[Principal Component Analysis]

HW 4: 7 pts in total

ASU Id- 1222389090

April 7, 2022

In the following figure, each of × marker corresponds to a data instance. There are 300 data instances that are generated by sampling from a bivariate normal distribution.

- [.5pt] What is the mean of this distribution (only approximately)? Estimate the answer visually and write down your guess in the nearest integers.

  Answer - μ_1= 5.0 ,μ_2= 5.0

- [.5pt] Which one would be larger? the variance of the first variable (on the horizontal axis) or the

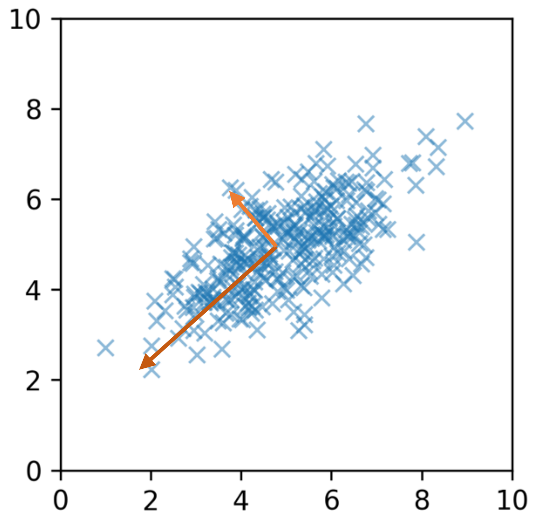
  variance of the second variable (on the vertical axis)

  Answer- Horizontal variance will be more than vertical variance.

- [.5pt] Would the off diagonal entries be [✓ positive?] negative? or close to zero? (Circle the correct one).

  Answer- Off diagonal entries will be positive.

- [.5pt] If you apply PCA and compute two principal components, how would they look? Draw your expected principal component directions in the figure above.



- [.5pt] Let’s denote the new coordinates, found by PCA, by z =[z1, z2] What would the covariance of z_1 and z_2 be like? positive? negative? [✓ or close to zero?] (Circle the correct one).

[Implementation] Now you will use PCA from scikit-learn to compute principal components.

1. [.5pt] Load the data from the npz file. The file contains a data in a dictionary format. Use a key 'data' to retrieve a matrix. The type of the matrix should be the NumPy array and the shape should be (300, 2) (i.e., 300 data instances).

   Done. Shown in code.

2. [1pt] Use PCA from scikit-learn to compute two principal components. Report the computed principal components. There could be two different answers for this. One from scaled data and another from the raw data (non-scaled). Either is okay.

   Done after scaling-
   PCA components - array([[-0.83774478, -0.54606197], [-0.54606197, 0.83774478]])

   Done before scaling-
   PCA components - array([[-0.70710678, -0.70710678], [-0.70710678, 0.70710678]])

