EECS 545 – Machine Learning - Homework #5

Due: 5:00PM 11/22/2018

Homework submission via Gradescope as usual.

1) **PCA (10 pts).**

Read over the proof of PCA based on the generalized Rayleigh quotient. Then solve the following problems. The first problem motivates a method for selecting k, as discussed at the end of the first set of notes on PCA.

a. (5 pts) Let $k \in \{0, 1, \ldots, d\}$ be arbitrary. Show that

$$\min_{\boldsymbol{\mu}, A, \{\boldsymbol{\theta}_i\}} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{\mu} - A\boldsymbol{\theta}_i\|^2 = n \sum_{j=k+1}^{d} \lambda_j,$$

where A ranges over all $d \times k$ matrices with orthonormal columns.

Hint: This is easy if you use properties of the trace operator.

b. (5 pts) Give a condition involving the spectral decomposition of the sample covariance matrix that is both necessary and sufficient for the subspace $\langle A \rangle$ in PCA to be unique.

2) **Eigenfaces (15 pts).**

In this exercise you will apply PCA to a modified version of the Extended Yale Face Database B. The modified database is available in the file `yalefaces.mat` on Canvas. The modification I made was simply to reduce the resolution of each image by a factor of $4 \times 4 = 16$ to hopefully avoid computational and memory bottlenecks.

For a whirlwind tour of the data, issue the following commands. In Matlab:

```
load yalefaces % loads the 3-d array yalefaces
for i=1:size(yalefaces,3)
    x = double(yalefaces(:,:,i));
    imagesc(x);
    colormap(gray)
    drawnow
    % pause(.1)
end
```

In Python:

```
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
import time
```

```
yale = sio.loadmat('yalefaces.mat')
yalefaces = yale['yalefaces']

fig, ax = plt.subplots()
for i in range(0,yalefaces.shape[2]):
    x = yalefaces[:,:,i]
    ax.imshow(x, extent=[0, 1, 0, 1])
    plt.imshow(x, cmap=plt.get_cmap('gray'))
    #time.sleep(0.1)
    plt.show()
```

Uncomment the `pause/sleep` command to slow things down a bit. What you will see are several different subjects (38 total) under a variety of lighting conditions.

a. (5 pts) By viewing each image as a vector in a high dimensional space, perform PCA on the full dataset. Hand in a plot the sorted eigenvalues (use the `semilogy` command in Matlab; `plt.semilogy` in Python) of the sample covariance matrix. How many principal components are needed to represent 95% of the total variation? 99%? What is the percentage reduction in dimension in each case? Useful commands in Matlab:`reshape, eig, svd, mean, diag`, and Python: `np.reshape, np.linalg.eig, np.linalg.svd, np.mean, np.diag`.

b. (5 pts) Hand in a 4×5 array of subplots showing principal eigenvectors ('eigenfaces') 0 through 19 as images, treating the sample mean as the zeroth order principal eigenvector. Comment on what facial or lighting variations some of the different principal components are capturing. Useful commands in Matlab: `subplot, imagesc, colormap(gray)`, in Python: `plt.imshow(x, cmap=plt.get_cmap('gray'))`, and for subplots a useful link is `http://matplotlib.org/examples/pylab_examples/subplots_demo.html`

c. (5 pts) Submit your code.

PLEASE NOTE: For uniformity, please do **not** standardize the features as described in the "preprocessing" section of the PCA notes.

3) **EM Algorithm for Mixed Linear Regression (20 pts).**

Consider regression training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, iid realizations of $(\mathbf{X}, Y) \in \mathbb{R}^d \times \mathbb{R}$ where the conditional distribution of Y given $\mathbf{X}$ is modeled by the pdf

$$f(y|\boldsymbol{x};\boldsymbol{\theta}) \sim \sum_{k=1}^{K} \epsilon_k \phi(y; \mathbf{w}_k^T \boldsymbol{x} + b_k, \sigma_k^2),$$

where $\boldsymbol{\theta} = (\epsilon_1, \ldots, \epsilon_K, \mathbf{w}_1, \ldots, \mathbf{w}_K, b_1, \ldots, b_K, \sigma_1^2, \ldots, \sigma_K^2)$ be a list of the model parameters. Derive an EM algorithm for maximizing the likelihood by following these steps.

