img2_train.reshape(28*28, -1).T[:80], img3_train.reshape(28*28, -1).
 ↪T[:80], img4_train.reshape(28*28, -1).T[:80])
X_train = np.concatenate(X_train)
Y_train = np.concatenate((y_train[indx2_train[0][:80]], y_train[␣
 ↪indx3_train[0][:80]], y_train[ indx4_train[0][:80]]))

print(X_train.shape)
print(Y_train.shape)
```

```
indx2_test = np.where(y_test == 2)
indx3_test = np.where(y_test == 3)
indx4_test = np.where(y_test == 4)

img2_test  = x_train[indx2_test]
img3_test  = x_train[indx3_test]
img4_test  = x_train[indx4_test]

X_test = (img2_test.reshape(28*28, -1).T[:20], img3_test.reshape(28*28, -1).T[:
 ↪20], img4_test.reshape(28*28, -1).T[:20])
X_test = np.concatenate(X_test)
Y_test = np.concatenate((y_test[[indx2_test[0][:20]]], y_test[indx3_test[0][:
 ↪20]], y_test[indx4_test[0][:20]]))

print(X_test.shape)
print(Y_test.shape)
```

```
(240, 784)
(240,)
(60, 784)
(60,)
```

```
/tmp/ipykernel_19391/2784805587.py:28: FutureWarning: Using a non-tuple sequence
for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of
`arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different
result.
  Y_test = np.concatenate((y_test[[indx2_test[0][:20]]],
y_test[indx3_test[0][:20]], y_test[indx4_test[0][:20]]))
```

[213]:
```
PCA = pca(data=X_train)
```

[214]:
```
X_train_pca, trans_mat = PCA.pca_comp(X_train)
X_test_pca = PCA.pre_process(X_test) @ trans_mat
```

[215]:
```
from sklearn.neighbors import KNeighborsClassifier
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train_pca, Y_train)
```

[215]:
```
KNeighborsClassifier()
```

[216]:
```
Y_pred = knn.predict(X_test_pca)

