ENSEMBLE CLASSIFIER

Authors : Anurag Chandra, Parijat Kawale
Instructor: Prof. Weijhe Zhao

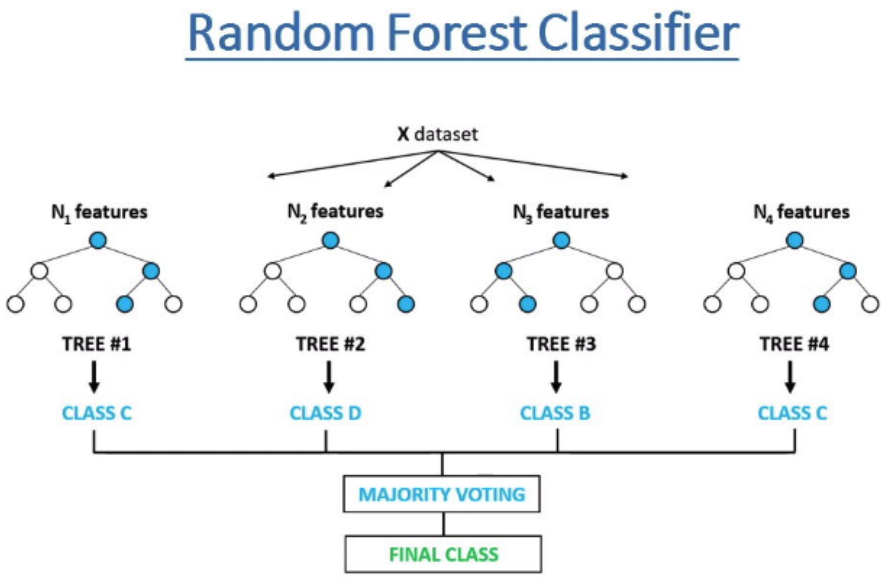
1. INTRODUCTION

As part of the course curriculum for the course *Topics in Intelligent Systems*, we chose the building an ensemble classifier that would be constructed based on our understanding of the Random Forest Algorithm. The motivation behind choosing this topic is to learn and understand the complete process of classification using Random Forest Algorithm.

The following documentation will go through the basic idea of Random Forest Algorithm followed by the features provided through this library. We trained and tested this model on Iris dataset and MNIST dataset. The baselines would be shared in further parts of the report.

2. What is Random Forest?

Random Forest algorithm is a supervised algorithm. It is one of the most flexible and easy to use algorithm. It creates decision trees on the given data samples, gets prediction from each tree and selects the best solution by means of voting. It is also a pretty good indicator of feature importance. Random forest algorithm combines multiple decision-trees, resulting in a forest of trees, hence the name Random Forest. In the random forest classifier, the higher the number of trees in the forest results in higher accuracy.



Steps of execution:

- In the first stage, we randomly select “k” features out of total m features and build the random forest. In the first stage, we proceed as follows:-

    1. Randomly select k features from a total of m features where k < m.
    2. The data rows in the dataset are randomly shuffled and data is picked by replacing it back into the dataset.
    3. Among the k features, calculate the node d using the best split point.
    4. Split the node into daughter nodes using the best split.
    5. Repeat 1 to 3 steps until the number of nodes has been reached.
    6. Build forest by repeating steps 1 to 4 for n number of times to create n number of trees.

- In the second stage, we make predictions using the trained random forest algorithm.
    1. We take the test features and use the rules of each randomly created decision tree to predict the outcome and store the predicted outcome.
    2. Then, we calculate the votes for each predicted target.
    3. Finally, we consider the maximum voted predicted target as the final prediction from the random forest algorithm.

3. Implementation details:

There are two types of decision trees implemented:

1. Using Entropy
2. Using Gini Index

Best split point selection:

- At each stage of tree building, we run a loop on all the features
- For each feature, the remaining dataset at that level is sorted based on the feature. Average of each adjacent data in the sorted feature is used to split the data into two halves
- Information gain is calculated using either Entropy or Gini and the feature (along with the average of adjacent data) is selected which results in maximum information gain.

