CSL 341: Assignment 2

Due Date: 11:50 pm, Thursday Oct 31, 2013. Total Points: 64

Notes:

- This assignment has a mix of theoretical as well as implementation questions.
- Only the implementation questions will be graded.
- All the coding problems should be done in Matlab.
- 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.
- For each problem that you implement, you should include a file <question-no.writeup.txt> which should have a brief explanation of what you did.
- 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. Any cheating will result in a zero on assignment and possibly even heavier penalties (including a fail grade).
- Some of the problems below have been adapted (or borrowed) from the Machine Learning course offered by Andrew Ng at Stanford University. Two of the coding problems (and the datasets thereof) have been borrowed/adapted from the Machine Learning courses offered at the University of Washington, Seattle.

1. **(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 [1]. The dataset provided to you has been split into 3 disjoint subsets: training data, validation data and test data [2]. 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.

   (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/"?" [3] 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.

[1] Read more about the distinction between democrats and republicans here.

[2] The splits were made available courtesy one of the machine learning courses offered at the University of Washington

[3] 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.

(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.)

2. **(22 points) Naïve Bayes for Newsgroup Classification** In this problem, we will use the naïve Bayes algorithm to learn a model for classifying an article into a given set of newsgroups. The data and its description is available through the UCI data repository. The original data is description available here. We have processed the data further to remove punctuation symbols, stopwords etc. The processed dataset contains the subset of the articles in the newsgroups rec.* and talk.*. This corresponds to a total of 7230 articles in 8 different newsgroups. The processed data is made available to you in a single file with each row representing one article. Each row contains the information about the class of an article followed by the list of words appearing in the article.

   (a) **(10 points)** Implement the Naïve Bayes algorithm to classify each of the articles into one of the newsgroup categories. Randomly divide your data into 5 equal sized splits of size 1446 each. Make sure NOT TO create the splits in any particular order (e.g. picking documents sequentially) otherwise it may lead to skewed distribution of classes across the splits. Now, perform 5-fold cross validation (train on each possible combination of 4 splits and the test on the remaining one). Report average test set accuracies.

   Notes:

   - Make sure to use the Laplace smoothing for Naïve Bayes (as discussed in class) to avoid any zero probabilities.
   - You should implement your algorithm using logarithms to avoid underflow issues.
   - You should implement naïve Bayes from the first principles and not use any existing Matlab modules.

   (b) **(2 points)**What is the accuracy that you would obtain by randomly guessing one of the newsgroups as the target class for each of the articles. How much improvement does your algorithm give over a random prediction?

   (c) **(2 points)** Some (4% in the original data) of the articles were cross-posted i.e. posted on more than one newsgroup. Does it create a problem for the naïve Bayes learner? Explain.

   (d) **(4 points)** For each of the train/test splits in part (a) above, vary the number of articles used in training from 1000 to 5784 (training split size) at the interval of 1000 and evaluate on the test split (sized 1446). Plot on a graph the train as well as the test set accuracies (y-axis) averaged over the 5 random splits, as you vary the number of training examples (x-axis). What do you observe? Explain.

   (e) **4 points** Read about the confusion matrix. Draw the confusion matrix for your results in the part (a) above. Which newsgroup has the highest value of the diagonal entry in the confusion matrix? Which two newsgroups are confused the most with each other i.e. which is the highest entry amongst the non-diagonal entries in the confusion matrix? Explain your observations. Include the confusion matrix in your submission.

   Note: You should submit all your code including any code written for processing the data.

3. **(20 points) K-means for Digit Recognition**

   In this problem, you will be working with a subset of the OCR (optical character recognition dataset) from the Kaggle website. This data was originally taken from the MNIST digit recognition database, but was

converted into an easier to use file format. Each of the images in the dataset is represented by a set of 784 (28×28) grayscale pixel values varying in the range $[0, 255]$. The data was further processed [4] so that it could be easily used to experiment with K-means clustering. The processed dataset contains 1000 images for four different $(1, 3, 5, 7)$ handwritten digits. Each image is represented by a sequence of 157 grayscale pixel values (a subset of the original 784 pixel values). A separate file is provided which contains the actual digit value for each of the images. We will implement the k-means clustering algorithm to categorize each of the images into one of the $k = 4$ digit clusters given the pixel values. Follow the instructions below. Be aware that some of these operations can take a while (several minutes even on a fast computer)!

(a) **(2 points)** Write a program to visualize the digits in the data. Your script should take an example index and display the image corresponding to the gray scale pixel values. You can assume a grayscale value of 0 for the pixel values not present in the file.

(b) **(10 points)** Implement the k-means ($k = 4$) clustering algorithm (as discussed in class) until convergence to discover the clusters in the data. Start from an initial random assignment of the cluster means. You can stop at 30 iterations if you find that the algorithm has still not converged. Report your findings.

(c) **(4 points)** One of the ways to characterize the "goodness" of the clusters obtained is to calculate the sum of squares of distance of each of the data points $x^{(i)}$ from the mean of the cluster it is assigned to i.e. the quantity $S = \Sigma_i (x^{(i)} - \mu_{k_i})^2$ where $x^{(i)}$ denotes the ith data point, k_i is the index of the cluster that $x^{(i)}$ belongs to ($1 \leq k_i \leq k$), k is the total number of clusters and μ_{k_i} denotes the mean of the cluster with index k_i. S essentially captures how close the points within each cluster are (or equivalently, how far points across different clusters are). Presumably, greater the value of S, better the clustering is. Plot the quantity S as we vary the number of iterations from 1 to 20. Comment on your findings.

(d) **(4 points)** Since we have the true labels of the data in this example, we can plot the error of clustering as we run the K-means algorithm. For each cluster obtained at a given step of k-means, assign to the cluster the label which occurs most frequently in the examples in the cluster. The remaining examples are treated as misclassified. Plot the error (ratio of the number of mis-classified examples to the total number of examples) as we vary the number of iterations from 1 to 20. Comment on your observations.

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

[4] Pre-processing was made available courtesy one of the machine learning courses offered at the University of Washington.

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

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

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

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. **Decision Trees**

(a) Give decision trees to represent the following Boolean functions:

i. $A \wedge \neg B$

ii. $A \vee [B \wedge C]$

iii. $A\ XOR\ B$

iv. $[A \wedge B] \vee [C \wedge D]$

(b) Consider the set of examples given in Table 3b:

Instance	Classification	a_1	a_2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

Table 1: Examples

i. What is the entropy of this collection of training examples with respect to the target classification?

ii. What is the information gain of a_2 relative to these training examples?

(c) Given two random variables X and Y, the mutual information between X and Y is defined as $I(X,Y) = H(Y) - \sum_{x_k} H(Y|X = x_k) * P(X = x_k)$. Show that mutual information is symmetric i.e. $I(X,Y) = I(Y,X)$.

4. **Neural Networks:**

(a) 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.

(b) 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.

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

ii. 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.

iii. 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.

iv. Derive the gradient descent rule for the error function defined in the part above.

v. Is your algorithm guaranteed to converge the global optimum? Why?