from sklearn.metrics import accuracy_score, confusion_matrix

print(f'Logistic testing score is {accuracy_score(Y_test,Y_pred)*100}')
```

```
Logistic testing score is 35.0
```

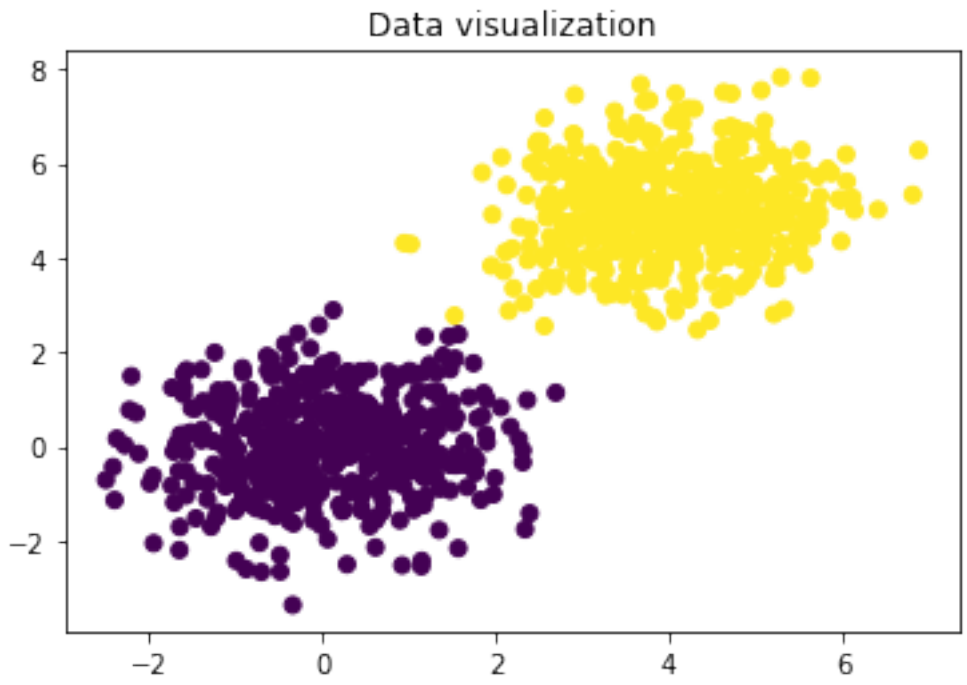
[217]:
```
print(f'Confusion matrix is \n {confusion_matrix(Y_test,Y_pred)}')
```

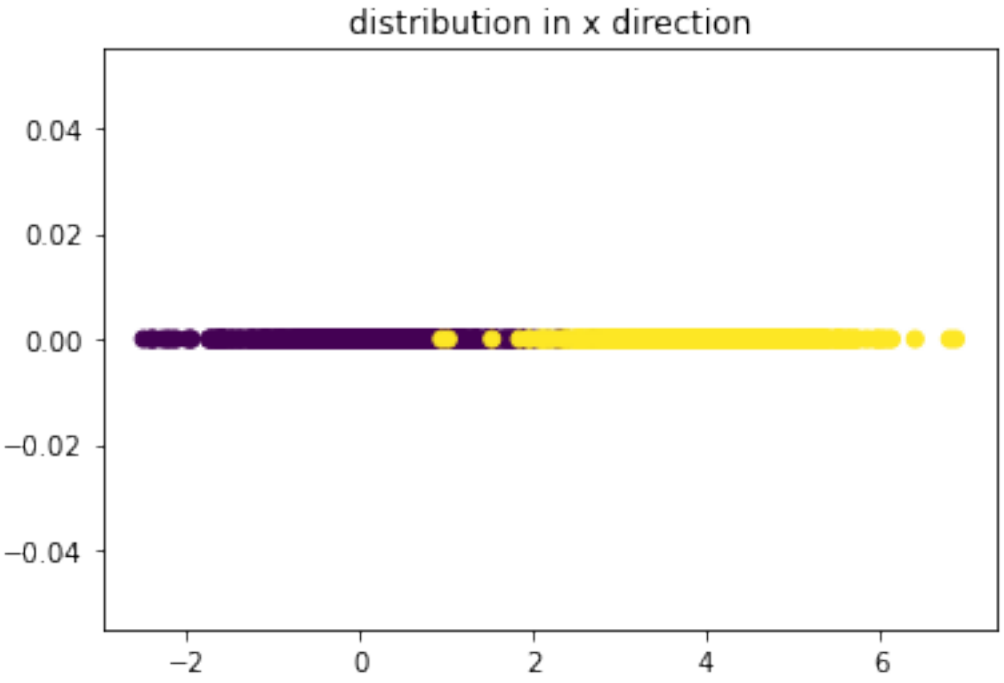
```
Confusion matrix is
 [[ 0  2 18]
 [ 1  1 18]
 [ 0  0 20]]
```

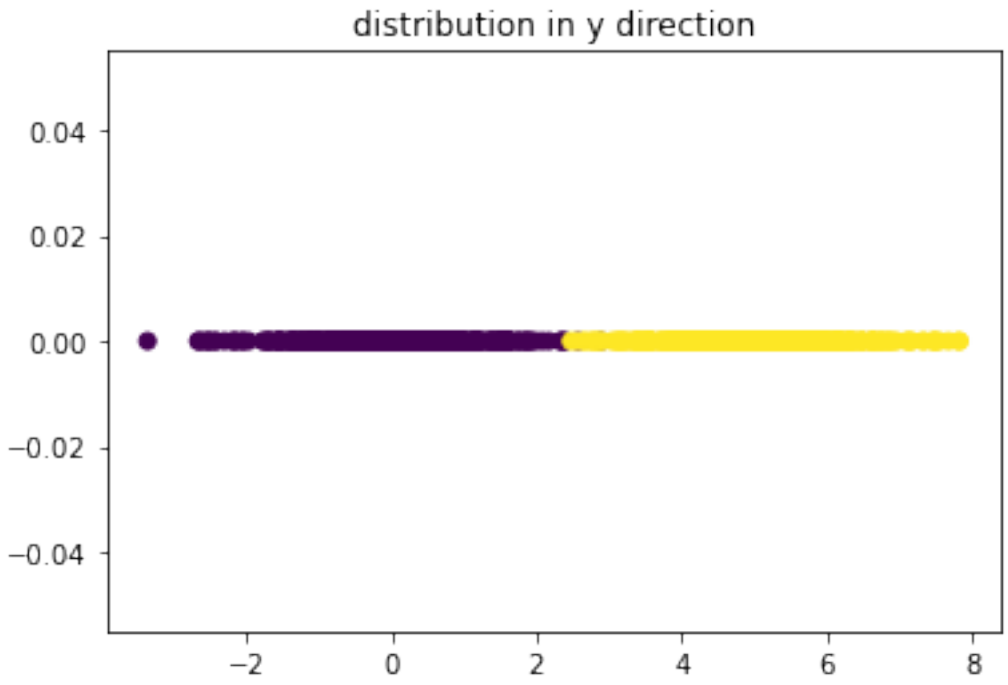
#LDA

[218]:
```
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')
```

[218]:
```
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

