알고리즘 응용(00) Lab04

201802161 조은빈

1. Goal

- Goal 1. 정규화한 데이터 그래프로 그리기
- Goal 2. Scikit-learn에서 제공하는 PCA 활용하여 결과 그래프로 그리기
- Goal 3. 구현한 PCA 활용하여 그래프 그리기

2. 코드 분석

```
1   import numpy as np
2   import copy
3   import matplotlib.pyplot as plt
4   from sklearn.preprocessing import MinMaxScaler
5   from sklearn.decomposition import PCA
```

코드 구현에 필요한 라이브러리를 import 한다.

```
7   def read_data():
8       f = open('seoul_student.txt','r',encoding = "utf-8")    # 파일 열기
9       lines = f.readlines()  # 데이터 세로 요소 길이 구하기 위해 모든 줄 읽는다
10      dataArr = np.zeros((len(lines) - 1, 2), dtype= float)   # 데이터 저장할 배열
11      f.seek(0)   # 파일 읽는 위치를 첫 줄로 이동
12      f.readline()    # 첫 줄은 버린다
13
14      index = 0
15      while True:
16          line = f.readline()
17          if line == "":
18              break
19
20          dataArr[index] = line.split('₩t')
21          index += 1
22
23      f.close()
24      return dataArr
```

read_data는 파일에 저장된 데이터를 읽어 배열로 만들어 반환한다. 파일 전체를 읽어 줄 수를 파악하고 이에 맞춰 배열 크기를 설정한다. 한 줄씩 데이터를 읽어 배열에 저장한다.

```
26  def normalization(dataArr):     # 정규화
27      scaler = MinMaxScaler()
28      return scaler.fit_transform(dataArr)
29
30  def skl_PCA(dataArr):
31      pca = PCA(n_components = 1)
32      return pca.fit_transform(dataArr)
```

normalization은 데이터 값들을 MinMax로 정규화하기 위한 메소드이다.

skl_PCA는 sklearn의 pca를 사용하기 위한 메소드이다. n-components는 축소할 차원의 수를 나타낸다.

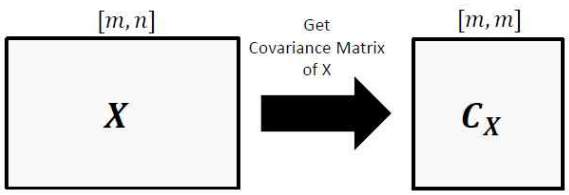
```
34      def get_mean(datas):
35          return (sum(datas) / len(datas))
36
37
38      def get_covariance_matrix(dataArr): # 공분산 행렬 만드는 메소드
39
40          dataArr[:,0] = dataArr[:,0] - get_mean(dataArr[:,0])
41          dataArr[:,1] = dataArr[:,1] - get_mean(dataArr[:,1])
42
43          dataArr_t = np.transpose(dataArr)
44
45          return np.dot(dataArr_t, dataArr) / len(dataArr)  # 2*2 행렬 만들어짐
46
47
```

get_mean은 평균값을 구하는 메소드이다.

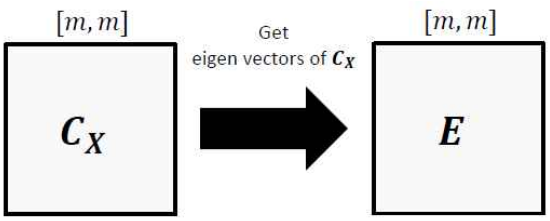
get_covariance_matrix는 공분산행렬을 만들어 주는 메소드이다. 공분산행렬을 만들기 위해 우선 데이터가 저장된 dataArr의 각 열의 요소들에서 각 열의 평균값을 뺀다. 각각의 열의 요소를 모두 더하면 0인 행렬이 만들어진다. 새로 얻어낸 dataArr로 공분산행렬 공식을 수행한다. 공분산행렬 공식은 $C_X = \frac{1}{n} XX^T$이다. 행렬의 행과 열을 바꿔주는 np.transpose를 통해 dataArr_t를 얻어내고 np.dot을 통해 dataArr와 행렬 곱해준다. 그 결과 만들어지는 공분산행렬은 2x2 모양의 행렬이다.

```
48      def e_sort(val, vec):
49          index = np.argsort(val)[::-1]   # 고유값 내림차순 정렬 인덱스
50          vec = vec[:,index]
51          val = val[index]
52          return vec
53
54      def pca(normal_Arr, dim = 1):
55          cor_matrix = get_covariance_matrix(copy.deepcopy(normal_Arr))
56          w, v = np.linalg.eig(cor_matrix)
57          vec = e_sort(w, v)
58          reduce_vec = vec[:,:dim] # 고유 벡터 축소
59          return np.dot(normal_Arr, reduce_vec)
```

