q1

February 25, 2021

[1]:
```
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.manifold import MDS
import matplotlib.pyplot as plt
from time import time
```

Part A

[2]:
```
# load the digits dataset
digits = load_digits()
feats = digits.data
targets = digits.target
names = digits.target_names
```

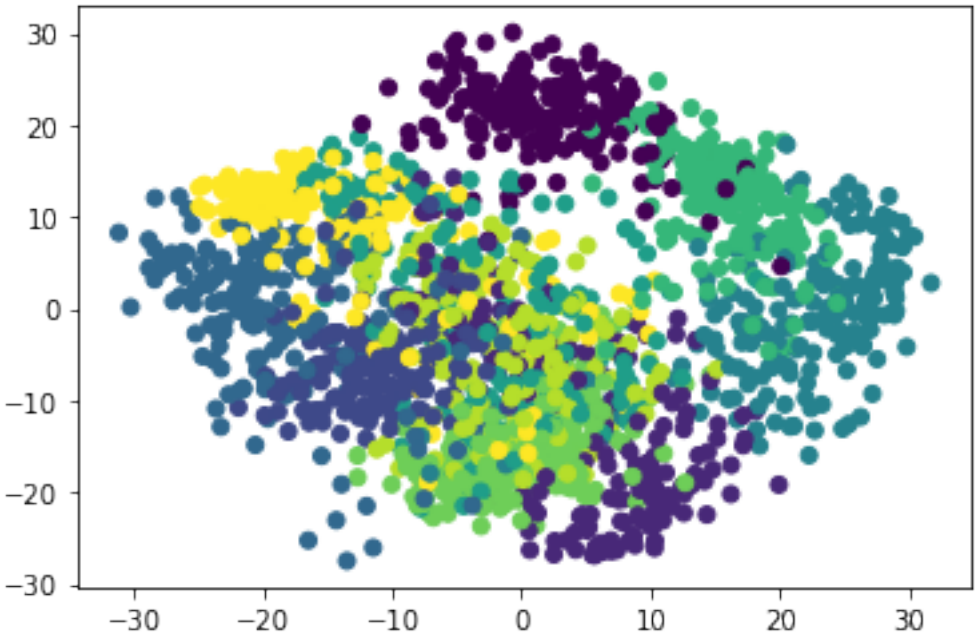
[3]:
```
# calculate pca runtime
pca_start = time()

# dedimensionalize data using PCA
pca = PCA(n_components = 2)
feats_transformed = pca.fit_transform(feats)

pca_end = time()

print("Time taken to run PCA: {} seconds".format(pca_end - pca_start))
```

```
Time taken to run PCA: 0.7480008602142334 seconds
```

[4]:
```
colors = ['blue', 'red', 'orange', 'green', 'pink', 'yellow', 'black',␣
 ↪'turquoise', 'navy', 'indigo']
lw = 2
```

[5]:
```
plt.scatter(feats_transformed[:, 0], feats_transformed[:, 1], c=targets)
```

