COL 774: Assignment 3

Due Date: 11:50 pm, April 5, 2019. Total Points: (35 + 35)

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 the 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. **(35 points) Decision Trees (and Random Forests):** In this problem, you will work with the Credit Card Defaults of Clients dataset available on the UCI repository. Read about the dataset in detail from the link given above. For the purpose of this assignment, the dataset has been processed and split into separate training, validation and testing sets. The training set consists of 18000 examples whereas the validation and the test sets consist of 6000 examples each. The dataset consists of binary, categorical and continuous attributes. The last entry in each row denotes the class label. You can read more about the attributes, in the README included with the processed dataset. You have to implement the decision tree algorithm for predicting whether a person defaulted or not based on various personal and economic 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 (continuous) 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 (categorical) 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) **(5 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) **(5 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) and (c) above?

(e) **(5 points)** The decision tree classifier in sciki-learn uses an optimised version of the CART algorithm for the tree construction. However, the scikit implementation does not support categorical variables. This makes the scikit decision tree treat the categorical variables as numerical (continuous) variables. One way to get around this issue is to convert the categorical variables into multiple binary variables using One-hot encoding. Transform the train, validation and the test sets by converting the categorical variables into binary variables as described above. Retrain a decision tree classifier on the transformed dataset and the parameter settings used in the previous part. Report training, validation and test set accuracies for this parameter setting. How do your numbers compare with those you obtained in part (b), (c) and (d) above?

(f) **(5 points)** Next, use the scikit-learn library to learn a random forest using the same transformation over the data, as described in the previous part. 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), (c), (d) and (e) above. Comment on your observations.

2. **(35 points) Neural Networks:** In this problem, you will work with the Poker Hand dataset available on the UCI repository. We will use the entire dataset for the purpose of this assignment. The training set contains 25010 examples whereas the test set contains 1000000 examples each. The dataset consists of 10 categorical attributes. The last entry in each row denotes the class label. You can read about the dataset and the attributes in detail from the link given above.

   (a) **(3 points)** The Poker Hand dataset described above has 10 categorical attributes. In the decision tree part, you have learned about one hot encoding as a way to convert categorical features to binary. Transform and save the given train and test sets using one hot encoding. We will use these new train and test sets for the subsequent parts. Note that the new dataset should have 85 features.

   (b) **(12 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:

   - the size of the batch for SGD
   - the number of inputs
   - a list of numbers where the size of the list denotes the number of hidden layers in the network and each number in the list denotes the number of units (perceptrons) in the corresponding hidden layer. Eg. a list [100 50] specifies two hidden layers; first one with 100 units and second one with 50 units.
   - the number of outputs i.e the number of classes

   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.

   (c) **(4 points)** In this part, we use the above implementation to experiment with a neural network having a **single** hidden layer. Vary the number of hidden layer units from the set {5, 10, 15, 20, 25}. Set the learning rate to 0.1. Choose a suitable stopping criterion and report it. Report and plot the accuracy on the training and the test sets, time taken to train the network. Plot the metric on the Y axis against the number of hidden layer units on the X axis. Additionally, report the confusion matrix for the test set, for each of the above parameter values. What do you observe? How do the above metrics and the confusion matrix change with the number of hidden layer units?

   (d) **(4 points)** In this part, we will experiment with a network having **two** hidden layers, each having the same number of neurons. Set the learning rate to 0.1 and vary the number of hidden layer units, as described in part (c). Report the metrics and the confusion matrix on the test set, as described in the previous part. How do the metrics and the confusion matrix change with the number of hidden layer units? What effect does increasing the number of hidden layers, keeping the number of hidden layer units same, have on the metrics and the confusion matrix?

   (e) **(6 points)** In both the previous parts, the value of the learning rate is fixed to 0.1, throughout the training. In this part, we will experiment with adaptive learning rate. We will fix the initial learning rate to 0.1. The learning rate is not changed as long as training loss keeps decreasing. Each time two consecutive epochs fail to decrease training loss by a fixed tolerance value, *tol*, the current learning rate is divided by 5. Repeat parts (c) and (d) by varying the learning rate as described, keeping $tol = 10^{-4}$. What affect does adaptive learning rate have on the metrics and the confusion matrix? Support your observations with reasons.

   (f) **(6 points)** In this part, we will use ReLU as the activation instead of the sigmoid function, only in the hidden layer(s). 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. Use the same definition of sub-gradient for the purpose of this assignment, as specified in the Minor 2 problem statement. Here is a resource to know more about sub-gradients. Repeat part (e) using ReLU as the activation function in the hidden layers and report the metrics and the confusion matrix and described previously. What effect does using ReLU have on each of the metrics as well as the confusion matrix? Support your observations with reasons.

(g) **Extra fun - No credits!** Observe that the poker hand dataset contains classes, which have a rare occurrence. For instance, there are only 4 instances of the Royal Flush in the train and test set combined. Data Augmentation is a popular technique which is widely used to make classifiers more robust. A popular use of data augmentation is in deep learning based image classification models, where the input images are flipped, rotated and then fed to the network. Upscaling or upsampling is a technique which is used handle class imbalance in a dataset. Here, we randomly choose a few examples from the rare class and feed them through the model multiple times. Now, observe that in the Poker Hand dataset, the order in which the cards are shown, does not determine the hand. Thus, any permutation of an input will also have the same hand as the input. In this part, you should experiment with upscaling the rare classes in the dataset by adding a suitable number of permutations of each input of the class, to the dataset. Check if this technique has any effect on the metrics and the confusion matrix described in the above parts.