Q2. Feature Extraction for Dataset B

In [1]:
```
# Python packages to import
import sys
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.manifold import LocallyLinearEmbedding
from matplotlib import offsetbox
from sklearn.model_selection import StratifiedKFold
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

In [2]:
```
# Display all the Columns in the dataset for better analysis
pd.set_option('display.max_columns',None)
```

In [3]:
```
# import DataB.csv
dataB = pd.read_csv('~/Documents/Uwaterloo_Study_Docs/657A_ECE/Assignments/ece_
657a_assignments/Assignment_1/Datasets/DataB.csv')
```

In [4]:
```
print("DataB: \n", dataB.head(1))
```

```
DataB:
   Unnamed: 0  fea.1  fea.2  fea.3  fea.4  fea.5  fea.6  fea.7  fea.8  fea.9  \
0           1      4      4      3      0      0      4      2      1      4   

   fea.10  fea.11  fea.12  fea.13  fea.14  fea.15  fea.16  fea.17  fea.18  \
0       1       0       4       1       4       0       3       1       4   

   fea.19  fea.20  fea.21  fea.22  fea.23  fea.24  fea.25  fea.26  fea.27  \
0       4       3       4       4       3       1       2       1       1   

   fea.28  fea.29  fea.30  fea.31  fea.32  fea.33  fea.34  fea.35  fea.36  \
0       3       4       1       3       2       0       2       0       2   

   fea.37  fea.38  fea.39  fea.40  fea.41  fea.42  fea.43  fea.44  fea.45  \
0       3       3       4       3       4       4       5       4       5   

   fea.46  fea.47  fea.48  fea.49  fea.50  fea.51  fea.52  fea.53  fea.54  \
0       1       4       3       1       3       0       4       5       4   

   fea.55  fea.56  fea.57  fea.58  fea.59  fea.60  fea.61  fea.62  fea.63  \
0       4       3       4       3       4       4       4       3       2   

   fea.64  fea.65  fea.66  fea.67  fea.68  fea.69  fea.70  fea.71  fea.72  \
0       5       1       0       3       4       2       2       3       2   

   fea.73  fea.74  fea.75  fea.76  fea.77  fea.78  fea.79  fea.80  fea.81  \
0       4       1       5       5       2       3       2       3       0   

   fea.82  fea.83  fea.84  fea.85  fea.86  fea.87  fea.88  fea.89  fea.90  \
0       0       2       3       2       4       2       0       2       4   

   fea.91  fea.92  fea.93  fea.94  fea.95  fea.96  fea.97  fea.98  fea.99  \
0       3       0       4       3       3       5       1       4       2   

   fea.100  fea.101  fea.102  fea.103  fea.104  fea.105  fea.106  fea.107  \
0        2        4        0        1        1        2        2        5   

   fea.108  fea.109  fea.110  fea.111  fea.112  fea.113  fea.114  fea.115  \
0        5        2        3        3        5        4        3        4   

   fea.116  fea.117  fea.118  fea.119  fea.120  fea.121  fea.122  fea.123  \
0        3        1        4        3        4        2        5        1   

   fea.124  fea.125  fea.126  fea.127  fea.128  fea.129  fea.130  fea.131  \
0        1        2        2        1       52      162      255      161   

   fea.132  fea.133  fea.134  fea.135  fea.136  fea.137  fea.138  fea.139  \
0       51        4        5        4        3        0        4        4   

   fea.140  fea.141  fea.142  fea.143  fea.144  fea.145  fea.146  fea.147  \
0        3        2        3        1        2        4        3        3   

   fea.148  fea.149  fea.150  fea.151  fea.152  fea.153  fea.154  fea.155  \
0        4        2        4        3        5        5        3       52   

   fea.156  fea.157  fea.158  fea.159  fea.160  fea.161  fea.162  fea.163  \
0      241      253      254      254      242        1        0        4   

   fea.164  fea.165  fea.166  fea.167  fea.168  fea.169  fea.170  fea.171  \
0        2        3        4        3        4        5        2        4   

   fea.172  fea.173  fea.174  fea.175  fea.176  fea.177  fea.178  fea.179  \
0        3        3        3        1        3        5        3        2   
```

As we can see from the above data is that there is a Unnamed column in the dataset that need to be removed. The dimension of the Dataset is (2066, 786)

In [5]:
```
dataB_updated = dataB.drop(['Unnamed: 0'], axis=1)
print("DataB: \n", dataB_updated.head(1))
```

```
DataB:
   fea.1  fea.2  fea.3  fea.4  fea.5  fea.6  fea.7  fea.8  fea.9  fea.10  \
0      4      4      3      0      0      4      2      1      4       1

   fea.11  fea.12  fea.13  fea.14  fea.15  fea.16  fea.17  fea.18  fea.19  \
0       0       4       1       4       0       3       1       4       4

   fea.20  fea.21  fea.22  fea.23  fea.24  fea.25  fea.26  fea.27  fea.28  \
0       3       4       4       3       1       2       1       1       3

   fea.29  fea.30  fea.31  fea.32  fea.33  fea.34  fea.35  fea.36  fea.37  \
0       4       1       3       2       0       2       0       2       3

   fea.38  fea.39  fea.40  fea.41  fea.42  fea.43  fea.44  fea.45  fea.46  \
0       3       4       3       4       4       5       4       5       1

   fea.47  fea.48  fea.49  fea.50  fea.51  fea.52  fea.53  fea.54  fea.55  \
0       4       3       1       3       0       4       5       4       4

   fea.56  fea.57  fea.58  fea.59  fea.60  fea.61  fea.62  fea.63  fea.64  \
0       3       4       3       4       4       4       3       2       5

   fea.65  fea.66  fea.67  fea.68  fea.69  fea.70  fea.71  fea.72  fea.73  \
0       1       0       3       4       2       2       3       2       4

   fea.74  fea.75  fea.76  fea.77  fea.78  fea.79  fea.80  fea.81  fea.82  \
0       1       5       5       2       3       2       3       0       0

   fea.83  fea.84  fea.85  fea.86  fea.87  fea.88  fea.89  fea.90  fea.91  \
0       2       3       2       4       2       0       2       4       3

   fea.92  fea.93  fea.94  fea.95  fea.96  fea.97  fea.98  fea.99  fea.100  \
0       0       4       3       3       5       1       4       2        2

   fea.101  fea.102  fea.103  fea.104  fea.105  fea.106  fea.107  fea.108  \
0        4        0        1        1        2        2        5        5

   fea.109  fea.110  fea.111  fea.112  fea.113  fea.114  fea.115  fea.116  \
0        2        3        3        5        4        3        4        3

   fea.117  fea.118  fea.119  fea.120  fea.121  fea.122  fea.123  fea.124  \
0        1        4        3        4        2        5        1        1

   fea.125  fea.126  fea.127  fea.128  fea.129  fea.130  fea.131  fea.132  \
0        2        2        1       52      162      255      161       51

   fea.133  fea.134  fea.135  fea.136  fea.137  fea.138  fea.139  fea.140  \
0        4        5        4        3        0        4        4        3

   fea.141  fea.142  fea.143  fea.144  fea.145  fea.146  fea.147  fea.148  \
0        2        3        1        2        4        3        3        4

   fea.149  fea.150  fea.151  fea.152  fea.153  fea.154  fea.155  fea.156  \
0        2        4        3        5        5        3       52      241

   fea.157  fea.158  fea.159  fea.160  fea.161  fea.162  fea.163  fea.164  \
0      253      254      254      242        1        0        4        2

   fea.165  fea.166  fea.167  fea.168  fea.169  fea.170  fea.171  fea.172  \
0        3        4        3        4        5        2        4        3

   fea.173  fea.174  fea.175  fea.176  fea.177  fea.178  fea.179  fea.180  \
0        3        3        1        3        5        3        2        3
```

After removing the 'Unnamed' Column the dimension becomes (2066, 785) as evident from above

Missing Value Check in the Dataset

In [6]:
```
dataB_updated.isna().sum()
```

Out[6]:
```
fea.1      0
fea.2      0
fea.3      0
fea.4      0
fea.5      0
          ..
fea.781    0
fea.782    0
fea.783    0
fea.784    0
gnd        0
Length: 785, dtype: int64
```

In [7]:
```
dataB_updated.isna().sum().sum()
```

Out[7]:
```
0
```

As we can see from the above value that there are no missing values in the complete dataset

In [8]:
```
dataB_updated['gnd'].unique()
```

Out[8]:
```
array([0, 1, 2, 3, 4])
```

Here we can see that there are 5 different class labels in the 'gnd' column

In [9]:
```
dataB_X = dataB_updated.drop(['gnd'], axis=1)
dataB_y = dataB_updated['gnd']
print("Features: \n",dataB_X.head(5))
print("Targets: \n", dataB_y.head(5))
```

```
       0
2      3      222      254      255      187        1        2        1
3      5       37      254      251      213       49        5        2
4      4        4        5      163      255      255      117        2

   fea.445  fea.446  fea.447  fea.448  fea.449  fea.450  fea.451  fea.452  \
0        2        1        4        0        4        5        4        3
1        5        1        1        2        2        5        4        2
2        2        2        0        1        2        2        2        4
3        5        2        2        2        1        2        2        3
4        2        1        2        3        4        1        2        1

   fea.453  fea.454  fea.455  fea.456  fea.457  fea.458  fea.459  fea.460  \
0        3        5       85      255      230       25        1        4
1        4      191      253      252      252        3        2        1
2        2        5        2        1       66      221      254      251
3        1        5        1      195      252      254      255      130
4        4      122      255      255      154        4        4        1

   fea.461  fea.462  fea.463  fea.464  fea.465  fea.466  fea.467  fea.468  \
0        2        5        5        5        1        4       11      140
1        3        3        1        4        1        5        2      101
2      219        3        0        1       89      238      252      251
3        8        1        4        3        4       39      252      254
4        1        2        4        2        5        5        1      157

   fea.469  fea.470  fea.471  fea.472  fea.473  fea.474  fea.475  fea.476  \
0      253      190       14        0        2        1        1        1
1      255      254       32        4        1        3        4        2
2       80        5        3        0        4        1        4        3
3       56        2        2        4        0        4        2        4
4      255      255      119        4        2        5        4        4

   fea.477  fea.478  fea.479  fea.480  fea.481  fea.482  fea.483  fea.484  \
0        1        1        5        3        3        2       89      255
1        2        0        4        4        2      191      254      252
2        5        2        1        1        5        2        2        4
3        2        1        1        1        3        1        2      117
4        1        3        3        0        2      120      255      254

   fea.485  fea.486  fea.487  fea.488  fea.489  fea.490  fea.491  fea.492  \
0      227        4        3        2        3        1        1        1
1      115        2        1        4        1        4        3        2
2      113      251      255      253      218        1        0       15
3      252      254      255      190       23        2        5       33
4      154        3        2        5        2        4        2        4

   fea.493  fea.494  fea.495  fea.496  fea.497  fea.498  fea.499  fea.500  \
0        1        9      131      253      226       72        5        3
1        0        1       42      238      252      220       24        3
2      161      253      255      100        2        1        1        2
3      113      130      254      178      105        0        4        2
4        2        4        5      157      254      255      118        3

   fea.501  fea.502  fea.503  fea.504  fea.505  fea.506  fea.507  fea.508  \
0        1        1        0        4        4        4        1        0
1        5        2        3        2        2        1        4        5
2        4        2        1        0        5        1        3        4
3        2        3        2        4        0        3        0        1
4        2        1        5        4        0        4        5        3

   fea.509  fea.510  fea.511  fea.512  fea.513  fea.514  fea.515  fea.516  \
0        2        4       89      254      149        4        2        2
```

Separting the data based on the features and the target('gnd') column where all the features are taken into 'dataB_X' variable where all the target values are taken in 'dataB_y' variable

Q2.1 EigenVectors and EigenValues Calculation of the Dataset

In order to caluculate the EigenValues and EigenVectors of the dataset first step is construct the Covariance Matrix.

Standardisation of the Dataset

In [10]:
```
standarscaler = StandardScaler()
standarscaler.fit(dataB_X)
dataB_X_std = standarscaler.transform(dataB_X)
dataB_y_values = dataB_y.values
print("dataB_X_std: \n", dataB_X_std)
```

```
dataB_X_std:
 [[ 1.01007711  0.96678184  0.35959365 ... -1.03428476  1.04733254
   1.64964331]
 [ 1.68717617 -1.02992387  1.02648816 ...  0.30710401  1.70203421
   0.98463588]
 [-1.02122007  0.30121327 -1.64108987 ... -1.03428476 -0.2620708
   0.98463588]
 ...
 [-0.34412101  0.30121327 -0.30730086 ... -1.03428476 -0.91677248
   0.31962845]
 [ 1.68717617 -0.3643553   1.02648816 ...  0.9777984   0.39263087
   0.98463588]
 [ 0.33297805  0.30121327 -0.97419536 ... -1.03428476  0.39263087
  -1.01038641]]
```

The Complete dataB here is standardised in order to calculate the EigenValues and EigenVectors of the Dataset

Step.1: Covariance Matrix Calculation for the Dataset

In [11]:
```
dataB_CovMat = np.cov(dataB_X_std.T)
EigenValues, EigenVectors = np.linalg.eigh(dataB_CovMat)
```

In [12]:
```
print("\n EigenValues.shape: ", EigenValues.shape)
print("\n EigenValues: \n", EigenValues)
```

```
EigenValues.shape:  (784,)