[5]:
```
<matplotlib.collections.PathCollection at 0x271240ee370>
```



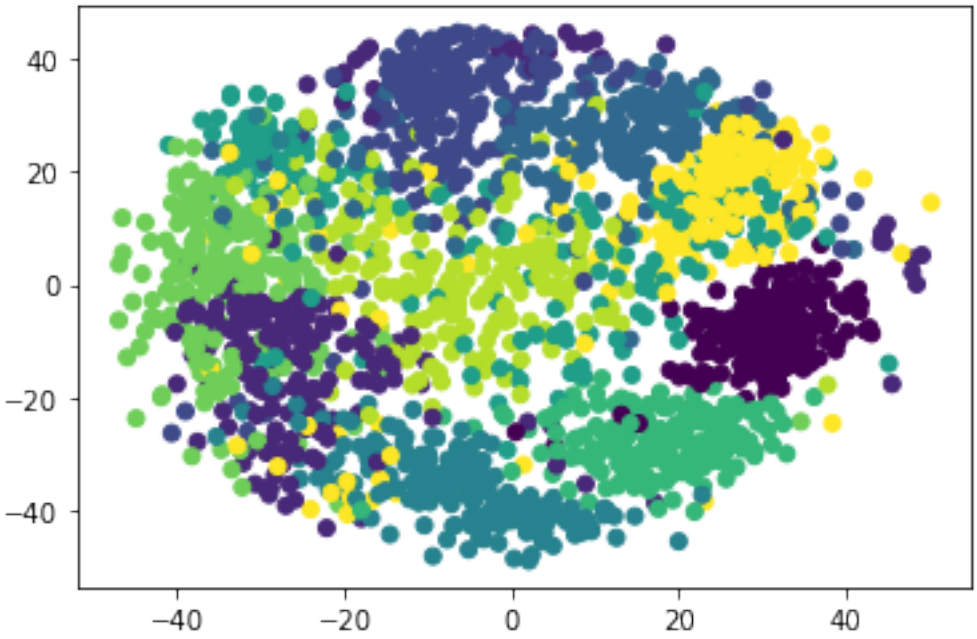
Part B

[6]:
```
# repeat the above but using multidimensional scaling

# calculate mds runtime
mds_start = time()

mds = MDS(n_components=2, n_jobs=20)
feats_transformed = mds.fit_transform(feats)

mds_end = time()

print("Time taken to run MDS: {} seconds".format(mds_end - mds_start))
```

```
Time taken to run MDS: 116.73923897743225 seconds
```

[7]:
```
plt.scatter(feats_transformed[:, 0], feats_transformed[:, 1], c=targets)
```

[7]:
```
<matplotlib.collections.PathCollection at 0x2712429d280>
```



Part C

Based on the plots above, I do believe that these modelling techniques would be able to differentiate among classes with some success. We can see distinct "clouds" of color that represent the different classes, and in many places, they do not overlap, suggesting that the modeling technique is able to, for the most part, group classes into separate groups. There is still a fair amount of "overlap" though, which means that we are bound to get a good amount of wrong predictions as well, because the model will have trouble determining which class an input belongs to.

MDS seems to do a much better job of suggesting spatial separation among the various digits. The data groupings are much more tighly knit, as shown by the MDS scatterplot versus that of PCA. Overlap between classes is less common, and space between points in preserved. Because groups of data points are better preserved, it is likely that this modelling technique would better be able to differentiate between classes.

The runtime of MDS is much greater than that of PCA, suggesting that a more complicated algorithm allows better preservation of space between data classes. PCA took about 2 seconds on the IPython compiler, whereas MDS took about 130 seconds.

q2

February 25, 2021

[1]:
```
from sklearn.decomposition import PCA
from numpy import genfromtxt as readfile
from numpy import cumsum as cumulative
from numpy import arange
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

[2]:
```
pca_data = readfile('pcaData.csv', delimiter=',')
```

Part A

[3]:
```
pca = PCA().fit(pca_data)
```

[4]:
```
explained_variance = pca.explained_variance_ratio_
explained_variance
```

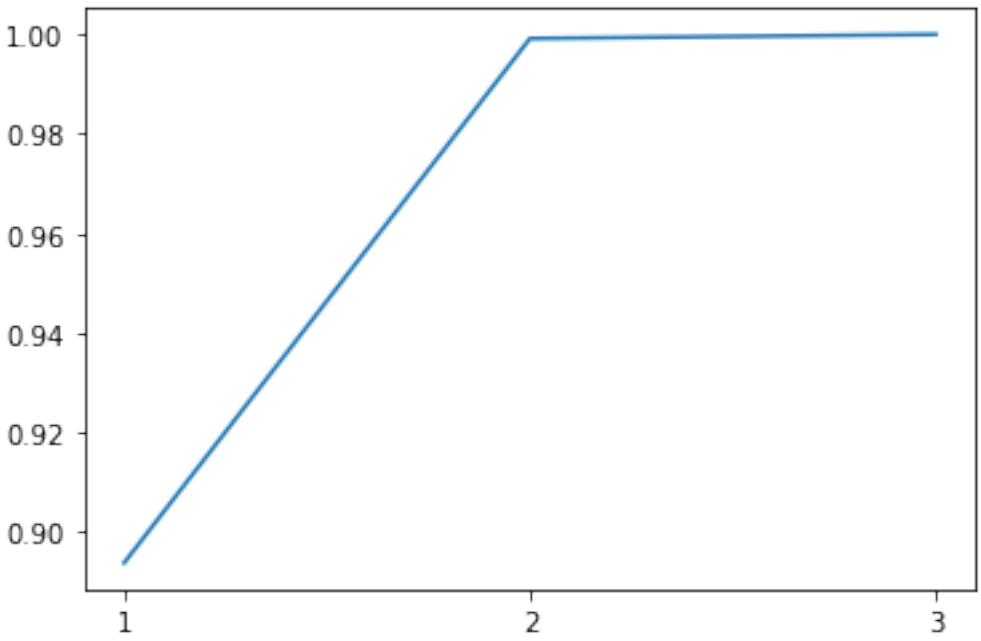
[4]:
```
array([8.93824494e-01, 1.05354439e-01, 8.21066741e-04])
```

[5]:
```
# print out the preserved variances
for index, variance in enumerate(explained_variance):
    print("Variance preserved in component {}: {}".format(index + 1,␣
 ↪format(variance)))
```

