PRINCIPAL COMPONENT ANALYSIS(PCA)

By-Bhumika Mahor

07101172020

CSE-AI (2020-2024)

IGDTUW

3rd Year-6th semester

Topics Covered

1. What is PCA
2. Common terms in PCA
3. Working/ Steps in PCA
4. Code of PCA using Python
5. Variations of PCA
6. Applications of PCA
7. Advantages of PCA
8. Disadvantages of PCA

What is PCA?

- Principal Component Analysis is an unsupervised learning algorithm
- It is used for the dimensionality reduction in machine learning.
- It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation
- These new transformed features are called the Principal Components.
- It is one of the popular tools that is used for exploratory data analysis and predictive modeling.

Common Terms in PCA

1. ***Dimensionality:-*** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset
2. ***Correlation:-*** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
3. ***Orthogonal:-*** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero. Eigenvectors: If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.

4. ***Covariance Matrix:-*** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

5. ***Principal Components:-*** the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:-

-the principal component must be the linear combination of the original features.
-These components are orthogonal, i.e., the correlation between a pair of variables is zero.
-The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

Working/Steps in PCA

- The PCA algorithm is based on some mathematical concepts such as:
  Variance and Covariance
  Eigenvalues and Eigen factors

- *Steps are->*
  i. Define the data
  ii. make data mean center
  iii. Find which factors are related
  iv. Find Covariance matrix
  v. find eigen value and eigen vector for the covariance matrix.
  vi. select principal components-
  -eigen value with maximum magnitute becomes the principal component 1, respectively select all the pc's based onn the magnitude of eigen values.
  vii. Project original data to the next axis using the principal components.
  viii. Removing less or unimportant features from new dataset.

Code of PCA in Python

```
In [ ]: # importing hand written digits dataset from sklearn library
```

```
In [2]: import pandas as pd
        from sklearn.datasets import load_digits

        dataset= load_digits()
        dataset.keys()
```

```
Out[2]: dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

```
In [5]: dataset.data.shape  ## flat 1-D array
```

```
Out[5]: (1797, 64)
```

```
In [6]: dataset.data[0].reshape(8,8)  ## Reshaping into a 2X2 matrix
```

```
Out[6]: array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
               [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
               [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
               [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
               [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
               [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
               [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
               [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

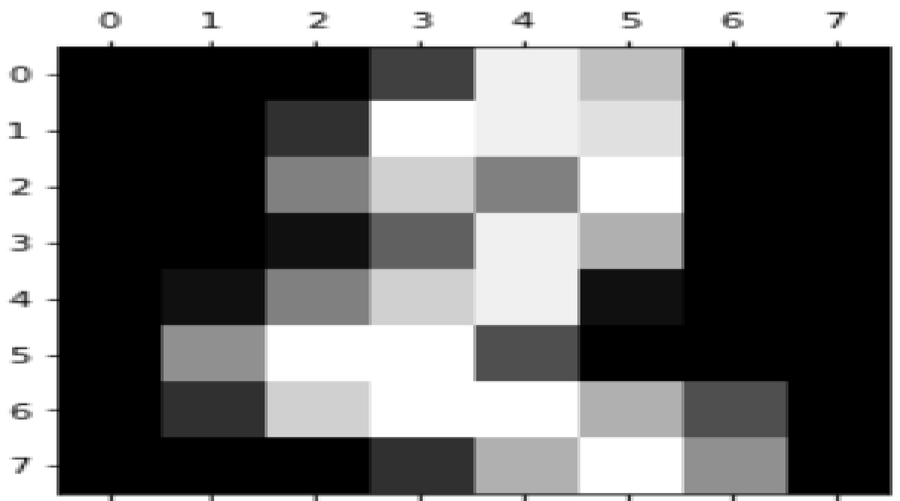
```
In [9]: # data visualization using matplotlib

        from matplotlib import pyplot as plt
        %matplotlib inline

        plt.gray() # Ploting image in gray scale
        plt.matshow(dataset.data[2].reshape(8,8)) # showing plot as matrix for 2nd sample
```

```
Out[9]: <matplotlib.image.AxesImage at 0x1cdc25fd7f0>

        <Figure size 432x288 with 0 Axes>
```