EigenValues:
[7.10466561e-03 7.43794649e-03 7.71149989e-03 8.14488845e-03
8.34035124e-03 8.54048562e-03 8.82056525e-03 8.93192486e-03
9.09831448e-03 9.31307758e-03 9.58849106e-03 9.69892410e-03
1.00322024e-02 1.00736363e-02 1.01457638e-02 1.06412158e-02
1.08129008e-02 1.09433143e-02 1.11619155e-02 1.12373849e-02
1.13628067e-02 1.15190236e-02 1.16052660e-02 1.18335915e-02
1.20787835e-02 1.22316057e-02 1.24225729e-02 1.26167660e-02
1.27207821e-02 1.28369117e-02 1.29080794e-02 1.30555115e-02
1.33046542e-02 1.34022076e-02 1.35174614e-02 1.36723998e-02
1.39527048e-02 1.41431281e-02 1.42711101e-02 1.44948392e-02
1.45343097e-02 1.47397414e-02 1.48912318e-02 1.50939880e-02
1.53408727e-02 1.54806987e-02 1.56594762e-02 1.57584247e-02
1.59376580e-02 1.60528103e-02 1.63249513e-02 1.63564608e-02
1.66122410e-02 1.67151921e-02 1.67939333e-02 1.71649898e-02
1.72645039e-02 1.73777548e-02 1.76953947e-02 1.78165760e-02
1.79045769e-02 1.79698710e-02 1.83372227e-02 1.84729470e-02
1.85131345e-02 1.88559855e-02 1.89298080e-02 1.90708791e-02
1.91693235e-02 1.93290883e-02 1.94912911e-02 1.97981614e-02
2.00692306e-02 2.01420891e-02 2.02710347e-02 2.05913791e-02
2.06974275e-02 2.10086462e-02 2.11151985e-02 2.12710252e-02
2.16129301e-02 2.16714068e-02 2.20473726e-02 2.24326849e-02
2.25304264e-02 2.25853884e-02 2.28175385e-02 2.30548370e-02
2.31125457e-02 2.34748437e-02 2.36595171e-02 2.38470137e-02
2.39878391e-02 2.41307523e-02 2.46732490e-02 2.47746319e-02
2.49695989e-02 2.53424289e-02 2.54467493e-02 2.56857193e-02
2.59201790e-02 2.62203192e-02 2.63324902e-02 2.65527684e-02
2.68883330e-02 2.69391617e-02 2.72990830e-02 2.74932374e-02
2.75616741e-02 2.77680664e-02 2.79547100e-02 2.81321689e-02
2.82134229e-02 2.85458236e-02 2.88452743e-02 2.90108159e-02
2.95315621e-02 2.96221792e-02 2.96749209e-02 3.01982378e-02
3.04923418e-02 3.06232530e-02 3.07603437e-02 3.11915807e-02
3.14690733e-02 3.18139552e-02 3.21068628e-02 3.23264251e-02
3.25170230e-02 3.26197513e-02 3.30842952e-02 3.32442373e-02
3.35890936e-02 3.36265712e-02 3.38517135e-02 3.42047572e-02
3.43939298e-02 3.46734865e-02 3.51586346e-02 3.53589967e-02
3.57157819e-02 3.58007915e-02 3.60109143e-02 3.62540599e-02
3.67223394e-02 3.72281748e-02 3.73445686e-02 3.75380764e-02
3.77157303e-02 3.79922830e-02 3.82158133e-02 3.85149257e-02
3.95369551e-02 3.99198276e-02 4.00258084e-02 4.05162084e-02
4.05862111e-02 4.06782667e-02 4.12948752e-02 4.17011860e-02
4.19048229e-02 4.20642971e-02 4.26163913e-02 4.28078357e-02
4.29655689e-02 4.33645804e-02 4.37841578e-02 4.40169979e-02
4.45812991e-02 4.47860905e-02 4.51517821e-02 4.55270415e-02
4.55698242e-02 4.62093948e-02 4.66088195e-02 4.67459736e-02
4.77925592e-02 4.79605073e-02 4.82230419e-02 4.83905905e-02
4.87384503e-02 4.91383694e-02 4.93420812e-02 4.98603957e-02
5.02855429e-02 5.05584286e-02 5.09995363e-02 5.11206748e-02
5.17381102e-02 5.19094853e-02 5.27466768e-02 5.33580911e-02
5.35166013e-02 5.40339091e-02 5.42798615e-02 5.47622827e-02
5.52925538e-02 5.54994483e-02 5.60466599e-02 5.62265295e-02
5.68412903e-02 5.76518061e-02 5.77712617e-02 5.81079453e-02
5.84101870e-02 5.89186821e-02 5.91766347e-02 5.97392837e-02
6.04153253e-02 6.07133938e-02 6.10446346e-02 6.12706423e-02
6.21808584e-02 6.24854806e-02 6.29538177e-02 6.35157398e-02
6.38173827e-02 6.40103113e-02 6.49313582e-02 6.52307311e-02
6.54685573e-02 6.64154703e-02 6.65768226e-02 6.78279910e-02
6.84815711e-02 6.86499091e-02 6.91524368e-02 6.96339653e-02
7.01675276e-02 7.10129231e-02 7.14166133e-02 7.20888012e-02
7.30031184e-02 7.30521527e-02 7.34738511e-02 7.39776374e-02
7.44115386e-02 7.52486183e-02 7.56396413e-02 7.60884809e-02
```

As we can see that the dimension of EigenValues of the Dataset B is (784,1) in addition to all the EigenValues of the Dataset.

In [13]:
```
print("\n EigenVectors.shape: ", EigenVectors.shape)
print("\n EigenVectors: \n", EigenVectors)
```

```
 EigenVectors.shape:  (784, 784)

 EigenVectors:
 [[ 0.00178531 -0.00069225  0.00294198 ... -0.00037529  0.00493308
   0.00197863]
 [ 0.00479506  0.0040447   0.00188671 ...  0.00258725 -0.00640373
   0.00151307]
 [-0.00816424  0.00090736  0.00838343 ... -0.00372451 -0.00156563
  -0.00049178]
 ...
 [-0.00498406 -0.00319481  0.00055758 ... -0.00335936  0.00300533
  -0.0001125 ]
 [-0.00351318 -0.00012294  0.00186681 ...  0.00553066  0.00947149
  -0.00132315]
 [ 0.00323845 -0.00054035  0.0001313  ...  0.00624184  0.00287621
   0.00591181]]
```

The dimension of the EigneVectors is (784, 784) and the list of all the EigenVectors of the Dataset is shown.

Note: The EigenValues define the magnitude of the EigenVectors in terms of its variance.

In [14]:
```
# Calculate the pairwise from top EigenVales and EigenVectors of the Dataset in
order to find the variances in the higher dimensional space
EigenPairs = [(np.abs(EigenValues[index]), EigenVectors[:, index]) for index in
range(len(EigenValues))]
EigenPairs.sort(key = lambda k: k[0], reverse=True)
```

Step.2 Projection Matrix Calculation with Principle Components(PC)-1 and Principle Component(PC)-2

In [15]:
```
# Projection Matrix is
ProMatrix = np.hstack((EigenPairs[0][1][:, np.newaxis], EigenPairs[1][1][:, np.
newaxis]))
print(" Projection Matrix: \n", ProMatrix)
```

```
Projection Matrix:
 [[ 0.00197863  0.00493308]
 [ 0.00151307 -0.00640373]
 [-0.00049178 -0.00156563]
 ...
 [-0.0001125   0.00300533]
 [-0.00132315  0.00947149]
 [ 0.00591181  0.00287621]]
```

The Projection matrix consists of PC-1 and PC-2 which has the higest variance in the dataset.

Step.3 Transformation of Dataset using Projection Matrix on to the PC-1 and PC-2

In [16]:
```
dataB_X_std_pro = dataB_X_std.dot(ProMatrix)
print("\n Dimension of the Projected DatasetB: ", dataB_X_std_pro.shape)
print("\n Projection of DatasetB using PC-1 and PC-2 Projetion Matrix: \n",data
B_X_std_pro)
```

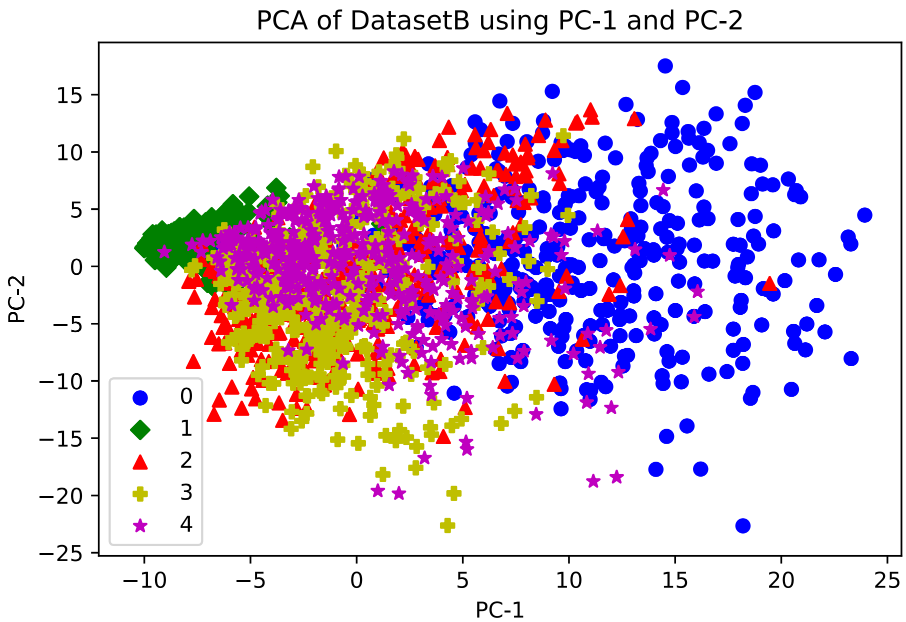
```
 Dimension of the Projected DatasetB:  (2066, 2)

 Projection of DatasetB using PC-1 and PC-2 Projetion Matrix:
 [[ 9.97069222  6.18172201]
 [11.41599978  6.94158705]
 [ 3.69011918  4.69309729]
 ...
 [-0.34942153  0.93368106]
 [-3.11526327  2.09047425]
 [-5.64409375 -0.24616663]]
```

The DatasetB is projected using the PC-1 and PC-2 as shown above

Q2.2 Plot of the Dataset B using the Projection Matrix which consists of PC-1 and PC-2

In [ ]:

In [17]:
```
colors = ['b', 'g', 'r', 'y', 'm']
markers = ['o', 'D', '^', 'P', '*']

for index, color, marker in zip(np.unique(dataB_y), colors, markers):
    plt.scatter(dataB_X_std_pro[dataB_y_values == index, 0], dataB_X_std_pro[da
taB_y_values ==index, 1], c=color, label= index, marker= marker)

plt.xlabel('PC-1')
plt.ylabel('PC-2')
plt.legend(loc='best')
plt.tight_layout()
plt.title("PCA of DatasetB using PC-1 and PC-2")
plt.show()
```



PCA is performed on the DatasetB using the PC-1 and PC-2. In the above graph each class label as shown in the graph is plotted with a different color. It is evident form the above graph that the DatasetB cann't be seperated Linearly.

Q2.3 Projection of the DatasetB using PC-5 and PC-6 Projection Matrix

In [18]:
```
# Projection Matrix is
ProMatrix_PC56 = np.hstack((EigenPairs[4][1][:, np.newaxis], EigenPairs
[5][1][:, np.newaxis]))
print(" Projection Matrix: \n", ProMatrix_PC56)
```

```
Projection Matrix:
 [[-0.0032507   0.00096031]
 [ 0.00376721 -0.0065041 ]
 [ 0.00138427  0.00086984]
 ...
 [ 0.00138768  0.01078037]
 [-0.00260272 -0.00577003]
 [-0.00052576 -0.00856238]]
```

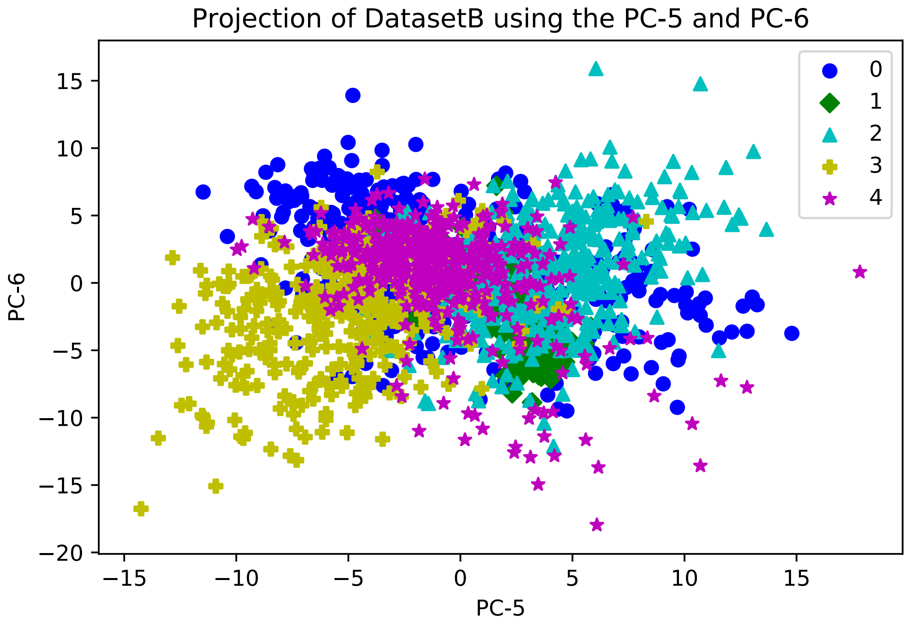
Transformation of Dataset using Projection Matrix consisting of PC-5 and PC-6

In [19]:
```
dataB_X_std_pro_PC56 = dataB_X_std.dot(ProMatrix_PC56)
print("\n Dimension of the Projected DatasetB: ", dataB_X_std_pro_PC56.shape)
print("\n Projection of DatasetB using PC-5 and PC-6 Projetion Matrix: \n",data
B_X_std_pro_PC56)
```

