Decision Tree on Time Series Data

Shubham Mishra
18CS10066
Somnath Jena
18CS30047

***Abstract*—Here we try to make a decision tree classifier for a time series dataset.**

We have taken the dataset from UCI Machine Learning Repository.

***Index Terms*—Decision Tree**

I. Introduction

This report is based on building a decision tree from scratch without using explicit Python libraries like sklearn for building decision tree. This is part of our Assignment in Machine Learning course taught by Professor Jayanta Mukhopadhyay at IIT Kharagpur. The dataset used in this assignment is the **EEG Steady-State Visual Evoked Potential Signals Data Set**[1]

II. Background Information

A. Decision Tree

Decision Tree is one of the simplest yet expressive machine learning models.

It classifies the data using a tree where each node directs the classification of the data at hand in a certain direction. The leaf nodes represent final verdict regarding labels, and each non-leaf splits the data points on certain attributes/features. A decision tree tries to emulate how a human being deductively reasons about classifying certain scenarios. Programmatically, a decision tree learns a few data specific *if-else* rules that segregates different labels well. A decision tree inherently represents a classification algorithm, which is what was required in our assignment. But it can also perform regression.

Essentially, a decision tree divides the population space into small regions, and then learns simple linear functions in those regions. The hypothesis space of Decision Trees is more expressive than that of Concept Learning.

B. ID3 Algorithm

ID3 (Iterative Dichotomiser 3) [2] is an algorithm invented by Ross Quinlan used to generate a decision tree from a dataset.

On a training dataset, it first calculates the entropy of the different labels, given by the formula:

$S = -\sum_i p_i \cdot log(p_i)$ where p_i denotes the probability of occurrence of the i^{th} label. Entropy calculates the amount of randomness in the data.

Then ID3 splits the training examples into multiple sections according to the value of a specific attribute, and then calculates the total entropy of the split, weighted by the number of examples in each of them. Naturally, a good split will reduce the randomness in the data. The difference between the entropy before and after the split is called **Information Gain**.

The algorithm selects the attribute with the best information gain, then splits the dataset on that attribute into several child nodes. Then it keeps on computing such splits recursively until the information gain drops below a threshold.

For continuous attributes, ID3 calculates the best place where it can do a *less than* - *greater than* split.

C. Reduced Error Pruning

Decision trees are very prone to overfitting. To reduce that, we remove some of the subtrees and replace by the majority decision at that subtree's root node. Finding the nodes that should be pruned optimally is a np-hard problem. Hence we take a greedy approach called reduced error pruning.

Steps:

1. The training data is partitioned into "grow" and "validation" set in 80:20 ratio
2. Complete tree is grown using the "grow" set.
3. Until validation accuracy increases, we do bottom-up pruning.

Bottom-up Pruning Algorithm:

1. For each non-leaf node recursively call prune function
2. Once we reach leaf-node, return to to prune call in its parent node
3. Temporarily prune the node, assigning its decision as the majority vote
   - If the validation accuracy increases, permanently cut/prune the node
   - Then go upwards in the tree (returns to parent's prune call)

We can also statistical tests like *Student's t-Test* to decide which nodes to prune.

D. Dataset Information

The dataset was collected from UCI Machine Learning Repository. It records EEG signals of different electrodes during several visual tasks carried out by the subjects.

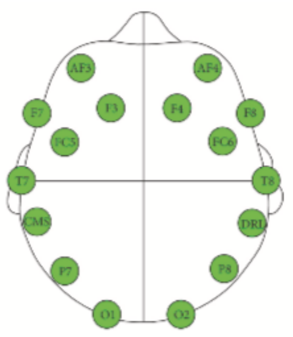


FIGURE 4: International 10-20 system for Emotiv Epoc+.

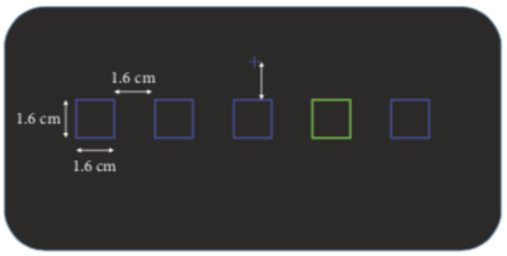


FIGURE 5: Five box task configuration.

