Principal Component Analysis

The goal of this question is to build a conceptual understanding of dimensionality reduction using PCA and implement it on a toy dataset. You'll only have to use numpy and matplotlib for this question.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
# (a) Load data (features)
X = 0
def load_data(X):
    X = 'features.npy'
    data = np.load(X)

    mu = np.mean(data,axis=0)
    sigma = np.std(data,axis=0)
    data-=mu
    data/=sigma
    return data

test = load_data(X)
print(test.shape)
print('Mean: {}\nSTD: {}'.format(np.mean(test), np.std(test)))
```

```
(150, 8)
Mean: -6.158037043254202e-16
STD: 1.0
```

In [3]:

```
# (b) Perform eigen decomposition and return eigen pairs in desecending order
def eigendecomp(X):
    covariance = np.cov(X.T)
#     print(covariance)
#     print(covariance.shape)
    sorted_eig_vals,sorted_eig_vecs = np.linalg.eig(covariance)
#     print(w,v)
    return (sorted_eig_vals, sorted_eig_vecs)
eigendecomp(test)
```

Out[3]:
```
(array([ 4.74298961e+00,  2.29585309e+00,  7.76910512e-01,  2.04172901e-01,
         3.37651661e-02, -7.03281987e-16,  6.34199026e-18,  8.43406727e-16]),
 array([[-0.39124937,  0.13884872, -0.46160937,  0.58034539,  0.24934936,
          0.20978664, -0.42116938,  0.09320476],
        [ 0.11687696, -0.4391715 , -0.78711289, -0.2905579 , -0.12725786,
          0.07280114,  0.20605814,  0.14672413],
        [-0.40655289,  0.29080021, -0.13961871, -0.12636707, -0.54994554,
         -0.59631459, -0.06875535,  0.21509872],
        [-0.39944906,  0.26454833, -0.16206048, -0.54404218,  0.49904279,
         -0.06355642,  0.05521708, -0.43718158],
        [-0.3778555 , -0.35426671,  0.07790627,  0.42060984,  0.12822569,
         -0.32964346,  0.66179491, -0.14645517],
        [-0.09816172, -0.64299795,  0.11941452, -0.04972667, -0.0795516 ,
         -0.20011335, -0.56640575, -0.40331041],
        [-0.45509399, -0.03231459,  0.12200908, -0.08034689, -0.51935676,
          0.65722825,  0.07577872, -0.23707109],
        [-0.38587285, -0.30545597,  0.29393481, -0.28457653,  0.27864817,
          0.10197996, -0.08859901,  0.70148315]]))
```

In [4]:
```python
# (c) Evaluate using variance_explained as the metric
#ratio between sum of k eigenvalues and sum of all eigenvalues
def eval(X):
    eigenval,eigenvec = eigendecomp(X)

    for k in range(1,9):
        var_expl = np.sum(eigenval[0:k])
        var_tot = np.sum(eigenval)
        ratio = var_expl/var_tot
        print('Ratio:',ratio)
    print('Eigenvalue',eigenval)
eval(test)
```

```
Ratio: 0.5889212098295772
Ratio: 0.8739896347022311
Ratio: 0.9704560233211404
Ratio: 0.9958074918820439
Ratio: 1.0
Ratio: 1.0
Ratio: 1.0
Ratio: 1.0
Eigenvalue [ 4.74298961e+00  2.29585309e+00  7.76910512e-01  2.04172901e-01
  3.37651661e-02 -7.03281987e-16  6.34199026e-18  8.43406727e-16]
```

In [5]:
```
def project(X):
    eigenval,eigenvec = eigendecomp(X)
#     print(eigenvec)
    eigenvector = np.column_stack((eigenvec[:,0],eigenvec[:,1]))
    print('eigenvector',eigenvector)
#     print(arr.shape)
#     print(X.shape)
    reduced_data = (np.matmul(X,eigenvector))
    return reduced_data
print('reduced_data',project(test))
```

```
eigenvector [[-0.39124937  0.13884872]
 [ 0.11687696 -0.4391715 ]
 [-0.40655289  0.29080021]
 [-0.39944906  0.26454833]
 [-0.3778555  -0.35426671]
 [-0.09816172 -0.64299795]
 [-0.45509399 -0.03231459]
 [-0.38587285 -0.30545597]]
reduced_data [[ 1.14283537 -2.75916236]
 [ 4.47917874  1.87206668]
 [ 2.36294554 -1.18487643]
 [ 2.11785321 -1.23400373]
 [ 2.35720963 -1.59610139]
 [ 2.11794046 -1.66871235]
 [ 2.59256157 -1.1733813 ]
 [ 1.69738736 -1.94274515]
 [ 3.84415322  0.91236648]
 [ 2.54114976 -0.61023934]
 [ 0.53307793 -3.52114295]
 [ 2.1342099  -1.52068107]
 [ 2.20338749 -0.9757627 ]
 [ 3.35078423 -0.23733063]
 [ 2.15681408 -2.04798058]
 [ 2.26610464 -2.28577345]
 [ 1.14461493 -3.08406768]
 [ 2.49984619 -1.06451121]
 [ 2.18818227 -1.28000327]
 [ 2.83611365 -1.13516124]
 [ 2.7612342  -0.24427366]
 [ 3.15987429 -0.47080825]
 [ 1.58717236 -3.08977007]
 [ 0.66469116 -2.43034637]
 [ 2.7406176  -0.60365427]
 [ 3.18585395  0.57768826]
 [ 0.81361677 -2.69041059]
 [ 2.8319995  -0.635948  ]
 [ 1.5236855  -2.05718464]
 [ 2.49499246 -0.82244322]
 [ 3.791253    0.94977631]
 [ 1.6103999  -1.51222567]
 [ 3.54346342 -0.9785532 ]
 [ 2.93280851 -1.59445256]
 [ 4.23927503  1.37510478]
```

