COL 774: Assignment 3

Due Date: 11:50 pm, Tuesday April 4, 2017. Total Points: 52

Notes:

- This assignment has two implementation questions.
- You should submit all your code (including any pre-processing scripts written by you) and any graphs that you might plot.
- Do not submit the datasets. Do not submit any code that we have provided to you for processing.
- 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/Python for the neural network question (Question 2). For decision tree implementation (Question 1), you are also free to use C++/Java.
- 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, a penalty of -10 points and possibly much stricter penalties (including a **fail grade** and/or a **DISCO**).

1. **(28 points) Decision Trees (and Random Forests):**
   In this problem, you will work with the Covertype Dataset available for download from the UCI repository. Read about the dataset in detail from the link given above. You have been provided with a subset of dataset to work with (available for download from the course website). Specifically, you have been provided with separate trainig, validation and testing sets. The first row in each file specifies the type of each attribute. The last entry in each row denotes the class label. You have to implement the decision tree algorithm for predicting the forest cover type based on a variety of cartographic features. You will also experiment with random forests in the last part of this problem.

   (a) **(10 points)** Construct a decision tree using the given data to predict the forest cover type. You should use mutual information as the criteria for selecting the attribute to split on. While calcuating the mutual information, for discrete attributes, consider a multi-way split on each of the attribute values. For handling continuous attributes, you should use the following procedure. At any given internal node of the tree, a numerical attribute is considered for a two way split by calculating the median attribute value from the data instances coming to that node, and then computing the information gain if the data was split based on whether the numerical value of the attribute is greater than the median or not. Note that in this setting, a numerical attribute can be considered for splitting multiple number of times. At any step, choose the attribute which results in highest mutual information. 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.

.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
x	o	.	.	.	.	.
o	x	x	o	x	.	.
x	o	o	x	o	x	o

Table 1: A typical Connect-4 board configuration

(b) **(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 (a) above using the validation set. Again plot 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. We have posted the relevant sections from Mitchell's textbook for this part on the course website (aceessible only through an internal link inside IIT).

(c) **(6 points)** A number of libraries are available for decision tree implmentation. Use the scikit-learn library of Python to grow a decision tree. Click here to read the documentation and the details of various parameter options. Try growing different trees by playing around with parameter values. Some parameters that you should specifically experiment with include min_samples_split, min_samples_leaf and max_depth (feel free to vary other paramters as well). How does the validation set accuracy change as you try various parameter settings? Comment. Find the setting of parameters which gives you best accuracy on the **validation set**. Report training, validation and test set accuracies for this parameter setting. How do your numbers compare with those you obtained in part (b) above (obtained after pruning)?

(d) (**6 points**) Next, use the scikit-learn library to learn a random forest over the data. Click here to read the documentation and the details of various parameter settings. Try growing different forests by playing around with parameter values. Some parameters that you should specifically experiment with include n_estimators, max_features and bootstrap (feel free vary other parameters as well). How does the validation set accuracy change as you try various parameter settings? Comment. Find the setting of parameters which gives you best accuracy on the **validation set**. Report training, validation and test set accuracies for this parameter setting. How do your numbers compare with those you obtained in parts (b) and (c) above. Comment on your observations. ~~You can modify the sample code provided above for decision trees apprpriately to work with random forests.~~

2. **(24 Points) Neural Networks:**
Read about the game of Connect-4 online. It is essentially a 2-player game where the goal of each player is to get four of her markers contiguously in a row, column or diagonal. The player who gets to do this first wins the game. Let us denote the two players by 'x' and 'o'. A typical Connect-4 board configuration looks like Table 1 where 'x' denotes a position occupied by the first player, 'o' denotes a position occupied by the second player and '.' denotes an empty position. The goal of the learning problem is: given a valid intermmediate board configuration (where possibly neither of the players has won yet), learn the game theoretic outcome (i.e. when the two players play optimally) for the first ('x') player. Each example is represented by a set of $42 * 3$ attributes where there is a triplet of attributes for each of the 42 board positions. For each such triplet, the attribute value is ('1','0','0') if the position is occupied by the 'x' player, ('0','1','0') if the position is occupied by the 'o' player, and ('0','0','1') otherwise. The class label is 'win', 'draw' or 'lose' depending the game theoretic outcome for 'x' player. You are provided with a total of 50,000 training examples and 17,557 test examples. In this problem, we learn a nerual network to predict the final game outcome given an intermmediate board configuration as described above. Click here to read more about the dataest. You should use the processed version of the dataset provided to you on the course website.

(a) **(12 points)** Train a neural network with one hidden layer using the backpropagation algorithm. You should implement the algorithm from first principles and not use any existing matlab/python modules. You will need to experiment with the right number of hidden units to use. To start off, set the number

of hidden units to 100. You should use the stochastic gradient version of the algorithm. Choose an appropriate (fixed) learning rate and stopping criteria for your algorithm and report them. Report the train as well as test set accuracies obtained by your model. Also, report the time taken by your algorithm to learn the parameters.

(b) (**4 points**) Now, vary the learning rate η being inversely proportional to number of iterations i.e. $\eta \propto \frac{1}{\sqrt{t}}$. What happens to your convergence rate? Report your train and test set accuracies. Do you get the same set of accuracies that you obtained in the part (a) above? Comment.

(c) (**4 points**) Vary the number of hidden units in the range 50 to 500 at intervals of 50 (using the fixed learning rate used in the part (a) above). Plot the accuracy over the test set as you vary the number of units. What do you observe?

(d) (**4 points**) In class, we used the sigmoid function to introduce non-linearity in neural networks. Another possible alternative for introducing non-linearity is Rectified Linear Unit (ReLU). Train your neural network with ReLU as the activation function. Specifically, you should use the softplus function described on the page above. Train your neural network using the same number of units as in part (a). Use a fixed learning rate (you might have to experiment for finding the right learning rate for this problem setting). Report the train and test set accuracies. Does this setting work any better than the sigmoid case? Comment.