COL 774: Assignment 3
Sem I, 2021-2022

Due Date: Monday, 25 Oct 2021, 11:50 pm. Total Points: 64

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.
- Include **a write-up (pdf) file**, one (consolidated) for each part, 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 (one for each of the parts A and B).
- You should use Python as your programming language.
- 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 Piazza 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):**
   For this part of the assignment you will work on Bank Marketing dataset. You can read about the dataset in detail from this link but work only on the data provided in the download link on the course page. You have been provided with a pre-defined set of test, train and validation of dataset to work with (available for download from the course website). In this dataset, the first row represents the labels of each column such that the last column represents whether the client subscribed (y/n) and following that, each row represents an example. You have to implement the decision tree algorithm for predicting whether client subscribed a term deposit. You will also experiment with Random Forests in the last part of this problem.

   (a) **(12 points) Decision Tree Construction** Construct a decision tree using the given data to predict which clients subscribed a term deposit. You should use mutual information as the criteria for selecting the attribute to split on. At each node, you should select the attribute which results in maximum decrease in the entropy of the class variable (i.e. has the highest mutual information with respect to the class variable). This problem has some attributes as numerical and some as categorical. For handling numerical 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. For example, if you have have 10 instances coming to a node, with values of an attribute being

(0,0,0,1,1,2,2,2,3,4) in the 10 instances, then we will split on value 1 of the attribute (median). At any step, choose the attribute which results in highest mutual information by splitting on its median value as described above. For categorical attributes, you can use the following two strategies (a) You can do a multi-way split by creating one decision tree branch for each possible value of the attribute. (b) You can do a two way split by first converting the attribute to a one-hot encoding, and then, treating each one-hot value as a separate Boolean valued attribute. Try both these variations in your implementation. 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) Decision Tree Post Pruning** 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. Read the following notes on pruning decision trees to avoid overfitting (also available from the course website). 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) **(10 points) Random Forests:** As discussed in class, Random Forests are extensions of decision trees, where we grow multiple decision trees in parallel on bootstrapped samples constructed from the original training data. A number of libraries are available for learning Random Forests over a given training data. In this particular question you will use the scikit-learn library of Python to grow a Random Forest. Click here to read the documentation and the details of various parameter options. Try growing different forests by playing around with various parameter values. Especially, you should experiment with the following parameter values (in the given range): (a) *n_estimators* (50 to 450 in range of 100). (b) *max_features* (0.1 to 1.0 in range of 0.2) (c) *min_samples_split* (2 to 10 in range of 2). You are free to try out non-default settings of other parameters too. Use the out-of-bag accuracy (as explained in the class) to tune to the optimal values for these parameters. You should perform a grid search over the space of parameters (read the description at the link provided for performing grid search). Report training, out-of-bag, validation and test set accuracies for the optimal set of parameters obtained. How do your numbers, i.e., train, validation and test set accuracies compare with those you obtained in part (b) above (obtained after pruning)?
One must note that if you encode categorical attributes to integers, scikit will assume they are ordinal. For example, if (Male, Female, Others) is mapped to (0, 1, 2) scikit will treat them as integers with 1 <2 <3. So you may need to encode in a way this ordering can be avoided. Scikit uses two split for handling categorical

(d) **(4 points) Random Forests - Parameter Sensitivity Analysis:** Once you obtain the optimal set of parameters for Random Forests (part (c) above), vary one of the parameters (in a range) while fixing others to their optimum. Plot the validation and test Repeat this for each of the parameters considered above. What do you observe? How sensitive is the model to the value of each parameter? Comment.

2. **(32 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 to learn a model for multi-class classification using one-hot encoding as described above. You will implement the backpropagation algorithm (from first principles) to train your network. You should use mini-batch Stochastic Gradient Descent (mini-batch SGD) algorithm to train your network. Use the Mean Squared Error

(MSE) over each mini-batch as your loss function. Given a total of m examples, and M samples in each batch, the loss corresponding to batch #b can be described as:

$$J^b(\ ) = \frac{1}{2M} \sum_{i=(b-1)M}^{bM} \sum_{l=1}^{r} (y_l^{(i)} - o_l^{(i)})^2 \tag{1}$$

Here each $y^{(i)}$ is represented using one-hot encoding as described above. You will use the sigmoid as activation function for the units in **output** layer as well as in the hidden layer (we will experiment with other activation units in one of the parts below). Your implementation (including back-propagation) MUST be from first principles and not using any pre-existing library in Python for the same. It should be generic enough to create an architecture based on the following input parameters:

- Mini-Batch Size (M)
- Number of features/attributes (n)
- Hidden layer architecture: List of numbers denoting the number of 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.
- Number of target classes (r)

Assume a fully connected architecture i.e., each unit in a hidden layer is connected to every unit in the next layer.

(c) **(4 points)** 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. Use a mini-batch size of 100 examples. This will remain constant for the remaining experiments in the parts below. 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 change with the number of hidden layer units? NOTE: For accuracy computation, the inferred class label is simply the label having the highest probability as output by the network.

(d) **(4 points)** Use an adaptive learning rate inversely proportional to number of epochs i.e. $_e = \frac{\eta_0}{\sqrt{e}}$ where $_0 = 0.1$ is the seed value and e is the current epoch number[1]. See if you need to change your stopping criteria. Report your stopping criterion. As before, plot the train/test set accuracy, as well as training time, for each of the number of hidden layers as used in 2c above using this new adaptive learning rate. Also, report the confusion matrix over the test set in each case. How do your results compare with those obtained in the part above? Does the adaptive learning rate make training any faster? Comment on your observations.

(e) **(4 points)** Several activation units other than sigmoid have been proposed in the literature such as tanh, and RelU to introduce non linearity into the network. ReLU is defined using the function: g(z) = max(0, z). In this part, we will replace the sigmoid activation units by the ReLU for all the units in the hidden layers of the network (the activation for units in the output layer will still be sigmoid to make sure the output is in the range (0, 1)). You can read about relative advantage of using the ReLU over several other activation units on this blog.

Change your code to work with the ReLU activation unit. Note that there is a small issue with ReLU that it non-differentiable at $z = 0$. This can be resolved by making the use of sub-gradients - intuitively, sub-gradient allows us to make use of any value between the left and right derivative at the point of non-differentiability to perform gradient descent (see this Wikipedia page for more details). Implement a network with 2 hidden layers with 100 units each. Experiment with both ReLU and sigmoid activation units as described above. Use the adaptive learning rate as described in part 2d above. Report your training and test set accuracies in each case. Also, the report the confusion matrix over the test set. Make a relative comparison of test set accuracies (and confusion matrix) obtained using the two kinds of units. What do you observe? Which ones performs better? Also, how do these results compare with results in part 2c using a single hidden layer with sigmoid. Comment on your observations.

[1] One epoch corresponds to one complete pass through the data

(f) **(5 points)** Use MLPClassifier from scikit-learn library to implement a neural network with the same architecture as in Part 2e above, and same kind of activation functions (ReLU for hidden layers, sigmoid for output). Use Stochastic Gradient Descent as the solver. Note that MLPClassifier only allows for Cross Entropy Loss (log-loss function) over the final network output. Read about the binary cross entropy loss . Compare the performance with the results of Part 2e. How does training using existing library (and modified loss function) compare with your results in Part 2e above.

(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 train/test set accuracy, as well as the confusion matrix over the test set, obtained in the above parts.