```
Variance preserved in component 1: 0.893824493780275
Variance preserved in component 2: 0.10535443947879294
Variance preserved in component 3: 0.0008210667409321359
```

[6]:
```
plt.plot(cumulative(explained_variance))
plt.xticks(arange(3), ['1', '2', '3'])
```

[6]:
```
([<matplotlib.axis.XTick at 0x1d5e1788190>,
  <matplotlib.axis.XTick at 0x1d5e1788160>,
  <matplotlib.axis.XTick at 0x1d5e151dc70>],
 [Text(0, 0, '1'), Text(1, 0, '2'), Text(2, 0, '3')])
```



Based on the results, 2 is the minimum number of dimensions which would be sufficient to capture at least 99% of the data. The first components preserves about 89% of the variance (a little over), whereas the second preserves a little over 10% of the variance. Thus, when we sum the variations represented by these 2 dimensions, we get a little over 99%. No one dimnesion preserves 99% of the data, but 1 and 2 together preserve over 99%.

Part B

[10]:
```
%matplotlib notebook

fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")

x = pca_data[:, 0]
y = pca_data[:, 1]
z = pca_data[:, 2]

ax.set_xlabel('X-axis', fontweight ='bold')
ax.set_ylabel('Y-axis', fontweight ='bold')
ax.set_zlabel('Z-axis', fontweight ='bold')

scatter = ax.scatter(x, y, z, c=(x + y + z), alpha=.8)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

The fact that the datapoints form a hyperplane best supports the findings of part B. PCA dedi-

mensionalizes the original data while trying to preserve the original shape of the data. Because of the shape of the data as a hyperplane in 3D space, the z dimension has little effect on the variation of the data, since the majority of the variation (over 99%) is contained in the x and y axes of the data. It is this relationship that gives the data its shape. Thus, when we perform PCA on the data, the axes that account for the variation of the data are the first two, and we need only these two in order to explain most of the variance in the dataset.

q3

February 25, 2021

[1]:
```
from sklearn.datasets import load_iris
from numpy import cov as covM
from numpy import corrcoef as corrM
import matplotlib.pyplot as plt
from numpy import array
```

Part A

[2]:
```
iris_data = load_iris()
# iris_data
```

[3]:
```
iris_feats = iris_data.data
iris_targets = iris_data.target
```

[4]:
```
covariance_matrix = covM(iris_feats, rowvar=False)
print("Covariance matrix of the Iris Dataset\n\n", covariance_matrix)
```

```
Covariance matrix of the Iris Dataset

 [[ 0.68569351 -0.042434    1.27431544  0.51627069]
 [-0.042434    0.18997942 -0.32965638 -0.12163937]
 [ 1.27431544 -0.32965638  3.11627785  1.2956094 ]
 [ 0.51627069 -0.12163937  1.2956094   0.58100626]]
```

[5]:
```
correlation_matrix = corrM(iris_feats, rowvar=False)
print("Correlation matrix of the Iris Dataset\n\n", correlation_matrix)
```

