**CC484: Pattern Recognition** **Spring 2018**

Sheet 1: Principal Components Analysis

*Abdelrahman Mohamed Abdelnabi, 3466* *Febraury 13*

# 1 Sheet 1: Principal Component Analysis

In [1]:
```
%matplotlib inline

import numpy as np
from numpy import linalg
import matplotlib.pyplot as plt
```

In [2]:
```
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics.pairwise import euclidean_distances
from mpl_toolkits.mplot3d import Axes3D
```

In [3]:
```
# set printing of matrices to 3 decimal places for clarity
# this doesn't change the values stored inside the matrices
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})
```

## 1.1 Question 2

In [4]:
```
data_matrix = np.array([[10, 60, 10, 90],
                        [20, 50, 40, 70],
                        [30, 50, 30, 40],
                        [20, 50, 20, 60],
                        [10, 60, 30, 10]])
```

In [5]:
```
norm = linalg.norm(data_matrix, axis=1)
print norm
```

```
[109.087 96.954 76.811 83.066 68.557]
```

In [6]:
```
cosine_sim = cosine_similarity(data_matrix)
print cosine_sim
```

```
[[1.000 0.936 0.859 0.971 0.655]
 [0.936 1.000 0.953 0.981 0.767]
 [0.859 0.953 1.000 0.956 0.874]
 [0.971 0.981 0.956 1.000 0.773]
 [0.655 0.767 0.874 0.773 1.000]]
```

In [7]:
```
euclidean_distances(data_matrix)
```

Out[7]:
```
array([[0.000, 38.730, 58.310, 34.641, 82.462],
       [38.730, 0.000, 33.166, 22.361, 62.450],
       [58.310, 33.166, 0.000, 24.495, 37.417],
       [34.641, 22.361, 24.495, 0.000, 52.915],
       [82.462, 62.450, 37.417, 52.915, 0.000]])
```

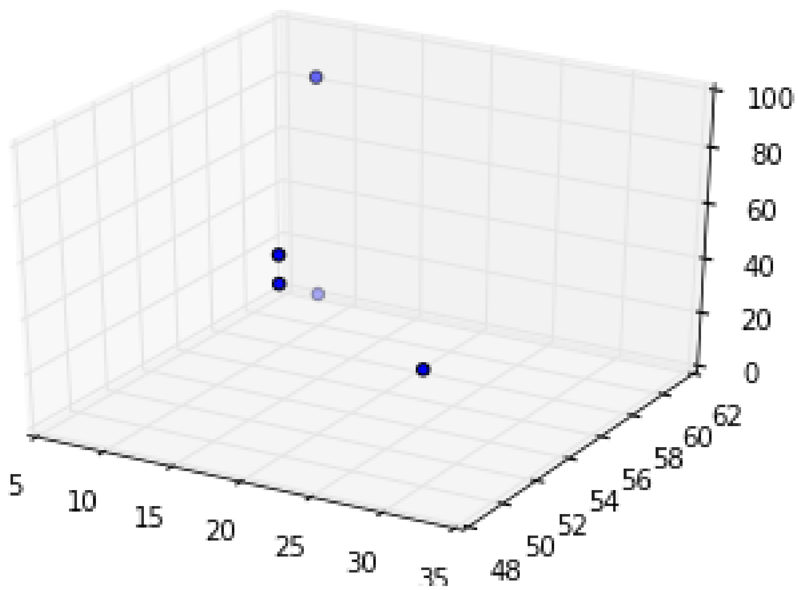
## 1.2 Question 3

```
In [8]: data_matrix = np.array([[10, 60, 90],
                                [20, 50, 70],
                                [30, 50, 40],
                                [20, 50, 60],
                               [10, 60, 10]])
```

### 1.2.1 Scatter Plot

```
In [9]: fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(data_matrix[:,0], data_matrix[:,1], data_matrix[:,2], marker='o')

Out[9]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0xa3231d0>
```




### 1.2.2 Mean Vector and data centering

```
In [10]: mean = np.mean(data_matrix, axis=0)
         print mean

[18.000 54.000 54.000]

In [11]: centered_data = np.empty([5,3])
         for i in range(0,3):
             centered_data[:,i] = data_matrix[:,i] - mean[i]
         print data_matrix

[[10 60 90]
 [20 50 70]
 [30 50 40]
 [20 50 60]
 [10 60 10]]
```

#### 1.2.3 covariance matrix

```
In [12]: # rowvar=False means that columns are variables (and rows are instances)
         covariance = np.cov(centered_data, rowvar=False)
         print covariance
```

```
[[70.000 -40.000 -15.000]
 [-40.000 30.000 -20.000]
 [-15.000 -20.000 930.000]]
```

#### 1.2.4 eigen values and eigen vectors

```
In [13]: eigenvalues, eigenvectors = linalg.eigh(covariance)
```

```
In [14]: for i in range(0,3):
             print "{}, {}".format(eigenvalues[i], eigenvectors[i])
```

```
4.60865706553, [0.526 0.850 -0.016]
94.7153745646, [0.850 -0.526 -0.021]
930.67596837, [0.027 0.003 1.000]
```

```
In [15]: # verify that (eigenvec * (eigenvals * eigenvec.transpose)) = Covariance
         np.matmul(eigenvectors, np.matmul(np.diag(eigenvalues),eigenvectors.transpose()))
         print eigenvectors.transpose()
```

```
[[0.526 0.850 0.027]
 [0.850 -0.526 0.003]
 [-0.016 -0.021 1.000]]
```

#### 1.2.5 variance and projection to lower dimensions

The variance by the eigenvector of the largest eigenvalue corresponds to $\frac{930.7}{930.7+94.7+4.6} = 0.903$ of the total variance which is good enough to represent the original dataset.

```
In [16]: # compute the projection matrix
         # (column vectors corresponding to the eigenvectors of the highest eigenvalues)
         proj_matrix = np.column_stack((eigenvectors[1], eigenvectors[2]))
```

The projection matrix $P$ corresponding to the top two eigen vectors of the martix $U$ is

$$P = \begin{bmatrix} 0.850 & 0.027 \\ -0.526 & 0.003 \\ -0.021 & 1.000 \end{bmatrix}$$

The projected data is just transforming the original dataset to the new subspace by applying the projection matrix on the dataset.

```
In [17]: # computing the projected data
         proj_data = np.matmul(data_matrix, proj_matrix)
```

$$ProjData = \begin{bmatrix} -25.014 & 90.396 \\ -10.821 & 70.646 \\ -1.677 & 40.926 \\ -10.606 & 60.649 \\ -23.296 & 10.425 \end{bmatrix}$$

### 1.2.6 Plotting the new projected data

In [18]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(proj_data[:,0], proj_data[:,1], marker='o')
```

Out[18]: <matplotlib.collections.PathCollection at 0xa5a3710>