```
 Dimension of the Projected DatasetB:  (2066, 2)

 Projection of DatasetB using PC-5 and PC-6 Projetion Matrix:
 [[-2.77786998  2.84476143]
 [-1.84494034  0.14643064]
 [-6.81180551  3.22610667]
 ...
 [-2.81443825  2.82286727]
 [-0.03630758  2.5864369 ]
 [-2.47709658  3.19848918]]
```

Here the datasetB in the original dimension has been projected using the PC-5 and PC-6 as shown above

In [20]:
```
colors = ['b', 'g', 'c', 'y', 'm']
markers = ['o', 'D', '^', 'P', '*']

for index, color, marker in zip(np.unique(dataB_y), colors, markers):
    plt.scatter(dataB_X_std_pro_PC56[dataB_y_values == index, 0], dataB_X_std_p
ro_PC56[dataB_y_values ==index, 1], c=color, label= index, marker= marker)

plt.xlabel('PC-5')
plt.ylabel('PC-6')
plt.legend(loc='best')
plt.tight_layout()
plt.title("Projection of DatasetB using the PC-5 and PC-6")
plt.show()
```



Analysis and Comparision between PC-1, PC-2 and PC-5, PC-6 plot:

In the plot of PC-1 vs PC-2, the x-limit and the y-limit for PC-1 and PC-2 respectively stretches from -10 to 25 and from the plot it is clear that the dataset in this cannot be seperated Linearly. In addition, most of the data is projected on to the range of -10 to 5 on the x-axis(along PC-1) and -10 to 15 on the y-axis(along PC-2). The majority of the label '0' data is projected further apart form the other labels with variance in the label higher than other labels. The Labels '1', '2', '3' and '4' overlaps one another in the 2D plot with variance for label '1' being quite low, it is the lowest among other labels. Majority of the data from label '2', '3' and '4' are projected on to one another which make it difficult to identify visually.

From the plot of PC-5, PC-6 the majority data is projected between -15 to 15 along the x-axis(PC-5) and -15 to 10 along the y-axis(PC-6). It is more centered towards 0 which different from that of the plot by PC-1 and PC-2. In addition the variance in label '0' is still higher than other labels. The label '1' is not visually clear in the 2D space as the other labels are projected along with it. Here again, the data cann't be seperated linearly.

Q2.4 Naive Bayes Classifier

1. Naive Bayes on first 2 Components (PC-1 and PC-2)

In [21]:
```
# Split the Datset in order to use the Naive Bayes Classifier

X_train_q2, X_test_q2, y_train_q2, y_test_q2 = train_test_split(dataB_X, dataB_
y, test_size=0.4, random_state=42)

ss = StandardScaler()

ss.fit(X_train_q2)
X_train_std_q2 = ss.transform(X_train_q2)

ss.fit(X_test_q2)
X_test_std_q2 = ss.transform(X_test_q2)
y_train_values_q2 = y_train_q2.values
y_test_values_q2 = y_test_q2.values
```

In [22]:
```
pca_q2_2PC = PCA(n_components=2)
X_train_std_q2_pca_2PC = pca_q2_2PC.fit_transform(dataB_X_std)
# X_test_std_q2_pca_2PC = pca_q2_2PC.fit_transform(X_test_std_q2)
gnb_q2_2PC = GaussianNB()
gnb_q2_2PC.fit(X_train_std_q2_pca_2PC, dataB_y_values)
y_pred_q2_2PC = gnb_q2_2PC.predict(X_train_std_q2_pca_2PC)
accuracy_2PC = accuracy_score(dataB_y_values, y_pred_q2_2PC)
print("\n Accuracy Score for Test Set using 2PCs: {0:.3f}%".format( accuracy_2P
C*100))
```

```
 Accuracy Score for Test Set using 2PCs: 58.422%
```

The Accuracy Score for the Test set with 30% after using 2 PCs of the Data is 44.839%.

In [23]:
```
# Classification Error is given as
print("Number of Mislabelled points out of total %d points: %d"%(X_train_std_q2
_pca_2PC.shape[0], (dataB_y_values != y_pred_q2_2PC).sum()))
```

```
Number of Mislabelled points out of total 2066 points: 859
```

In [24]:
```
# Explained Variace ratio
print("\n Explained Variance for 2PCs: ",pca_q2_2PC.explained_variance_ratio_.c
umsum())
```

```
 Explained Variance for 2PCs:  [0.06601053 0.10272855]
```

In [25]:
```
# classification error and retained variance list for all the sets
global classification_error
classification_error = []
global retained_variance
retained_variance = []
classification_error_2PC = (1 - accuracy_2PC)
retained_variance_2PC = pca_q2_2PC.explained_variance_ratio_.cumsum()
classification_error.append(classification_error_2PC)
retained_variance.append(retained_variance_2PC[-1])
print(classification_error)
print(retained_variance)
```

```
[0.4157792836398838]
[0.10272854843026871]
```

Naive Bayes on first 4 Components

In [26]:
```
pca_q2_4PC = PCA(n_components=4)
X_train_std_q2_pca_4PC = pca_q2_4PC.fit_transform(dataB_X_std)
# X_test_std_q2_pca_4PC = pca_q2_4PC.fit_transform(X_test_std_q2)
gnb_q2_4PC = GaussianNB()
gnb_q2_4PC.fit(X_train_std_q2_pca_4PC, dataB_y_values)
y_pred_q2_4PC = gnb_q2_4PC.predict(X_train_std_q2_pca_4PC)
accuracy_4PC = accuracy_score(dataB_y_values, y_pred_q2_4PC)
print("\n Accuracy Score for Test Set using 4PCs: {0:.3f}%".format( accuracy_4P
C*100))
```

```
 Accuracy Score for Test Set using 4PCs: 79.526%
```

In [27]:
```
# Classification Error is given as
print("Number of Mislabelled points out of total %d points: %d"%(X_train_std_q2
_pca_4PC.shape[0], (dataB_y_values != y_pred_q2_4PC).sum()))
```

```
Number of Mislabelled points out of total 2066 points: 423
```

In [28]:
```
# Explained Variace ratio
print("\n Explained Variance for 4PCs: \n",pca_q2_4PC.explained_variance_ratio
_.cumsum())
```

```
 Explained Variance for 4PCs:
 [0.06601053 0.10272855 0.13685859 0.16736722]
```

In [29]:
```
# classification error and retained variance list for all the sets
classification_error_4PC = (1 - accuracy_4PC)
retained_variance_4PC = pca_q2_4PC.explained_variance_ratio_.cumsum()
classification_error.append(classification_error_4PC)
retained_variance.append(retained_variance_4PC[-1])
print(classification_error)
print(retained_variance)
```

```
[0.4157792836398838, 0.20474346563407553]
[0.10272854843026871, 0.16736721603112412]
```

Naive Bayes Classification on First 10 PC

In [30]:
```
pca_q2_10PC = PCA(n_components=10)
X_train_std_q2_pca_10PC = pca_q2_10PC.fit_transform(dataB_X_std)
# X_test_std_q2_pca_10PC = pca_q2_10PC.fit_transform(X_test_std_q2)
gnb_q2_10PC = GaussianNB()
gnb_q2_10PC.fit(X_train_std_q2_pca_10PC, dataB_y_values)
y_pred_q2_10PC = gnb_q2_10PC.predict(X_train_std_q2_pca_10PC)
accuracy_10PC = accuracy_score(dataB_y_values, y_pred_q2_10PC)
print("\n Accuracy Score for Test Set using 10PCs: {0:.3f}%".format( accuracy_1
0PC*100))
```

```
 Accuracy Score for Test Set using 10PCs: 90.368%
```

In [31]:
```
print("Number of Mislabelled points out of total %d points: %d"%(X_train_std_q2
_pca_10PC.shape[0], (dataB_y_values != y_pred_q2_10PC).sum()))
```

```
Number of Mislabelled points out of total 2066 points: 199
```

In [32]:
```
# Explained Variace ratio
print("\n Explained Variance for 10PCs: \n",pca_q2_10PC.explained_variance_rati
o_.cumsum())
```

```
 Explained Variance for 10PCs:
 [0.06601053 0.10272855 0.13685859 0.16736722 0.19487308 0.21513562
 0.23280808 0.24799466 0.26158673 0.27410905]
```

In [33]:
```
# classification error and retained variance list for all the sets
classification_error_10PC = (1 - accuracy_10PC)
retained_variance_10PC = pca_q2_10PC.explained_variance_ratio_.cumsum()
classification_error.append(classification_error_10PC)
retained_variance.append(retained_variance_10PC[-1])
print(classification_error)
print(retained_variance)
```

```
[0.4157792836398838, 0.20474346563407553, 0.09632139399806394]
[0.10272854843026871, 0.16736721603112412, 0.2741090530740235]
```

Naive Bayes Classification on First 30 PC

In [34]:
```
pca_q2_30PC = PCA(n_components=30)
X_train_std_q2_pca_30PC = pca_q2_30PC.fit_transform(dataB_X_std)
# X_test_std_q2_pca_30PC = pca_q2_30PC.fit_transform(X_test_std_q2)
gnb_q2_30PC = GaussianNB()
gnb_q2_30PC.fit(X_train_std_q2_pca_30PC, dataB_y_values)
y_pred_q2_30PC = gnb_q2_30PC.predict(X_train_std_q2_pca_30PC)
accuracy_30PC = accuracy_score(dataB_y_values, y_pred_q2_30PC)
print("\n Accuracy Score for Test Set using 30PCs: {0:.3f}%".format( accuracy_3
0PC*100))
```

```
 Accuracy Score for Test Set using 30PCs: 90.900%
```

In [35]:
```
print("Number of Mislabelled points out of total %d points: %d"%(X_train_std_q2
_pca_30PC.shape[0], (dataB_y_values != y_pred_q2_30PC).sum()))
```

```
Number of Mislabelled points out of total 2066 points: 188
```

In [36]:
```
# Explained Variace ratio
print("\n Explained Variance for 30PCs: \n",pca_q2_30PC.explained_variance_rati
o_.cumsum())
```

```
 Explained Variance for 30PCs:
 [0.06601053 0.10272855 0.13685859 0.16736722 0.19487308 0.21513562
 0.23280811 0.24799475 0.26158691 0.27411093 0.28572945 0.29629569
 0.3058193  0.31498173 0.32392895 0.3325077  0.34037509 0.34806276
 0.35567622 0.36300059 0.37009498 0.37690205 0.38367002 0.39013437
 0.39636988 0.40238498 0.40805604 0.41367824 0.41921845 0.42460451]
```

In [37]:
```
# classification error and retained variance list for all the sets
classification_error_30PC = (1 - accuracy_30PC)
retained_variance_30PC = pca_q2_30PC.explained_variance_ratio_.cumsum()
classification_error.append(classification_error_30PC)
retained_variance.append(retained_variance_30PC[-1])
print(classification_error)
print(retained_variance)
```

```
[0.4157792836398838, 0.20474346563407553, 0.09632139399806394, 0.09099709583736
693]
[0.10272854843026871, 0.16736721603112412, 0.2741090530740235, 0.42460450889383
08]
```

Naive Bayes Classification on First 60 PC

In [38]:
```
pca_q2_60PC = PCA(n_components=60)
X_train_std_q2_pca_60PC = pca_q2_60PC.fit_transform(dataB_X_std)
X_test_std_q2_pca_60PC = pca_q2_60PC.fit_transform(X_test_std_q2)
gnb_q2_60PC = GaussianNB()
gnb_q2_60PC.fit(X_train_std_q2_pca_60PC, dataB_y_values)
y_pred_q2_60PC = gnb_q2_60PC.predict(X_train_std_q2_pca_60PC)
accuracy_60PC = accuracy_score(dataB_y_values, y_pred_q2_60PC)
print("\n Accuracy Score for Test Set using 60PCs: {0:.3f}%".format( accuracy_6
0PC*100))
```

```
 Accuracy Score for Test Set using 60PCs: 80.784%
```

In [39]:
```
print("Number of Mislabelled points out of total %d points: %d"%(X_train_std_q2
_pca_60PC.shape[0], (dataB_y_values != y_pred_q2_60PC).sum()))
```

```
Number of Mislabelled points out of total 2066 points: 397
```

In [40]:
```
# Explained Variace ratio
print("\n Explained Variance for 60PCs: \n",pca_q2_60PC.explained_variance_rati
o_.cumsum())
```

```
 Explained Variance for 60PCs:
 [0.06666315 0.10288598 0.13772673 0.1695333  0.19761394 0.22010355
 0.2384579  0.25427235 0.26875234 0.28215208 0.29461619 0.30639337
 0.31760769 0.32793428 0.33819283 0.34775482 0.35680059 0.36572013
 0.37419452 0.38217266 0.38979461 0.39720553 0.40438677 0.41140726
 0.41806392 0.42467588 0.43116106 0.43753193 0.44358797 0.44956291
 0.45541073 0.46112362 0.46663989 0.47204999 0.47741551 0.48244524
 0.48745409 0.49240703 0.49720491 0.50196846 0.50660854 0.51119385
 0.51567509 0.52008713 0.52446392 0.52869674 0.53284797 0.53695128
 0.54102592 0.54498686 0.54893393 0.55285294 0.55671719 0.56044695
 0.5641     0.56770341 0.57122541 0.57472838 0.57816402 0.58156588]
```

In [41]:
```
# classification error and retained variance list for all the sets
classification_error_60PC = (1 - accuracy_60PC)
retained_variance_60PC = pca_q2_60PC.explained_variance_ratio_.cumsum()
classification_error.append(classification_error_60PC)
retained_variance.append(retained_variance_60PC[-1])
print(classification_error)
print(retained_variance)
```