```
[ 3.43117407  0.23166945]
[ 2.58921603 -0.82150777]
[ 2.76985217 -0.34285448]
[ 1.6173334  -1.9381603 ]
[ 2.23395945 -1.23642152]
[ 1.75067988 -2.08873766]
[ 2.79267307  0.86709121]
[ 1.74023069 -2.14990566]
[ 3.17847114  0.11295206]
[ 4.12713083  0.75973692]
[ 1.88802255 -1.12823371]
[ 0.623944   -3.76026107]
[ 2.85033664 -0.62469429]
[ 1.57186285 -2.38565561]
[ 1.79454623 -1.72078895]
[-1.81141033 -0.59303167]
[-0.19671106  0.79021338]
[-0.89276166  0.82653295]
[ 1.42975232  3.01725168]
[-0.76704061  0.98262477]
[ 0.53563806  1.5880808 ]
[ 0.11689792  1.14697328]
[ 0.86288386  0.89300451]
[-0.64077874  0.81528976]
[ 0.92841179  1.52203367]
[ 1.05511764  2.04630892]
[-0.45219697  0.24391029]
[ 0.46395778  2.13644889]
[-2.40742934 -1.46768404]
[ 0.33058612  0.46747946]
[-1.86848459 -0.92708591]
[ 0.26256682  1.05063124]
[-0.94171816 -0.48419685]
[-1.47260851  0.9176662 ]
[ 0.57256585  1.45306373]
[-1.57699445 -0.6861078 ]
[-2.24604408 -1.69485953]
[-1.52264838  0.68241772]
[-1.97882408 -1.00510469]
[-2.22760787 -1.40597853]
[ 0.97456275  2.49554929]
[-0.9760806   1.00580133]
[-1.39963675  0.53923257]
[-1.35747427 -0.3497214 ]
[ 0.18060287  0.51056642]
[ 0.34461951  1.21593044]
[ 1.77058821  2.70561789]
[-0.28925582  0.35615519]
[-2.08138365 -0.31642132]
[-0.39294209  0.12625758]
[-1.20532415 -0.95229301]
[-0.38477179  1.12374948]
[-1.51957044  0.47845785]
[-0.95558824 -0.86719815]
[-0.39527753  0.52810915]
[ 0.14437388  1.15060741]
[-1.64648773 -0.82510304]
```

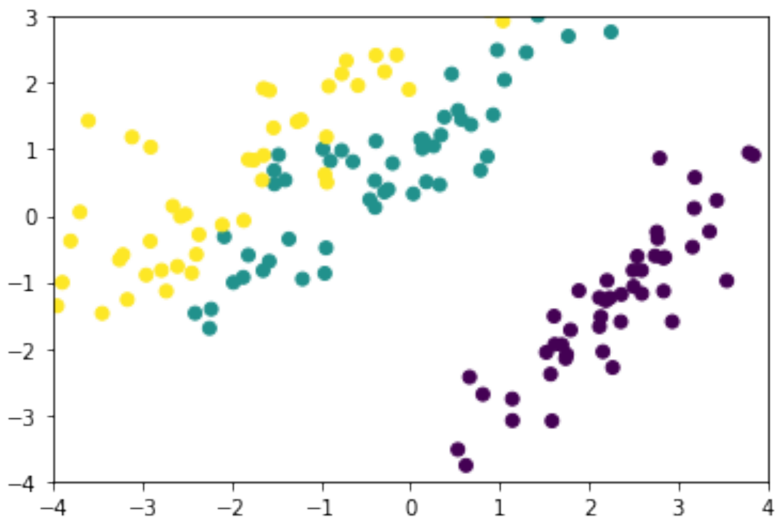
```
[ 1.29930976  2.46047417]
[ 2.24785603  2.76894025]
[ 0.13605657  1.01158084]
[ 0.03469556  0.33081092]
[ 0.68078327  1.37200898]
[ 0.38178132  1.48676681]
[ 0.78699576  0.68462402]
[-0.24537126  0.39756981]
[-1.75175341  0.836874  ]
[-0.93719603  1.18765157]
[-3.8935865  -1.00393043]
[-2.95450856 -0.89267797]
[-1.5298337   1.32441284]
[-2.90357248  1.03397004]
[ 1.81110092  3.41291557]
[-1.64096172  1.91845919]
[-1.57399002  1.88688541]
[-4.51177198 -2.14279665]
[-2.72993696 -1.13579289]
[-1.26492019  1.41717621]
[-0.76465418  2.13983383]
[-0.15090453  2.42213493]
[-0.58474807  1.96258338]
[-3.16607528 -1.26164261]
[-1.65646998  0.53583669]
[-3.95218976 -1.35726227]
[-3.60131966  1.43234824]
[-1.64027473  0.91040176]
[-4.18658328 -1.78237345]
[-0.01335868  1.90145402]
[-3.11272766  1.18509785]
[-2.78310603 -0.82231002]
[-3.44714984 -1.46981494]
[-0.91088588  1.95182671]
[-1.86220954 -0.07171595]
[-0.95493     0.62394468]
[-3.25328195 -0.66092292]
[-2.90881505 -0.38353612]
[-3.80045695 -0.38016355]
[-4.47063545 -2.22969086]
[-2.65589151  0.14562659]
[ 0.87444483  3.09050969]
[-0.28584465  2.17004306]
[-3.69602821  0.05595666]
[-2.10297758 -0.13699038]
[-2.3929747  -0.5819506 ]
[-0.93224893  0.50211494]
[-0.38310161  2.41785277]
[-0.71429891  2.33569728]
[-1.22125173  1.44612127]
[-2.60478212 -0.761988  ]
[-3.21436214 -0.58600703]
[-2.56922658 -0.01093208]
[-2.5142908   0.02345174]
[-1.81080307  0.8473443 ]
[-2.36399184 -0.28317222]
[-2.44436643 -0.86193624]
```

