Lecture 15

PCA

SVD implementation: $X = ASB^T$

$$\left(\frac{X^T}{\sqrt{N-1}} \cdot \frac{X}{\sqrt{N-1}}\right) \quad \text{eigenvector } B$$

$$= \frac{X^T X}{N-1}$$

```
[A,S,B]= svd(X/sqrt(N-1),'econ');
>> whos A
  Name      Size        Bytes  Class     Attributes
  A         100x2        1600  double

S =
   2.0093        0
        0   1.0042

>> S.^2
ans =
   4.0373        0
        0   1.0084


>> B
B =
  -0.7478  -0.6639
  -0.6639   0.7478
```

```
[V,D] = eig(C);

D =
   1.0084        0
        0   4.0373


V =
   0.6639  -0.7478
  -0.7478  -0.6639
```

Covariance

For two random variable vectors A and B, the covariance is defined as

$$\operatorname{cov}(A,B) = \frac{1}{N-1}\sum_{i=1}^{N}(A_i - \mu_A)^*(B_i - \mu_B)$$

where μ_A is the mean of A, μ_B is the mean of B, and $*$ denotes the complex conjugate.

The *covariance matrix* of two random variables is the matrix of pairwise covariance calculations between each variable,

$$C = \begin{pmatrix} \operatorname{cov}(A,A) & \operatorname{cov}(A,B) \\ \operatorname{cov}(B,A) & \operatorname{cov}(B,B) \end{pmatrix}.$$

http://www.ece.uah.edu/~dwpan/course/ee610/code/Unsupervised%20Learning/pca_demo.m

```
>> % Empirical sample covariance matrix
(X'*X)/(N-1)

ans =

   2.7023    1.5038
   1.5038    2.3434

>> C = cov(X)

C =

   2.7023    1.5038
   1.5038    2.3434


V1 = V(:,1);
V2 = V(:,2);
% Display the eigenvectors for data projection
hor = min(X(:,1)): 0.01: max(X(:,1));
ver = V1(2)/V1(1)*hor;
plot(hor, ver, 'g');
axis equal

hold on;
hor = min(X(:,1)): 0.01: max(X(:,1));
ver = V2(2)/V2(1)*hor;
plot(hor, ver, 'r');

>> % Variance after projection
Y1 = X*V1;
var(Y1)
figure;  scatter(Y1,zeros(100,1))


ans =

   1.0084


>> % Correlation between projections
Y1'*Y2

ans =

  -9.2371e-14
```

```
[V,D] = eig(C);
>> C*V

ans =

    0.6695   -3.0192
   -0.7541   -2.6803

>> V*D

ans =

    0.6695   -3.0192
   -0.7541   -2.6803


>> V

V =

    0.6639   -0.7478
   -0.7478   -0.6639


D =

    1.0084         0
         0    4.0373


>> Y2 = X*V2;
var(Y2)
figure;  scatter(Y2,zeros(100,1))
grid

ans =

    4.0373
```

```
>> % Use pca function
[coeff,score,latent] = pca(Xraw);

>> coeff

coeff =

   0.7478   -0.6639
   0.6639    0.7478
```

PCA : data reduction method

data compression { lossless, lossy }

EE 614 : Data Compression

```
% Reconstruction using only the principal components
Y1_drop = zeros(N,1);  % Setting the first component to zeros
Y = [Y1_drop, Y2];
%Y = [Y1, Y2];  % Keep both components
% inv(V) = V = V'
X_rec = Y*V;

figure;
plot(X_rec(:,1),X_rec(:,2),'b.','MarkerSize',12)
grid

diff = (X_rec - X);
diff_sq = diff(:,1).^2 + diff(:,2).^2;
% Average distortion (mean square error)
sum(diff_sq)/(N-1)   % Same as the eigenvalue of the 1st component

% Repeat the above steps using the score matrix
score_truncated = score;
score_truncated(:,2) = 0;
X_rec2 = score_truncated * coeff';
figure;
plot(X(:,1),X(:,2),'b.','MarkerSize',12)
hold on;
plot(X_rec2(:,1),X_rec2(:,2),'r.','MarkerSize',12)
legend('Original Data', 'Reconstructed Data','Location','SE');
grid
axis equal
```

```
diff = (X_rec2 - X);
diff_sq = diff(:,1).^2 + diff(:,2).^2;
% Average distortion (mean square error)
sum(diff_sq)/(N-1)

% Average distortion (mean square error)
sum(diff_sq)/(N-1)

ans =

 1.008407590798483

  D =

   1.0084        0
        0   4.0373
```

http://www.ece.uah.edu/~dwpan/course/ee610/code/Unsupervised%20Learning/pca_demo.py

```
import numpy as np
infile = "pca.csv"
dataset = np.loadtxt(infile, delimiter=',')
X = dataset[:, 0:2]


from sklearn.decomposition import PCA

pca = PCA(n_components=2)
# sklearn automatically centers the input raw data
pca.fit(X)

# Eigenvectors (loadings)
print(pca.components_)

# Eigvenvalues (latent)
print(pca.explained_variance_)

# Scores
Y = pca.transform(X)

#axis = 0, along the columb; ddof = 1 for dividing by (N-1);
np.var(Y, axis = 0, ddof=1)

# Reconstruction by keeping only the 1st principal component
# setting the 2nd component in Y to zero
Y_trunc = Y
Y_trunc[:,1] = 0

X_rec = pca.inverse_transform(Y_trunc)

# Centered (instead of the raw) input to compare with the reconstructed data
#X_center = X - np.mean(X)
X_center = X

# Mean square error
diff = X_rec - X_center
diff_sq = diff[:,0]**2 + diff[:,1]**2
np.sum(diff_sq)/(np.size(diff_sq)-1)
```

```
pca.fit(X)
Out[10]: PCA(n_components=2)

print(pca.components_)
[[ 0.74783215  0.66388785]
 [-0.66388785  0.74783215]]
```

Matlab

```
>> V

V =

   0.6639   -0.7478
  -0.7478   -0.6639
```

```
print(pca.explained_variance_)
[4.03733216 1.00840759]
```

Matlab

```
>> format long
>> D

D =

  1.008407590798482                  0
                  0   4.037332163848554
```

```
np.var(Y, axis = 0, ddof=1)
Out[14]: array([4.03733216,
1.00840759])
```

https://numpy.org/doc/stable/reference/generated/numpy.var.html

```
np.sum(diff_sq)/(np.size(diff_sq)-1)
Out[30]: 1.0084075907984822
```