```
[0.4157792836398838, 0.20474346563407553, 0.09632139399806394, 0.09099709583736
693, 0.1921587608906099]
[0.10272854843026871, 0.16736721603112412, 0.2741090530740235, 0.42460450889383
08, 0.5815658836005093]
```

Naive Bayes Classification on First 200 PC

In [42]:
```
pca_q2_200PC = PCA(n_components=200)
X_train_std_q2_pca_200PC = pca_q2_200PC.fit_transform(dataB_X_std)
X_test_std_q2_pca_200PC = pca_q2_200PC.fit_transform(X_test_std_q2)
gnb_q2_200PC = GaussianNB()
gnb_q2_200PC.fit(X_train_std_q2_pca_200PC, dataB_y_values)
y_pred_q2_200PC = gnb_q2_200PC.predict(X_train_std_q2_pca_200PC)
accuracy_200PC = accuracy_score(dataB_y_values, y_pred_q2_200PC)
print("\n Accuracy Score for Test Set using 200PCs: {0:.3f}%".format( accuracy_
200PC*100))
```

```
 Accuracy Score for Test Set using 200PCs: 75.895%
```

In [43]:
```
print("Number of Mislabelled points out of total %d points: %d"%(X_train_std_q2
_pca_200PC.shape[0], (dataB_y_values != y_pred_q2_200PC).sum()))
```

```
Number of Mislabelled points out of total 2066 points: 498
```

In [44]:
```
# Explained Variace ratio
print("\n Explained Variance for 200PCs: \n",pca_q2_200PC.explained_variance_ra
tio_.cumsum())
```

```
 Explained Variance for 200PCs:
 [0.06666315 0.10288598 0.13772673 0.1695333  0.19761394 0.22010355
 0.2384579  0.25427235 0.26875234 0.28215208 0.29461619 0.30639337
 0.31760769 0.32793428 0.33819283 0.34775482 0.35680059 0.36572013
 0.37419453 0.38217266 0.38979462 0.39720554 0.40438678 0.4114073
 0.41806398 0.42467599 0.43116119 0.43753209 0.44358815 0.44956318
 0.45541115 0.46112418 0.46664065 0.47205097 0.47741695 0.48244737
 0.48745659 0.49241034 0.49720982 0.50197531 0.50661909 0.51120584
 0.51568978 0.52011017 0.52449147 0.52873186 0.53288687 0.53699484
 0.54107604 0.54504876 0.54900863 0.55293459 0.55681014 0.56057167
 0.56426398 0.56789385 0.57144771 0.57498215 0.57845029 0.58188717
 0.58526622 0.58860274 0.59190168 0.5951657  0.59835395 0.60146329
 0.60455497 0.60762767 0.61065175 0.61362267 0.61657675 0.61950214
 0.62242382 0.62527683 0.62810527 0.63090906 0.63366348 0.63641069
 0.63914434 0.64184958 0.64449937 0.64713757 0.64974373 0.65233323
 0.65490867 0.65744094 0.65996693 0.66245691 0.66493798 0.66741547
 0.66986068 0.67228225 0.67468531 0.67708568 0.67947712 0.68184882
 0.68421488 0.68654094 0.68886449 0.69116642 0.69345896 0.69570981
 0.69795339 0.70017377 0.70238204 0.70456723 0.70673186 0.70888317
 0.71102127 0.71315824 0.71526702 0.71737278 0.71945642 0.72152488
 0.72357924 0.72561736 0.72763625 0.72964548 0.73164289 0.73362934
 0.73560267 0.73756709 0.73951575 0.74145754 0.74337794 0.74528693
 0.74718625 0.74907476 0.75094312 0.75278834 0.75462688 0.75646155
 0.75829046 0.76010198 0.76189626 0.76367872 0.76544591 0.76720532
 0.76894386 0.77068023 0.77240968 0.77411731 0.77581316 0.77750241
 0.77918321 0.78086222 0.78253015 0.78418589 0.78582946 0.78745305
 0.78906243 0.79065749 0.7922417  0.79381697 0.79538388 0.79694501
 0.7984963  0.80004593 0.80157999 0.80310381 0.80461693 0.80611626
 0.80760316 0.80907795 0.81053785 0.811994   0.81343407 0.81486275
 0.81628678 0.81770726 0.81910431 0.82049251 0.82187226 0.82324294
 0.82460041 0.82594156 0.82727748 0.82861019 0.82993018 0.83123482
 0.83252968 0.83381618 0.83509104 0.83635693 0.83761695 0.83885971
 0.84009385 0.84132578 0.84254494 0.84375631 0.84496389 0.8461495
 0.84733145 0.84850361 0.84966608 0.85080942 0.85194398 0.85307084
 0.85418961 0.85530086]
```

In [45]:
```
# classification error and retained variance list for all the sets
classification_error_200PC = (1 - accuracy_200PC)
retained_variance_200PC = pca_q2_200PC.explained_variance_ratio_.cumsum()
classification_error.append(classification_error_200PC)
retained_variance.append(retained_variance_200PC[-1])
print(classification_error)
print(retained_variance)
```

```
[0.4157792836398838, 0.20474346563407553, 0.09632139399806394, 0.09099709583736
693, 0.1921587608906099, 0.24104549854791868]
[0.10272854843026871, 0.16736721603112412, 0.2741090530740235, 0.42460450889383
08, 0.5815658836005093, 0.8553008593813782]
```

Naive Bayes Classification on First 500 PC

In [46]:
```
pca_q2_500PC = PCA(n_components=500, svd_solver='full')
X_train_std_q2_pca_500PC = pca_q2_500PC.fit_transform(dataB_X_std)
X_test_std_q2_pca_500PC = pca_q2_500PC.fit_transform(X_test_std_q2)
gnb_q2_500PC = GaussianNB()
gnb_q2_500PC.fit(X_train_std_q2_pca_500PC, dataB_y_values)
y_pred_q2_500PC = gnb_q2_500PC.predict(X_train_std_q2_pca_500PC)
accuracy_500PC = accuracy_score(dataB_y_values, y_pred_q2_500PC)
print("\n Accuracy Score for Test Set using 500PCs: {0:.3f}%".format( accuracy_
500PC*100))
```

```
 Accuracy Score for Test Set using 500PCs: 76.041%
```

In [47]:
```
print("Number of Mislabelled points out of total %d points: %d"%(X_train_std_q2
_pca_500PC.shape[0], (dataB_y_values != y_pred_q2_500PC).sum()))
```

```
Number of Mislabelled points out of total 2066 points: 495
```

In [48]:
```
# Explained Variace ratio
print("\n Explained Variance for 500PCs: \n",pca_q2_500PC.explained_variance_ra
tio_.cumsum())
```

```
Explained Variance for 500PCs:
[0.06666315 0.10288598 0.13772673 0.1695333  0.19761394 0.22010355
0.2384579  0.25427235 0.26875234 0.28215208 0.29461619 0.30639337
0.31760769 0.32793428 0.33819283 0.34775482 0.35680059 0.36572013
0.37419453 0.38217266 0.38979462 0.39720554 0.40438679 0.4114073
0.41806398 0.424676   0.43116119 0.43753209 0.44358815 0.44956319
0.45541115 0.46112419 0.46664066 0.47205099 0.47741697 0.48244739
0.48745662 0.49241038 0.49720986 0.50197536 0.50661915 0.5112059
0.51568986 0.52011027 0.52449159 0.52873199 0.53288702 0.536995
0.54107622 0.54504898 0.5490089  0.55293491 0.5568105  0.56057209
0.56426444 0.56789437 0.57144833 0.57498286 0.57845104 0.58188804
0.58526715 0.58860374 0.59190285 0.59516702 0.59835546 0.60146497
0.60455685 0.60762985 0.61065412 0.61362523 0.61657954 0.61950523
0.62242733 0.62528074 0.62810964 0.63091383 0.63366869 0.63641667
0.6391509  0.64185662 0.64450684 0.64714597 0.64975259 0.65234317
0.65491944 0.6574529  0.65997994 0.66247075 0.66495272 0.66743108
0.66987727 0.67229998 0.67470396 0.67710546 0.67949778 0.68187074
0.68423775 0.68656512 0.68889036 0.6911945  0.6934879  0.69574023
0.69798478 0.70020774 0.70241803 0.70460544 0.70677301 0.70892686
0.7110671  0.71320573 0.71531968 0.71742801 0.71951409 0.72158649
0.72364287 0.72568409 0.72770601 0.72971821 0.73171982 0.73371156
0.73569128 0.73765913 0.73961421 0.74155967 0.74348453 0.74539823
0.74730302 0.74919819 0.75107126 0.75292343 0.75476942 0.75661046
0.75844427 0.76026192 0.76206331 0.76385322 0.7656327  0.7674026
0.76915228 0.77089943 0.77263424 0.774353   0.77605827 0.77775734
0.77945207 0.78114196 0.78282433 0.78449013 0.78614567 0.78778292
0.78940919 0.79102235 0.79262269 0.79421737 0.79580056 0.79737718
0.79895118 0.80051567 0.80207448 0.80362054 0.80515815 0.80667782
0.80818809 0.80969544 0.81119092 0.81267594 0.8141477  0.81560377
0.81705765 0.81850094 0.81991922 0.82133611 0.822751   0.82415243
0.82554462 0.82693173 0.82831084 0.82968312 0.83104639 0.83240156
0.83374898 0.83509314 0.83642878 0.83775269 0.839069   0.84036896
0.8416662  0.84296057 0.84424143 0.84551407 0.84677709 0.84803675
0.8492856  0.85052812 0.85176757 0.85300201 0.85422898 0.85543736
0.85663997 0.85783619 0.85902558 0.86020755 0.86138322 0.86254399
0.86369972 0.86484842 0.8659918  0.86711843 0.86824398 0.86935906
0.87046857 0.87157146 0.87266931 0.87375332 0.87482872 0.8759031
0.87696967 0.8780304  0.8790906  0.88014095 0.88117932 0.88221307
0.88323118 0.88424326 0.88525139 0.88625283 0.88724755 0.88823078
0.88920893 0.89018431 0.89115554 0.89210737 0.8930566  0.8939997
0.89493496 0.89586445 0.89678463 0.89770363 0.89861778 0.8995278
0.90042244 0.90130918 0.90219162 0.90306432 0.90393647 0.90480713
0.90566866 0.90652225 0.90736704 0.90821    0.90904952 0.90987577
0.91070001 0.91151215 0.91232243 0.91312896 0.91392814 0.9147229
0.91551093 0.91629714 0.91707448 0.91784708 0.91861335 0.9193754
0.9201293  0.92087438 0.92161328 0.92234896 0.92308421 0.92380872
0.92452968 0.92524836 0.92595758 0.92665935 0.92735706 0.92804859
0.92873494 0.92941608 0.93009611 0.93077292 0.93144464 0.93210482
0.93276194 0.93341247 0.93406166 0.9347064  0.93534499 0.93597756
0.93660434 0.937226   0.93784553 0.93846289 0.9390698  0.93967414
0.94027149 0.94086615 0.94145577 0.9420386  0.94261816 0.94319176
0.94376236 0.94432806 0.94489161 0.94544965 0.94600334 0.94655053
0.94709497 0.94763556 0.94817339 0.94870359 0.94922968 0.94974976
0.95026704 0.95078071 0.95129068 0.9517944  0.95229641 0.9527937
0.95328408 0.95377141 0.95425476 0.95473202 0.95520836 0.95567933
0.95614811 0.9566133  0.95707318 0.95753088 0.95798292 0.95843321
0.95887781 0.95931497 0.95974992 0.96018017 0.96060748 0.96102916
0.96144634 0.96185969 0.96227169 0.96267879 0.96308252 0.96348134
0.96387604 0.96426851 0.96465977 0.96504531 0.96542875 0.96580716
0.96617981 0.96655169 0.96692157 0.96728757 0.96765091 0.96801053
0.96836931 0.96872153 0.96906981 0.96941514 0.96975593 0.97009551
0.97043026 0.97076101 0.97108821 0.97141279 0.97173416 0.97205355
0.97236968 0.97268108 0.97298854 0.97329304 0.97359254 0.97388719
```

In [49]:
```
# classification error and retained variance list for all the sets
classification_error_500PC = (1 - accuracy_500PC)
retained_variance_500PC = pca_q2_500PC.explained_variance_ratio_.cumsum()
classification_error.append(classification_error_500PC)
retained_variance.append(retained_variance_500PC[-1])
print(classification_error)
print(retained_variance)
```

```
[0.4157792836398838, 0.20474346563407553, 0.09632139399806394, 0.09099709583736
693, 0.1921587608906099, 0.24104549854791868, 0.23959341723136496]
[0.10272854843026871, 0.16736721603112412, 0.2741090530740235, 0.42460450889383
08, 0.5815658836005093, 0.8553008593813782, 0.9943347485917375]
```

Naive Bayes Classification on all 784 PC

In [50]:
```
pca_q2_784PC = PCA(n_components=784)
X_train_std_q2_pca_784PC = pca_q2_784PC.fit_transform(dataB_X_std)
X_test_std_q2_pca_784PC = pca_q2_784PC.fit_transform(X_test_std_q2)
gnb_q2_784PC = GaussianNB()
gnb_q2_784PC.fit(X_train_std_q2_pca_784PC, dataB_y_values)
y_pred_q2_784PC = gnb_q2_784PC.predict(X_train_std_q2_pca_784PC)
accuracy_784PC = accuracy_score(dataB_y_values, y_pred_q2_784PC)
print("\n Accuracy Score for Test Set using 784PCs: {0:.3f}%".format( accuracy_
784PC*100))
```

```
 Accuracy Score for Test Set using 784PCs: 77.977%
```

In [51]:
```
print("Number of Mislabelled points out of total %d points: %d"%(X_train_std_q2
_pca_784PC.shape[0], (dataB_y_values != y_pred_q2_784PC).sum()))
```

