Linear Algebra Assignment 4

Sriram Radhakrishna PES1UG20CS435 Section : 'H'

Principal Component Analysis applied on MNIST Dataset

Python code (executed on a kaggle notebook, space separated based on cell content) :

```
# initial library import
import numpy as np
import pandas as pd
import seaborn as sns

# data import
data = pd.read_csv('../input/mnist-data/train.csv')
data.head()

# dropping unnecessary labels
label = data['label'] # save label data for later use
data.drop('label', axis = 1, inplace = True)
data.head()

# scaling data to have a mean of 0 and standard deviation of 1
from sklearn.preprocessing import StandardScaler
data_standardized = StandardScaler().fit_transform(data)
data_standardized

# covariance matrix to determine dimensional relationships
covMatrix = np.matmul(data_standardized.T ,data_standardized)
covMatrix

# eigenvalue & eigenvector calculation to determine principal
components
from scipy.linalg import eigh
```

```
values, vector = eigh(covMatrix,eigvals=(782,783))
vector = vector.T
values

# projecting vector on standardized data
projectedData = np.matmul(vector, data_standardized.T)
projectedData

# preparing stacked data for visualization
reducedData = np.vstack((projectedData, label)).T
reducedData = pd.DataFrame(reducedData, columns = ['pca_1', 'pca_2',
'label'])

# data visualization
sns.FacetGrid(reducedData, hue = 'label', size = 8).map(sns.scatterplot,
'pca_1', 'pca_2').add_legend()

# visualization of what the dataset actually represents
import matplotlib.pyplot as plt

index = 1234 # random index chosen for representation purposes
fig_data = np.array(data.iloc[index]).reshape(28,28)
plt.imshow(fig_data, interpolation = None, cmap = 'gray')
plt.show()
print('Digit represented : ', label[index])
```

Output screenshots :

[32]:

```python
# initial library import
import numpy as np
import pandas as pd
import seaborn as sns
```

Principal component analysis of a dataset :

- Unlike what the name suggests, it is a dimension reduction technique for easier data processing.
- In this notebook, we'll demonstrate the same by converting an of 784 dimensions from the MNIST dataset into a 2D visualization.

[33]:

```python
# data import
data = pd.read_csv('../input/mnist-data/train.csv')
data.head()
```

[33]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 785 columns

[34]:

```python
# dropping unnecessary labels
label = data['label'] # save label data for later use
data.drop('label', axis = 1, inplace = True)
data.head()
```

[34]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel77
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

5 rows × 784 columns

Data standardization :

PCA gives more emphasis to variables with high variance. Therefore, if the dimensions are not scaled, we will get inconsistent results. For example, the value for one variable might lie in the range 50-100 and the other one 5-10. In this case, PCA will give more weight to the first variable. Such issues can be resolved by standardizing the dataset before applying PCA.

[35]:
```
# scaling data to have a mean of 0 and standard deviation of 1
from sklearn.preprocessing import StandardScaler
data_standardized = StandardScaler().fit_transform(data)
data_standardized
```

[35]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

[36]:
```
# covariance matrix to determine dimensional relationships
covMatrix = np.matmul(data_standardized.T ,data_standardized)
covMatrix
```

[36]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

[37]:
```
# eigenvalue & eigenvector calculation to determine principal components
from scipy.linalg import eigh
values, vector = eigh(covMatrix,eigvals=(782,783))
vector = vector.T
values
```

[37]:
```
array([1222652.44613786, 1709211.41082575])
```

[38]:
```
# projecting vector on standardized data
projectedData = np.matmul(vector, data_standardized.T)
projectedData
```

[38]:
```
array([[-5.2264454 ,  6.03299601, -1.70581328, ...,  7.07627667,
        -4.34451279,  1.55912058],
       [-5.14047772, 19.29233234, -7.64450341, ...,  0.49539137,
         2.30724011, -4.80767022]])
```

[39]:
```
# preparing stacked data for visualization
reducedData = np.vstack((projectedData, label)).T
reducedData = pd.DataFrame(reducedData, columns = ['pca_1', 'pca_2', 'label'])
```

Visualization using FacetGrid :

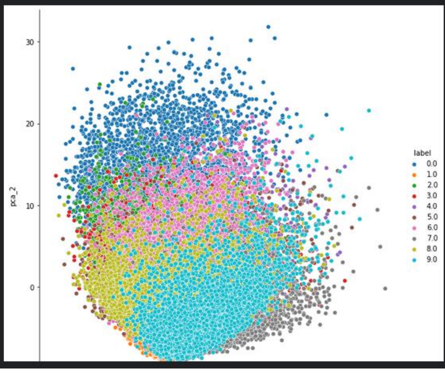
FacetGrid is used for plotting conditional relationships. The basic workflow is to initialize the FacetGrid object with the dataset, and the variables used to structure the grid. Then one or more plotting functions can be applied to each subset by calling FacetGrid.map() or FacetGrid.map_dataframe(), and then the other customizations can also be done.

+ Code   + Markdown

[40]:
```
# data visualization
sns.FacetGrid(reducedData, hue = 'label', size = 8).map(sns.scatterplot, 'pca_1', 'pca_2').add_legend()
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/axisgrid.py:337: UserWarning: The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

[40… <seaborn.axisgrid.FacetGrid at 0x7fa5c8a0bad0>



[41]:
```
# visualization of what the dataset actually represents
import matplotlib.pyplot as plt

index = 1234 # random index chosen for representation purposes
fig_data = np.array(data.iloc[index]).reshape(28,28)
plt.imshow(fig_data, interpolation = None, cmap = 'gray')
plt.show()
print('Digit represented : ', label[index])
```