Ph21 Problem Set 6

Thomas Alford

March 11, 2019

1

Imports

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Principle Component Analysis

```
def PCA(X):
    # here X is an m x n matrix which contains m measurement types and n samples
    Xprime = np.array([x - np.mean(x) for x in X])
    covariance = np.cov(Xprime)
    eigenvals, eigenvecs = np.linalg.eig(covariance)
    return eigenvals, eigenvecs
```

2

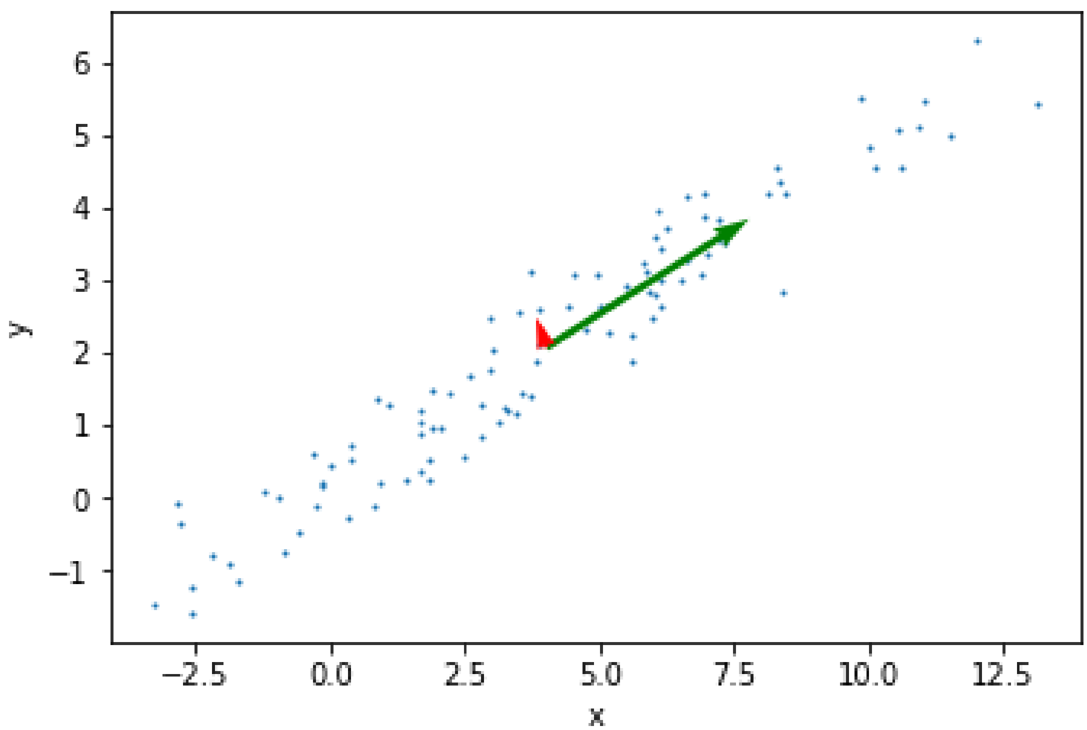
Simulated Data Set (2d)

Here we'll create a data set which has x linearly dependent of y but with a random amount of Gaussian noise involved:

```
def create_2d_data(n, mu, sigma, noise_sigma):
    y_vals = np.random.normal(mu, sigma, n)
    # create a linear relation with noise
    x_vals = 2 * y_vals + np.random.normal(0, noise_sigma, n)
    return x_vals, y_vals
```

```
data = create_2d_data(100, 2, 2, 1)
evals, evecs = PCA(data)
```

```
plt.scatter(data[0], data[1], s=1)
origin = (np.mean(data[0]), np.mean(data[1]))
# multiply by sqrt(lambda)
V = evecs.T * np.array(2 * [np.sqrt(evals)]).T
plt.quiver(*origin, V[:,0], V[:,1], color=['g', 'r'],
           angles='xy', scale=1, scale_units='xy')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```




```
print('eiegnvals = ', evals)
print('eiegnvecs = ', evecs)
```

```
eiegnvals =  [24.43196754  0.21322486]
eiegnvecs =  [[ 0.90075419 -0.43432923]
 [ 0.43432923  0.90075419]]
```

Here these values make sense, as x is very dependent on y here, and y is not dependent on x at all (the noise makes it so that this dependence isn't perfect, however).

3

Higher Dimensional Data Set

Here now we'll simulate a higher dimensional dataset where each element is slightly dependent on the other elements. We'll choose 5 dimensions here.

```
def create_5d_data(n, y_mu, y_sigma, noise_sigma):
    y_vals = np.random.normal(y_mu, y_sigma, n)
    # create a linear relation with noise
    x_vals = 2 * y_vals + np.random.normal(0, noise_sigma, n)
    z_vals = .5 * y_vals - .7 * x_vals + np.random.normal(0, noise_sigma, n)
    w_vals = np.random.normal(y_mu + 10, y_sigma, n)
    v_vals = -.3 * w_vals + 1.5 * z_vals + np.random.normal(0, noise_sigma, n)
    return v_vals, w_vals, x_vals, y_vals, z_vals
```

```
data_5d = create_5d_data(100, 2, 2, 1)
evals_5d, evecs_5d = PCA(data_5d)
```

```
list(evals_5d)
```

```
[33.47792791833488,
5.314471270308553,
2.1399570795143403,
0.24691964621245038,
0.1475084147204272]
```

```
evecs_5d.T
```

```
array([[-0.55094958,  0.02537261,  0.68413801,  0.31660392, -0.35711054],
[ 0.33225504, -0.89601528,  0.25838908,  0.1358716 ,  0.03920705],
[ 0.62138085,  0.43289284,  0.50514353,  0.2891418 ,  0.29616998],
[-0.4418413 , -0.09446484, -0.01958823,  0.24750149,  0.8568617 ],
[-0.06872597, -0.01389335,  0.45786088, -0.85816135,  0.22137352]])
```

This values make sense for us. v contains data about every single variable, since it is a linear combination of w and z, which is then a linear combination of x and y. Since x is also dependent on y, this makes z, x, and y all very redundant variables, which is why their components are so low. w is the second-highest due to the fact that the only variable dependent on it is v.