```
Number of Mislabelled points out of total 2066 points: 455
```

In [52]:
```
# Explained Variace ratio
print("\n Explained Variance for 784PCs: \n",pca_q2_784PC.explained_variance_ra
tio_.cumsum())
```

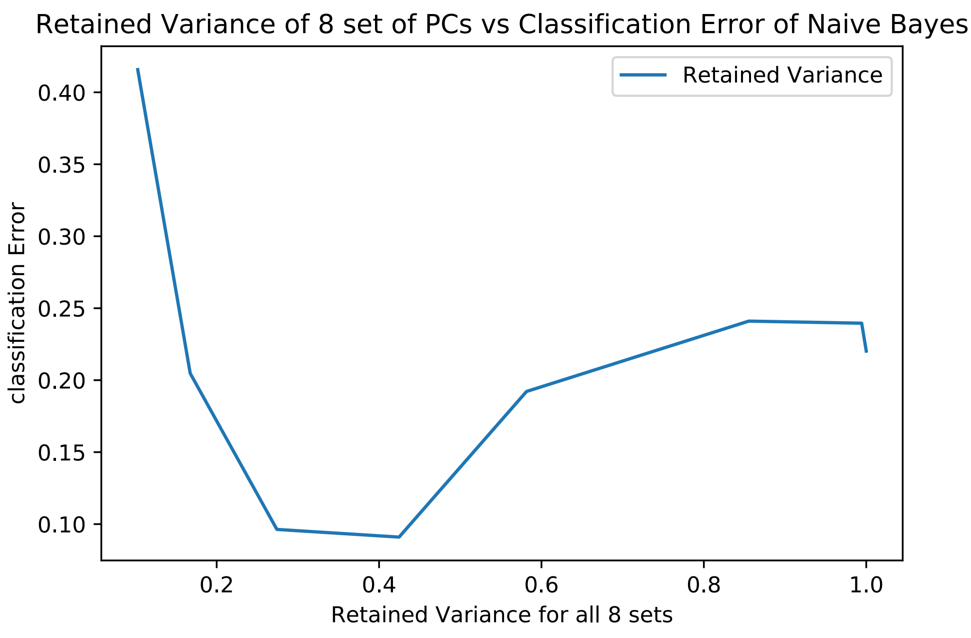
```
Explained Variance for 784PCs:
[0.06666315 0.10288598 0.13772673 0.1695333  0.19761394 0.22010355
 0.2384579  0.25427235 0.26875234 0.28215208 0.29461619 0.30639337
 0.31760769 0.32793428 0.33819283 0.34775482 0.35680059 0.36572013
 0.37419453 0.38217266 0.38979462 0.39720554 0.40438679 0.4114073
 0.41806398 0.424676   0.43116119 0.43753209 0.44358815 0.44956319
 0.45541115 0.46112419 0.46664066 0.47205099 0.47741697 0.48244739
 0.48745662 0.49241038 0.49720986 0.50197536 0.50661915 0.5112059
 0.51568986 0.52011027 0.52449159 0.52873199 0.53288702 0.536995
 0.54107622 0.54504898 0.5490089  0.55293491 0.5568105  0.56057209
 0.56426444 0.56789437 0.57144833 0.57498286 0.57845104 0.58188804
 0.58526715 0.58860374 0.59190285 0.59516702 0.59835546 0.60146497
 0.60455685 0.60762985 0.61065412 0.61362523 0.61657954 0.61950523
 0.62242733 0.62528074 0.62810964 0.63091383 0.63366869 0.63641667
 0.6391509  0.64185662 0.64450684 0.64714597 0.64975259 0.65234317
 0.65491944 0.6574529  0.65997994 0.66247075 0.66495272 0.66743108
 0.66987727 0.67229998 0.67470396 0.67710546 0.67949778 0.68187074
 0.68423775 0.68656512 0.68889036 0.6911945  0.6934879  0.69574023
 0.69798478 0.70020774 0.70241803 0.70460544 0.70677301 0.70892686
 0.7110671  0.71320573 0.71531968 0.71742801 0.71951409 0.72158649
 0.72364287 0.72568409 0.72770601 0.72971821 0.73171982 0.73371156
 0.73569128 0.73765913 0.73961421 0.74155967 0.74348453 0.74539823
 0.74730302 0.74919819 0.75107126 0.75292343 0.75476942 0.75661046
 0.75844427 0.76026192 0.76206331 0.76385322 0.7656327  0.7674026
 0.76915228 0.77089943 0.77263424 0.774353   0.77605827 0.77775734
 0.77945207 0.78114196 0.78282433 0.78449013 0.78614567 0.78778292
 0.78940919 0.79102235 0.79262269 0.79421737 0.79580056 0.79737718
 0.79895118 0.80051567 0.80207448 0.80362054 0.80515815 0.80667782
 0.80818809 0.80969544 0.81119092 0.81267594 0.8141477  0.81560377
 0.81705765 0.81850094 0.81991922 0.82133611 0.822751   0.82415243
 0.82554462 0.82693173 0.82831084 0.82968312 0.83104639 0.83240156
 0.83374898 0.83509314 0.83642878 0.83775269 0.839069   0.84036896
 0.8416662  0.84296057 0.84424143 0.84551407 0.84677709 0.84803675
 0.8492856  0.85052812 0.85176757 0.85300201 0.85422898 0.85543736
 0.85663997 0.85783619 0.85902558 0.86020755 0.86138322 0.86254399
 0.86369972 0.86484842 0.8659918  0.86711843 0.86824398 0.86935906
 0.87046857 0.87157146 0.87266931 0.87375332 0.87482872 0.8759031
 0.87696967 0.8780304  0.8790906  0.88014095 0.88117932 0.88221307
 0.88323118 0.88424326 0.88525139 0.88625283 0.88724755 0.88823078
 0.88920893 0.89018431 0.89115554 0.89210737 0.8930566  0.8939997
 0.89493496 0.89586445 0.89678463 0.89770363 0.89861778 0.8995278
 0.90042244 0.90130918 0.90219162 0.90306432 0.90393647 0.90480713
 0.90566866 0.90652225 0.90736704 0.90821    0.90904952 0.90987577
 0.91070001 0.91151215 0.91232243 0.91312896 0.91392814 0.9147229
 0.91551093 0.91629714 0.91707448 0.91784708 0.91861335 0.9193754
 0.9201293  0.92087438 0.92161328 0.92234896 0.92308421 0.92380872
 0.92452968 0.92524836 0.92595758 0.92665935 0.92735706 0.92804859
 0.92873494 0.92941608 0.93009611 0.93077292 0.93144464 0.93210482
 0.93276194 0.93341247 0.93406166 0.9347064  0.93534499 0.93597756
 0.93660434 0.937226   0.93784553 0.93846289 0.9390698  0.93967414
 0.94027149 0.94086615 0.94145577 0.9420386  0.94261816 0.94319176
 0.94376236 0.94432806 0.94489161 0.94544965 0.94600334 0.94655053
 0.94709497 0.94763556 0.94817339 0.94870359 0.94922968 0.94974976
 0.95026704 0.95078071 0.95129068 0.9517944  0.95229641 0.9527937
 0.95328408 0.95377141 0.95425476 0.95473202 0.95520836 0.95567933
 0.95614811 0.9566133  0.95707318 0.95753088 0.95798292 0.95843321
 0.95887781 0.95931497 0.95974992 0.96018017 0.96060748 0.96102916
 0.96144634 0.96185969 0.96227169 0.96267879 0.96308252 0.96348134
 0.96387604 0.96426851 0.96465977 0.96504531 0.96542875 0.96580716
 0.96617981 0.96655169 0.96692157 0.96728757 0.96765091 0.96801053
 0.96836931 0.96872153 0.96906981 0.96941514 0.96975593 0.97009551
 0.97043026 0.97076101 0.97108821 0.97141279 0.97173416 0.97205355
 0.97236968 0.97268108 0.97298854 0.97329304 0.97359254 0.97388719
```

In [53]:
```
# classification error and retained variance list for all the sets
classification_error_784PC = (1 - accuracy_784PC)
retained_variance_784PC = pca_q2_784PC.explained_variance_ratio_.cumsum()
classification_error.append(classification_error_784PC)
retained_variance.append(retained_variance_784PC[-1])
print(classification_error)
print(retained_variance)
```

```
[0.4157792836398838, 0.20474346563407553, 0.09632139399806394, 0.09099709583736
693, 0.1921587608906099, 0.24104549854791868, 0.23959341723136496, 0.2202323330
106486]
[0.10272854843026871, 0.16736721603112412, 0.2741090530740235, 0.42460450889383
08, 0.5815658836005093, 0.8553008593813782, 0.9943347485917375, 1.0000000000000
002]
```

Plot of retained Variance vs Error

In [54]:
```
plt.plot(retained_variance, classification_error, label = 'Retained Variance',
linestyle='-')
plt.xlabel('Retained Variance for all 8 sets')
plt.ylabel('classification Error')
plt.legend(loc='best')
plt.tight_layout()
plt.title("Retained Variance of 8 set of PCs vs Classification Error of Naive B
ayes")
plt.show()
```

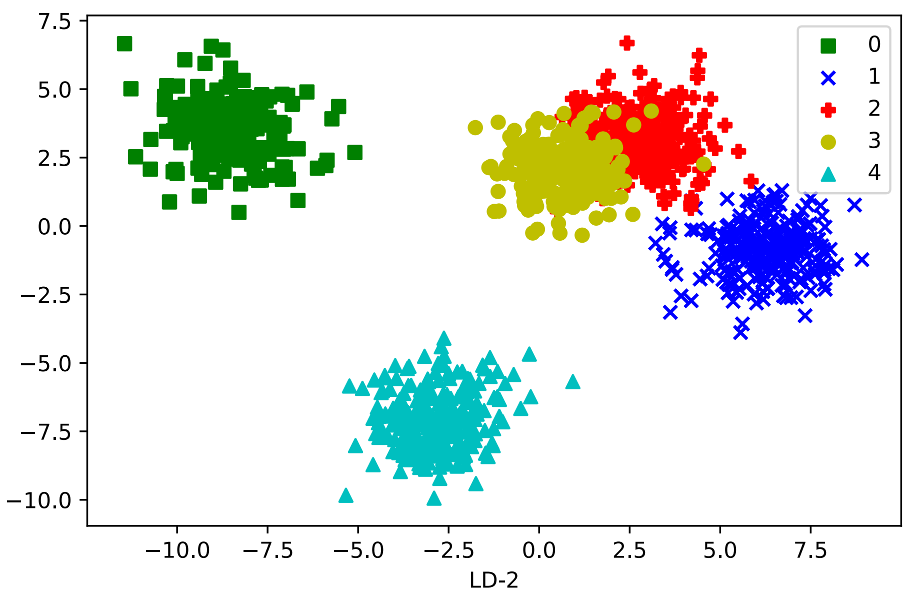


Analysis:

As seen from the graph it is clear that the classification error is highest when the variance in the dataset is slowest for 2PCs and for this the classification error is higher than 40%. However this error decreases with increase in principle components up until 30 PCs and at this set of PCs the accuracy score is highest given by the classifier to 90.949% and the classificaiton error is lowest among all PCs. As evident from the graph the classification error increase after the set of 30 PCs and for all PCs (784 for this dataset) the classification error is at around 22%.

Dimensionality Reduction using LDA

In [55]:
```
lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train_std_q2, y_train_values_q2)

colors = ['g', 'b', 'r', 'y', 'c']
markers = ['s', 'x', 'P', 'o', '^']
for index, color, marker in zip(np.unique(y_train_values_q2), colors, markers):
    plt.scatter(X_train_lda[y_train_values_q2==index, 0], X_train_lda[y_train_v
alues_q2==index, 1] * (-1), c = color, label = index, marker = marker)

plt.xlabel('LD-1')
plt.xlabel('LD-2')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```



Analysis:

On comparing the dataset when it is projected using the first two components PCA and LDA, LDA gives a plot in which the class labels '0' and '4' are linearly sepearable from other classes as compared the plot by PCA. The class label '1' in the plot by LDA is distinguishable from the class labels '2' and '3' however a few points overlap with the class label '2' points. In the same plot, class label '3' and class label '2' cannot be seperated linearly. In the plot by PCA the all class labels are linearly inseperable from one another and they overlap one another.