In [10]:
```python
dataset.target[2] # target is our final digit at 2nd image
```
Out[10]: 2

In [11]:
```python
# converting it into dataframe using pandas
pd.DataFrame(dataset.data)
```
Out[11]:

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	4.0	0.0	0.0	0.0	2.0	14.0	15.0	9.0	0.0	0.0
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	6.0	16.0	14.0	6.0	0.0	0.0
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	9.0	13.0	6.0	0.0	0.0
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	0.0	5.0	12.0	16.0	12.0	0.0	0.0
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	8.0	0.0	0.0	1.0	8.0	12.0	14.0	12.0	1.0	0.0

1797 rows × 64 columns

In [12]:
```python
dataset.feature_names  # column names in dataset
```

```
Out[12]: ['pixel_0_0',
          'pixel_0_1',
          'pixel_0_2',
          'pixel_0_3',
          'pixel_0_4',
          'pixel_0_5',
          'pixel_0_6',
          'pixel_0_7',
          'pixel_1_0',
```

```
In [17]: df= pd.DataFrame(dataset.data, columns= dataset.feature_names) # using above feature names as column names for dataset
         df.head(5)
```

Out[17]:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_6	pixel_6_7	pixel_7_0	pixel_7_1	pix
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	

5 rows × 64 columns

In [18]:
```python
df.describe() # describing data (0 means Black, 16 means white)
```

Out[18]:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_6	pix
count	1797.0	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	...	1797.000000	1797.
mean	0.0	0.303840	5.204786	11.835838	11.848080	5.781859	1.362270	0.129661	0.005565	1.993879	...	3.725097	0.
std	0.0	0.907192	4.754826	4.248842	4.287388	5.666418	3.325775	1.037383	0.094222	3.196160	...	4.919406	0.
min	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.
25%	0.0	0.000000	1.000000	10.000000	10.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.
50%	0.0	0.000000	4.000000	13.000000	13.000000	4.000000	0.000000	0.000000	0.000000	0.000000	...	1.000000	0.
75%	0.0	0.000000	9.000000	15.000000	15.000000	11.000000	0.000000	0.000000	0.000000	3.000000	...	7.000000	0.
max	0.0	8.000000	16.000000	16.000000	16.000000	16.000000	16.000000	15.000000	2.000000	16.000000	...	16.000000	13.

8 rows × 64 columns

In [19]:
```python
X= df
y= dataset.target
```

In [20]:
```python
# feature scaling using sklearn library
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler() # creating object of standardscaler
X_scaled= scaler.fit_transform(X)  # scale a new transformed df
X_scaled    # dataset is scaled
```

```
Out[20]: array([[ 0.        , -0.33501649, -0.04308102, ..., -1.14664746,
                 -0.5056698 , -0.19600752],
                [ 0.        , -0.33501649, -1.09493684, ...,  0.54856067,
                 -0.5056698 , -0.19600752],
                [ 0.        , -0.33501649, -1.09493684, ...,  1.56568555,
                  1.6951369 , -0.19600752],
                ...,
                [ 0.        , -0.33501649, -0.88456568, ..., -0.12952258,
                 -0.5056698 , -0.19600752],
                [ 0.        , -0.33501649, -0.67419451, ...,  0.8876023 ,
                 -0.5056698 , -0.19600752],
                [ 0.        , -0.33501649,  1.00877481, ...,  0.8876023 ,
                 -0.26113572, -0.19600752]])
```

```
In [21]: # splitting into training and testing dataset
from sklearn.model_selection import train_test_split

# using random state, the result will be same after calling upon for same value.
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size= 0.2, random_state= 30)
```

```
In [22]: # starting with logistic regression
from sklearn.linear_model import LogisticRegression
model= LogisticRegression() # creating model
model.fit(X_train, y_train) # fitting/training model on xtrain, ytrain
model.score(X_test, y_test) # predicting score of model on xtest, ytest

Out[22]: 0.9722222222222222
```

In [23]:
```
X.head(3)
```

Out[23]:

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_6	pixel_6_7	pixel_7_0	pixel_7_1	pix
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	

