Algorithms and datastructures
Decision trees

May 8 + May 15, 2020

1 Introduction

Decision tree learning is one of the predictive modeling approaches used in statistics, data mining and machine learning. It uses a decision tree to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves)[1].

In this assignment, we will build a decision tree to infer what animal the user was thinking of by asking a sequence of yes/no questions. The decision tree can then be modeled by a binary tree. You are given some text files containing a list of questions and animals together with all the answers to the questions for each animal. Your goal is to convert this knowledge into a decision tree like the one shown in figure 1.

We will start with a naive implementation and improve this step by step to make it more efficient.

[1] https://en.wikipedia.org/wiki/Decision_tree_learning

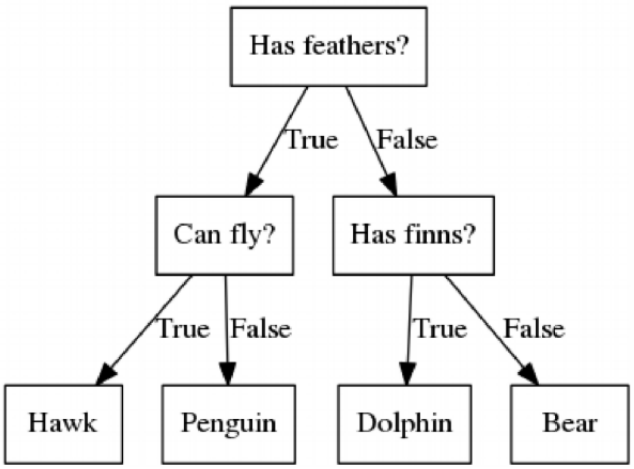

Figure 1: An example decision tree. (https://mc.ai/machine-learning-algorithms-decision-trees)

2 Build the decision tree

The start code contains a *main.cpp* file with the required code to read a text file. Your task is to complete the *BinaryTree.h* file. You are free to restructure the code any way you see fit and to add methods. All the tasks can be implemented in an elegant recursive way.

- Complete the *build* method. This method adds all the questions one by one in the same order they are given in the text file until you end up with only one possible animal.
- Implement the *operator«* method. This prints the entire tree. An example output is given below.
- You can test this for the *animals_small.txt* and *animals_medium.txt* files.

```
Is this animal a mammal?
--Y--> Is this animal a carnivore?
        --Y--> bear
        --N--> deer
--N--> Is this animal a carnivore?
        --Y--> owl
        --N--> pigeon
```

3 Invalid trees

To uniquely identify an animal, we need a set of questions that distinguish it from all other animals.

- Modify your code to check this while building the tree.
- The *build* method should return a boolean indicating whether there are conflicts or not.
- The *build* method should also write messages to the screen if it detects animals that can not be distinguished (an example is shown below).
- You can test this on *animals_invalid.txt*.

```
Can not distinguish between cavy, calf, buffalo, boar, bear, antelope, aardvark,
Can not distinguish between catfish, carp, bass,
Not enough information to build a tree
```

4 Height of the decision tree

- Implement the *height()* method. The height of the tree is the longest path from the root to a leaf. In the context of a decision tree, this indicates how many steps we need to take until we obtain an answer in the worst case.
- Implement the *numberOfLeaves()* and *numberOfSplits()* methods. Leaves correspond to animals, internal nodes to questions.

- Implement the *averageDepth()* method. This returns the average depth of each leaf in the tree. This indicates how many steps we need to take on average to obtain an answer.

A possible output is shown below.

```
Does this animal have hair?
--Y--> Does this animal have feathers?
        --N--> Does this animal lay eggs?
                --Y--> honeybee
                --N--> Does this animal produce milk?
                        --Y--> Can this animal fly?
                                --Y--> fruitbat
                                --N--> Does this animal live in the water?
                                        --Y--> seal
                                        --N--> aardvark
--N--> Does this animal have feathers?
        --Y--> Does this animal lay eggs?
                --Y--> Does this animal produce milk?
                        --N--> Can this animal fly?
                                --Y--> Does this animal live in the water?
                                        --Y--> gull
                                        --N--> parakeet
        --N--> Does this animal lay eggs?
                --Y--> Does this animal produce milk?
                        --N--> Can this animal fly?
                                --Y--> gnat
                                --N--> frog

The height of the tree is: 7
The tree has 8 leaves
The average depth of a leaf is 5.25
The tree has 14 splits
```

5 Optimization 1: Remove redundant questions

We should try to make the tree as small as possible. A small tree with a low average depth means that we have to ask fewer questions before we can given an answer. As a first optimization, we can remove redundant questions. In the tree from previous section we first ask "Does this animal have hair". If it does, we then ask "Does this animal have feathers". This is a redundant question since there are no animals with both hair and feathers in our system.

We can recognize redundant questions as internal nodes with only one child.

- Make a copy of your code from *BinaryTree.h* to *optimization1.h* and modify the *build* method in this second file to skip redundant questions while building the tree.
- An example output is given below. This shows that instead of asking 5.25 questions on average, we can now do it with only 3.125.

```
Does this animal have hair?
--Y--> Does this animal lay eggs?
        --Y--> honeybee
        --N--> Can this animal fly?
                --Y--> fruitbat
                --N--> Does this animal live in the water?
                        --Y--> seal
                        --N--> aardvark
--N--> Does this animal have feathers?
        --Y--> Does this animal live in the water?
                --Y--> gull
                --N--> parakeet
        --N--> Can this animal fly?
                --Y--> gnat
                --N--> frog

The height of the tree is: 5
The tree has 8 leaves
The average depth of a leaf is 3.125
The tree has 7 splits
```

6 Optimization 2: Optimize the order of the questions

In the previous sections, we added one question at a time in the order they were given in the text files. This is probably not the optimal order. In the worst case, each question splits of one animal from all other remaining animals. The height of the tree would then be $O(n)$. In the best case, each question would allow us to eliminate half of the remaining questions, resulting in a height of $O(lg\ n)$. You can visualize this with *best.txt* and *worst.txt*.

We will take a greedy approach to check at each internal node what question would be the most useful to ask here. A good question gives us a high **information gain**[2]. A question that splits of one animal from all other remaining animals has not learned us much averaged over all remaining animals while a question that eliminates half of the remaining questions does.

- Make a copy of your code to *optimization2.h* and modify the *build* method in this second file to always pick the most useful remaining question at each level.
- An example output is shown below. By reordering the questions, we can reduce the average depth to 3 and the height of the tree to 4. This is a big improvement from our first approach with an average depth of 5.25 and a height of 7.
- You can also try your code on the *animals_large.txt* file with 55 animals and 22 questions.

[2]This is exactly what the ID3 algorithm for building decision trees does. Given a set of attributes for a dataset, it calculates the information gain for each attribute at each level and splits based on that attribute (https://en.wikipedia.org/wiki/ID3_algorithm).

```
Successfully built tree
Does this animal have hair?
--Y--> Can this animal fly?
        --Y--> Does this animal lay eggs?
                --Y--> honeybee
                --N--> fruitbat
        --N--> Does this animal live in the water?
                --Y--> seal
                --N--> aardvark
--N--> Does this animal have feathers?
        --Y--> Does this animal live in the water?
                --Y--> gull
                --N--> parakeet
        --N--> Can this animal fly?
                --Y--> gnat
                --N--> frog

The height of the tree is: 4
The tree has 8 leaves
The average depth of a leaf is 3
The tree has 7 splits
```