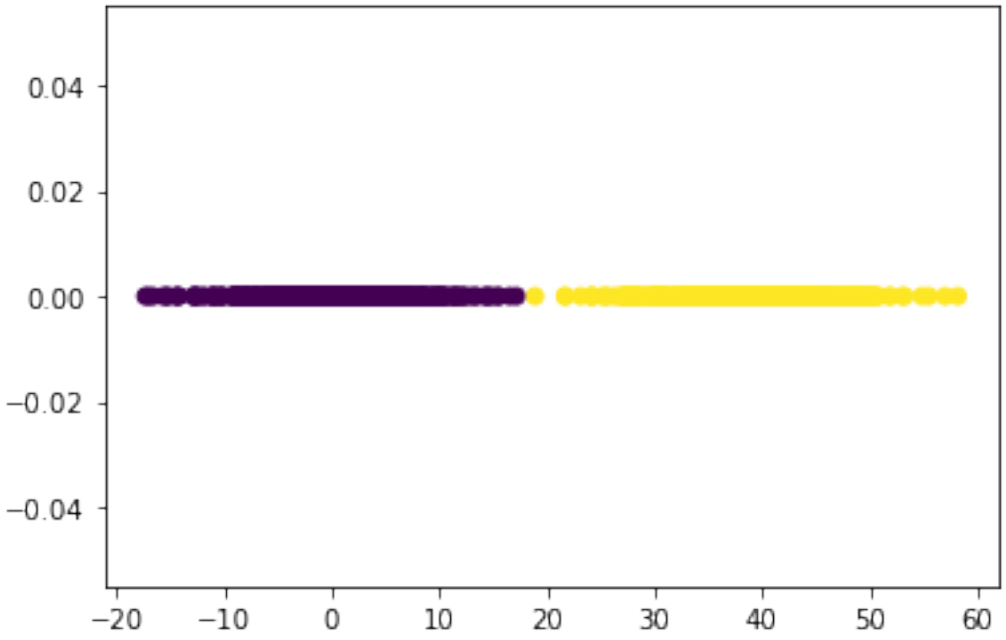


[219]:
```
# perform 2-class and m-class LDA
def LDA(data,label):
  id={}
  data_l={}
  mean_l={}
  cov_l={}
  S_w=np.zeros((data.shape[1],data.shape[1]))
  cls=np.unique(label)

  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]=((data_l[i]-mean_l[i]).T @ (data_l[i]-mean_l[i]))/(data_l[i].
 ↪shape[0]-1)
    S_w=S_w+cov_l[i]


  S_w=S_w/len(data_l)

  if len(data_l)==2:
    S_b=(mean_l[1]-mean_l[0]).T @ (mean_l[1]-mean_l[0])
    w=np.linalg.pinv(S_w) @ (mean_l[1]-mean_l[0]).T
```

```
  else:
    S_t=np.cov(data,rowvar=False)
    S_b=S_t-S_w
    u,_,_=np.linalg.svd(np.linalg.pinv(S_w) @ S_b)
    w=u[:,:len(data_l)-1]

  return w
```

