Competitive Programming Honours: Jan 2023 Batch

Emir Demirović

December 23, 2022

For the January session we are going to work on a single problem of *decision trees*. This is a challenging problem that we encountered as part of our research on optimisation algorithms. In the Algorithmics group at TU Delft, we are further looking into this problem and looking for ways to extend our approach. This problem will be a somewhat of a test of your algorithmic skills!

Note that the problem is difficult, but make sure to give it a go before the session. We divided tasks into easy, medium, and hard tasks to help you start out. With enough effort, you should be able to solve all of the tasks, given that all the techniques required have in some shape or form been taught in Algorithm Design!

1 Informal Description and Motivation

Let us first motivate and understand the problem we would like to solve. Consider the *decision tree* given in Figure 1. It models the decision making process of a particular individual, specifically their thought process on how they decide to go to work.

These type of tree-like models are called *decision trees*. You may encountered decision trees already as part of machine learning. We will consider decision trees

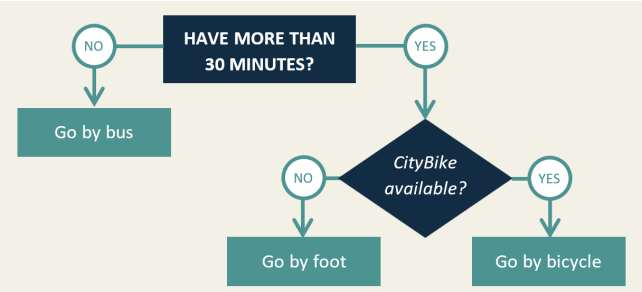

Figure 1: A Decision Tree for Commuting to Work

that are *binary trees*[a]. Making decisions based on a decision tree is intuitive. Even without describing the procedure, you probably have a good idea on how to read a decision tree. Nevertheless, for completeness, let us discuss this in more detail.

There are two types of nodes in decision trees that we consider[b]: *predicate* nodes and *classification* nodes.

A predicate node is a node that has a *predicate* associate with it, i.e., a function that takes some values as input and outputs either *true* (represented as '1') or *false* (represented as '0'). Predicates typically capture something important that one needs to consider in the decision making process. For example, in Figure 1, an example of a predicate is "Is CityBike available?", which we could view as a function that takes the current state of the surroundings as input and returns true or false depending on whether there is a city bike available. Another example of a predicate is "Have more than 30 minutes?", which could be viewed as a function that takes the current time as input and outputs *true* or *false* depending whether there is at least 30 minutes until the start of the working day (say 9.00 am).

The other type of node is a *classification* node. These nodes do not have predicates attached to it as predicate nodes do, but instead have a fixed constant or *class* assigned to it. In Figure 1, these correspond to the childless nodes (leaf nodes) at the bottom of tree, i.e., "go by foot", "go by bike", and "go by bus".

There is one more component that is important for decision trees: the state, often labelled as *instance*, based on which we be making decisions. In our example, this is simply the information about available bikes and time.

Given a decision tree, the decision making process is as recursive procedure. We start at the root node. Given a node (at the start this is the root node), first check if the node is a classification node. If it is, we stop our decision procedure and take the class given by the node. Otherwise, we evaluate the predicate at the node with respect to our instance. If the answer is *true*, we look at the *right child* of the node and apply this procedure recursively starting from that node. Similarly, if the predicate evaluated to *false*, we would do the same but this time consider the *left child*. Since at each step we are either going down the tree, or stopping, we know that eventually this procedure will terminate.

For example, if there is a city bike available and we have plenty of time, then based on Figure 1, we may conclude that we go to work by bike. However even if there is a city bike available but we short on time, then the decision would be to go by bus.

We may construct decision trees intuitively to describe some procedure. However in modern times, constructing decision trees based on *data* is a good idea. For example, say we made a large database by tracking the movement of many people[c] and recorded their choice of transportation, the weather, time,

[a] Other generalisations exist, e.g., the tree does not have to binary, it can be ternary, etc., but here we only consider binary trees, i.e., trees such that each node has at most two children.

[b] A (final!) reminder that other types of decision trees also exist, e.g., regression trees, but we do not consider these here

[c] We will disregard GDPR and other ethical questions for a moment here...

and many other factors. And say that we would like to understand the trends. One way to understand what is going on is to construct a decision tree.

In particular, we may be interested in constructing a *small* decision tree to capture the main factors of the decision making process of the particular set of people we are considering. It is likely that there is no *small* tree that *completely* accuracy describes everyone, and therefore we would then be interested in the *most accurate* tree, i.e., the tree that covers as many instances as possible.

Another use of decision trees is to compactly represent Boolean functions. This may be useful for control problems with embedded devices. Processing power and memory are scarce resources for embedded devices, so it may be important to have the smallest representation of the function that the device will perform. In this case the task is to find the decision tree that *perfectly* represents the data, since we want the device to adhere to our precise specification.

