cmps320_hw2_SammanBhetwal

October 8, 2024

0.0.1 Question number 2

Imports

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn import decomposition
from sklearn import preprocessing
from sklearn import metrics
%matplotlib inline
plt.style.use('seaborn-white')
```

Q.2 a) Load the training set optdigits.tra, which has sixty-four (p = 64) inputs plus the target variable that indicates the digit 0-9

[2]:
```
data_values = pd.read_csv('optdigits.tra', header=None)
data_values.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3823 entries, 0 to 3822
Data columns (total 65 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       3823 non-null   int64
 1   1       3823 non-null   int64
 2   2       3823 non-null   int64
 3   3       3823 non-null   int64
 4   4       3823 non-null   int64
 5   5       3823 non-null   int64
 6   6       3823 non-null   int64
 7   7       3823 non-null   int64
 8   8       3823 non-null   int64
 9   9       3823 non-null   int64
 10  10      3823 non-null   int64
 11  11      3823 non-null   int64
 12  12      3823 non-null   int64
 13  13      3823 non-null   int64
```

```
14  14      3823 non-null   int64
15  15      3823 non-null   int64
16  16      3823 non-null   int64
17  17      3823 non-null   int64
18  18      3823 non-null   int64
19  19      3823 non-null   int64
20  20      3823 non-null   int64
21  21      3823 non-null   int64
22  22      3823 non-null   int64
23  23      3823 non-null   int64
24  24      3823 non-null   int64
25  25      3823 non-null   int64
26  26      3823 non-null   int64
27  27      3823 non-null   int64
28  28      3823 non-null   int64
29  29      3823 non-null   int64
30  30      3823 non-null   int64
31  31      3823 non-null   int64
32  32      3823 non-null   int64
33  33      3823 non-null   int64
34  34      3823 non-null   int64
35  35      3823 non-null   int64
36  36      3823 non-null   int64
37  37      3823 non-null   int64
38  38      3823 non-null   int64
39  39      3823 non-null   int64
40  40      3823 non-null   int64
41  41      3823 non-null   int64
42  42      3823 non-null   int64
43  43      3823 non-null   int64
44  44      3823 non-null   int64
45  45      3823 non-null   int64
46  46      3823 non-null   int64
47  47      3823 non-null   int64
48  48      3823 non-null   int64
49  49      3823 non-null   int64
50  50      3823 non-null   int64
51  51      3823 non-null   int64
52  52      3823 non-null   int64
53  53      3823 non-null   int64
54  54      3823 non-null   int64
55  55      3823 non-null   int64
56  56      3823 non-null   int64
57  57      3823 non-null   int64
58  58      3823 non-null   int64
59  59      3823 non-null   int64
60  60      3823 non-null   int64
61  61      3823 non-null   int64
```

```
 62  62      3823 non-null   int64
 63  63      3823 non-null   int64
 64  64      3823 non-null   int64
dtypes: int64(65)
memory usage: 1.9 MB
```

[4]:
```
# we will now display the first five rows of the DataFrame
# we are doing this to inspect the structure and verify if the data has been␣
 ↪loaded correctly
data_values.head()
```

[4]:

	0	1	2	3	4	5	6	7	8	9	…	55	56	57	58	59	60	61	\
0	0	1	6	15	12	1	0	0	0	7	…	0	0	0	6	14	7	1	
1	0	0	10	16	6	0	0	0	0	7	…	0	0	0	10	16	15	3	
2	0	0	8	15	16	13	0	0	0	1	…	0	0	0	9	14	0	0	
3	0	0	0	3	11	16	0	0	0	0	…	0	0	0	0	1	15	2	
4	0	0	5	14	4	0	0	0	0	0	…	0	0	0	4	12	14	7	

	62	63	64
0	0	0	0
1	0	0	0
2	0	0	7
3	0	0	4
4	0	0	6

[5 rows x 65 columns]

[5]:
```
# since it has we will move forward to the next part of the problem
```

Q 2b) Remove any unary column (i.e., containing only one value)

[6]:
```
unary_cols = data_values.columns[data_values.nunique() == 1]
data_values = data_values.drop(columns=unary_cols)
data_values
```

[6]:

	1	2	3	4	5	6	7	8	9	10	…	55	56	57	58	59	60	61	\
0	1	6	15	12	1	0	0	0	7	16	…	0	0	0	6	14	7	1	
1	0	10	16	6	0	0	0	0	7	16	…	0	0	0	10	16	15	3	
2	0	8	15	16	13	0	0	0	1	11	…	0	0	0	9	14	0	0	
3	0	0	3	11	16	0	0	0	0	5	…	0	0	0	0	1	15	2	
4	0	5	14	4	0	0	0	0	0	13	…	0	0	0	4	12	14	7	
…	..	..	..	..	..	..	..	..	..	..	…	..	..	..	..	..	..	..	
3818	0	5	13	11	2	0	0	0	2	15	…	0	0	0	8	13	15	10	
3819	0	0	1	12	1	0	0	0	0	0	…	0	0	0	0	4	9	0	
3820	0	3	15	0	0	0	0	0	0	11	…	0	0	0	4	14	16	9	
3821	0	6	16	2	0	0	0	0	0	15	…	0	0	0	5	16	16	16	
3822	0	2	15	16	13	1	0	0	0	3	…	0	0	0	4	14	1	0	