[220]:
```
# after LDA projection

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

[220]: <matplotlib.collections.PathCollection at 0x7f7e17e40460>



[221]:
```
#classification using LDA
#use k-nearest neighbour classifier after dimensionality reduction


from sklearn.neighbors import KNeighborsClassifier

LDA_data= data @ w[:,np.newaxis]
k=5
knn = KNeighborsClassifier(n_neighbors=k)
```

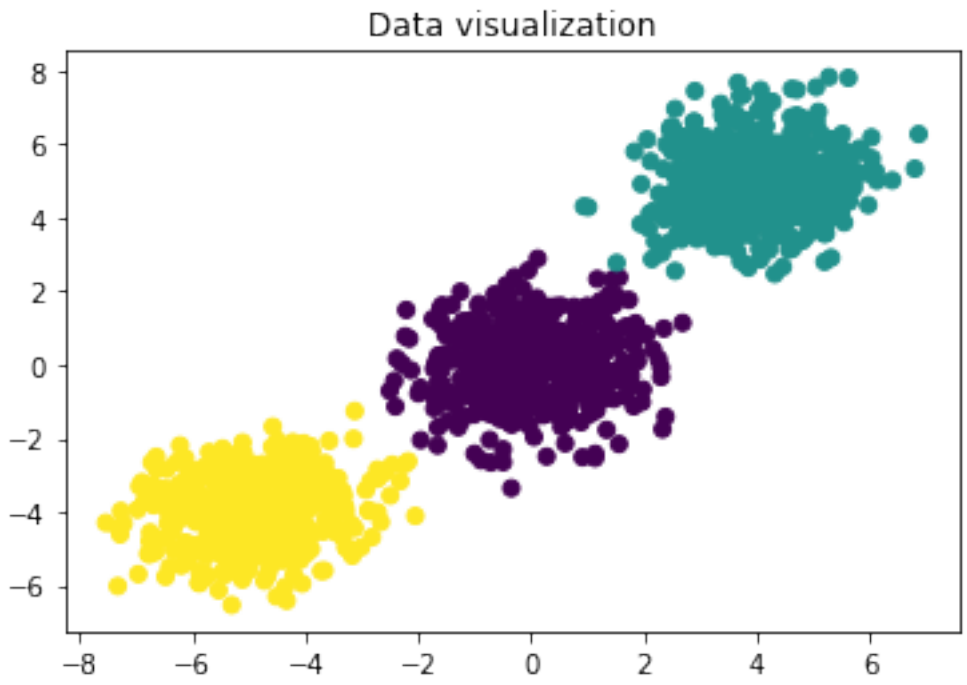
```
knn.fit(LDA_data, label)

print('KNN 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)
data_tst=np.concatenate((data1,data2))
tst_label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