We now continue to formally describe decision trees, and then formulate a natural question: how can we construct the best tree based on data? Note that you may come up with different heuristics to construct the tree, but here we are truly considered with finding the tree that most accurately represents the data.

2 Formal Preliminaries

A *feature* is a variable that encodes information about an object. In this lecture we only consider *binary features*, meaning each feature is either zero or one[d]. A *feature vector* is a vector of features. An *instance* is a pair that consists of a feature vector and a value representing the *class*. For our purposes, we only consider *binary classes*, i.e., a class is either zero or one[e]. A *dataset*, or simply *data*, is a set of instances. The assumption is that the features describe certain characteristics about some objects[f], and the *i-th* feature of each feature vector refers to the same characteristic of interest.

A *decision tree* is a machine learning model that takes the form of a tree (see Figure 1). We consider binary trees, i.e., trees that contain nodes with at most two child nodes. We call leaf and non-leaf nodes *classification* and *predicate* nodes, respectively. The left and right edges of a predicate node are associated with the values zero and one, respectively. Each classification node is assigned a fixed class. We note that other variations of decision trees are possible, e.g., more than two child nodes, but these are not considered in this course.

A decision tree may be viewed as a function that performs classification according to the following recursive procedure. Given a feature vector, it starts by considering the root node. If the considered node is a classification node, its class determines the class of the feature vector and the procedure terminates. Otherwise, the node is a predicate node, and the left child node will be con-

[d] You may think of the predicates we previously discussed as binary features.

[e] For example, we can associate 'class 0' with 'go by foot' and 'class 1' with 'go by bike'. It is straight-forward to consider any discrete value for a class but we restrict ourselves here for simplicity.

[f] For example, the state of the day, which includes the time, the weather, and so forth

sidered next if the predicate of the node evaluates to zero, and otherwise the right child node is selected. The process recurses until a class is determined. The *misclassification score* of a decision tree on data is the number of instances for which the classification produces an incorrect class considering the data as ground truth.

The *depth* of a decision tree is the maximum number of feature nodes any instance may encounter during classification. The *size* of a decision tree is the number of feature nodes. For example, the decision tree in Fig. 1 has the depth and size equal to two. It follows that the maximum size of a decision tree with depth d is $2^d - 1$. An alternative size definition may consider the total number of nodes in the tree. Note that these definitions are equivalent, as a tree with n predicate nodes has $n + 1$ classification nodes.

Since we only consider binary features, our predicates take a special form, i.e., they simply return the value of a particular feature. For this reason we refer to predicate nodes as *feature nodes*, as the predicate depends solely on one feature.

The process of *decision tree learning* seeks to compute a decision tree that minimises a target metric under a set of constraints. In this lecture we are concerned with minimising the misclassification score given a maximum depth. Given the misclassification score, classification nodes will be assigned the class that minimises the misclassifications on the given dataset. This corresponds to computing the instances that reach the classification node during classification and selecting the class of the node according to the majority class.

3 Problem Statement

We may now state our decision tree problem as follows:

Input: A binary dataset $\mathcal{D}$ and an integer k representing the depth of the tree.

Output: A single integer representing the minimum classification score that may be achieved by using a decision tree of depth k on the dataset $\mathcal{D}$.

(We only concern ourselves with *binary classification*, meaning the class of each instances will be either '0' or '1')

The bad news is that the above described problem has been shown to be **NP**-hard. On the positive side, the problem is still solvable for *smaller* trees of limited depth, which is valuable as smaller trees can be practically relevant.

Can you construct accurate decision trees for all datasets? The reference algorithm solves each dataset within about a second with a reasonably good implementation, but we have given you three minutes (subject to server variability!) for each dataset. As you may see, it may not be straight-forward to get the fastest runtime!

The main runtime improves will come from good algorithmic ideas. To give you some hints: 1) consider a top-down dynamic programming approach, 2)

but carefully evaluate what is the best way to compute decision trees for depths 1 and 2, and 3) think whether certain divide and conquer calls can be pruned away, and 4) evaluate symmetries that may be present in the problem. There are a few other ideas to try out, however you may not need *all* of these ideas. Remember this is a challenge problem, have fun!

4 File Format

You may find input files in the same folder where you found this text. The format of an input file is as follows:

[depth of the tree]
[class for first data point] [list of feature values for first data point]
[class for second data point] [list of feature values for second data point]
...

For example, consider the following file:

2
0 0 0 0
0 1 0 0
0 0 1 1
1 1 1 1

In the above case, the task is to compute a decision tree of (decision) depth of at most two that minimises the misclassification score of a dataset that consists of four instances (three 'negative' and one 'positive' instance), each with three binary features.

To help you solve the problem, we are also providing the minimum misclassification score that you should achieve for each task.