```
      62  63  64
0      0   0   0
1      0   0   0
2      0   0   7
3      0   0   4
4      0   0   6
…     ..  ..  ..
3818   1   0   9
3819   0   0   4
3820   0   0   6
3821   5   0   6
3822   0   0   7

[3823 rows x 63 columns]
```

[7]:
```
#Now we need to find out any missing values
```

Q2 c) Are there any missing values?

[8]:
```
missing_vals = data_values.isnull().sum().sum()
print(f"Total missing values in the dataset: {missing_vals}")
```

```
Total missing values in the dataset: 0
```

Q2 d) Data preprocessing: Check to see if the data is standardized. If not, standardize the data matrix X so that all variables are given a mean of zero and a standard deviation of one. Remember to exclude the target variable. After identifying the target column from optdigits.names, we will move forward

[10]:
```
X = data_values.iloc[:, :-1]
y = data_values.iloc[:, -1]
X.describe()
```

[10]:
```
                1            2            3            4            5   \
count  3823.000000  3823.000000  3823.000000  3823.000000  3823.000000
mean      0.301334     5.481821    11.805912    11.451478     5.505362
std       0.866986     4.631601     4.259811     4.537556     5.613060
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     1.000000    10.000000     9.000000     0.000000
50%       0.000000     5.000000    13.000000    13.000000     4.000000
75%       0.000000     9.000000    15.000000    15.000000    10.000000
max       8.000000    16.000000    16.000000    16.000000    16.000000

                6            7            8            9            10  …  \
count  3823.000000  3823.000000  3823.000000  3823.000000  3823.000000  …
mean      1.387392     0.142297     0.002093     1.960502    10.577295  …
std       3.371444     1.051598     0.088572     3.052353     5.435481  …
min       0.000000     0.000000     0.000000     0.000000     0.000000  …
```

```
25%       0.000000     0.000000     0.000000     0.000000     7.000000  …
50%       0.000000     0.000000     0.000000     0.000000    13.000000  …
75%       0.000000     0.000000     0.000000     3.000000    15.000000  …
max      16.000000    16.000000     5.000000    15.000000    16.000000  …

                54           55           56           57           58  \
count  3823.000000  3823.000000  3823.000000  3823.000000  3823.000000
mean      3.743918     0.148313     0.000262     0.283024     5.855872
std       4.901657     0.767761     0.016173     0.928046     4.980012
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     1.000000
50%       1.000000     0.000000     0.000000     0.000000     5.000000
75%       7.000000     0.000000     0.000000     0.000000    10.000000
max      16.000000    12.000000     1.000000    10.000000    16.000000

                59           60           61           62           63
count  3823.000000  3823.000000  3823.000000  3823.000000  3823.000000
mean     11.942977    11.461156     6.700497     2.105676     0.202197
std       4.334508     4.991934     5.775815     4.028266     1.150694
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%      10.000000     9.000000     0.000000     0.000000     0.000000
50%      13.000000    13.000000     6.000000     0.000000     0.000000
75%      15.000000    16.000000    12.000000     2.000000     0.000000
max      16.000000    16.000000    16.000000    16.000000    16.000000

