COL341 Spring 2023
Assignment 3: Decision Tree and Random Forest
(To be done Individually)

Due Date: 14th April 2023, Friday, 11:55 PM (No extensions)
Total Points: 50

1 Introduction

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes, and leaf nodes. Decision tree learning employs a divide-and-conquer strategy by conducting a greedy search to identify the optimal split points within a tree. This process of splitting is then repeated in a top-down, recursive manner until all, or the majority of records have been classified under specific class labels.
In this assignment, you will implement the decision tree algorithm on a given data and analyse the results.

2 Dataset

Given dataset is an image dataset containing 2400 samples from 4 distinct classes. The classes include airplane, car, dog, and person-face. Image size is 32x32x3. You need to implement various tree-based models as instructed below for the classification task. We have split the data into 2000 train and 400 validation images into two separate folders as train and validation
All images are provided at this link with separate folders for each class in train and validation folders. There is a seperate folder of test sample images in test_sample folder which can be used during evaluation.

2.1 Binary Classification Data

For Binary classification task, you will be implementing **Face Classification** task. Extract all images and label all face images as **1** and rest as **0**. Train your model on images in train folder and use validation folder images for validation. Perform all tasks as mentioned in **section 3.1** and report the results.

2.2 Multi-Class Classification Data

For the multi-class classification problem, you need to classify images into all 4 classes. Label the classes as given : `Cars - 0, Faces - 1, Airplanes - 2, Dogs - 3`. Train your model on images in train folder and use validation folder images for validation. Perform all tasks as mentioned in **section 3.2** and report the results.

3 Task:

3.1 Binary Classification

a) **Decision Tree from scratch (7 marks)** Implement the decision tree model for binary classification. Use **Gini Index** and **Information Gain** as splitting criteria in two separate implementations. You can read about Gini Index from here.

Specify an appropriate halting mechanism (stopping criteria) to stop the growth of the tree. For this, we would like you to experiment with the following hyperparameters:

(a) `max_depth`: which will control the depth of the tree. (Set it as 10)

(b) `min_samples_split`: which will control the minimum number of samples present at a node for the best split to take place. (Set it as 7)

You must report your model's Accuracy, Precision, and Recall value for `train` and `validation` data, also report the total training time for both cases.

b) **Decision Tree sklearn (2 marks)** In this part implement Decision Tree model using scikit learn library with hyperparameters `max_depth` and `min_samples_split` set with the value as given in the first part. Report model's Accuracy, Precision and Recall value for `train` and `validation` data and also total time taken to train the model and compare them with the model implemented by you in part(a).

c) **Decision Tree Grid-Search and Visualisation (3 marks)** Next, select top-10 features from the data and build a Decision Tree over those features. Use sklearn `feature_selection` class to select top-10 features and then visualize the tree graphically. Next Perform a grid search on these top-10 features `(use parameter cv=5)` over the space mapped by following parameters:

```
criterion: {'gini', 'entropy'},
max_depth: {None, 5, 7, 10, 15},
min_samples_split: {2, 4, 7, 9}
```

You can also try to vary any other parameters that you may find relevant. Report training and validation accuracies for the optimal set of parameters obtained. Comment on your observations. Specifically, compare the tree with those obtained in part**(a)** and part**(b)** above.

d) **Decision Tree Post Pruning with Cost Complexity Pruning (3 marks)** 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. Minimal cost complexity pruning recursively finds the node with the "weakest link". An effective alpha characterizes the weakest link, 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 their accuracy on the validation set. To understand 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 the total impurity of leaves vs the effective alphas of pruned tree. Plot the number of nodes vs alpha and the depth of the 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 training and validation accuracy for the best tree. Visualize the best-pruned tree.

e) **Random Forest (3 marks)** As we know, Random Forest is an extension of the Decision Tree Algorithm where multiple decision trees are grown in parallel with bootstrapped data. In this part, we will implement random forest using scikit-learn library. You can read here the documentation of random forest and various hyper-parameters used in it. First, use default hyper-parameter values to report the **training** and **validation** accuracy, precision, and recall values. Then perform Grid Search `(with parameter cv=5)` over given hyper-parameters list:

```
[ n_estimators: {80,100,150,200},
  criterion: {'gini', 'entropy'},
  max_depth: {None,5,7,10},
  min_samples_split: {5,7,10} ]
```

and report the **training** and **validation** accuracy, precision, and recall values respectively obtained with the best parameter values. Report the best set of parameters.

