COL 774: Assignment 3
Semester I, 2022-2023

Due Date: Wednesday October 26 (2022), 11:50 pm. Total Points: 47+ ____

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, an additional penalty equivalent of the weight of the assignment and possibly much stricter penalties (including a **fail grade** and/or a **DISCO**).

1. **(47 points) Decision Trees, Random Forests and Gradient Boosted Trees:**
   In this problem, we will work with two different datasets.

   Dataset 1:
   Machine learning has been deployed in various domains of computer science as well as other engineering fields. In this problem you will work on predicting the severity (benign or malignant) of a mammographic mass lesion. Mammography is the most effective method for breast cancer screening available today. However, each mammogram interpretation may require significant time from a trained medical personnel, and that is where Machine Learning can be of help. Our objective in this problem is to help physicians in deciding whether to perform a breast biopsy on a suspicious lesion seen in a mammogram or to perform a short-term follow-up examination instead. We will work on a mammography dataset available from the UCI repository, and you can read more about this dataset from this link. We will use a processed version of this dataset for this assignment available here. The dataset contains 5 features: (a) BI-RADS assessment (b) Age (c) Shape (d) Margin (e) Density. The value of the first is based on a human assessment of the mammographic mass for additional analysis, and therefore you **should not** use this in learning your models below. Using the four features: Age, Shape, Margin and Density, you need to grow a decision tree/random forest/gradient boosted trees to predict the severity of a mammographic mass lesion. Note that the target variable as the last (sixth) column in the csv files provided to you. We have split the dataset into train, validation, and test set.

(a) **(2 points) Decision Tree Construction and Visualization** There are several missing values in the dataset splits. One way to handle missing values is to ignore those examples. Drop the samples with any missing attribute and train a decision tree on the training split and report training accuracy, validation accuracy, and test accuracy. Visualize the decision tree. You should use the default parameter values while defining the tree in this part.

(b) **(4 points) Decision Tree Grid Search** Next, perform a grid search over the space of parameters including `max_depth`, `min_samples_split` and `min_samples_leaf`. You can also try to vary any other parameters that you may find relevant. Report training, validation, and test set accuracies for the optimal set of parameters obtained. Visualize the optimal decision tree. Comment on your observations. Specifically, you should compare the tree with that obtained in the part (a) above.

(c) **(3 points) Decision Tree Post Pruning (Cost Complexity Pruning)** In the previous part, we have seen that there are various parameters to prevent Decision tree classifiers from overfitting. Cost complexity pruning provides another option to control the size of a tree, and is a variation of the idea of reduced error pruning discussed in the class. Minimal cost complexity pruning recursively finds the node with the "weakest link". The weakest link is characterized by an effective alpha, and the nodes with the smallest effective alpha are pruned first. In this part we study the effect of `ccp_alpha`(a parameter in the scikit-learn implementation of decision tree) on regularizing the trees based on its accuracy on the validation set. To get an idea of what values of `ccp_alpha` could be appropriate, scikit-learn provides `DecisionTreeClassifier.cost_complexity_pruning_path` that returns the effective alphas and the corresponding total leaf impurities at each step of the pruning process. Use the training split and plot total impurity of leaves vs effective alphas of pruned tree. Plot the number of nodes vs alpha and the depth of tree vs alpha. Plot training accuracy, validation accuracy, and test accuracy vs alpha. What are your observations? Use the validation split to determine the best performing tree and report the train, validation, and test accuracy with respect to the best tree. Visualize the best pruned tree. Comment on your observations. Specifically, you should compare the tree with that obtained in the part (a) and (b) above.

(d) **(4 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 : (a) `n_estimators` (b) `max_features` (c) `min_samples_split`. 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.

(e) **(4 points) Missing Data Imputation** Another way to handle missing value in a dataset is called imputation, the process of replacing missing data with substituted values. The simplest way is to replace missing values of missing attributes in an example with some aggregate function of the attribute's value in other examples where it is present. This is done as a pre-processing step before the tree (forest) is grown. Two simplest aggregate functions to try are median and mode. Try these two aggregate functions for imputation (a) - (d). Note that you may need to perfrom the grid search again to obtain the best set of parameters. Compare the results with those obtained by simply ignoring any examples with missing values. What do you see? Comment on your observations.

