COL 774: Assignment 2

Due Date: 11:50 pm, Sunday Mar 15, 2015. Total Points: 72 (+ 8 extra credit)

Notes:

- This assignment has a mix of theoretical as well as implementation questions.
- Only the implementation questions will be graded.
- You are strongly encouraged to try out theoretical questions though they are not graded.
- You should submit all your code as well as any graphs that you might plot. Do not submit answers to theoretical questions.
- Do not submit the datasets.
- Include a **single write-up (pdf) file** which includes a brief description for each question explaining what you did. Include any observations and/or plots required by the question in this single write-up file.
- You should use MATLAB for first two programming questions.
- You have a choice of using MATLAB/C/C++/Java/Python for the third programming question.
- Your code should have appropriate documentation for readability.
- You will be graded based on what you have submitted as well as your ability to explain your code.
- Refer to the course website for assignment submission instructions.
- This assignment is supposed to be done individually. You should carry out all the implementation by yourself.
- We plan to run Moss on the submissions. We will also include submissions from previous years since some of the questions may be repeated. Any cheating will result in a zero on the assignment and possibly much stricter penalties (including a **fail grade** and/or a **DISCO**).
- Many of the problems (and the datasets) below have been adapted from the Machine Learning courses offered by Andrew Ng, Carlos Guestrin, Pedro Domingos and possibly other researchers at their respective universities.

1. **(30 points) Spam Classification**
   In this problem, we will use Support Vector Machines (SVMs) to build a spam classifier. We will be solving the SVM optimization problem using a general purpose convex optimization package as well using a customized solver known as libSVM. The dataset we will be using is a subset of 2005 TREC Public Spam Corpus. It contains a training set and a test set. Both files use the same format: each line represents the space-delimited properties of an email, with the first one being the email ID, the second one being whether it is a spam or ham (non-spam), and the rest are words and their occurrence numbers in this email. The dataset presented to you is processed version of the original dataset where non-word characters have been removed and some basic feature selection has been done. You can download the dataset from here.

(a) **(8 points)** Download and install the CVX package. Express the SVM dual problem (with a linear kernel) in the a form that the CVX package can take. You will have to think about how to express the SVM dual objective in the form $\alpha^T Q\alpha + b^T\alpha + c$ matrix where Q is an $m \times m$ matrix (m being the number of training examples), b is an m-sized column vector and c is a constant. For your optimization problem, remember to use the constraints on α_i's in the dual. Use $C = 1$. Report the set of support vectors obtained from your optimization.

(b) **(6 points)** Calculate the weight vector w and the intercept term b using the solution in the part above. Classify the test examples as spam or non-spam. Report the average accuracy obtained.

(c) **(6 points)** Now solve the dual SVM problem using a Gaussian kernel with the bandwidth parameter $\gamma = 2.5 * 10^{-4}$. Think about how the Q matrix will be represented. What are the set of support vectors in this case? Note that you may not be able to explicitly store the weight vector (w) or the intercept term (b) in this case. Use your learned model to classify the test examples and report the accuracies obtained. How do these compare with the ones obtained with the linear SVM?

(d) **(10 points)** Now train an SVM on this dataset using the LibSVM library, available for download from LibSVM. Repeat the parts above using a linear Kernel as well as a Gaussian kernel with $\gamma = 2.5 * 10^{-4}$. Use $C = 1$ in both cases, as before. Report the set of support vectors obtained as well as the test set accuracies for both linear as well as the Gaussian kernel setting. How do these compare with the numbers obtained using the CVX package. Comment.

Note: You do not need to submit the CVX or LibSVM code. But you should submit any code that you wrote as a wrapper to get them to run on this dataset.

2. **(20 points + 8 extra credit) Digit Recognition**
In this problem, you are given the MNIST handwritten digit dataset[1] (mnist_all.mat) that contains 60K training and 10K testing examples of handwritten digits. Each example in the dataset is represented by 784 features corresponding to (28×28) pixel values ([0, 255]). The classes are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 corresponding to each digit. Examples are partitioned based on the class to which they belong. We will implement the neural network learning algorithm (using backpropagation) to recognize the digits given the pixel values.

(a) **(4 points)** Write a script to visualize the digits in the data. Your script should take a file/example index and display the image corresponding to the gray scale pixel values.

(b) **(12 points)** Extract the data for classes 3 and 8 from the original mnist_all.mat file to create a new mnist_bin38.mat file for binary classification. Train a neural network with one hidden layer (with 100 units) using the backpropagation algorithm. You should implement the algorithm from first principles and not use any existing matlab modules. Use the stochastic gradient version of the algorithm. Use a variable learning rate given as $\alpha_t = \frac{1}{\sqrt{t}}$ where t denotes the learning iteration. Choose an appropriate stopping criteria based on the change in value of the error function. Report the stopping criteria that you chose.