```
 [ 1.03880181  2.93560245]]
```

In [6]:

```python
Y = 'labels.npy'
labels = np.load(Y,allow_pickle=True)
# (d) Visualize after projecting to 2-D space
def viz(X):
    plt.ylim(-4,3)
    plt.xlim(-4,4)
    data_reduced = project(X)
    plt.scatter(data_reduced[:,0],data_reduced[:,1],c=labels)
viz(test)
```

```
eigenvector [[-0.39124937  0.13884872]
 [ 0.11687696 -0.4391715 ]
 [-0.40655289  0.29080021]
 [-0.39944906  0.26454833]
 [-0.3778555  -0.35426671]
 [-0.09816172 -0.64299795]
 [-0.45509399 -0.03231459]
 [-0.38587285 -0.30545597]]
```



In [7]:

```python
# def main():
#     eval()
#     viz()

# if __name__ == "__main__":
#     main()
```

(e1): If the number of features is 1000 and the number of data points is 10, what will be the dimension of your covariance matrix? Can you suggest what can be changed to improve the performance?1000 by 1000. Number of features is greater than the number of samples and to improve the performance the number of samples must be greater than 1000.(e2): Assume you have a dataset with the original dimensionality as 2 and you have to reduce it to 1. Provide a sample scatter plot of the original data (less than 10 datapoints) where PCA might produce misleading results. You can plot it by hand and then take a picture. In the next cell, switch to Markdown mode and use the command: `![title]()`

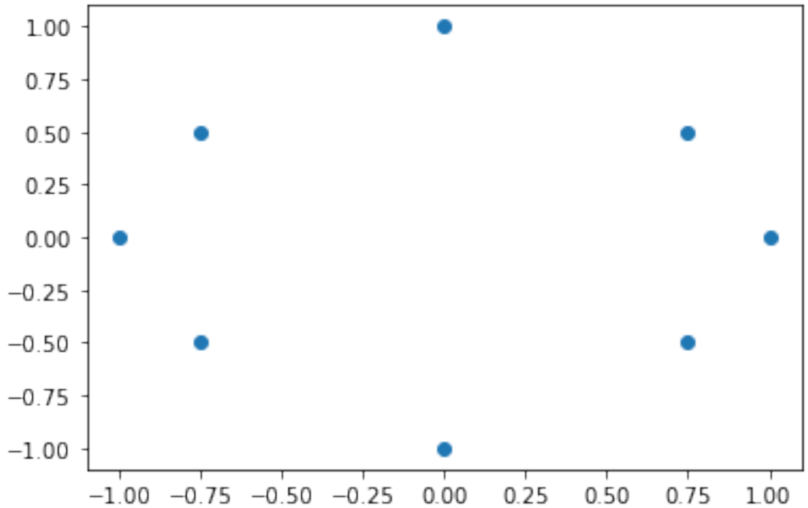
In [8]:

```
x_dat = np.array([-1,1,0,0,0.75,-0.75, 0.75,-0.75])
y_dat = np.array([0,0,1,-1,0.5,0.5,-0.5,-0.5])

plt.scatter(x_dat,y_dat)
```

Out[8]:

```
<matplotlib.collections.PathCollection at 0x7fb8902d4940>
```



PCA does not work well for non-linear data