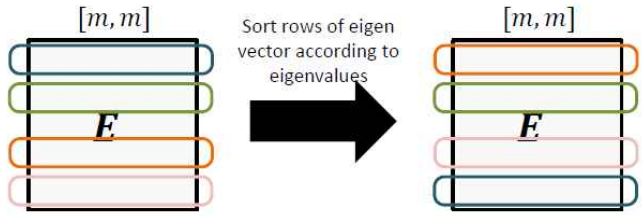
pca는 직접 구현한 pca 메소드이다. 위에서 구현한 get_covariance_matrix로 공분산행렬을 만들어 cor_matrix에 저장한다.



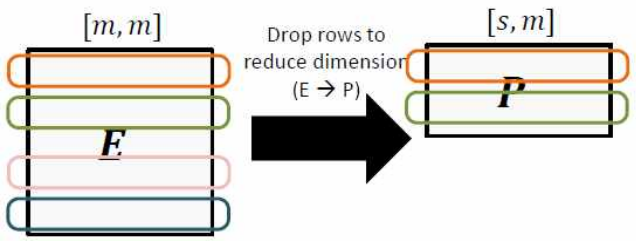
np.linalg.eig를 통해 cor_matrix의 고유값 w와 고유벡터 v을 구한다.



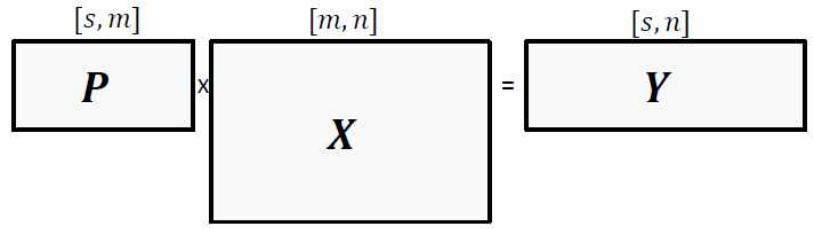
w, v를 파라미터로 e_sort를 수행한다. e_sort는 고유값의 크기에 따라 고유값과 고유벡터를 정렬하는 메소드이다. np.argsort를 통해 고유값을 내림차순으로 정렬하는데 [::-1]을 붙이면 내림차순, 붙이지 않고 사용하면 올림차순으로 정렬된다. np.argsort를 통해 반환되는 값은 내림차순으로 정렬된 인덱스이므로 이에 맞춰 고유값과 고유벡터를 정렬시켜준다.



정렬된 고유벡터를 반환받아 vec에 저장한다. 설정한 차원만큼 고유벡터를 축소시킨다.



축소시킨 고유벡터인 reduce_vec과 원래 데이터를 행렬 곱 한다.




```
61  def draw_graph_2d(data):    # 데이터가 2차원일 경우 그래프 시각화
62      plt.figure()
63      plt.scatter(data[:,0], data[:,1], cmap = 'rainbow')
64      plt.show()
65
66  def draw_graph_1d(data):    # 데이터가 1차원일 경우 그래프 시각화
67      plt.figure()
68      plt.scatter(data, [0] * len(data), cmap = 'rainbow')
69      plt.show()
```