(f) **(4 points) Gradient Boosted Trees:** XGBoost (Extreme Gradient Boosting) is functional gradient boosting based approach where an ensemble of "weak learners" (decision trees in our case) is used with the goal to construct a model with less bias, and better predictive performance as discussed in the class. You can read about the XGBoost implementation here. Implement an XGBoost classifier and experimenting with different parameter values (in the given range): (a) `n_estimators` (10 to 50 in range of 10) (b) `subsample` (0.1 to 0.6 in range of 0.1) (c) `max_depth`(4 to 10 in range of 1). You should perform a grid search over the space of parameters (read the description at the link provided for performing grid search). Report training, validation and test set accuracies for the optimal set of parameters obtained. Note that XGBoost by itself takes care of missing values, so you do not have to worry about ignoring any examples with missing values, or performing imputation as a pre-processing step.

Dataset 2: In this problem, we will work another dataset related to prediction of rating associated medical drugs, where the attributes are primarily textual. Any dataset with text attributes requires conversion of text to numerical attributes for computations to be performed. One such dataset is provided here. In this dataset there are two text attibutes, namely **conditon** and **review**. The review attribute contains reviews about drugs collected from patients with specified conditions. We are going to use these two text attributes to predict the rating of a drug. In order to make the specified text attributes as numerical attributes, two well known techniques are CountVectorizer and TF-IDFVectorizer. Use these techniques to convert text attibutes to numerical attributes. In the provided dataset, the text attributes (particularly reviews) have a lot of stopwords (considered insignificant like 'a', 'an', 'the'). Remove these using information provided here. Also remove unnecessary characters (commas, full stop) and backslash character constants (carriage-return, linefeed). In case a review is missing, consider the review to be an empty string. The date attribute must be split into three numerical attributes (day, month and year).

(a) **(2 points) Decision Tree Construction:** Repeat part(a) above for the Dataset 1. No need to visualise in this case as the number of attributes are really high.

(b) **(4 points) Decision Tree Grid Search:** Repeat part(b) above for the Dataset 1. No need to visualise in this case as the number of attributes are really high.

(c) **(3 points) Decision Tree Post Pruning:** Repeat part(c) above for the Dataset 1. No need to visualise in this case as the number of attributes are really high.

(d) **(4 points) Random Forests:** Repeat part(d) above for the Dataset 1. You should experiment with the following parameter values : (a) `n_estimators` (50 to 450 in range of 50) (b) `max_features`(0.4 to 0.8 in range of 0.1) (c) `min_samples_split (2 to 10 in range of 2)`.

(e) **(4 points) Gradient Boosted Trees (XGBoost):** Repeat part(f) above for the Dataset 1. You should experiment with the following parameter values : (a) `n_estimators` (50 to 450 in range of 50) (b) `subsample` (0.4 to 0.8 in range of 0.1) (c) `max_depth`(40 to 70 in range of 10).

(f) **(4 points) GBM (Gradient Boosted Machines):** LightGBM is a gradient boosting framework that implements another boosting framework, Gradient Boosted Machines. One of the primary advantages of GBMs is its scalability to large datasets. Carefully read the following paper. Try various parameter settings for LightGBM on the above dataset, and see if you can find a setting which improves the performance compared to the earlier trained models. Compare the running times for standard decision tree learner (part (b)), decision tree with pruning (part (c)), random forest (part (c)), XGBoost (part(d)) and LightGBM (part (f)).

(g) **(5 points) Training with Varying amount of data** Randomly sample n number of examples from the training data, with $n \in \{20k, 40k, 60k, 80k, 100k, 120k, 140k, 160k\}$, and repeat the parts (a) - (g) above for each value of n. Note that validation and test sets remain the same. Plot the test set accuracy on y-axis and value of n on x-axis, with one curve for each of the algorithms tried above. Plot a similar curve for the amount of time taken for training on y-axis and n on x-axis. Comment on your observations.

2. **Neural Networks :** Coming Soon