EECS445: Introduction to Machine Learning, Fall 2014
Homework #4

Due date: 6pm on Tuesday, November 11

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. HW4 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 don't have to print your code for submission any more.

1 [24 points] K-means for image compression

In this problem, we will apply the K-means algorithm to lossy image compression, by reducing the number of colors used in an image. CTools contains a 512x512 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 $262144 \times 3 = 786432$ 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) 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 K-means 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. Comment on what regions of the original image are better preserved and what regions are not preserved well.

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

$$\text{compression factor} = \frac{\text{space used to store original image}}{\text{space used to store compressed image}}$$

2 [20 points] Cross-Validation on Hyper-Parameters of SVM

In the previous assignment (Homework3), you have implemented an SVM classifier using two different optimization methods, Batch Gradient Descent and Stochastic Gradient Descent (SGD). In this question, we will only consider SGD since it converges faster and requires fewer iterations over the training set. SGD solves this minimization:

$$\min_{\mathbf{w},b} E(\mathbf{w},b) = \sum_{i=1}^{N} E^{(i)}(\mathbf{w},b)$$
$$\text{where, } E^{(i)}(\mathbf{w},b) = \frac{1}{2N}||\mathbf{w}||^2 + C\max\left(0, 1 - t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)\right)$$

The pseudo-code for SGD minimization is summarized in Homework3 Q3, in that question, you were given the values for C and η_0 to use. Your task in this question is find good values for C and η_0 (a.k.a hyper-paremeters[1]) using cross validation. Load the matlab file `q2_data.mat`, which loads the variables `q2x_train`, `q2t_train`, `q2x_test` and `q2t_test`.

[Note: the dataset `svm_data.mat` is different than dataset of Homework3 Q3, therefore the C and η_0 may be different than what was recommended for use in Homework3. Nonetheless, you may reuse parts of your code from Homework3]

(a) **[3 points]** Write down the expressions for $\nabla_{\mathbf{w}} E(\mathbf{w},b)$ and $\frac{\partial}{\partial b}E(\mathbf{w},b)$, as well as the SGD update rules for both parameters.

(b) **[8 points]** Implement "hold-out cross validation" using a $(50\% : 50\%)$ split. In other words, train on the first half of the training examples, and validate on the second half of the training examples to choose a good configuration of C and η_0. Try all pairs of C and η_0 where $C \in \{0.01, 0.1, 1, 10, 100, 1000\}$ and $\eta_0 \in \{0.01, 0.5, 1.0, 10, 100\}$. Run 200 iterations of SGD for each combination of C and η_0. Report the C and η_0 that minimize the validation error. If there were multiple pairs (C, η_0) that achieve the minimum error, choose the pair for which η_0 is smallest. Finally, using those cross validated hyper-parameters, train your SVM using the entire training set then compute and report the test error (i.e. number of mis-classified test examples).

(c) **[9 points]** Implement "k-fold cross validation" using $k = 10$, known as 10-fold validation. Try all pairs of C and η_0 where $C \in \{0.01, 0.1, 1, 10, 100, 1000\}$ and $\eta_0 \in \{0.01, 0.5, 1.0, 10, 100\}$. Run 200 iterations of SGD for each combination of C and η_0. Report the C and η_0 that minimize the validation error. Finally, using those cross validated hyper-parameters, train your SVM using the entire training set then compute and report the test error (i.e. number of mis-classified test examples).

3 [20 points] Intuitions behind Kernels in classification

In the previous assignment, you have proven some useful properties of kernels. In addition, you have Kernelized an algorithm. In this (short) question, you will explore why Kernels are useful. For this question, load the matlab file `q3_data.mat`, which loads the variables `q3x_train`, `q3t_train`, `q3x_test` and `q3t_test`.

[1] The parameters of SVM are $\mathbf{w}$ and b, which are fit to the training data. The hyper-parameters are *chosen* before training begins.

(a) **[6 points]** Train an SVM over the training data using matlab's built-in `svmtrain` using a linear kernel (i.e. no kernel). Plot the training data and the separating hyperplane. Use `svmclassify` to classify all test examples. Calculate and report the classification accuracy.

[hint: svmtrain accepts a 'showplot' argument which plots the training points and the separating hyperplane]

(b) **[5 points]** Train an SVM over the training data using a Radial Basis Function (RBF) kernel, also known as Gaussian Kernel. Plot the training data and the separating hyperplane. Report the test classification accuracy using the RBF kernel.