3 rows × 64 columns

In [25]:
```
# applying pca
from sklearn.decomposition import PCA
pca= PCA(0.95) # computing new principal components while capturing 95% variation (extracting 95% information from features)
X_pca= pca.fit_transform(X) # calling fit transform method
X_pca.shape # calling shape of new df (x_pca) with reduced features from 64 to 29(principal components)
```

Out[25]: (1797, 29)

```
In [26]: X_pca # it is in form of numpy array
```

```
Out[26]: array([[ -1.25946645,  21.27488348,  -9.46305462, ...,   3.67072108,
                  -0.9436689 ,  -1.13250195],
                [  7.9576113 , -20.76869896,   4.43950604, ...,   2.18261819,
                  -0.51022719,   2.31354911],
                [  6.99192297,  -9.95598641,   2.95855808, ...,   4.22882114,
                   2.1576573 ,   0.8379578 ],
                ...,
                [ 10.8012837 ,  -6.96025223,   5.59955453, ...,  -3.56866194,
                   1.82444444,   3.53885886],
                [ -4.87210009,  12.42395362, -10.17086635, ...,   3.25330054,
                   0.95484174,  -0.93895602],
                [ -0.34438963,   6.36554919,  10.77370849, ...,  -3.01636722,
```

```
In [28]: pca.n_components_ # final principal components
```

```
Out[28]: 29
```

In [27]:
```
pca.explained_variance_ratio_ # variation/information captured by each 29 pc's
```
Out[27]:
```
array([0.14890594, 0.13618771, 0.11794594, 0.08409979, 0.05782415,
       0.0491691 , 0.04315987, 0.03661373, 0.03353248, 0.03078806,
       0.02372341, 0.02272697, 0.01821863, 0.01773855, 0.01467101,
       0.01409716, 0.01318589, 0.01248138, 0.01017718, 0.00905617,
       0.00889538, 0.00797123, 0.00767493, 0.00722904, 0.00695889,
       0.00596081, 0.00575615, 0.00515158, 0.0048954 ])
```
In [29]:
```
# calling train test split on pca
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size= 0.2 ,random_state=30)
```
In [31]:
```
model = LogisticRegression(max_iter = 1000) # fitting new pca df in logistic regression model
model.fit(X_train_pca, y_train)
model.score(X_test_pca, y_test) # score is almost 97%(similar as before)
```
Out[31]:
```
0.9694444444444444
```

Variations of PCA

i). Robust PCA:-

-This variation of PCA is designed to handle datasets with outliers or noise.
-It separates the data into a low-rank component and a sparse component, where the sparse component represents the outliers or noise.

ii). Randomized PCA:-

- it is a variation of Principal Component Analysis (PCA) that is designed to approximate the first k principal components of a large dataset efficiently .
-Instead of computing the eigenvectors of the covariance matrix of the data, randomized PCA uses a random projection matrix to map the data to a lower-dimensional subspace.

iii). Incremental PCA:-

-it is useful for large training datasets as it splits the data into min-batches and feeds it to Incremental PCA, one batch at a time.
-This is called as on-the-fly learning.
-As not much data is present in the memory at a time thus memory usage is controlled.

iv). Kernal PCA:-

-This variation of PCA uses a kernel trick to transform the data into a higher-dimensional space where it is more easily linearly separable.
-This can be useful for handling non-linearly separable data.

v). Sparse PCA:-

-This variation of PCA adds a sparsity constraint to the PCA problem, which encourages the algorithm to find a lower-dimensional representation of the data with fewer non-zero components

Applications of PCA

- Some real-world applications of PCA are image processing, movie recommendation system, optimizing the power allocation in various communication channels.
- PCA is mainly used as the dimensionality reduction technique in various AI applications such as computer vision, image compression, etc.
- It can also be used for finding hidden patterns if data has high dimensions.
- Some fields where PCA is used are Finance, data mining, Psychology, etc.
- It is used to find inter-relation between variables in the data.

Advantages of PCA

- The PCA can counteract the issues of a high-dimensional data set.
- Correlated features removed.
- Speeds up other machine learning algorithms
- Improves visualization.

Disadvantages of PCA

- Data normalization required before performing the PCA.
- We may lose some valuable information.
- Major components may be difficult to understand.

THANK YOU