a. (5 pts) Denote $\underline{\boldsymbol{x}} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and $\underline{y} = (y_1, \ldots, y_n)$. First, write down the formula for the log-likelihood $\ell(\boldsymbol{\theta}; \underline{y}|\underline{\boldsymbol{x}}) = \log f(\underline{y}|\underline{\boldsymbol{x}};\boldsymbol{\theta})$ where $f(\underline{y}|\underline{\boldsymbol{x}};\boldsymbol{\theta})$ is the model (with parameters $\boldsymbol{\theta}$) for $\underline{y}$ given $\underline{\boldsymbol{x}}$. Then, introduce an appropriate hidden variable and write down the complete-data log-likelihood. Here and

below, use notation consistent with the EM algorithm for GMMs whenever possible. Finally, determine the E-step. Give an explicit formula for $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)})$ in terms of $\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)}$, and the data.

b. (5 pts) Determine the M-step. *Suggestions:* Use Lagrange multiplier theory to optimize the weights ϵ_k. To optimize $(\mathbf{w}_k, b_k, \sigma_k^2)$, first hold σ_k^2 fixed and find the optimal $(\mathbf{w}_k, b_k)$, then plug that in and find the optimal σ_k^2. Just treat σ_k^2 as a variable (not the square of a variable).

c. (10 pts) Now let's put these ideas into practice. Generate training data using the following Matlab code: In Matlab

```
clear all
close all
rng(0);

n = 200; % sample size
K = 2; % number of lines

e = [.7 .3]; % mixing weights
w = [-2 1]; % slopes of lines
b = [.5 -.5]; % offsets of lines
v = [.2 .1]; % variances

for i=1:n
    x(i) = rand;
    if rand < e(1);
        y(i) = w(1)*x(i) + b(1) + randn*sqrt(v(1));
    else
        y(i) = w(2)*x(i) + b(2) + randn*sqrt(v(2));
    end
end
plot(x,y,'bo')
hold on
t=0:0.01:1;
plot(t,w(1)*t+b(1),'k')
plot(t,w(2)*t+b(2),'k')
```

In Python:

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

np.random.seed(0)
n = 200 #sample size
K = 2 #number of lines
e = np.array([0.7,0.3]) #mixing weights
w = np.array([-2,1]) #slopes of lines
b = np.array([0.5,-0.5]) #offsets of lines
v = np.array([0.2,0.1]) #variances
x = np.zeros([n])
y = np.zeros([n])
for i in range(0,n):
    x[i] = np.random.rand(1)
    if np.random.rand(1) < e[0]:
        y[i] = w[0]*x[i] + b[0] + np.random.randn(1)*np.sqrt(v[0])
```

```
        else:
            y[i] = w[1]*x[i] + b[1] + np.random.randn(1)*np.sqrt(v[1])

    plt.plot(x,y,'bo')
    t = np.linspace(0, 1, num=100)
    plt.plot(t,w[0]*t+b[0],'k')
    plt.plot(t,w[1]*t+b[1],'k')

    plt.show()
```

Implement and run the EM algorithm, and report or turn in the following:

- The number of iterations to reach convergence
- A plot of the log-likelihood as a function of iteration number
- The estimated model parameters
- A plot showing the data, true lines (solid), and estimated lines (dotted) together.
- Your code.

Comments:

- Initialize your variables as follows. In Matlab:

  ```
  = [.5 .5]
  = [1 -1];
  = [0 0];
  = repmat(var(y),1,2);
  ```

  In Python

  ```
  = np.array([.5 , .5])
  = np.array([1 ,-1])
  = np.array([0, 0])
  = np.array([np.var(y),np.var(y)])
  ```

  where you get to define the variable names, and the variables are listed in the same order as above.

- Stop iterating when the increase in log-likelihood is less than 10^{-4}.
- In Matlab, use the command `normpdf` to evaluate the Gaussian pdf. Note that the third argument is the standard devation, not the variance. In Python, use

  ```
  from scipy.stats import norm
  norm.pdf()
  ```

- This is optional, but it is helpful and interesting to plot the current estimate at each iteration, and watch the estimate evolve. To update the plot while the program is running, you can use `drawnow` in Matlab. The command `clf` will clear the current plot. In Python look at `plt.ion()` and `plt.clf()`. For more information, see here or here.

4) **Ncut and Normalized Spectral Clustering (10 pts).**

Assuming $K = 2$, show that a relaxation of the Ncut problem discussed in class is solved by normalized spectral clustering, i.e., spectral clustering with the normalized graph Laplacian $\tilde{L} = D^{-1}L$. *Hint:* First, define f_A in an analogous way to the treatment of RatioCut. Verify analogous formulas for $f_A^T L f_A$, $\mathbf{1}^T D f_A$, and $f_A^T D f_A$ to formulate the relaxation. Then make the substitution $g = D^{1/2} f$ and reformulate the relaxation with g as the variable. Once you solve for g, don't forget to transform back to f.