[8 rows x 62 columns]
```

From the above observations we can see that the data is not standardized as most of the columns do not have mean whose absolute value equal to 0 or standard deviation whose absolute value that equal to 1. Therefore we will standardize the data.

[13]:
```
scaler = StandardScaler()
scaled_X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

[14]:
```
scaled_X.mean()
```

[14]:
```
1     6.870140e-16
2    -1.158721e-17
3    -1.623371e-16
4     2.239032e-16
5    -1.388142e-17
          …
59    4.445103e-16
60    3.178206e-16
61   -1.305086e-16
62    3.680318e-16
63   -1.956307e-15
Length: 62, dtype: float64
```

[15]:
```
scaled_X.std()
```

[15]:
```
1     1.000131
2     1.000131
3     1.000131
4     1.000131
5     1.000131
        …
59    1.000131
60    1.000131
61    1.000131
62    1.000131
63    1.000131
Length: 62, dtype: float64
```

Q2 e) PCA:

i) Run the PCA algorithm on the standardized data with number of components equal to the total number of columns in the data.

[17]:
```
p_c_a = decomposition.PCA(n_components=62)
PC = p_c_a.fit_transform(X_scaled)
```

ii) Plot the Proportion of Variance Explained (PVE) of each principal component (i.e., a scree plot) and the cumulative PVE of each principal component

[18]:
```
proportion_explained_variance = p_c_a.explained_variance_ratio_
```

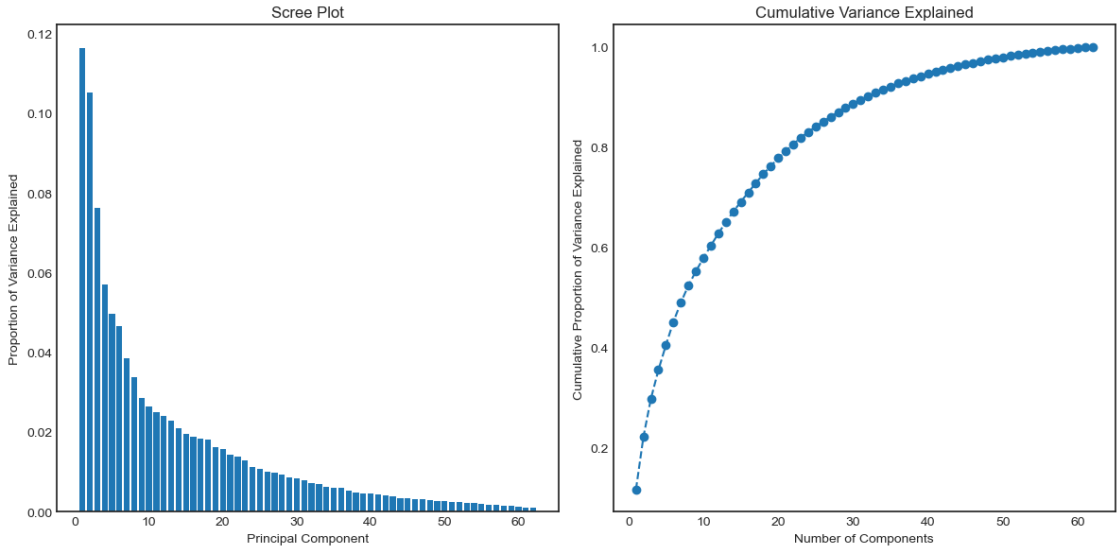
[19]:
```
cumulative_variance = np.cumsum(proportion_explained_variance)

# Plotting
plt.figure(figsize=(12, 6))

# Scree plot
plt.subplot(1, 2, 1)
plt.bar(range(1, len(proportion_explained_variance)+1),␣
 ↪proportion_explained_variance)
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.title('Scree Plot')

# Cumulative Variance explained
plt.subplot(1, 2, 2)
plt.plot(range(1, len(cumulative_variance)+1), cumulative_variance,␣
 ↪marker='o',linestyle='--')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Proportion of Variance Explained')
plt.title('Cumulative Variance Explained')
plt.tight_layout()
```

```
plt.show()
```