f) **Gradient Boosted Trees and XGBoost(3 marks)** In this task you need to execute Gradient Boosted Trees and then its extreme version XGBoost. You can read about them from here. 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. You can read about the XGBoost implementation here. Implement an Gradient Boosted Classifier using `sklearn.ensemble.GradientBoostingClassifier`, then XGBoost classifier and experiment with different parameter values (in the given range): (a) n estimators (20 to 50 in range of 10) (b) subsample (0.2 to 0.6 in range of 0.1) (c) max depth(5 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) and report the **train** and **validation** accuracies, precision, recall values obtained with the best parameter values and time taken to train both models. Report the best set of parameters.

g) **Confusion Matrix (1 mark)** Generate confusion matrix in all the parts above.

h) **Competitive Part (5 marks)** This section will be graded on the rank in the leader-board on the hidden test set. The top-k submissions will receive full marks, and the rest will receive diminishing marks. The exact scheme will be decided later.

3.2 Multi-Class Classification

(a) **Decision Tree sklearn (1 marks)** Repeat part(b) of section 3.1 for this multi-class classification task and report **train** and **validation** accuracies and time taken to train the model.

(b) **Decision Tree Grid Search and visualisation (2 marks)** Repeat part(c) of 3.1 for this task, report **train** and **validation** accuracies and time taken to train the model for the best set of parameters obtained. Compare the result with part(a) above.

(c) **Decision Tree Post Pruning with Cost Complexity Pruning (3 marks)** Repeat part(d) of section 3.1 for this dataset. Use the validation split to determine the best-performing tree and report the **train** and **validation** accuracy for the best tree.

(d) **Random Forest (3 marks)** Repeat part(e) of section 3.1 for this task. Report the **train** and **validation** accuracies obtained with the best parameter values. Report the best set of parameters.

(e) **Gradient Boosted Trees and XGBoost (3 marks)** Repeat part(f) of section 3.1 for this task. Report the **train** and **validation** accuracies obtained with the best parameter values. Report the best set of parameters.

(f) **Confusion Matrix (3 marks)** Plot confusion matrix in all the parts above.

(g) **Real-time Application (3 marks)** Test your model with the image of your own face with different view angles and partially visible face. Report your accuracy with 10 such images.

(h) **Competitive Part (5 marks)** This section will be graded on the rank in the leaderboard on the hidden test set. The top-k submissions will receive full marks and the rest will receive diminishing marks. The exact scheme will be decided later.

4 Evaluation

The efficacy of your solution will be judged by running your implemented algorithm. You need to create `main_binary.py` and `main_multi.py` files to train your model and generate the labels for binary and multi-class classification respectively. Both main files should take 3 paths as argument : 1 - train data path, 2 - test data path, 3 - output path. Your code should generate predicted output lables for test data `(sample test data is provided in test_sample folder)` and save them into a csv file with image name `(image_id)` as 1st column and predicted lables `(labels)` as 2nd column in the output path **(do not include column headers in csv files)**. Generate a separate csv file for all required parts with name as `test_section.csv`, for example : for 3.1.a part generate a csv file with name `test_31a.csv`. The usage of the `main_binary and main_multi.py` should be the following to generate the output on `test` data:

```
python main_binary.py or main_multi.py
        --train_path=<path to train folder> \
        --test_path=<path to test folder> \
        --out_path=<path to store csv file> \
```

The output format is the following for test data:

```
<image id 1>, <output label 1>
    .
    .
    .
<image id N>, <output label N>
```

5 Submission

You are required to submit your complete source code (including the completed `main_binary.py and main_multi.py`) and a report containing the following information:

a) Detailed analysis regarding the performance of your model on `train` and `validation` data for the different experimental setups. Include all numbers and plots.

b) Your observation on the Visualization part.

Submit a `README.txt` for your code, if necessary with all instructions for generating the required output. You must zip the code file, and any other files that may have been created to be used by both main code files and report in a single zip file, rename the zip file as `<Your-Entry-Number>.zip` (e.g., `2019CSZ8406.zip`), and submit in Moodle.

6 Constraints

a) You need to do this assignment individually.

b) You need to use `Python 3.6+` for implementation. You are allowed to use basic standard libraries like `math`, `numpy`, `pandas, matplotlib, seaborn` etc. You can use `cv2` or `skimages` library for images processing task.You are allowed to use `sklearn` except for 3.1.a part. No other third-party library is allowed. Please clarify any further doubts using piazza.

c) Any plagiarism will attract penalties as described in the course policy.

d) There are no late submissions or extensions allowed. Medical/ emergency cases will be dealt with on a case-to-case basis.