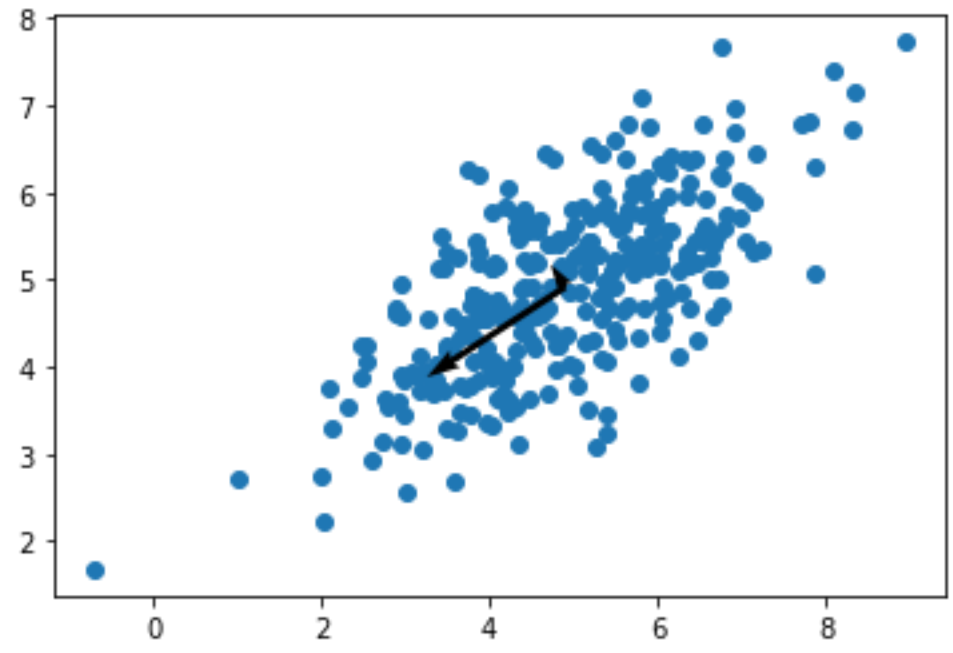
3. [2pt] Implement your own version of PCA algorithm without using scikit-learn API.

   Done. Shown below in code.

4. [1pt] Produce a plot with data instances and computed principal components.
   - for plotting data instances, use either plot or scatter of Matplotlib.
   - for plotting the principal components, use quiver. Plot two arrows that are centered around mean, in the direction of principal components, and of lengths proportional to explained variance. An example is shown below. To obtain the information on the mean, the principal component directions, and the explained variances, you will have to read the scikit-learn document carefully.

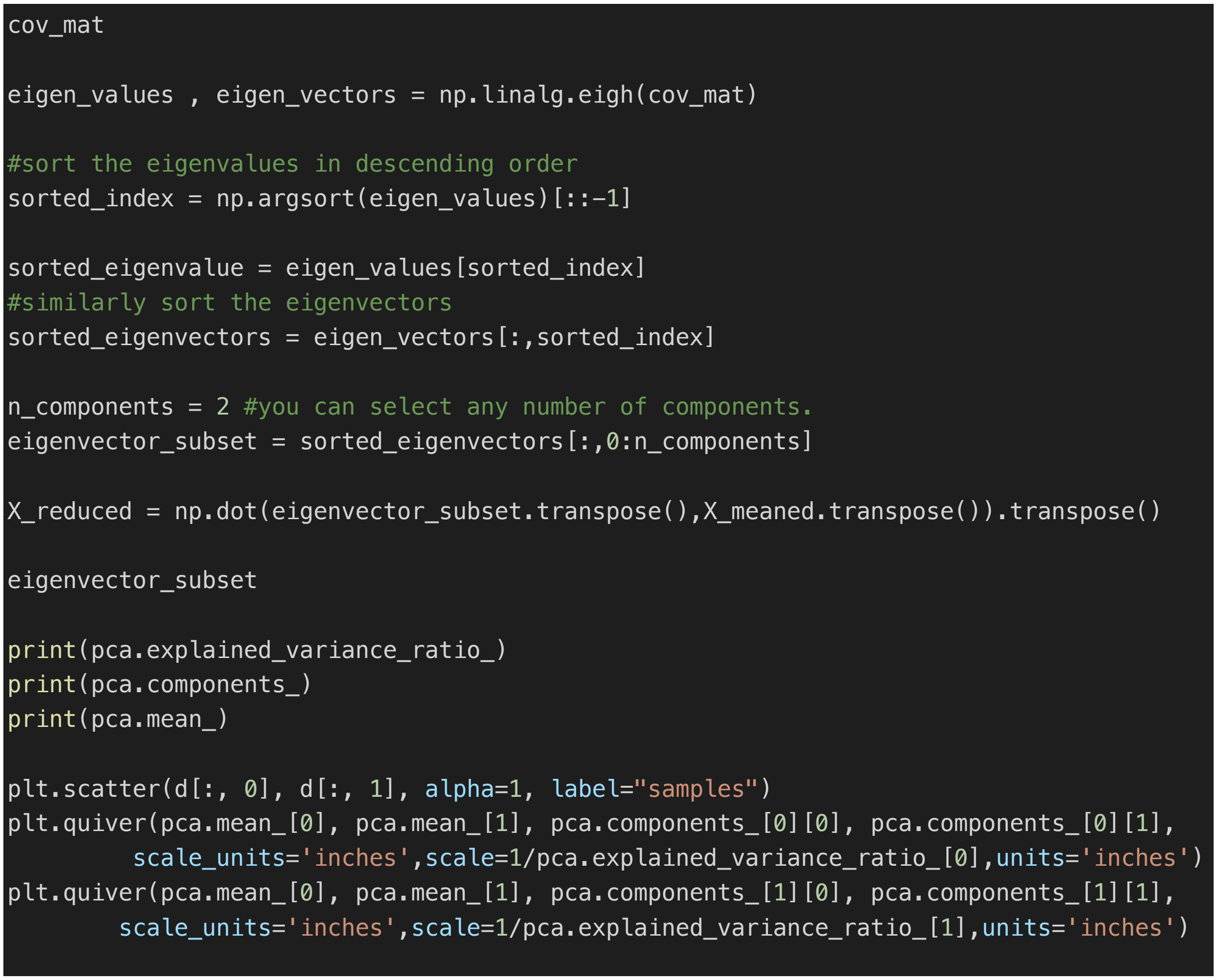
   Code Snippet-

```
plt.scatter(d[:, 0], d[:, 1], alpha=1, label="samples")

plt.quiver( pca.mean_[0], pca.mean_[1], pca.components_[0][0], pca.components_[0][1], scale_units='inches',
scale=1/pca.explained_variance_ratio_[0],units='inches')

plt.quiver( pca.mean_[0], pca.mean_[1], pca.components_[1][0], pca.components_[1][1], scale_units='inches'
,scale=1/pca.explained_variance_ratio_[1], units='inches')
```



