COL 774: Assignment 3

Due Date: 11:50 pm, Sunday April 8, 2018. Total Points: (32+36)

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 both questions.
- 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. **(32 points) Decision Trees (and Random Forests):**
   In this problem, you will work with the Adult Dataset available on the UCI repository. Read about the dataset in detail from the link given above. For the purpose of this assignment, you have been provided with a subset of this dataset available for download from the course website. Specifically, you have been provided with separate training, 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 been provided a file which specifies which attributes are discrete and which are continuous. You have also been provided with a script to read the data into desired format (python/matlab). You have to implement the decision tree algorithm for predicting whether a person is rich (1) or not (0) based on various personal attributes. You will also experiment with random forests in the last part of this problem.

   (a) **(10 points)** Construct a decision tree for the above prediction problem. Preprocess (before growing the tree) each numerical attribute into a Boolean attribute by a) computing the median value of the attribute in the training data (make sure not to ignore the duplicates) b) replacing each numerical value by a 1/0 value based on whether the value is greater than the median threshold or not. Note that this process should be repeated for each attribute independently. For non-Boolean (discrete valued) attributes, you should use a multi-way split as discussed in class. Use information gain as the criterion for choosing the attribute to split on. In case of a tie, choose the attribute which appears first in the ordering as given in the training data. 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) **(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.

(c) **(6 points)** In Part(a) we used the median value of a numerical attribute to convert it in a 1/0 valued attribute as a pre-processing step. In a more sophisticated setting, no such pre-processing is done in the beginning. 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. As earlier, the node is split on the attribute which maximizes the information gain. Note that in this setting, the original value of a numerical attribute remains intact, and a numerical attribute can be considered for splitting in the tree multiple times. Implement this new way of handing numerical attributes. Report the numerical attributes which are split multiple times in a branch (along with the maximum number of times they are split in any given branch and the corresponding thresholds). Replot the curves in part (a). Comment on which results (Part (a) or Part (c)) are better and why. You dont have to implement pruning for this part.

(d) **(5 points)** A number of libraries are available for decision tree implementation. 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 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 part (b) above (obtained after pruning)?

(e) (**5 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.

2. **(36 Points) Neural Networks:**

In this problem, we will work with neural network learning.

(a) **(10 points)** Write a program to implement a generic neural network architecture. Implement the backpropagation algorithm to train the network. You should train the network using Stochastic Gradient Descent (SGD) where the batch size is an input to your program. Your implementation should be generic enough so that it can work with different architectures. Specifically, your program should be able to accept the following parameters: # of inputs and a list of numbers. Here, the size of the list denotes the # of hidden layers in the network and each number in the list denotes the number of units (perecptrons) in the corresponding hidden layer. E.g., a list [100 50] means that there are two hidden layers with first one having 100 units and second one having 50 units. You can assume that the output is Boolean valued. Additionally, you should input a parameter denoting the size of the batch for SGD. Assume a fully connected architecture i.e., each unit in a hidden layer is connected to every unit in the next layer. You should implement the algorithm from first principles and not use any existing matlab/python modules. Use the sigmoid function as the activation unit.

(b) **Visualizing Decision Boundary: (14 points)** You have been given a 2-dimensional dataset. You have also been provided with a script containing functions to visualize it. Since this is a small dataset, we will use the standard gradient descent for training. (i.e., batch size = # of examples).

i. (**2 points**) Train your network using the logistic regression learner available from the scikit-learn library of Python: logistic regression learner. Compute the train and test set accuracies. Also, plot the decision boundary along with the train samples. Repeat this for the test samples.

ii. (**4 points**) Next, use your implementation to train your neural network with a single hidden layer having 5 units. Compute the train and test set accuracies obtained by your trained model. Plot the decision boundary and visualize the train samples along with the decision boundary. Repeat the same for the test samples. How do your results compare with the ones obtained using a logistic regression classifier? Comment on your observations.

iii. (**4 points**) Vary the number of units in the hidden layer as (1,2,3,10,20,40). Note that we are still working with a single hidden layer architecture. Report the train and test set accuracies in each case. Also, plot the decision boundary along with the test samples (no need to plot the train samples for this part). How does the decision boundary change as we increase the number of hidden units? What is the optimal # of units for this problem based on your experimentation? Why?

iv. (**4 points**) Train a network with two hidden layers each having 5 units. Again report the train and test accuracies. Plot the decision boundary along with test samples. How do your results compare with accuracies obtained in the previous part(s)? Comment.

(c) **Working with MNIST (12 points):** In this part, we will use our algorithm to train a model on the MNIST dataset used in Assignment 2 (SVM problem). We will restrict our attention to only two classes: digits 6 and 8. For your convenience, the two class dataset has been again released for this problem. While training your neural network for this problem, use a batch size of 100 in SGD.

i. (**3 points**) Use LIBSVM to train a linear classifier as done in Assignment 2. Use C=1. Next, use your implementation to train a network with a single perceptron on this dataset. Compare the train and test set accuracies obtained in each case. Comment.

ii. (**5 points**) Use your implementation to learn a network with a single hidden layer having 100 units. vary the learning rate η being inversely proportional to number of iterations i.e. $\eta \propto \frac{1}{\sqrt{t}}$ where t is the current learning iteration number. Choose an appropriate stopping criterion based on the change in value of the error function. Report your stopping criterion and the train and test set accuracies. How do your results compare with those obtained in the part above? Comment on your observations.

iii. (**4 points**) In this part, we will use ReLU as the activtion instead of the sigmoid function. ReLU is defined using the function: $g(z) = max(0, z)$. Change your code to work with the ReLU activation unit. Make sure to correctly implement gradient descent by making use of sub-gradient at $z = 0$. Note that ReLU should only be used as the activation in the hidden layer(s). You should still use sigmoid in the final (output) layer since we are dealing with Boolean output. Train a network with a single hidden layer having 100 units using ReLU as the activation function. Report your train and test set accuracies. ~~Also, plot the decision boundary along with the test samples.~~ Do you see any improvements in the results compared to the case when sigmoid is used as the activation function? Comment.