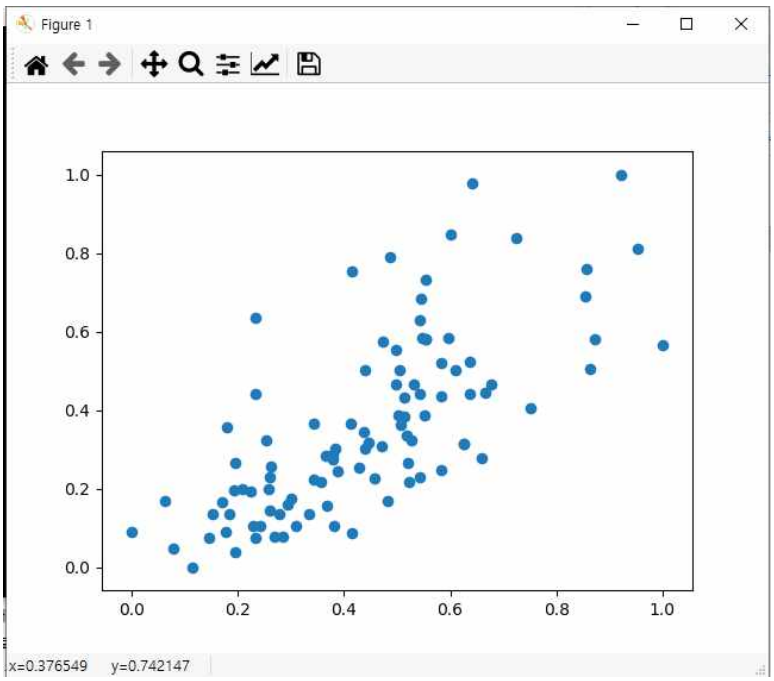
데이터를 그래프로 시각화하는 메소드이다.

```
71  def main():
72      dataArr = read_data()
73      normal_Arr = normalization(dataArr)
74      draw_graph_2d(normal_Arr)
75      draw_graph_1d(skl_PCA(copy.deepcopy(normal_Arr)))
76      draw_graph_1d(pca(copy.deepcopy(normal_Arr)))
77
78  if __name__ == '__main__':
79      main()
```

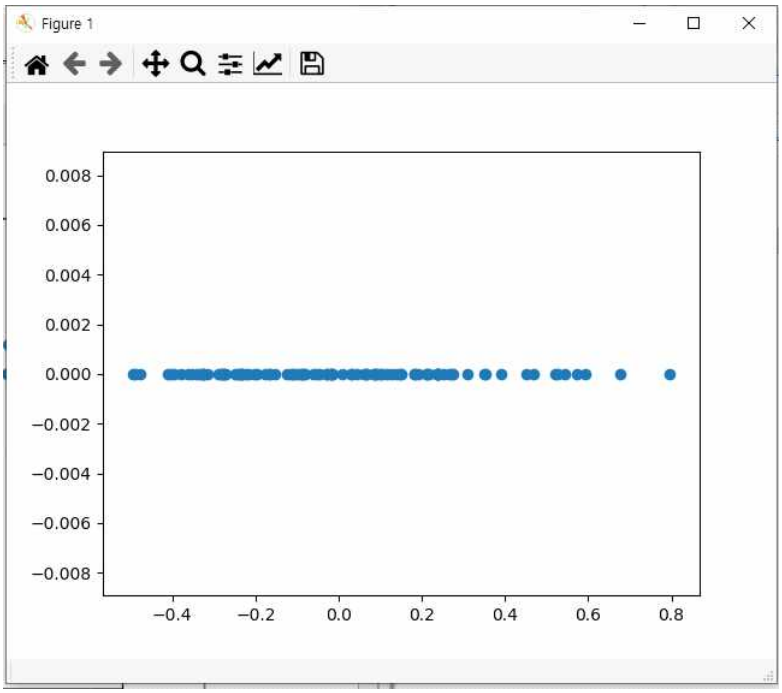
main에서 전체 메소드를 실행시킨다. 메소드를 실행시킬 때 파라미터로 사용한 데이터에 영향이 가지 않도록 하기 위해 deepcopy를 사용했다. 첫 번째로 그려지는 그래프가 정규화한 데이터로 그린 그래프, 두 번째가 sklearn의 pca로 그린 그래프, 세 번째가 직접 구현한 pca로 그린 그래프이다.

3. 실행 결과

- Goal 1. 정규화하여 그래프 그리기



- Goal2. sklearn pca활용하여 그래프 그리기



- Goal3. 구현한 pca활용하여 그래프 그리기