(c) **(4 points)** Report your accuracies over the test set using the learned network. Also report the training times of your algorithm.

(d) (**Extra Credit: 8 points**) Train the neural network classifier on the original multiclass MNIST dataset with the same experimental settings as in the binary case above. How many output units would you need in this case? Report the accuracies over the test set. Do you see any difference in training times compared to the binary setting?

3. **(22 points) Decision Trees for Classification**
In this problem, we will work with one of the UCI datasets on US congressional voting. Click here to read the data description. The goal is to build a decision tree which would learn a model to predict whether a US congressman (equivalent of a Member of Parliament in India) is democrat or republican based their voting pattern on various issues [2]. The dataset provided to you has been split into 3 disjoint subsets: training data, validation data and test data [3]. For this problem, you may find it helpful to read Chapter 3 of Mitchell's book and/or the original paper on ID3 algorithm by Ross Quinlan (available on the website) in addition to the class notes/slides.

[1] data is available at `http://www.cs.nyu.edu/~roweis/data.html`
[2] Read more about the distinction between democrats and republicans here.
[3] The splits were made available courtesy one of the machine learning courses offered at the University of Washington

(a) **(10 points)** Construct a decision tree using the given data to classify a congressman as democrat or republican. Use net error as the criterion for choosing the attribute to split on. In case of a tie, choose the attribute with the lowest index as the splitting attribute. For now, you can treat the missing values ("?") simply as another attribute value. Consider splitting each attribute using a 3-way split i.e. using the values y/n/"?" [4] Plot the train, validation and test set accuracies against the number of nodes in the tree as you grow the tree. On X-axis you should plot the number of nodes in the tree and Y-axis should represent the accuracy. Comment on your observations.

(b) **(4 points)** Repeat the part above using the information gain as the criterion. Again plot the train, validation and test set accuracies as you grow the tree. Comment on your observations. Is the tree obtained in this part very different from the one obtained in part(a) above? Comment.

(c) **(6 points)** One of the ways to reduce overfitting in decision trees is to grow the tree fully and then use post-pruning based on a validation set. In post-pruning, we greedily prune the nodes of the tree (and sub-tree below them) by iteratively picking a node to prune so that resultant tree gives maximum increase in accuracy on the validation set. In other words, among all the nodes in the tree, we prune the node such that pruning it (and sub-tree below it) results in maximum increase in accuracy over the validation set. This is repeated until any further pruning leads to decrease in accuracy over the validation set. Post prune the tree obtained in step (b) above using the validation set. Again plot the the training, validation and test set accuracies against the number of nodes in the tree as you successively prune the tree. Comment on your findings.

(d) **(2 points)** In the above problem, we simply treated the missing values ("?") as another attribute value. What might be a more principled way of handling the missing data? (You do not have to write any code for this part.)

Following problems are for your practice and will not be graded.

1. **Constructing Kernels**

In class, we saw that by choosing a kernel $K(x, z) = \phi(x)^T\phi(z)$, we can implicitly map data to a high dimensional space, and have the SVM algorithm work in that space. One way to generate kernels is to explicitly define the mapping ϕ to a higher dimensional space, and then work out the corresponding K.

However in this question we are interested in direct construction of kernels. I.e., suppose we have a function $K(x, z)$ that we think gives an appropriate similarity measure for our learning problem, and we are considering plugging K into the SVM as the kernel function. However for $K(x, z)$ to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping ϕ. Mercer's theorem tells us that $K(x, z)$ is a (Mercer) kernel if and only if for any finite set $\{x^{(1)}, \cdots, x^{(m)}\}$, the matrix K is symmetric and positive semidefinite, where the square matrix $K \in R^{m\times m}$ is given by $K_{ij} = K(x^{(i)}, x^{(j)})$.

Now here comes the question: Let K_1 , K_2 be kernels over $R^n \times R^n$, let $a \in R^+$ be a positive real number, let $f : R^n \longmapsto R$ be a real-valued function, let $\phi : R^n \longmapsto R^d$ be a function mapping from R^n to R^d , let K_3 be a kernel over $R^d \times R^d$, and let $p(x)$ a polynomial over x with positive coefficients. For each of the functions K below, state whether it is necessarily a kernel. If you think it is, prove it; if you think it isn't, give a counter-example.