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

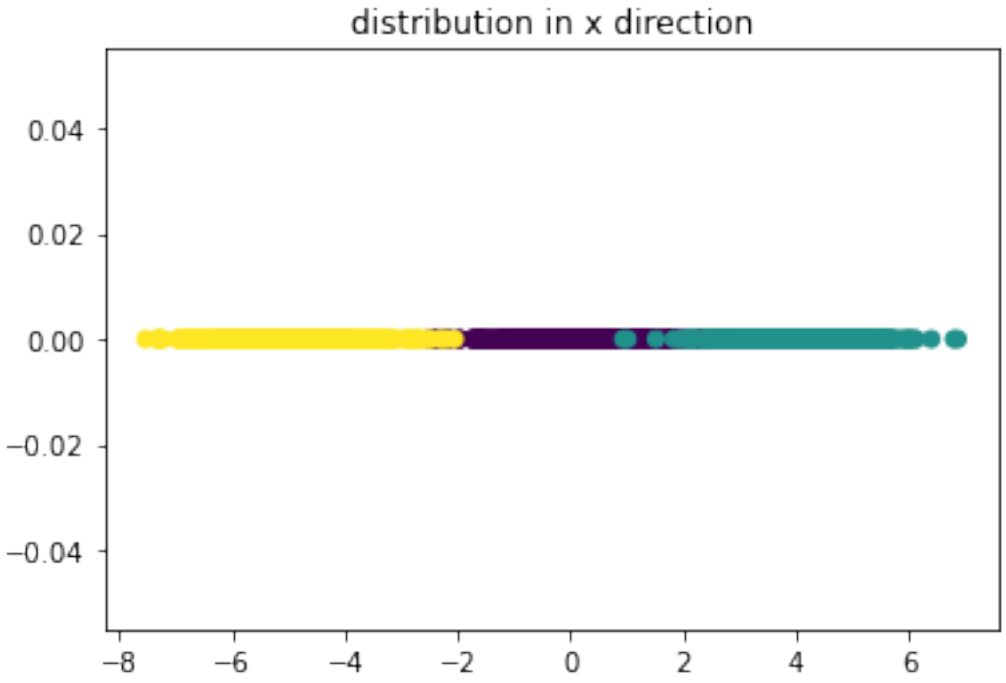
```
KNN Training accuracy = 100.0
KNN Testing accuracy = 100.0
```

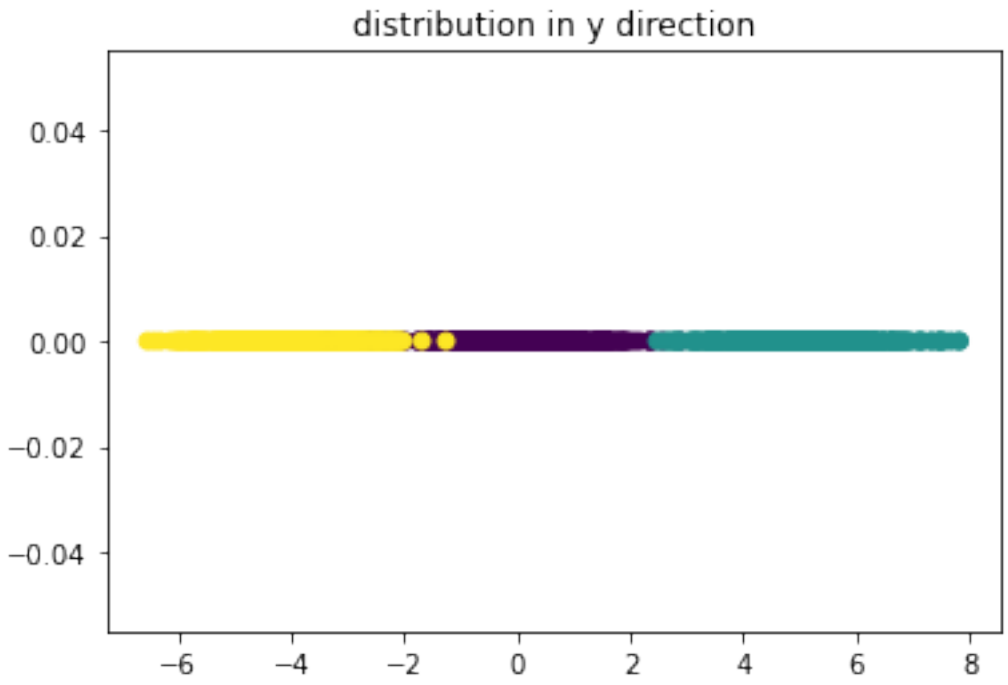
##LDA Multiclass

[222]:
```
import numpy as np
import matplotlib.pyplot as plt

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')
```

