CSL 865: Assignment 3

Due Date: Friday November 16, 2012 (in class). Coding Problem due on Saturday November 24.

Note: Refer to the course website for assignment submission instructions.

1. **[15 points]VC dimension**

   Let the domain of the inputs for a learning problem be $X = R$. Consider using hypotheses of the following form:

   $$h_\theta(x) = 1\{\theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_d x^d \geq 0\},$$

   and let $H = \{h_\theta : \theta \in R^{d+1}\}$ be the corresponding hypothesis class. What is the VC dimension of H? Justify your answer.

   [Hint: You may use the fact that a polynomial of degree d has at most d real roots. When doing this problem, you should not assume any other non-trivial result (such as that the VC dimension of linear classifiers in d-dimensions is $d + 1$) that was not formally proved in class.]

2. **[15 points]LOOCV and SVM**

   (a) **Linear Case.** Consider training an SVM using a linear Kernel $K(x, z) = x^T z$ on a training set $\{(x^{(i)}, y^{(i)}) : i = 1, \ldots, m\}$ that is linearly separable, and suppose that it is noise free (i.e. we use the simplest SVM model without any slack variables). Let $|SV|$ be the number of support vectors obtained when training on the entire training set. (Recall $x(i)$ is a support vector if and only if $\alpha_i > 0$.) Let $\widehat{\epsilon}_{LOOCV}$ denote the leave one out cross validation error of our SVM.
   Prove that

   $$\widehat{\epsilon}_{LOOCV} \leq \frac{|SV|}{m}$$

   (b) **General Case.** Consider a setting similar to in part (a), except that we now run an SVM using a general (Mercer) kernel. Assume that the data is linearly separable in the high dimensional feature space corresponding to the kernel. Does the bound in part (a) on $\widehat{\epsilon}_{LOOCV}$ still hold? Justify your answer.

3. **[15 points]MAP estimates and weight decay**

   Consider using a logistic regression model $h_\theta(x) = g(\theta^T x)$ where g is the sigmoid function, and let a training set $\{(x^{(i)}, y^{(i)}); i = 1, ..., m\}$ be given as usual. The maximum likelihood estimate of the parameters θ is given by

   $$\theta_{ML} = arg\max_\theta \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta).$$

   Consider using a Bayesian prior $\theta \sim (0, \tau^2 I)$ (here, $\tau > 0$, and I is the $(n + 1) \times (n + 1)$ identity matrix). The MAP estimate is given by

   $$\theta_{MAP} = arg\max_\theta p(\theta) \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}, \theta).$$

   Prove that

   $$||\theta_{MAP}||_2 \leq ||\theta_{ML}||_2$$

   [Hint: Consider using a proof by contradiction.]
   Remark. Above form of prior is sometimes called **weight decay**, since it encourages the weights (meaning parameters) to take on generally smaller values.

4. **[15 points]EM for MAP estimation**

The EM algorithm that we covered in class was for solving a maximum likelihood estimation problem in which we wished to maximize

$$\prod_{i=1}^{m} p(x^{(i)};\theta) = \prod_{i=1}^{m} \sum_{z^{(i)}} p(x^{(i)}, z^{(i)};\theta)$$

where the $z^{(i)}$'s were latent random variables. Suppose we are working in a Bayesian framework, and would like to find the MAP estimate of the parameters θ by maximizing

$$\Big(\prod_{i=1}^{m} p(x^{(i)}|\theta)\Big)p(\theta) = \Big(\prod_{i=1}^{m} \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}|\theta)\Big)p(\theta).$$

Here, $p(\theta)$ is our prior on the parameters. Generalize the EM algorithm to work for MAP estimation. You may assume that $\log p(x, z|\theta)$ and $\log p(\theta)$ are both concave in θ, and hence, maximizing any linear combination of these quantities is tractable. Show that your M-step is tractable, and also prove that $\prod_{i=1}^{m} p(x^{(i)}|\theta)p(\theta)$ (viewed as a function of θ) monotonically increases with each iteration of your algorithm.

5. **[15 points]PCA**

In class, we showed that PCA finds the "variance maximizing"directions onto which to project the data. In this problem, we will look at another interpretation of PCA. Suppose we are given a set of points $\{x^{(1)}, ..., x^{(m)}\}$. Let us assume that we have as usual pre-processed the data to have zero-mean and unit variance in each coordinate. For a given unit-length vector u, let $f_u(x)$ denote the projection of point x onto the direction given by u. I.e., if $\mathcal{V} = \{\alpha u : \alpha \in R\}$, then

$$f_u(x) = arg\min_{v\in\mathcal{V}} ||x - v||^2.$$

Show that the unit-length vector u that minimizes the mean squared error between projected points and original points corresponds to the first principal component for the data. I.e., show that

$$arg\min_{u:u^Tu=1} \sum_{i=1}^{m} ||x^{(i)} - f_u(x^{(i)})||_2^2.$$

gives the first principal component.

Remark. If we are asked to find a k-dimensional subspace onto which to project the data so as to minimize the sum of squared distances between the original data and their projections, then we should choose the k-dimensional subspace spanned by the first k principal components of the data. This problem shows that this result holds for the case of $k = 1$.

6. **[25 points]K-means for compression**

In this problem, we will apply the K-means algorithm to do lossy image compression, by reducing the number of colors used in an image.

Download the 512×512 image of a mandrill represented in 24-bit color. This means that, for each of the 262144 pixels in the image, there are three 8-bit numbers (each ranging from 0 to 255) that represent the red, green, and blue intensity values for that pixel. The straightforward representation of this image therefore takes about 262144 x 3 = 786432 bytes (a byte being 8 bits). To compress the image, we will use K-means to reduce the image to k = 16 colors. More specifically, each pixel in the image is considered a point in the three-dimensional (r, g, b)-space. To compress the image, we will cluster these points in color-space into 16 clusters, and replace each pixel with the closest cluster centroid.

Follow the instructions below. Be warned that some of these operations can take a while (several minutes even on a fast computer)!

(a) In MATLAB, use the command A = double (imread(mandrill.tiff)) to read in the image. This loads the image into a three-dimensional matrix A where A(:,:,1), A(:,:,2) and A(:,:,3) are 512x512 arrays that respectively contain the red, green, and blue values for each pixel. You can use imshow(uint8(round(A))) to display the image.

(b) Since the large image has 262144 pixels and would take a while to cluster, we will instead run quantization on this smaller image. Treating each pixel's (r, g, b) values as an element of R^3 , run K-means[1] with 16 clusters on the pixel data from this smaller image, iterating (preferably) to convergence, but in no case for less than 30 iterations. For initialization, set each cluster centroid to the (r, g, b)-values of a randomly chosen pixel in the image.

(c) Take the matrix A from mandrill-large.tiff, and replace each pixel's (r, g, b) values with the value of the closest cluster centroid. Display the new image, and compare it visually to the original image. Hand in all your code and a printout of your compressed image (printing on a black-and-white printer is fine).

(d) If we represent the image with these reduced (16) colors, by (approximately) what factor have we compressed the image?

Note: Make sure to submit a printout of your code and figures.

[1] You should implement K-means yourself, rather than using built-in functions from MATLAB.