(a) $K(x, z) = K_1(x, z) + K_2(x, z)$

(b) $K(x, z) = K_1(x, z) - -K_2(x, z)$

(c) $K(x, z) = aK_1(x, z)$

(d) $K(x, z) = -aK_1(x, z)$

(e) $K(x, z) = K_1(x, z)K_2(x, z)$

(f) $K(x, z) = f(x)f(z)$

(g) $K(x, z) = K_3(\phi(x), \phi(z))$

(h) $K(x, z) = p(K_1(x, z))$

[4] Once you implement this basic version, you are free to try more fancy multiple two-way splits e.g. y/others followed by a possible further split on n/"?" at a descendant node.

2. **Kernelizing the Perceptron**

Let there be a binary classification problem with $y \in \{0, 1\}$. The perceptron uses hypotheses of the form $h_\theta(x) = g(\theta^T x)$, where $g(z) = \mathbf{1}\{z \geq 0\}$. In this problem, we will consider a stochastic gradient descent-like implementation of the perceptron algorithm where each update to the parameters θ is made using only one training example. However, unlike stochastic gradient descent, the perceptron algorithm will only make one pass through the entire training set. The update rule for this version of the perceptron algorithm is given by

$$\theta^{(i+1)} := \theta^{(i)} + \alpha[y^{(i+1)} - h_{\theta^{(i)}}(x^{(i+1)})]x^{(i+1)}$$

where $\theta^{(i)}$ is the value of the parameters after the algorithm has seen the first i training examples. Prior to seeing any training examples, $\theta^{(0)}$ is initialized to $\vec{0}$.

Let K be a Mercer kernel corresponding to some very high-dimensional feature mapping ϕ. Suppose ϕ is so high-dimensional (say, ∞-dimensional) that it's infeasible to ever represent $\phi(x)$ explicitly. Describe how you would apply the "kernel trick" to the perceptron to make it work in the high-dimensional feature space ϕ, but without ever explicitly computing $\phi(x)$. [Note: You don't have to worry about the intercept term. If you like, think of ϕ as having the property that $\phi_0(x) = 1$ so that this is taken care of.] Your description should specify

(a) How you will (implicitly) represent the high-dimensional parameter vector $\theta^{(i)}$, including how the initial value $\theta^{(0)} = \vec{0}$ is represented (note that $\theta^{(i)}$ is now a vector whose dimension is the same as the feature vectors $\phi(x)$);

(b) How you will efficiently make a prediction on a new input $x^{(i+1)}$. I.e., how you will compute $h_{\theta^{(i)}}(x^{(i+1)}) = g({\theta^{(i)}}^T \phi(x^{(i+1)}))$, using your representation of $\theta^{(i)}$; and

(c) How you will modify the update rule given above to perform an update to θ on a new training example $(x^{(i+1)}, y^{(i+1)})$; i.e., using the update rule corresponding to the feature mapping ϕ:

$$\theta^{(i+1)} := \theta^{(i)} + \alpha[y^{(i+1)} - h_{\theta(i)}(x^{(i+1)})]\phi(x^{(i+1)})$$

[Note: If you prefer, you are also welcome to do this problem using the convention of labels $y \in \{-1, 1\}$, and $g(z) = sign(z) = 1$ if $z \geq 0, -1$ otherwise.]

3. **Neural Networks**

One of the ways to avoid overfitting during neural network learning is to add a penalty term to the cost function, which penalizes large weights. This has the effect of trading-off minimization of the squared error with keeping some function of the weights' magnitude low. Typically, 2-norm of the weight vector is used as the penalty term. Consider the alternate error function defined as:

$$J(\theta) = \frac{1}{2} \sum_{i \in \{1 \cdots m\}} \sum_{k \in outputs} (y_k^{(i)} - o_k^{(i)})^2 + \gamma \sum_{l,k} \theta_{kl}^2$$

Derive the stochastic gradient descent update rule for this definition of J. Show that it can be implemented by multiplying each weight by some constant before performing the standard gradient descent update.

4. **Dealing with Non-Linear Hypotheses**

Consider learning the target concepts corresponding to circles in the x, y plane. Describe each hypothesis h_{cr} by the co-ordinates of the center (c_x, c_y) and the radius r of the circle. An instance (x, y) is labeled positive by hypothesis h_{cr} if and only if the point (x, y) lies inside the corresponding circle.

(a) Write the mathematical expression for the classification rule imposed by a hypothesis h_{cr} as defined above.

(b) Let the training error of a hypothesis be defined as in the case of perceptron i.e. number of examples classified incorrectly by the hypothesis. Can you devise a gradient descent algorithm to the learn the hypothesis minimizing this error function. Argue.

(c) Devise an alternate error function so that error is a continuous function of the inputs. You should come up with an error function which is reasonable i.e. makes intuitive sense.

(d) Derive the gradient descent rule for the error function defined in the part above.