4. Features provided in current implementation:

- Platforms supported: MacOS, Linux (Ubuntu), Windows
- Languages supported: Python, C++, Java
- Multi-threading support in Python and C++
- Illegal Input Handling

5. **Details of execution and User Input handling**:

- Supported parameters in Python:
    1. -h, --help show this help message and exit
    2. -n N number of trees
    3. -s S minimum samples to split
    4. -d D maximum depth of the tree
    5. -f F maximum number of features to use in a tree
    6. -i I information gain type gini/entropy
- Decision Tree support parameters:

    - min_samples_split:
    - max_depth:
- Parameters in C++:
    1. [-n <num_trees>]: Number of Trees to use in the forest
    2. [-s <min_samples_split>]: Minimum number of samples required to perform a split.
    3. [-d <max_depth>]: Maximum depth of a tree.
    4. [-f <max_features>]: Maximum number of features to use in bootstrapping.
    5. [-i <input-type: iris/mnist>]: Input data to use.
    6. [-t <tree-type: gini/entropy>]: Decision tree to use in forest, using gini index or entropy to find the best split.
    7. [-h this help message]: Prints a user help message.

  Note: All the above mentioned parameters are optional, i.e., if not specified, they will be taken default values as:
    1. num_trees: 100
    2. min_samples_split: 3
    3. max_depth: 10
    4. max_features: 0
- In case of **illegal inputs** such as negative hyperparameters, or floating point values, the programs in both the languages will capture such inputs and exit with an error message.
- Further, if the **input is positive but a value which is not legal for the dataset**, for example, the min_samples_split is greater than the dataset, or max_features is greater than the total number of features in the dataset, the program will catch such inputs and exit with printing an error.

6. **Prerequisites**:
    - For Python-based implementation:
        - Sklearn
        - pandas
    - For C++ - based implementation:
        - OpenMP
7. **How to run the model?**
    - PYTHON:
        - Download the “pk7145_ac5068_csci_739_project.zip”
        - Unzip the folder and inside the folder run, run **python3 randomForestClassifier.py**
    - C++:
        - Download the “pk7145_ac5068_csci_739_project.zip”
        - Unzip the folder and run following commands:
            - For **LINUX** users,
                - **`cd src &&`**

```
g++ RandomForest.cc DecisionTree.cc
iris_read.h InputRead.h -o rf -fopenmp &&
./rf
```

- For **Mac** Users,
  - **`cd src && clang++ RandomForest.cc DecisionTree.cc -o rf -fopenmp && ./rf`**
- **Windows machines:**
  If the windows machine has g++ installed, we can compile it using following command:
  `g++ RandomForest.cc DecisionTree.cc iris_read.h InputRead.h -o rf`
  Then it can be run using the command: **`.\rf.exe`**
  NOTE: Openmp is not supported natively on Windows, so we didn't use the `-fopenmp` flag in this case.

```
PS D:\Master's\src> g++ RandomForest.cc DecisionTree.cc iris_read.h InputRead.h -o rf
PS D:\Master's\src>
PS D:\Master's\src> .\rf.exe
Loading data...
file opened successfully
Starting with training...
Training Ended


Accuracy is: 54.6667%
PS D:\Master's\src>
```

- **Java language support:**
  The Python script is being called from Java using the **ProcessBuilder** and **Process** class:

```
String command = "python3 randomForestClassifier.py -n 100 -s 3
-d 10";
ProcessBuilder pb = new ProcessBuilder(command.split(" "));
Process p = pb.start();
```

```
→  CSCI739-EnsembleClassifier git:(main) ✗ javac DecisionTreeCaller.java
→  CSCI739-EnsembleClassifier git:(main) ✗ java DecisionTreeCaller
Hello Random Forester!
Accuracy: 0.95
Process exited with code 0
→  CSCI739-EnsembleClassifier git:(main) ✗ java -version
openjdk version "1.8.0_382"
OpenJDK Runtime Environment (build 1.8.0_382-8u382-ga-1~22.04.1-b05)
OpenJDK 64-Bit Server VM (build 25.382-b05, mixed mode)
→  CSCI739-EnsembleClassifier git:(main) ✗ cat DecisionTreeCaller.java
```

8. **Benchmarking with baselines using sample datasets:**

DATASETS	BASE ACCURACY	MODEL ACCURACY
IRIS	94% Time of execution: 2s	65%-93% Time of execution: 30s
TITANIC	85% Time of execution: 5s	62%-82% Time of execution: 38s
MNIST	94% Time of execution: 2-5min	52% - 70% Time of execution: 10min - 40min

NOTE: The benchmarking of the data is based on different sizes of the data ranging from 1000 rows to 10000 rows. The accuracy and time of execution is based on the same subset of the data due to the entire dataset being too big for testing purposes.