Figure 1. Placement of the electrodes and task configuration

There are 5 different types of tests performed on 30 subjects

1. SB1- Five Box Visual Test 1
2. SB2- Five Box Visual Test 2
3. SB3- Five Box Visual Test 3
4. SV1- Visual Image Search
5. SM1- Motor Images (Handshake experiment)

There are a total 142 tests performed on these 30 subjects. Each subject undergoes different tests which are provided in .csv format as follows: suppose you have a .csv whose name is `A001SB1_1`. This means the data corresponds to group A (only Group A is provided at present), subject 001, Test `SB1` (Five Box Visual Test), and first experiment.

E. Attribute Information

The time series .csv files contain 16 attributes, of which last 14 are the signals coming from the electrodes. We use these 14 signals as attributes of our decision tree. The attributes are namely- **AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4**. Also note that the attributes are continuous-valued.

III. Methodology

A. Data Preprocessing

Since the data is in the form of time series, we can't apply a Decision Tree directly. First, we need some aggregating metric representative of the time series, so that we can train a decision tree on it.

We take two approaches here. These are called `modes` in the code.

1. Variance of the signals from each electrode.
   This has been intuitively used as a good heuristic measure for filtering the dataset.
2. Rolling window mean of the signals from each electrode
   Rolling window mean reduces fluctuations in the time series, but preserves essential characteristics. After many cycles of searching for a feasible window size, we took a **window size of 10 and took the first 10 means of each time series**.

B. Learning Task

The learning task for our decision tree is to predict the type of test from given 14 dimensional data of the electrodes signals.

C. Finding the best split

Our dataset has continuous attributes only. The implementation of other parts of ID3 and Pruning were as described in the *Background Information* section. However, the algorithm for finding the best split for a continuous attribute requires special attention in our context. The algorithm used is given below.[3][4]

Iterative Square Root Decomposition:

1. Find `min` and `max` values of the attributes from the given examples.
2. Divide the `[min, max]` interval into $\lfloor\sqrt{max - min}\rfloor$ sized bins.
3. Traverse the through the bins, calculating the entropy of splitting at the lower bound of each bin.
4. Find the minimum entropy of such splitting. Then that value occur at x.
   - If a threshold minimum bin size is reached or number of iterations has exceeded a threshold value, stop and return x as the best splitting point.
   - Else set `min = x - bin_size` and `max = x + bin_size` and go to step 2.

This algorithm gives the best possible splitting point. But due to multiple iterations it is quite slow.

D. Train Test Splitting

For the purpose of training, pruning and testing, we have divided the dataset into `train` and `test` set. We further split the `train` set into `grow` and `validation` set.

The ratio of sizes of `grow`, `validation` and `test` set is $64 : 16 : 20$.

While training a full height tree, we have used the full `train` set. However, for height bound tree growth and subsequent pruning, we have used the `grow` set from training and `validation` set for pruning. In all cases, final testing is done on the `test` set.

IV. Results

A. Processed Dataset Sizes

Aggregation led to considerable reduction in dataset size. The reduced dataset sizes are as follows:

Mode	Total Number of Rows
Variance	142
Rolling mean	1562

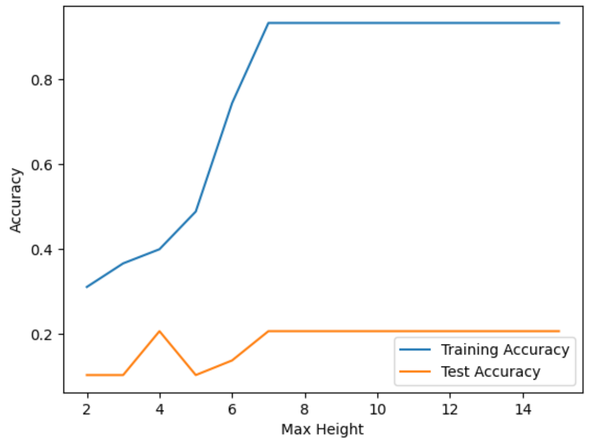
B. Accuracy of Full Grown Trees

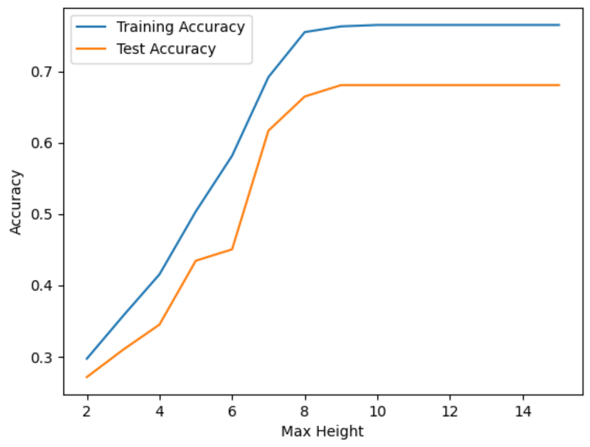
Trees are grown to their full depth over 10 fold random test train splits. Variance mode shows terrible overfitting. The results for Rolling mean mode indicates that the tree can generalise well to the test set.

Mode	Train Accuracy Max	Train Accuracy Average	Test Accuracy Max	Test Accuracy Average
Variance	100%	91%	24%	12%
Rolling mean	78%	76%	72%	68%

C. Accuracy vs Height

We took one `grow`, `validation` and `test` split and trained the trees with maximum heights ranging from 2 to 16. The accuracy vs height curves of variance and rolling mean modes are respectively as follows:





The best heights are as follows:

Mode	Best Height
Variance	8
Rolling mean	10

D. Accuracy after Pruning

The best height trees are then taken and pruned. In this dataset, pruning *does not* bring about any drastic change in the accuracy.

Mode	Test Accuracy after Pruning
Variance	17%
Rolling mean	67%

E. Structure of the Trees

The trees have been stored as pdf images and graphviz files in the `outputs` folder of the code. To reduce cluttering we are not including it here.

F. Running time

On an Intel 10th Generation Core i5 processor with 16GB RAM, running both modes simultaneously took **1 hour** to complete all the analysis.

V. Conclusion and Future goals

Through this exercise we learnt about the details of Decision Tree Implementation and Application. Future goals that can be achieved from here are: - Improving upon the run-time by using more parallelized algorithms. - Using `gini` index in place of `entropy` for splitting. - Using better algorithms like `C4.5` in place of ID3.

VI. Appendix

A. Code Structure

```
|- dtree: Contains the core decision tree package
|    |- __init__.py: Exports the
                    decision tree APIs.
|    |- tree.py: Defines the basic
```

```
                 Decision Tree Data Structure.
|    |- learner.py: Defines the ID3 algorithm.
|    |- pruning.py: Defines the
                 Reduced Error Pruning algorithm.
|    |- api.py: Defines an api similar to Sklearn
                for using the tree.
|- analysis: Contains code for data analysis
|    |- <mode>: Two folders, one for each mode,
                has the same structure.
|    |    |-analysis.py: Full analysis to generate
                      all the result for the mode.
|    |    |-data_processor.py: Creates the
                                aggregated dataset
                                for the mode.
|    |    |- benchmark.py: Runs the same analysis
                            but using Sklearn
                            on mode's dataset.
|- tests: Contains some scripts
          for testing the dtree package.
|- run_tests.py: Driver script to run all tests.
|- run_analysis.py: Driver script to run all analysis
                    and store the outputs.
|- run_data_processor.py: Driver script to
                          run data aggregation.
|- run_benchmark.py: Driver script to
                     run benchmarks.
```

B. *Output Structure*

```
|- outputs
|    |- <mode>
|    |    |- accuracy.png: Accuracy vs Height graph.
|    |    |- data.csv: Data corresponding to mode.
|    |    |- Final_tree.gv.pdf: Tree after pruning.
|    |    |- Max_height.gv.pdf: Tree with max height.
```

References

[1] Marco Aceves, Ph.D., Autonomous University of Queretaro, “EEG steady-state visual evoked potential signals data set.” https://archive.ics.uci.edu/ml/datasets/EEG+Steady-State+Visual+Evoked+Potential+Signals.

[2] Wikipedia contributors, “ID3 algorithm — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=ID3_algorithm&oldid=970826747, 2020.

[3] “How to discretise continuous attributes while implementing the id3 algorithm?” https://stats.stackexchange.com/questions/72720/how-to-discretise-continuous-attributes-while-implementing-the-id3-algorithm, 2020.

[4] “Decision tree with continuous variables.” https://discuss.analyticsvidhya.com/t/decision-tree-with-continuous-variables/201, 2020.