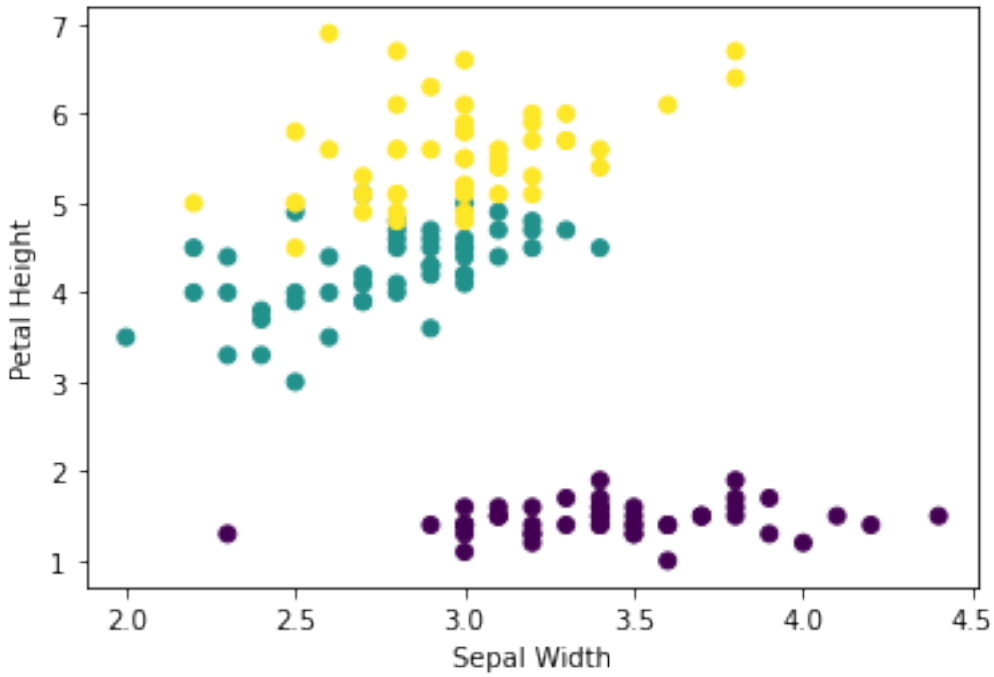
```
Correlation matrix of the Iris Dataset

 [[ 1.         -0.11756978  0.87175378  0.81794113]
 [-0.11756978  1.         -0.4284401  -0.36612593]
 [ 0.87175378 -0.4284401   1.          0.96286543]
 [ 0.81794113 -0.36612593  0.96286543  1.        ]]
```

Part B

[6]:
```
# focusing on attributes 1 and 2

attr1_column = iris_feats[:, 1]
attr2_column = iris_feats[:, 2]
```

```
plt.xlabel('Sepal Width')
plt.ylabel('Petal Height')

plt.scatter(attr1_column, attr2_column, c=iris_targets)
```

[6]: <matplotlib.collections.PathCollection at 0x1698c2538b0>



What is surprising is that the correlation matrix for the Iris dataset gives a negative covariance between sepal length and sepal width (about -.11), however, when we look at each class alone, the relation appears to be positively correlated (even to a high magnitude). So, the two variables are negatively correlated on the whole of the dataset (as determined above) but appear to be positively correlated when looking at the class alone. The scatterplot reflects this global negative covariance (note that it is to a small magnitude) but positive correlation among individual classes.

Part C

[7]:
```
target_names = iris_data.target_names
label_dict = {identifier : target_names[identifier] for identifier in range(3)}

label_dict
```

[7]: {0: 'setosa', 1: 'versicolor', 2: 'virginica'}

[8]:
```
print("Correlation matrix: \nrow1, col1 = sepal width\nrow2, col2 = petal␣
 ↪height")

# for each class, compute the correlation coefficient
for key in label_dict.keys():

    print("\nCorrelation matrix for {}".format(label_dict[key]))

    attr1 = attr1_column[iris_targets == key]
    attr2 = attr2_column[iris_targets == key]

    corr_matrix = corrM(attr1, attr2, rowvar=False)
    print(corr_matrix)
```

```
Correlation matrix:
row1, col1 = sepal length
row2, col2 = sepal width

Correlation matrix for setosa
[[1.         0.17769997]
 [0.17769997 1.        ]]

Correlation matrix for versicolor
[[1.         0.56052209]
 [0.56052209 1.        ]]

Correlation matrix for virginica
[[1.         0.40104458]
 [0.40104458 1.        ]]
```

The per-class correlations all are positive and of relatively high magnitude, suggesting that sepal length and width are positively correlated when analyzing individual classes. This is different from the overall correlation, as explained in the preceding part, in which the correlation was negative. The results from this part of the question appear to confirm our hypothesis from the previous part in that sepal length and width are positively correlated for individual classes. We could see this relationship in the previous plot by looking at the individual color-coded classes and noting that as sepal length increases, sepal width tends to increase as well, suggesting positive correlation