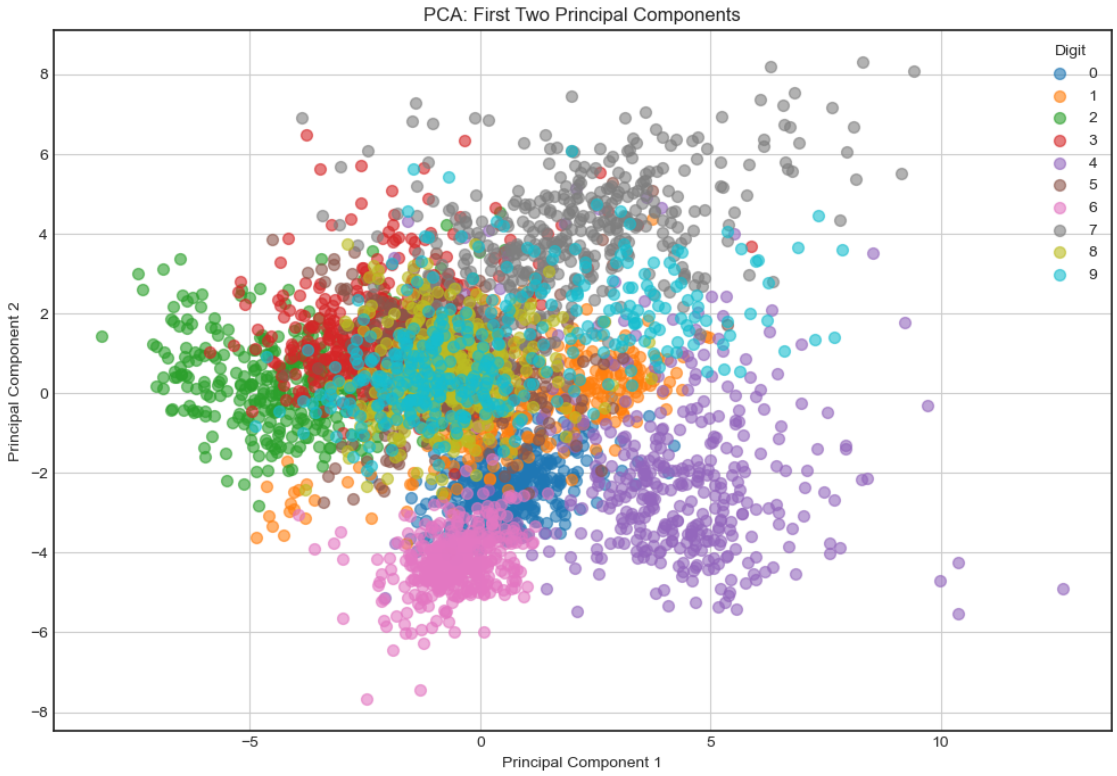
iii. Write your observations on the variance explained by the principal components. The first few principal components capture most of the variation between the data points. As more components are added, each successive one contributes less new information. About 40 components capture 95% of the total information in the data, allowing us to work with just these 40 instead of all 64 features with minimal loss of accuracy. By applying PCA, we can reduce the number of features while preserving most of the original data, simplifying analysis and improving efficiency.

iv)Create a scatter plot for the first two principal components and show the target class variable (i.e., digit number) with different symbols and colors. Write your observations on the plot.

[20]:
```
# Scatter plot of the first two principal components
plt.figure(figsize=(12, 8))

for i in range(10):
    subset = PC[y == i]
    plt.scatter(subset[:, 0], subset[:, 1], label=str(i), alpha=0.6, s=50)

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA: First Two Principal Components')
plt.legend(title='Digit')
plt.grid(True)
plt.show()
```


PCA: First Two Principal Components
Digit
0
1
2
3
4
5
6
7
8
9
Principal Component 2
Principal Component 1
8
6
4
2
0
−2
−4
−6
−8
−5
0
5
10

[ ]: