Machine Learning

COMP 5/630, Spring 2020

HW00: Dimensionality reduction

Remember that only PDF submissions are accepted.

1. A dataset D has 64 points. Each point in D is a vector of length 6830, in other words there are6830 features. If we do Principal Component Analyses (PCA) of this dataset D, how many principal components with non-zero variance would we get? Explain why?

   To make my argument clear: N= 64, P=6830

   The rule: A sample of size N with P dimensions has at most N−1 principal components if N≤P.

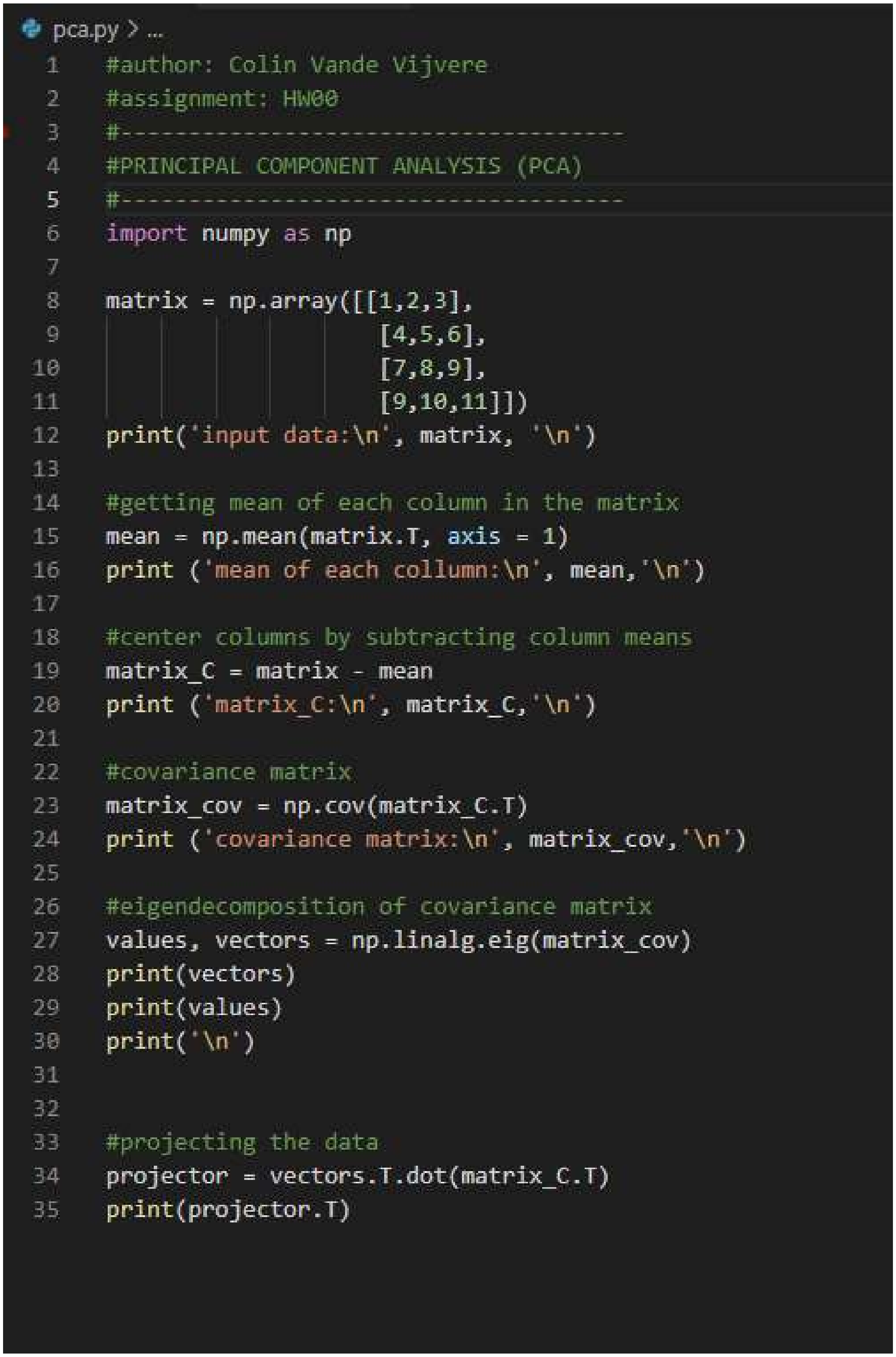
   This means that the answer to our question is 63 principal components.

   Why N-1 principal components?

   Principal components are the axes of most variance in a dataset. Let's imagine that we have 2 points in that dataset, that would mean that there is only 1 axis of variance between them. This example scales up to data sets with a higher dimension as well. Therefore, the answer is N-1 or 63.

2. (Programming) Read this tech blog (https://sebastianraschka.com/Articles/2014_pca_step_by_step. html) (also uploaded to Canvas) and implement a PCA on your own. Feel free to call any command to compute eigenvector, eigenvalue, covariance matrix, SVD, etc.

   Just hand-in your code, and you don't have to play with data for now. (But you are encouraged to play with some toy data you find on your own to validate if your implementation is correct or not). You can reuse the code to this question in your mini-project, and play with data then. **See next page**



pca.py > ...

```python
#author: Colin Vande Vijvere
#assignment: HW00
#-------------------------------------
#PRINCIPAL COMPONENT ANALYSIS (PCA)
#-------------------------------------
import numpy as np

matrix = np.array([[1,2,3],
                    [4,5,6],
                    [7,8,9],
                    [9,10,11]])
print('input data:\n', matrix, '\n')

#getting mean of each column in the matrix
mean = np.mean(matrix.T, axis = 1)
print ('mean of each collumn:\n', mean,'\n')

#center columns by subtracting column means
matrix_C = matrix - mean
print ('matrix_C:\n', matrix_C,'\n')

#covariance matrix
matrix_cov = np.cov(matrix_C.T)
print ('covariance matrix:\n', matrix_cov,'\n')

#eigendecomposition of covariance matrix
values, vectors = np.linalg.eig(matrix_cov)
print(vectors)
print(values)
print('\n')


#projecting the data
projector = vectors.T.dot(matrix_C.T)
print(projector.T)
```