Q3.1 LLE on Digit '3'

In [56]:
```
# Select the data for the digit 3 in the Dataset

dataB_updatedDig3 = (dataB_updated.loc[dataB_updated['gnd'] == 3])
print("\n DataB after removing Unnamed column: \n",dataB_updatedDig3.head(1))
```

```
 DataB after removing Unnamed column:
      fea.1  fea.2  fea.3  fea.4  fea.5  fea.6  fea.7  fea.8  fea.9  fea.10  \
1237      3      4      2      1      3      3      0      1      2       0

      fea.11  fea.12  fea.13  fea.14  fea.15  fea.16  fea.17  fea.18  fea.19  \
1237       3       0       5       3       3       2       1       3       0

      fea.20  fea.21  fea.22  fea.23  fea.24  fea.25  fea.26  fea.27  fea.28  \
1237       4       2       3       3       3       1       4       3       1

      fea.29  fea.30  fea.31  fea.32  fea.33  fea.34  fea.35  fea.36  fea.37  \
1237       1       5       3       4       2       5       3       4       4

      fea.38  fea.39  fea.40  fea.41  fea.42  fea.43  fea.44  fea.45  fea.46  \
1237       2       4       3       4       4       3       3       2       0

      fea.47  fea.48  fea.49  fea.50  fea.51  fea.52  fea.53  fea.54  fea.55  \
1237       2       1       3       1       1       1       2       3       1

      fea.56  fea.57  fea.58  fea.59  fea.60  fea.61  fea.62  fea.63  fea.64  \
1237       1       4       3       4       3       2       1       1       1

      fea.65  fea.66  fea.67  fea.68  fea.69  fea.70  fea.71  fea.72  fea.73  \
1237       3       2       1       2       0       1       0       2       3

      fea.74  fea.75  fea.76  fea.77  fea.78  fea.79  fea.80  fea.81  fea.82  \
1237       3       3       0       3       3       2       4       0       2

      fea.83  fea.84  fea.85  fea.86  fea.87  fea.88  fea.89  fea.90  fea.91  \
1237       3       5       5       3       2       2       3       5       5

      fea.92  fea.93  fea.94  fea.95  fea.96  fea.97  fea.98  fea.99  fea.100
\
1237       4       2       2       3       0       2       3       1        2

      fea.101  fea.102  fea.103  fea.104  fea.105  fea.106  fea.107  fea.108  \
1237        3        3        3        3        2        4        5        1

      fea.109  fea.110  fea.111  fea.112  fea.113  fea.114  fea.115  fea.116  \
1237        3        1        4        3        0        4        1        0

      fea.117  fea.118  fea.119  fea.120  fea.121  fea.122  fea.123  fea.124  \
1237        2        3        1        0        3        0        5        2

      fea.125  fea.126  fea.127  fea.128  fea.129  fea.130  fea.131  fea.132  \
1237        1        3        4        1        3        3        1        3

      fea.133  fea.134  fea.135  fea.136  fea.137  fea.138  fea.139  fea.140  \
1237        2        0        3        4        4        3        1        1

      fea.141  fea.142  fea.143  fea.144  fea.145  fea.146  fea.147  fea.148  \
1237        1        0        4        1        3        4        4        4

      fea.149  fea.150  fea.151  fea.152  fea.153  fea.154  fea.155  fea.156  \
1237        2        4        5       41       45      110      255      253

      fea.157  fea.158  fea.159  fea.160  fea.161  fea.162  fea.163  fea.164  \
1237      255      255      255      255      175        7        1        2

      fea.165  fea.166  fea.167  fea.168  fea.169  fea.170  fea.171  fea.172  \
1237        0        1        1        3        0        4        1        3

      fea.173  fea.174  fea.175  fea.176  fea.177  fea.178  fea.179  fea.180  \
```

In [57]:
```
print("\n Total number of samples for class label 3: ",(dataB_updatedDig3.gnd.v
alue_counts()))
```

```
 Total number of samples for class label 3:  3    398
Name: gnd, dtype: int64
```

Total number of class label '3' samples are 398

In [58]:
```
dataB_updatedDig3_X = dataB_updatedDig3.drop(['gnd'], axis=1)
print("\n Feature set for class label'3': \n", dataB_updatedDig3_X.head(1))
```

```
 Feature set for class label'3':
      fea.1  fea.2  fea.3  fea.4  fea.5  fea.6  fea.7  fea.8  fea.9  fea.10  \
1237      3      4      2      1      3      3      0      1      2       0

      fea.11  fea.12  fea.13  fea.14  fea.15  fea.16  fea.17  fea.18  fea.19  \
1237       3       0       5       3       3       2       1       3       0

      fea.20  fea.21  fea.22  fea.23  fea.24  fea.25  fea.26  fea.27  fea.28  \
1237       4       2       3       3       3       1       4       3       1

      fea.29  fea.30  fea.31  fea.32  fea.33  fea.34  fea.35  fea.36  fea.37  \
1237       1       5       3       4       2       5       3       4       4

      fea.38  fea.39  fea.40  fea.41  fea.42  fea.43  fea.44  fea.45  fea.46  \
1237       2       4       3       4       4       3       3       2       0

      fea.47  fea.48  fea.49  fea.50  fea.51  fea.52  fea.53  fea.54  fea.55  \
1237       2       1       3       1       1       1       2       3       1

      fea.56  fea.57  fea.58  fea.59  fea.60  fea.61  fea.62  fea.63  fea.64  \
1237       1       4       3       4       3       2       1       1       1

      fea.65  fea.66  fea.67  fea.68  fea.69  fea.70  fea.71  fea.72  fea.73  \
1237       3       2       1       2       0       1       0       2       3

      fea.74  fea.75  fea.76  fea.77  fea.78  fea.79  fea.80  fea.81  fea.82  \
1237       3       3       0       3       3       2       4       0       2

      fea.83  fea.84  fea.85  fea.86  fea.87  fea.88  fea.89  fea.90  fea.91  \
1237       3       5       5       3       2       2       3       5       5

      fea.92  fea.93  fea.94  fea.95  fea.96  fea.97  fea.98  fea.99  fea.100
\
1237       4       2       2       3       0       2       3       1        2

      fea.101  fea.102  fea.103  fea.104  fea.105  fea.106  fea.107  fea.108  \
1237        3        3        3        3        2        4        5        1

      fea.109  fea.110  fea.111  fea.112  fea.113  fea.114  fea.115  fea.116  \
1237        3        1        4        3        0        4        1        0

      fea.117  fea.118  fea.119  fea.120  fea.121  fea.122  fea.123  fea.124  \
1237        2        3        1        0        3        0        5        2

      fea.125  fea.126  fea.127  fea.128  fea.129  fea.130  fea.131  fea.132  \
1237        1        3        4        1        3        3        1        3

      fea.133  fea.134  fea.135  fea.136  fea.137  fea.138  fea.139  fea.140  \
1237        2        0        3        4        4        3        1        1

      fea.141  fea.142  fea.143  fea.144  fea.145  fea.146  fea.147  fea.148  \
1237        1        0        4        1        3        4        4        4

      fea.149  fea.150  fea.151  fea.152  fea.153  fea.154  fea.155  fea.156  \
1237        2        4        5       41       45      110      255      253

      fea.157  fea.158  fea.159  fea.160  fea.161  fea.162  fea.163  fea.164  \
1237      255      255      255      255      175        7        1        2

      fea.165  fea.166  fea.167  fea.168  fea.169  fea.170  fea.171  fea.172  \
1237        0        1        1        3        0        4        1        3

      fea.173  fea.174  fea.175  fea.176  fea.177  fea.178  fea.179  fea.180  \
```

In [59]:
```
dataB_updatedDig3_y = dataB_updatedDig3['gnd']
print(" \n Target of the Samples: \n",dataB_updatedDig3_y)
```

```
 Target of the Samples:
 1237    3
1238    3
1239    3
1240    3
1241    3
       ..
1630    3
1631    3
1632    3
1633    3
1634    3
Name: gnd, Length: 398, dtype: int64
```

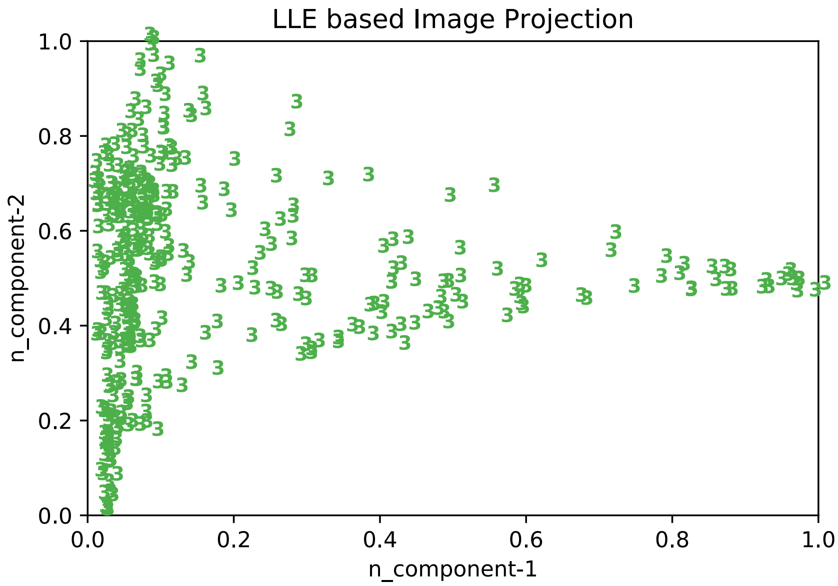
In [60]:
```
# Standardise the Dataset
ss = StandardScaler()
dataB_updatedDig3_X_std = ss.fit_transform(dataB_updatedDig3_X)
```

In [61]:
```
# import LLE from sklearn
lle = LocallyLinearEmbedding(n_neighbors= 5, n_components=2, eigen_solver= 'aut
o',n_jobs=-1)

dataB_updatedDig3_X_std_lletranformed = lle.fit_transform(dataB_updatedDig3_X_s
td)

print("\n LLE transformed.shape: ",dataB_updatedDig3_X_std_lletranformed.shape)
print("\n LLE transformed dataset: \n",dataB_updatedDig3_X_std_lletranformed)
```

```
LLE transformed.shape:  (398, 2)

LLE transformed dataset:
[[ 0.09067985  0.00561093]
[-0.02589247 -0.03926432]
[-0.02250927  0.07089376]
[ 0.02011714  0.02884113]
[-0.03206439  0.0242124 ]
[-0.03348035 -0.08439706]
[ 0.00865793  0.00150839]
[-0.01945895  0.11278798]
[-0.03235335 -0.01979973]
[-0.03262548  0.03860558]
[-0.03240989 -0.0171935 ]
[ 0.1298194  -0.00115743]
[ 0.04848913  0.01657109]
[-0.02659493 -0.03899922]
[ 0.13114936  0.00380842]
[-0.03301174  0.06564318]
[-0.01498219  0.06530604]
[ 0.15444512 -0.00468021]
[ 0.05489728 -0.00431067]
[ 0.04590709 -0.0160958 ]
[-0.02653838  0.00689605]
[ 0.06278132 -0.0053887 ]
[ 0.02753804 -0.03629725]
[ 0.11170507  0.02035988]
[-0.0345758   0.04378238]
[ 0.16135095  0.00056411]
[ 0.0646797   0.03968222]
[-0.0289477   0.01776706]
[-0.01648273  0.08224907]
[ 0.15875238 -0.00390819]
[ 0.08089038 -0.02315076]
[ 0.16373369 -0.00390418]
[ 0.116937   -0.00782903]
[-0.03564523  0.05751232]
[ 0.16292619 -0.00643281]
[ 0.12611709  0.00780192]
[-0.02074566  0.03466916]
[ 0.16182499 -0.00387207]
[-0.03151162 -0.07325323]
[ 0.10347558 -0.01410123]
[-0.02475307  0.03367282]
[ 0.16376173 -0.0040082 ]
[ 0.06825816 -0.01599626]
[ 0.14011009 -0.00436085]
[ 0.00355146  0.05853127]
[-0.02986912 -0.08186231]
[-0.02128005  0.04480767]
[-0.03149332  0.03724861]
[-0.01680148  0.07522176]
[ 0.00053973  0.04271477]
[-0.03077232 -0.03230978]
[ 0.03298948 -0.03472055]
[-0.03339177 -0.0956793 ]
[-0.02438939 -0.00794332]
[ 0.08489577 -0.01688854]
[ 0.10190859 -0.01256061]
[ 0.06210768 -0.01340948]
[-0.03238261 -0.12414915]
[ 0.04284434 -0.03261196]
[-0.01141069 -0.05969647]
```

In [62]:
```
# Plot the text of the target on the Transformed data in the 2D space
def plot_text(X):
    x_min, x_max = np.min(X, 0), np.max(X, 0)
    X = (X - x_min) / (x_max - x_min)
    for i in range(X.shape[0]):
        plt.text(X[i, 0], X[i, 1], str(dataB_updatedDig3_y.values[i]),
                 color=plt.cm.Set1(dataB_updatedDig3_y.values[i] / 10.),
                 fontdict={'weight': 'bold', 'size': 9})
```

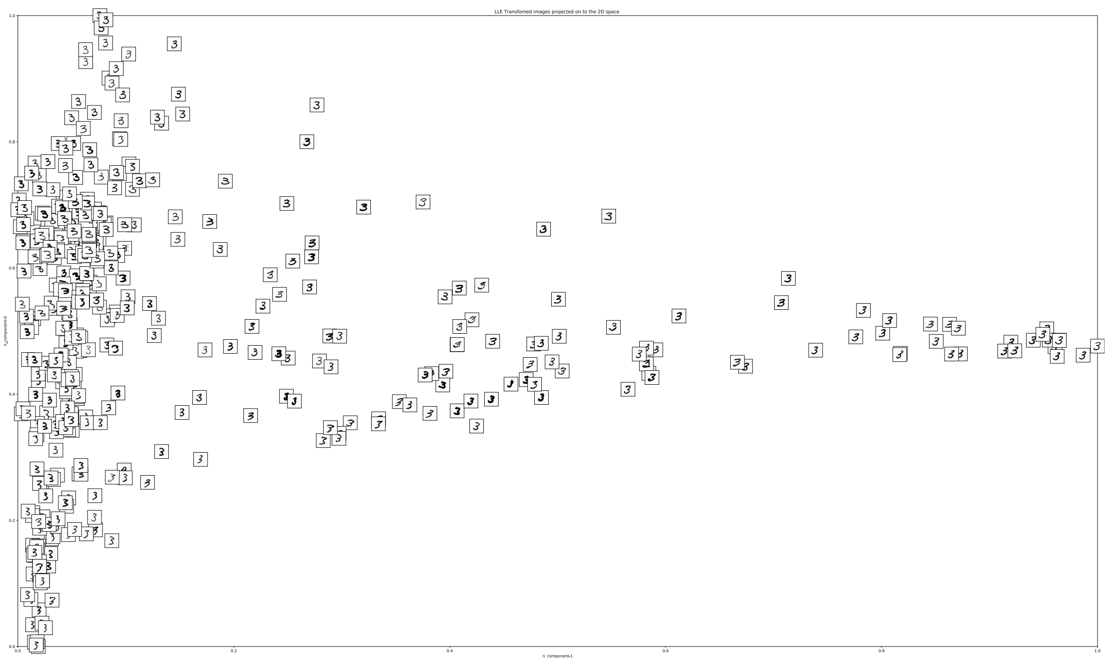
In [63]:
```
plot_text(dataB_updatedDig3_X_std_lletranformed)
plt.xlabel('n_component-1')
plt.ylabel('n_component-2')
plt.title('LLE based Image Projection')
plt.show()
```