Whole code-

```
# -*- coding: utf-8 -*-
"""hw4.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1sQTRntvaa0bASR4qiXrjlakBGTObO_9j
"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale, normalize

data = np.load('/content/drive/MyDrive/Colab Notebooks/SML/pca_data.npz')
d= data['data']

print(d.shape)

plt.scatter(d[:, 0], d[:, 1], alpha=1, label="samples")

pca = PCA(n_components=2)

pca.fit(d)

print(pca.explained_variance_ratio_)

print(pca.singular_values_)

pca.get_covariance()

pca.get_params([d])

pca.components_

pca.mean_

pca.explained_variance_

import numpy as np

X_meaned = d - np.mean(d , axis = 0)

cov_mat = np.cov(X_meaned , rowvar = False)
```



```
cov_mat

eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)

#sort the eigenvalues in descending order
sorted_index = np.argsort(eigen_values)[::-1]

sorted_eigenvalue = eigen_values[sorted_index]
#similarly sort the eigenvectors
sorted_eigenvectors = eigen_vectors[:,sorted_index]

n_components = 2 #you can select any number of components.
eigenvector_subset = sorted_eigenvectors[:,0:n_components]

X_reduced = np.dot(eigenvector_subset.transpose(),X_meaned.transpose()).transpose()

eigenvector_subset

print(pca.explained_variance_ratio_)
print(pca.components_)
print(pca.mean_)

plt.scatter(d[:, 0], d[:, 1], alpha=1, label="samples")
plt.quiver(pca.mean_[0], pca.mean_[1], pca.components_[0][0], pca.components_[0][1],
         scale_units='inches',scale=1/pca.explained_variance_ratio_[0],units='inches')
plt.quiver(pca.mean_[0], pca.mean_[1], pca.components_[1][0], pca.components_[1][1],
        scale_units='inches',scale=1/pca.explained_variance_ratio_[1],units='inches')
```