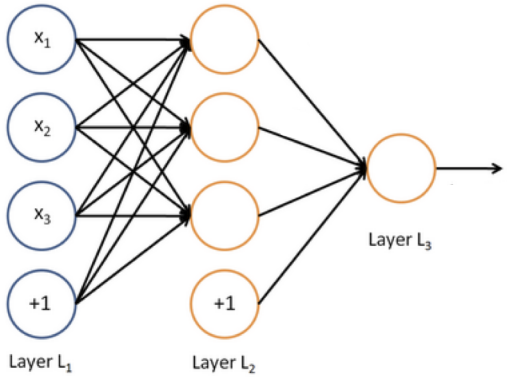
(c) **[3 points]** Which Kernel did better? RBF or linear (/ no kernel)? Why did this happen?

(d) **[6 points]** Use 5-fold cross validation to determine the best hyper-parameter, σ for the SVM with RBF kernel. Try $\sigma \in \{0.2, 0.5, 1, 1.5, 2, 2.5, 3\}$ *[Note: you can use the* `help` *command to find out how to setup hyper parameters when using the* `svmtrain` *command*

(e) **[Extra credit 8 points]** Repeat step a through d using LibSVM instead of Matlab's sum toolkit. You can find information about how to download and use LibSVM by Google *libsvm* and *download libsvm.*

4 [16 points] Update rules for a 2-layer Neural Network

In this question, you will derive the update rules for a 2-layer neural network. Our small Neural Network is depicted in this diagram:



- The first (input) layer takes a training example $\mathbf{x}$ that is 3-dimensional ($M = 3$).
- Given an example $\mathbf{x}$ on the input layer, the second layer (a.k.a first hidden layer) computes a tanh(.) transformation of the input layer[2]. In particular, each of the 3 depicted hidden neurons compute the the following:

$$h_k = \tanh\left(b_k + \sum_{j=1}^{3} w_{kj} x_j\right) = \tanh\left(b_k + \mathbf{w_k}^T \mathbf{x}\right); \text{ for } k = 1, 2, 3$$

Where w_{kj} is the weight of the parameter (or edge, as depicted in the graph) connecting x_j and h_k, and the vector $\mathbf{w_k}^T = [w_{k1}, w_{k2}, w_{k3}]$ As discussed in class, we can define a $\mathbf{W}$ matrix to contain all $\mathbf{w_k}$ vectors like:

$$\mathbf{W} = \begin{bmatrix} \text{\_\_\_\_\_} & \mathbf{w}_1^T & \text{\_\_\_\_\_} \\ \text{\_\_\_\_\_} & \mathbf{w}_2^T & \text{\_\_\_\_\_} \\ \text{\_\_\_\_\_} & \mathbf{w}_3^T & \text{\_\_\_\_\_} \end{bmatrix}$$

[2] the lecture slides define $z_k = b_k + \mathbf{w_k}^T \mathbf{x}$ and $h_k = \tanh(z_k)$

- Using the matrix $\mathbf{W}$, we can define the vector $\mathbf{h}$ to be:

$$\mathbf{h} = \tanh\left(\mathbf{b} + \mathbf{W}\mathbf{x}\right)$$

  Where $\mathbf{h} = [h_1, h_2, h_3]$ and $\mathbf{b} = [b_1, b_2, b_3]$

- Finally, the output layer (which contains only one, output neuron) is a logistic layer. In particular, the output neuron is computes the following:

$$\hat{t} = \sigma(b_4 + \theta^T \mathbf{h})$$

  where $\sigma(.)$ is the logistic function, $\sigma(a) = \frac{1}{1+e^{-a}}$, θ is the vector of containing the weights of edges connecting hidden neurons to the output neuron, and b_4 is the bias for the output neuron.

Given a training example $\mathbf{x}$ and its label t, we define the error on the example as the cross-entropy loss:

$$E(\mathbf{x}, t) = -\left(t \log(\hat{t}) + (1-t)\log(1-\hat{t})\right)$$

If we want to optimize the network parameters $(\theta, b_4, \mathbf{W}, \mathbf{b})$ via SGD, we would first need to find expressions for the derivatives of the error function above with respect to each of the parameters.

(a) **[6 points]** Given a training example $(\mathbf{x}, t)$, find the derivatives of the error function with respect to parameters of the output neuron. In particular, calculate expressions for $\nabla_\theta E(\mathbf{x}, t)$ and $\frac{\partial}{\partial b_4} E(\mathbf{x}, t)$.

(b) **[10 points]** Given a training example $(\mathbf{x}, t)$, find the derivatives of the error function with respect to parameters of the hidden layer. In particular, calculate expressions for $\nabla_{\mathbf{W}} E(\mathbf{x}, t)$ and $\nabla_{\mathbf{b}} E(\mathbf{x}, t)$.