In [64]:
```
# Image data from the flattened Dataset
global images_list
images_list = []
for value in range(dataB_updatedDig3_X.shape[0]):
    image = dataB_updatedDig3_X.iloc[[value],:]
    image_new = image.values.reshape(28,28)
    images_list.append(image_new)
```

In [65]:
```
# Graph plot for the Transformed Images
def plot_images(X):
    x_min, x_max = np.min(X, 0), np.max(X, 0)
    X = (X - x_min) / (x_max - x_min)

    plt.figure(figsize=(50,30))
    ax = plt.subplot(111)

    if hasattr(offsetbox, 'AnnotationBbox'):
        shown_images = np.array([[1., 1.]])
        for i in range(X.shape[0]):
            dist = np.sum((X[i] - shown_images) ** 2, 1)
            shown_images = np.r_[shown_images, [X[i]]]
            imagebox = offsetbox.AnnotationBbox(
                offsetbox.OffsetImage(images_list[i], cmap=plt.cm.gray_r),
                X[i])
            ax.add_artist(imagebox)
```

In [66]:
```
plot_images(dataB_updatedDig3_X_std_lletranformed)
plt.xlabel("n_component-1")
plt.ylabel("n_component-2")
plt.title("LLE Transfomed images projected on to the 2D space ")
plt.show()
```



Analysis:

The image of the target from the tranformed LLE space is projected onto the 2D space. On the x-axis in this projected group of images on the left side on the top some images have bold images and the font is thick and dark, whereas this is not the case on the left side below near the point 0 of the x-axis. In addition on the y-axis between 0.5 and 0.7 some images are left skewed whereas below the point 0.4 most of the images are straight or aligned. When we move along the direction of x-axis images slightly start to skew more towards the right mainly starting from 0.2 on the x-axis and this skewness towards the right increases when we move further along the x-axis and as we can see almost all the images are slightly skewed towards the right. In addition, most of the data is before the point 0.2 on the x-axis and between 0.8 and 0.2 on the y-axis. LLE is a local method or it is locally based.

Q3.2 Isomap

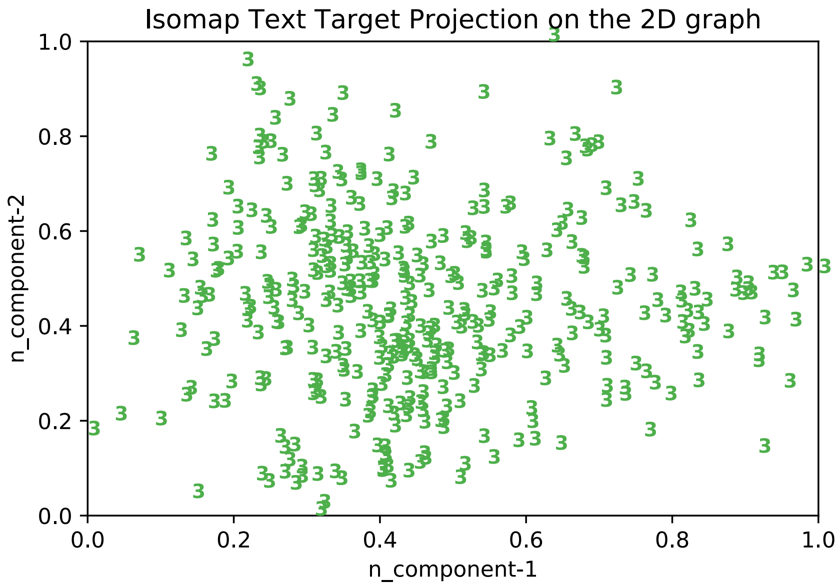
In [67]:
```
from sklearn.manifold import Isomap

iso = Isomap(n_neighbors=5, n_components=2, eigen_solver='auto', n_jobs=-1)
dataB_updatedDig3_X_std_iso = iso.fit_transform(dataB_updatedDig3_X_std)
print(dataB_updatedDig3_X_std_iso)
```

```
[[ 4.17628882e+01  2.97184109e+01]
 [ 1.22218469e+01 -1.29602130e+00]
 [-4.11315359e+01 -1.47833341e+01]
 [ 1.48745601e+01  3.49439933e+01]
 [-7.22317227e+01  3.67584434e+00]
 [-4.82496235e+01  5.86685573e+01]
 [ 3.02938431e+01  1.68316111e+01]
 [-7.04809014e+00 -3.46344784e+01]
 [-2.44896634e+01  8.20526431e+00]
 [-8.74492108e+01 -1.13534329e+01]
 [-2.20362925e+01  8.83215664e+00]
 [ 6.89864402e+01  1.12405376e+01]
 [ 4.34590278e+01  3.44601285e+01]
 [-6.16309866e+00  1.34355826e+01]
 [ 7.75609460e+01  4.96522542e+00]
 [-3.91986663e+01  7.09578154e+00]
 [-1.30951707e+01 -5.77542041e+00]
 [ 9.16906874e+01  2.21074843e+01]
 [ 4.30083293e+01  5.27538895e+01]
 [ 5.81499604e+01  7.79888912e+01]
 [-4.68323392e+01  3.26470550e+00]
 [ 4.86839300e+01  5.70361402e+01]
 [ 3.93574747e+01  9.68324926e+01]
 [ 5.29946742e+01 -8.31505018e+00]
 [-6.67424158e+01  3.64979314e+00]
 [ 7.96059721e+01 -1.52870948e+00]
 [ 4.24255349e+01 -2.12575398e+01]
 [-6.55080202e+01 -1.52423251e+01]
 [ 6.51896702e-01 -5.40280148e+01]
 [ 9.64817358e+01  4.71002607e+00]
 [ 4.76352121e+01  3.14780830e+01]
 [ 1.05498513e+02  1.19963276e+01]
 [ 7.29522687e+01 -3.30456486e+00]
 [-5.22148858e+01 -1.48425506e-01]
 [ 7.05786828e+01  2.20568130e+00]
 [ 8.54729041e+01  2.37236163e+00]
 [ 9.20348744e+00 -2.37830390e+01]
 [ 1.08288259e+02  1.21087056e+01]
 [-2.81373132e+01  3.02734330e+01]
 [ 6.34304724e+01  3.71934624e+01]
 [-2.94526860e+01  2.18525324e+00]
 [ 1.02964577e+02 -4.00530742e+00]
 [ 6.46577432e+01  4.54418133e+01]
 [ 7.88586224e+01 -1.07702022e+01]
 [ 1.24894581e+01 -5.61654399e+01]
 [-4.03866345e+01  7.39714899e+01]
 [ 8.13652583e+00  1.07385809e+01]
 [-6.82021795e+01 -7.46347108e-01]
 [ 2.84378435e-01 -5.23803667e+01]
 [ 2.61267106e+01 -1.03872668e+01]
 [-2.59022517e+01  4.79087376e+01]
 [ 5.27696845e+01  5.85309213e+01]
 [-4.94725891e+01  6.08604873e+01]
 [-1.75753198e-01 -3.70489837e+01]
 [ 5.95013797e+01  3.60186991e+01]
 [ 8.05295450e+01  3.06499930e+01]
 [ 8.26161364e+01  2.02776113e+01]
 [-9.16137646e+00  4.10434949e+01]
 [ 5.49975373e+01  4.20930707e+01]
 [ 1.82420469e+01  4.13118801e+01]
 [ 4.94120826e+01  5.58945301e+01]
 [ 2.12316124e+01  6.34367785e+00]
 [ 2.61374490e+01  3.67027046e+01]
```

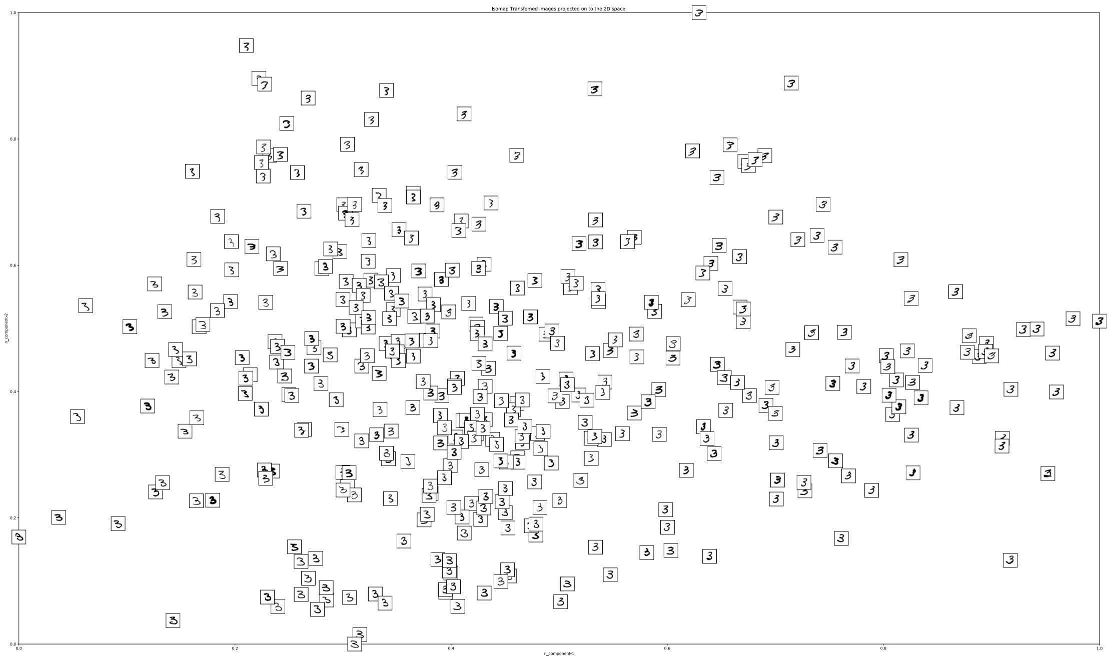
In [68]:
```
dataB_updatedDig3_X_std_iso.shape
```

Out[68]:
```
(398, 2)
```

In [69]:
```
plot_text(dataB_updatedDig3_X_std_iso)
plt.xlabel('n_component-1')
plt.ylabel('n_component-2')
plt.title('Isomap Text Target Projection on the 2D graph')
plt.show()
```



In [70]:
```
# Image projection on to the 2D space of the Isomap transformed dataset
plot_images(dataB_updatedDig3_X_std_iso)
plt.xlabel("n_component-1")
plt.ylabel("n_component-2")
plt.title("Isomap Transfomed images projected on to the 2D space ")
plt.show()
```



Analysis:

The plot above show the Isomap transformed data projected on to the 2D space. It is evident from the plot that the data is widely spread along both the axis. The data which are projected on to the left side of x-axis mainly before 0.2 on the x-axis and in between 0.3 and 0.6 on the y-axis are slightly skewed towards the left and this skewness slightly decreases when we move along the y-axis towards up. In addition, the data between 0.2 and 0.5 are aligned properly with some expception of data that begin to skew slightly towards the right. When we move along the x-axis after 0.5 on the x-axis it is evident that most of the data is skewed towards the right. Another point to note is that in the plot the images near to each other seems alike for example on the x-axis between the scale 0.2 and 0.4 there is a group of data that is skewed with similar skewness so it looks like a group. The Isomap is a Global Method whereas LLE is a Local method or Locally Based as mention in the above analysis.

Q3.3 Naive Bayes classifier using LLE

In [71]:
```
# Split the Dataset into 70% and 30% ratio
dataB_3X_train, dataB_3X_test, dataB_3y_train, dataB_3y_test = train_test_split
(dataB_X, dataB_y, test_size=0.30, random_state=42)

print("\n Dimension of Training set: ",dataB_3X_train.shape)
```

```
 Dimension of Training set:  (1446, 784)
```

In [72]:
```
dataB_3y_train_values = dataB_3y_train.values
dataB_3y_test_values = dataB_3y_test.values
print("Dimension of Training Labels: ",dataB_3y_train_values.shape)
```

```
Dimension of Training Labels:  (1446,)
```

In [73]:
```
ss = StandardScaler()
dataB_3X_train_std = ss.fit_transform(dataB_3X_train)
dataB_3X_test_std = ss.fit_transform(dataB_3X_test)
```

In [74]:
```
lle_q3 = LocallyLinearEmbedding(n_neighbors=5, n_components=4, eigen_solver='au
to', n_jobs=-1)
dataB_3X_train_std_lleq3 = lle_q3.fit_transform(dataB_3X_train_std)
dataB_3X_test_std_lleq3 = lle_q3.fit_transform(dataB_3X_test_std)
```

In [75]:
```
print("\n Dimension of LLE transformed dataset: ",dataB_3X_train_std_lleq3.shap
e)
```

```
 Dimension of LLE transformed dataset:  (1446, 4)
```

In [76]:
```
# Initialize the Gaussian Naive Bayes Classifier
gnb_lleq3 = GaussianNB()
gnb_lleq3.fit(dataB_3X_train_std_lleq3, dataB_3y_train_values)
lleq3_y_pred = gnb_lleq3.predict(dataB_3X_test_std_lleq3)
```

In [77]:
```
# Accuracy Score for Isomap dataset with Naive bayes classifier
accuracy_score_lleq3 = accuracy_score(dataB_3y_test_values, lleq3_y_pred)
print(" \n Accuracy Score for LLE Transformed dataset using Naive Bayes: {0:.3
f}".format(accuracy_score_lleq3 * 100))
```

```
 Accuracy Score for LLE Transformed dataset using Naive Bayes: 11.129
```

