EECS445: Introduction to Machine Learning, Fall 2014
Homework #5

Due date: Dec 2 (Tuesday), 6pm

Reminder: While you are encouraged to think about problems in small groups, all written solutions must be independently generated. Please type or hand-write solutions legibly. While these questions require thought, please be as concise and clear in your answer as possible. Please address all questions to `http://piazza.com/class#fall2014/eecs445` with a reference to the specific question in the subject line (E.g. HW5 Q1). For any solutions that require programming, please submit commented code along with any figures that you are asked to plot.

Submission Instructions: Please refer to previous assignment for submission instructions. Please also keep in mind that you are recommended to attach a cover page to your hardcopy which has your name, unique name, names of group members and submission time on it. It is not mandatory but in this way your grader can put your score on the second page so it won't be in plain sight when we return your homework.

1 [25 points] Gaussian mixtures for image compression

In this problem, you will redo the task of Homework 4 Problem 1. Instead of using K-means, you will be using Gaussian Mixture Model.

(a) Load `mandrill-large.tiff` into matlab by typing `A = double(imread('mandrill-large.tiff'));`. Now, `A` is a "three dimensional matrix," and `A(:,:,1)`, `A(:,:,2)` and `A(:,:,3)` are 512×512 arrays that respectively contain the red, green, and blue values for each pixel. Enter `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 vector quantization on a smaller image. Repeat part (a) with `mandrill-small.tiff`. Treating each pixel's (r, g, b) values as an element of $\mathbb{R}^3$, run GMM with full covariances and K=5 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. Use MAP estimation for the latent cluster-assignment variable for each pixel.

(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.

(d) If we represent the image with these reduced (K) colors, by (approximately) what factor have we compressed the image? Is It different from what you have using K-means? Would it be different if K was 16 (same as is it in K-means).

(e) After training the GMM, report the model parameters $\{(\mu_k, \Sigma_k) : k = 1, ..., 5\}$ and the data log-likelihood.

2 [25 points] Gaussian Processes

In this problem, you will be asked to implement Gaussian Process regression. We will assume a parameterization of covariance matrix C for Gaussian process regression as follows:

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{n,m}$$
$$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\left\{-\frac{1}{2\sigma^2}||\mathbf{x}_n - \mathbf{x}_m||^2\right\}$$

For more details of covariance matrix C and kernel function $k(\cdot,\cdot)$, please read Chapter 6.4.2 of the Bishop book. Now suppose that we have a training set $\{(\mathbf{x}_i, t_i); i = 1\ldots, N\}$ of N points. To make the problem simple and comparable to the earlier homework, we will use the dataset that was provided for homework #1 Q2 (for locally-weighted linear regression).

(a) **[15 points]** Implement Gaussian Process regression for arbitrary query points. As we did in Homework #1 Q2, densely draw samples for query points (Hint: in matlab notation, define a set of query points as `min(x):.2:max(x)`, where `x` is a vector of 1-dimensional training examples.), and for any query point $\mathbf{x}^*$, compute the mean and variance of the conditional distribution $p(t^*|\mathbf{x}_1, ..., \mathbf{x}_N, t_1, ..., t_N, \mathbf{x}^*)$. Please draw the original samples and overlay with the predicted regression curve with error bars. (Hint: use the `errorbar` function in matlab.) Try three different pairs of hyperparameter (σ, β^{-1}) values, such as $(1, 0.1)$, $(0.2, 0.5)$, $(10, 0.01)$. and plot the regression curve for each hyperparameter setting.

(b) **[2 points]** What does σ model? What happens when σ value is too small or too large?

(c) **[2 points]** What does β^{-1} model? What happens when β^{-1} value is too small or too large?

(d) **[6 points]** Now, we will search the hyperparameter values to maximize the data log-likelihood.[1]

(i) Concretely, search the grid of hyperparameters $\sigma \in \{0.2, 0.5, 1, 2, 5\}$ and $\beta^{-1} \in \{0.01, 0.03, 0.1, 0.3\}$. For each hyperparameter setting of (σ, β^{-1}) value, compute the log-likelihood of the data in your code (i.e., total 20 values for all possible combinations, but you don't have to report the numbers for all cases). (Hint: look at Section 6.4.3 of the Bishop book.) What are the best hyperparameters among the values you tried? What is the corresponding data-likelihood?

(ii) For the best hyperparameters that you have found, draw the predicted regression curve with error bars (and overlay with the original points).

3 [30 points] Training Sparse Autoencoders

In this question, you will implement a Sparse Autoencoder, by following Stanford's Unsupervised Feature Learning and Deep Learning (UFLDL) Tutorial. Recall from class, an autoencoder consists of 3 layers: the input layer, followed by an "encoder" layer, followed by a "decoder" layer. The encoder layer transforms the input features into a hidden representation (can be thought as an *encoding* of the input feature vector). The decoder layer outputs a vector who's dimensions is equal to the dimensions of the input vector. The autoencoder is trained such that the output of the decoder layer is as close as possible to the input feature vector. In other words, the autoencoder transforms an input example $\mathbf{x}$ into some "hidden representation" (encoding), then tries to *reconstruct* $\mathbf{x}$ from the hidden representation (decoding). The training of an auto encoder is concerned with minimizing the reconstruction error.

Standard Autoencoders make the hidden representation smaller than the dimensions of the input vector: $\mathbf{x} \in \mathbb{R}^M$, $\texttt{encode}(\mathbf{x}) \in \mathbb{R}^H$, where $H < M$. In contrast, sparse autoencoders have a hidden representation that is larger than the input vector (i.e. $H > M$). This setting allows sparse autoencoders to discover interesting structures in the training data. You should read through the first section of the UFLDL tutorial[2]

[1] To simplify the problem, we will simply search over a grid of hyperparameter values; however, it is possible to compute gradient (w.r.t the hyperparameters) and perform gradient descent to find a local maximum of the data-likelihood. For details, see Section 6.4.3 of the Bishop book.

[2] `http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial`

titled **Sparse Autoencoder**

(a) **[18 points + 5 extra credit]** Train and visualize an auto-encoder on natural images, by completing Steps 1 through 5 in: `http://ufldl.stanford.edu/wiki/index.php/Exercise:Sparse_Autoencoder`. Submit all your code. Attach the visualizations to your hardcopy submission (from step 5). It will help for the next part if you *vectorize*[3] your implementation. More importantly, you will get 5 extra credit points if you have a vectorized implementation! (what better than extra credit can be used to enforce good habits?)

(b) **[12 points]** Train your implementation of sparse auto encoder on hand-written digits. Then, map all training (and test) data to the encoding (hidden representation) produced by your trained auto encoder. Complete steps 1, 2 and 3 of: `http://ufldl.stanford.edu/wiki/index.php/Exercise:Self-Taught_Learning` (you are expected to reuse your code from part (a)). Finally, using the extracted features (i.e. the hidden representation, which is output of the encoder layer), use a classifier of your choice to do classification. You may want to use `liblinear` since you have experience with it at this point. Report your classification accuracy. Submit your code.

[3] `http://ufldl.stanford.edu/wiki/index.php/Vectorization`