[222]:
```
Text(0.5, 1.0, 'distribution in y direction')
```


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

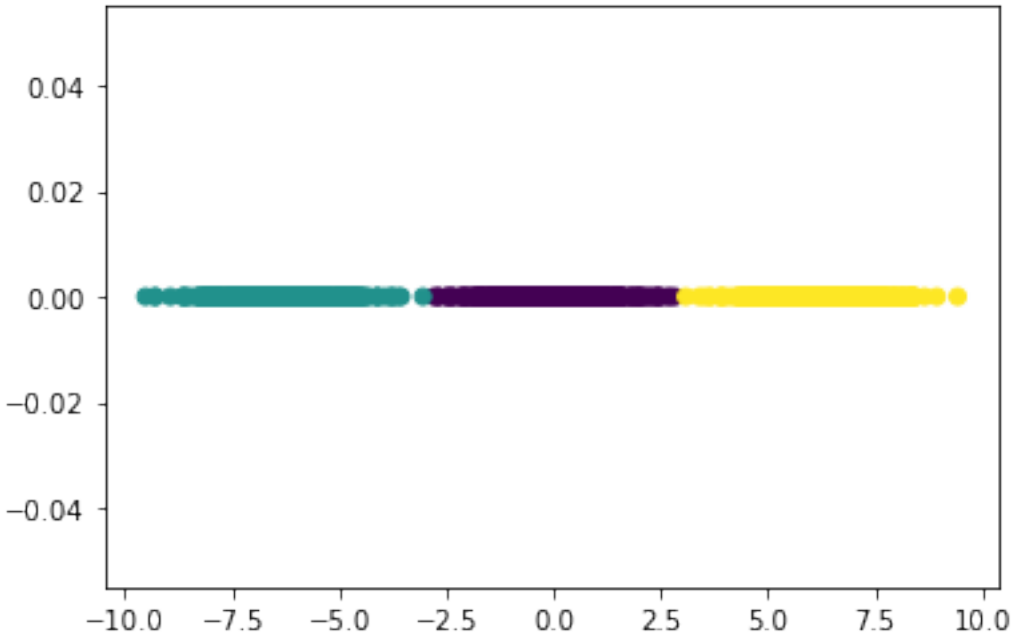

distribution in x direction
0.04
0.02
0.00
−0.02
−0.04
−8
−6
−4
−2
0
2
4
6



[223]:
```
# 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)
```

[223]:
```
<matplotlib.collections.PathCollection at 0x7f7dc8d6a980>
```



[224]:
```
# testing (using KNN)

from sklearn.neighbors import KNeighborsClassifier

LDA_data= data @ w
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(LDA_data, label)

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

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

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

[225]:
```
## Write your code here
# MNIST data
from tensorflow import keras

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()


cls = [2,3,4]

indx2_train = np.where(y_train == 2)
indx3_train = np.where(y_train == 3)
indx4_train = np.where(y_train == 4)


img2_train  = x_train[indx2_train]
img3_train  = x_train[indx3_train]
img4_train  = x_train[indx4_train]


X_train = (img2_train.reshape(28*28, -1).T[:350], img3_train.reshape(28*28, -1).
 ↪T[:350], img4_train.reshape(28*28, -1).T[:350])
DATA = np.concatenate(X_train)
Lable = np.concatenate((y_train[indx2_train[0][:350]], y_train[indx3_train[0][:
 ↪350]], y_train[indx4_train[0][:350]]))
```

[226]:
```
print(DATA.shape)
print(Lable.shape)
print(np.unique(Lable))
```

```
(1050, 784)
(1050,)
[2 3 4]
```

[227]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(DATA, Lable, test_size=0.1,␣
 ↪random_state=0)
```

[228]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(945, 784)
(945,)
```

[229]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA()
X_train = lda.fit_transform(X_train, y_train)
```

```
X_test = lda.transform(X_test)
```

[230]:
```
print(X_train.shape)
```

```
(945, 2)
```

[231]:
```
k=5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

print('KNN Training accuracy =',knn.score(X_train, y_train)*100)
```

```
KNN Training accuracy = 100.0
```

[232]:
```
y_pred = knn.predict(X_test)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy ' + str(accuracy_score(y_test, y_pred)*100))
```

```
[[28  0  0]
 [ 0 44  0]
 [ 0  0 33]]
Accuracy 100.0
```