Training the classifier with different folds of datasets with LLE

In [78]:
```
# Training with Straitified K fold as this preserves the class distribution
kfold = StratifiedKFold(n_splits = 10).split(dataB_3X_train_std_lleq3, dataB_3y
_train_values)
Accuracy_Score = []

for index, (train, test) in enumerate(kfold):
    gnb_lleq3.fit(dataB_3X_train_std_lleq3[train], dataB_3y_train_values[trai
n])
    lleq_y_pred = gnb_lleq3.predict(dataB_3X_train_std_lleq3[test])
    accuracy_score_kfold = accuracy_score(lleq_y_pred, dataB_3y_train_values[te
st])
    Accuracy_Score.append(accuracy_score_kfold)
    sys.stderr.write('\n Fold: {0}, Class Distribution: {1}, Accuracy_Score:
{2:.3f}'.format(index+1, np.bincount(dataB_3y_train_values[train]), accuracy_sc
ore_kfold))
    sys.stderr.flush()

sys.stderr.write("\n Mean Accuracy: {0:.3f} +/- {1:0.3f}".format(np.mean(Accura
cy_Score), np.std(Accuracy_Score)))
sys.stderr.flush()
```

```
 Fold: 1, Class Distribution: [228 280 282 235 276], Accuracy_Score: 0.807
 Fold: 2, Class Distribution: [228 280 282 235 276], Accuracy_Score: 0.855
 Fold: 3, Class Distribution: [228 280 281 236 276], Accuracy_Score: 0.834
 Fold: 4, Class Distribution: [228 280 281 236 276], Accuracy_Score: 0.848
 Fold: 5, Class Distribution: [228 280 281 236 276], Accuracy_Score: 0.793
 Fold: 6, Class Distribution: [227 280 282 236 276], Accuracy_Score: 0.814
 Fold: 7, Class Distribution: [227 280 282 236 277], Accuracy_Score: 0.819
 Fold: 8, Class Distribution: [227 280 282 236 277], Accuracy_Score: 0.840
 Fold: 9, Class Distribution: [228 279 282 236 277], Accuracy_Score: 0.826
 Fold: 10, Class Distribution: [228 280 282 236 276], Accuracy_Score: 0.792
 Mean Accuracy: 0.823 +/- 0.021
```

Analysis:

In the above code the training dataset is split into 10 folds and then the training of the Classifier is performed. Here with 10 fold the Mean Accuracy of the Classifier is 82.3% with standard devaition of 2.2%. One main reason for choosing 10 iteration is to put in account the variation in accuracy of the completed dataset.

Q3.3 Naive Bayes classifier using IsoMap

In [79]:
```
isomap_q3 = Isomap(n_neighbors=5, n_components=4, eigen_solver='auto', n_jobs=-
1)
dataB_3X_train_std_isoq3 = isomap_q3.fit_transform(dataB_3X_train_std)
dataB_3X_test_std_isoq3 = isomap_q3.fit_transform(dataB_3X_test_std)
```

In [80]:
```
gnb_isoq3 = GaussianNB()
gnb_isoq3.fit(dataB_3X_train_std_isoq3, dataB_3y_train_values)
isoq3_y_pred = gnb_isoq3.predict(dataB_3X_test_std_isoq3)
```

In [81]:
```
accuracy_score_isoq3 = accuracy_score(dataB_3y_test_values, isoq3_y_pred)
print(" \n Accuracy Score for Isomap Transformed dataset using Naive Bayes:
{0:.3f}", accuracy_score_isoq3 * 100)
```

```
 Accuracy Score for Isomap Transformed dataset using Naive Bayes: {0:.3f} 68.06
451612903226
```

Training the classifier with different folds of datasets

In [82]:
```
kfold = StratifiedKFold(n_splits = 10).split(dataB_3X_train_std_isoq3, dataB_3y
_train_values)
Accuracy_Score_Isomapq3 = []

for index, (train, test) in enumerate(kfold):
    gnb_isoq3.fit(dataB_3X_train_std_isoq3[train], dataB_3y_train_values[trai
n])
    isoq3_y_pred = gnb_isoq3.predict(dataB_3X_train_std_isoq3[test])
    accuracy_score_kfold_isomapq3 = accuracy_score(isoq3_y_pred, dataB_3y_train
_values[test])
    Accuracy_Score_Isomapq3.append(accuracy_score_kfold_isomapq3)
    sys.stderr.write('\n Fold: {0}, Class Distribution: {1}, Accuracy_Score:
{2:.3f}'.format(index+1, np.bincount(dataB_3y_train_values[train]), accuracy_sc
ore_kfold_isomapq3))
    sys.stderr.flush()

sys.stderr.write("\n Mean Accuracy: {0:.3f} +/- {1:0.3f}".format(np.mean(Accura
cy_Score_Isomapq3), np.std(Accuracy_Score_Isomapq3)))
sys.stderr.flush()
```

```
 Fold: 1, Class Distribution: [228 280 282 235 276], Accuracy_Score: 0.821
 Fold: 2, Class Distribution: [228 280 282 235 276], Accuracy_Score: 0.807
 Fold: 3, Class Distribution: [228 280 281 236 276], Accuracy_Score: 0.821
 Fold: 4, Class Distribution: [228 280 281 236 276], Accuracy_Score: 0.834
 Fold: 5, Class Distribution: [228 280 281 236 276], Accuracy_Score: 0.855
 Fold: 6, Class Distribution: [227 280 282 236 276], Accuracy_Score: 0.834
 Fold: 7, Class Distribution: [227 280 282 236 277], Accuracy_Score: 0.806
 Fold: 8, Class Distribution: [227 280 282 236 277], Accuracy_Score: 0.889
 Fold: 9, Class Distribution: [228 279 282 236 277], Accuracy_Score: 0.785
 Fold: 10, Class Distribution: [228 280 282 236 276], Accuracy_Score: 0.875
 Mean Accuracy: 0.833 +/- 0.031
```

Analysis:

The training dataset is split into 10 folds and then the training of the Classifier is performed with Isomap transformed dataset. Here with 10 fold the Mean Accuracy of the Classifier is 83.3% with standard devaition of 3.1%. One main reason for choosing 10 iteration is to put in account the variation in accuracy over the complete dataset.

Q3.3 Naive Bayes classifier using PCA transformed dataset

In [83]:
```
pca_q3 = PCA(n_components=4)
dataB_3X_train_std_pcaq3 = pca_q3.fit_transform(dataB_3X_train_std)
dataB_3X_test_std_pcaq3 = pca_q3.fit_transform(dataB_3X_test_std)
```

In [84]:
```
gnb_pcaq3 = GaussianNB()
gnb_pcaq3.fit(dataB_3X_train_std_pcaq3, dataB_3y_train_values)
isoq3_y_pred = gnb_pcaq3.predict(dataB_3X_test_std_pcaq3)
```

In [85]:
```
accuracy_score_isoq3 = accuracy_score(dataB_3y_test_values, isoq3_y_pred)
print(" \n Accuracy Score for Isomap Transformed dataset using Naive Bayes:
{0:.3f}", accuracy_score_isoq3 * 100)
```

```
 Accuracy Score for Isomap Transformed dataset using Naive Bayes: {0:.3f} 43.38
709677419355
```

Training the classifier with different folds of datasets

In [86]:
```
kfold = StratifiedKFold(n_splits = 10).split(dataB_3X_train_std_pcaq3, dataB_3y
_train_values)
Accuracy_Score_PCAq3 = []

for index, (train, test) in enumerate(kfold):
    gnb_pcaq3.fit(dataB_3X_train_std_pcaq3[train], dataB_3y_train_values[trai
n])
    pcaq3_y_pred = gnb_pcaq3.predict(dataB_3X_train_std_pcaq3[test])
    accuracy_score_kfold_q3_pca= accuracy_score(pcaq3_y_pred, dataB_3y_train_va
lues[test])
    Accuracy_Score_PCAq3.append(accuracy_score_kfold_q3_pca)
    sys.stderr.write('\n Fold: {0}, Class Distribution: {1}, Accuracy_Score:
{2:.3f}'.format(index+1, np.bincount(dataB_3y_train_values[train]), accuracy_sc
ore_kfold_q3_pca))
    sys.stderr.flush()

sys.stderr.write("\n Mean Accuracy: {0:.3f} +/- {1:0.3f}".format(np.mean(Accura
cy_Score_PCAq3), np.std(Accuracy_Score_PCAq3)))
sys.stderr.flush()
```

```
 Fold: 1, Class Distribution: [228 280 282 235 276], Accuracy_Score: 0.779
 Fold: 2, Class Distribution: [228 280 282 235 276], Accuracy_Score: 0.821
 Fold: 3, Class Distribution: [228 280 281 236 276], Accuracy_Score: 0.807
 Fold: 4, Class Distribution: [228 280 281 236 276], Accuracy_Score: 0.841
 Fold: 5, Class Distribution: [228 280 281 236 276], Accuracy_Score: 0.834
 Fold: 6, Class Distribution: [227 280 282 236 276], Accuracy_Score: 0.821
 Fold: 7, Class Distribution: [227 280 282 236 277], Accuracy_Score: 0.819
 Fold: 8, Class Distribution: [227 280 282 236 277], Accuracy_Score: 0.826
 Fold: 9, Class Distribution: [228 279 282 236 277], Accuracy_Score: 0.847
 Fold: 10, Class Distribution: [228 280 282 236 276], Accuracy_Score: 0.826
 Mean Accuracy: 0.822 +/- 0.018
```

Analysis:

In order to consistent with other methods the training dataset is split into 10 folds and then the training of the Classifier is performed with PCA transformed dataset. Here with 10 fold the Mean Accuracy of the Classifier is 82.2% with standard devaition of 1.8%. One main reason for choosing 10 iteration is to put in account the variation in accuracy over the complete dataset.

Q3.3 Naive Bayes classifier using LDA transformed dataset

In [87]:
```
# LDA transformation on the train and test data
lda_q3 = LDA(n_components=4)
dataB_3X_train_std_ldaq3 = lda_q3.fit_transform(dataB_3X_train_std, dataB_3y_tr
ain_values)
dataB_3X_test_std_ldaq3 = lda_q3.fit_transform(dataB_3X_test_std, dataB_3y_test
_values)
```

In [88]:
```
# Classifier initialiation
gnb_ldaq3 = GaussianNB()
gnb_ldaq3.fit(dataB_3X_train_std_ldaq3, dataB_3y_train_values)
ldaq3_y_pred = gnb_ldaq3.predict(dataB_3X_test_std_ldaq3)
```

In [89]:
```
# Accuracy Calculation of LDA
accuracy_score_ldaq3 = accuracy_score(dataB_3y_test_values, ldaq3_y_pred)
print(" \n Accuracy Score for LDA Transformed dataset using Naive Bayes: {0:.3
f}", accuracy_score_ldaq3 * 100)
```

```
 Accuracy Score for LDA Transformed dataset using Naive Bayes: {0:.3f} 100.0
```

Training the classifier with different folds of datasets

In [90]:
```
# Training with Straitified K fold as this preserves the class distribution
kfold = StratifiedKFold(n_splits = 10).split(dataB_3X_train_std_ldaq3, dataB_3y
_train_values)
Accuracy_Score_LDA = []

for index, (train, test) in enumerate(kfold):
    gnb_ldaq3.fit(dataB_3X_train_std_ldaq3[train], dataB_3y_train_values[trai
n])
    ldaq_y_pred = gnb_ldaq3.predict(dataB_3X_train_std_ldaq3[test])
    accuracy_score_kfold_lda = accuracy_score(ldaq_y_pred, dataB_3y_train_value
s[test])
    Accuracy_Score_LDA.append(accuracy_score_kfold_lda)
    sys.stderr.write('\n Fold: {0}, Class Distribution: {1}, Accuracy_Score:
{2:.3f}'.format(index+1, np.bincount(dataB_3y_train_values[train]), accuracy_sc
ore_kfold_lda))
    sys.stderr.flush()

sys.stderr.write("\n Mean Accuracy: {0:.3f} +/- {1:0.3f}".format(np.mean(Accura
cy_Score_LDA), np.std(Accuracy_Score_LDA)))
sys.stderr.flush()
```

```
 Fold: 1, Class Distribution: [228 280 282 235 276], Accuracy_Score: 1.000
 Fold: 2, Class Distribution: [228 280 282 235 276], Accuracy_Score: 1.000
 Fold: 3, Class Distribution: [228 280 281 236 276], Accuracy_Score: 1.000
 Fold: 4, Class Distribution: [228 280 281 236 276], Accuracy_Score: 1.000
 Fold: 5, Class Distribution: [228 280 281 236 276], Accuracy_Score: 1.000
 Fold: 6, Class Distribution: [227 280 282 236 276], Accuracy_Score: 1.000
 Fold: 7, Class Distribution: [227 280 282 236 277], Accuracy_Score: 1.000
 Fold: 8, Class Distribution: [227 280 282 236 277], Accuracy_Score: 1.000
 Fold: 9, Class Distribution: [228 279 282 236 277], Accuracy_Score: 1.000
 Fold: 10, Class Distribution: [228 280 282 236 276], Accuracy_Score: 1.000
 Mean Accuracy: 1.000 +/- 0.000
```

Analysis:

The training dataset is split into 10 folds and then the training of the Classifier is performed with LDA transformed dataset with 4 components. Here with 10 fold the Mean Accuracy of the Classifier is 100.0% with no standard devaition.

Analysis:

Among all the transformed dataset using LLE, Isomap, PCA and LDA, Naive Bayes(NB) classifier works best with LDA transformed dataset as the Mean Accuracy score is 100%. In addition, the NB classifier gives an Mean Accuracy score of 82.2% on PCA transformed dataset which is similar to that of LLE transformed dataset as for this the NB classifier gives a Mean accuracy of 82.3%. For Isomap transformed dataset the NB classifier gives Mean accuracy of 83.3% which slighlty better than that of PCA based NB classifier and LLE based NB classifier.