EECS 545 HW 7

Due Wednesday, Nov. 9, at 11 pm via gradescope.

0. Project (0 points)

Start thinking about the course project. There is nothing to turn in now, so the main thing to do is to find a group of 3-4 people and start talking about possible topics. The project will ask you to find a machine learning research paper, understand it, implement it and try it out, and provide a critical evaluation of the strengths and weaknesses of the method. Optionally, you can extend the method or develop your own for comparison. A typeset project report of about 8 pages will be the final deliverable. More details to follow next week.

1. Vector Quantization (15 points)

Download the image `mandrill.tiff` from Canvas. In this problem you will apply the k-means algorithm to image compression. In this context it is also known as the Lloyd-Max algorithm.

a. (10 points) First, use the following code to partition the image into $M \times M$ blocks, and reshape each block into a vector of dimension $3M^2$. The 3 comes from the fact that this is a color image, and so there are three intensities for each pixel. Assume that M, like the image dimensions, is a power of 2.

In Matlab:

```
\begin{quote}
% load image
y = double(imread('mandrill.tiff'));

% extract blocks
M = 2; % block side-length
n = prod(size(y))/(3*M*M); % number of blocks
d = size(y,1); % image length/width

c=0; % counter
x = zeros(n,3*M*M);
for i=1:M:d % loop through blocks
    for j=1:M:d
        c = c+1;
        x(c,:) =  reshape(y(i:i+M-1,j:j+M-1,:),[1,M*M*3]);
    end
end
\end{quote}
```

In Python:

```
\begin{quote}
from PIL import Image
```

```
import numpy as np

#To display this image
im = Image.open('mandrill.tiff')
#im.show()

y = np.array(im)
M = 2 #block side-length
n = np.prod(y.shape)/(3*M*M) #number of blocks
d = y.shape[0]
c = 0 # counter
x = np.zeros([n,3*M*M])
for i in range(0,d,M):
    for j in range(0,d,M):
        #print c,i,j,M,y[i:i+M,j:j+M,:].shape, M*M*3
        x[c,:] = np.reshape(y[i:i+M,j:j+M,:],[1,M*M*3])
        c = c+1
\end{quote}
```

Next, write a program that will cluster these vectors using the k-means algorithm. You should write the k-means algorithm yourself. The function `dist2` from a previous assignment can be used to eliminate all loops except for the outermost loop over the iterations of k-means. This will make your program much more efficient. Use the `min` (Matlab) or `np.min` (Python) command to determine the cluster assignments. To update the centroids, the `find` (Matlab) or `np.where` (Python) command is useful. Please initialize the cluster means to be randomly selected data points using the following code:

In Matlab

```
rng(0);
perm = randperm(n);
m = x(perm(1:k),:); % initial cluster centers
```

In Python

```
np.random.seed(0)
perm = np.random.permutation(n)
m = x[perm[0:k,:]] # initial cluster centers
```

Finally, write a program to reconstruct a quantized version of the original image by replacing each block in the original image by the nearest cluster center.

Deliverables: Test your code using $M = 2$ and $k = 100$. How many iterations does the k-means algorithm require to converge? Hand in a plot of the objective function value versus iteration. How does the compressed image look compared to the original? What regions of the image are best preserved, and which are not? Hand in a plot of the *difference* of the two images, using the command `imagesc` (Matlab) or `plt.imshow()` (Python) to display the difference image. You may need to normalize the pixel intensities in the difference image to get a good view (do not normalize your images for other parts of the problem). Also report the compression ratio (use your formula from part b.).

Finally, report the relative mean absolute error of the compressed image, defined as

$$\frac{\frac{1}{3N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{r=1}^{3}|I(i,j,r)-I'(i,j,r)|}{256}$$

where I and I' are the original and compressed images viewed as 3-D arrays. This quantity can be viewed as the average error in pixel intensity relative to the range of pixel intensities. Finally, submit your code.

Programming tip: If `C` is a matrix or multi-dimensional array in Matlab, `C(:)` converts `C` to a columnn vector (like using the `reshape` command).

b. (5 points) The original uncompressed image uses 24 bits per pixel (bpp), 8 bits for each color. Assuming an image of size $N \times N$, where N is a power of 2, what is the number of bits per pixel, as a function of N, k, and M, needed to store the compressed image? What is the compression ratio, which is defined as the ratio of bpp in the compressed image relative to the original uncompressed image? *Hint:* To calculate the bpp of the compressed image, imagine you need to transmit the compressed image to a friend who knows the compression strategy.

2. EM Algorithm for Mixed Linear Regression (20 points)

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

a. (5 points) 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.

c. (5 points) 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).

d. (10 points) 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.

3. KL Divergence and KDE Model Selection (5 points)

If g and h are two densities, the KL divergence between them is defined as

$$D(g||h) := \int \log\left(\frac{g(x)}{h(x)}\right) g(x)dx.$$

We have shown that the KL divergence is nonnegative and equal to zero iff $g = h$ (when we showed the EM algorithm has the ascent property). Therefore it is a performance measure that could be used for selecting the bandwidth of a KDE. Devise a leave-one-out type procedure for selecting the bandwidth that aims to minimize

$$D(f||\widehat{f}_\sigma)$$

where f is the true density.

SUPPLEMENTAL EXERCISES

Not to be turned in.

1. In the notes, it was shown that both the Eckart-Young theorem and the generalized Rayleigh quotient theorem imply the PCA theorem.

Can you use the Eckhart-Young theorem to prove the GRQ theorem? Can you use the GRQ theorem to prove the Eckart-Young theorem? Are the three theorems (including the PCA theorem) equivalent (i.e., each one can be used to prove the others)?

3. Let C_k^* be the globally optimal k-means cluster map. Show that $W(C_k^*)$ is nonincreasing as a function of k.

3. Let $C_k^{(j)}$ be estimate of C_k^* produced by the j-th iteration of k-means. Show that $W(C_k^{(j)})$ is nonincreasing as a function of j.

4. In the image compression problem above, play around with M and k.