Chapter 12 Learning with Trees

$N(logN)$

We are now going to consider a rather different approach to machine learning, starting with one of the most common and powerful data structures in the whole of computer science: the binary tree. The computational cost of making the tree is fairly low, but the cost of using it is even lower: $\mathcal{O}(logN)$, where N is the number of datapoints. This is important for machine learning, since querying the trained algorithm should be as fast as possible since it happens more often, and the result is often wanted immediately. This is sufficient to make trees attractive for machine learning. However, they do have other benefits, such as the fact that they are easy to understand (following a tree to get a classification answer is transparent, which makes people trust it more than getting an answer from a 'black box' neural network).

For these reasons, classification by **decision trees** has grown in popularity over recent years. You are very likely to have been subjected to decision trees if you've ever phoned a helpline, for example for computer faults. The phone operators are guided through the decision treed by your answers to their questions.

The idea of a decision tree is that we break classification down into a set of choices about each feature in turn, starting at the **root** (base) of the tree and progressing down to the **leaves**, where we receive the classification decision. The trees are very easy to understand, and can even be turned into a set of if-then rules, suitable for use in a **rule induction** system.

In terms of optimisation and search, decision trees use a greedy heuristic to perform search, evaluating the possible options at the current stage of learning and making the one that seems optimal at that point. This works well a surprisingly large amount of the time.

12.1 USING DECISION TREES

As a student it can be difficult to decide what to do in the evening. There are four things that you actually quite enjoy doing, or have to do: going to the pub, watching TV, going to a party, or even (gasp) studying. The choice is sometimes made for you – if you have an assignment due the next day, then you need to study, if you are feeling lazy then the pub isn't for you, and if there isn't party then you can't go to it. You are looking for a nice

algorithm that will let you decide what to do each evening without having to think about it every night.

Each evening you start at the top (root) of the tree and check whether any of your friends know about a party that night. If there is one, then you need to go, regardless. Only there is not a party do you worry about whether or not you have an assignment deadline coming up. If there is a crucial deadline, then you have to study, but if there is nothing that is urgent for the next few days, you think about how you feel. A sudden burst of energy might make you study, but otherwise you'll be slumped in front of the TV indulging your secret love of Shortland Street (or other soap opera of your choice) rather than studying. Of course, near the start of the semester when there are no assignments to do, and you are feeling rich, you'll be in the pub.

One of the reasons that decision trees are popular is that we can turn them into a set of logical disjunctions (if ... then rules) that then go into program code very simply - the first part of the tree above can be turned into:

if *there is a party* **then** *go to it*

if *there is not a party* **and** *you have an urgent deadline* **then** *study*

etc.

12.2 CONSTRUCTING DECISION TREES

In the example above, the three features that we need for the algorithm are the stated of your energy level, the date of your nearest deadline, and whether or not there is a party tonight. The question we need to ask is how, based on those features, we can construct the tree. There are a few different decision tree algorithms, but they are almost all variants of the same principle: the algorithms build the tree in a **greedy** manner starting at the root, choosing the most informative feature at each step. We are going to start by focusing on the most common: Quinlan's **ID3**, although we'll also mention its extension, known as **C4.5**, and another known as **CART**.

There was an important word hidden in the sentence above about how the trees work, which was **informative**. Choosing which feature to use next in the decision tree can be thought of as playing the game '20 Questions', where you try to elicit the item your opponent is thinking about by asking questions about it. At each stage, you choose a

question that gives you the most information given what you know already. Thus, you would as 'Is it an animal?' before you ask 'Is it a cat?'. The idea is to quantify this question of how much information is provided to you by knowing certain facts. Encoding this mathematically is the task of **information theory**.

12.2.1 Quick Aside: Entropy in Information Theory

Information theory was 'born' in 1948 when Claude Shannon published a paper called "A Mathematical Theory of Communication". In that paper, he proposed the measure of **information entropy**, which describes the amount of **impurity** in a set of features. The entropy H of a set of probabilities p_i is (for those who know some physics, the relation to physical entropy should be clear):

$$Entropy(p) = -\sum_i p_i log_2^{p_i} \tag{1}$$

where the logarithm is base 2 because we are imagining that we encode everything using binary digits (bits), and we define $0log0 = 0$. Suppose that we have a set of positive and negative examples of some feature (where the feature can only take 2 values: positive and negative). If all of the examples are positive, then we don't get any extra information from knowing the value of the features for any particular example, since whatever the value of the feature, the example will be positive. Thus, the entropy of that feature is 0. However, if the feature separates the examples into 50% positive and 50% negative, then the amount of entropy is at a maximum, and knowing about that feature is very useful to us. The basic concept is that it tells us how much extra information we would get from knowing the value of that feature. A function for computing the entropy is very simple.

For our decision tree, the best feature to pick as the one to classify on now is the one that gives you the most information, i.e., the one with the highest entropy. After using that feature, we re-evaluate the entropy of each feature and again pick the one with the highest entropy.

Information theory is a very interesting subject. It is possible to download Shannon's 1948 paper from the Internet, and also to find many resources showing where it has been applied. There are now while journals devoted to information theory because it is relevant to so many areas such as computer and telecommunication networks, machine learning, and data storage.

12.2.2 ID3

Now that we have a suitable measure for choosing which feature to choose next, entropy, we just have to work out how to apply it. The important idea is to work out how much the entropy of the whole training set would decrease if we choose each particular feature for the next classification step. This is known as the **information gain**, and it is defined as the entropy of the whole set minus the entropy when a particular feature is chosen. This is defined by (where S is the set of examples, F is a possible feature out of the set of all possible ones, and $|S_f|$ is a count of number of members of S that have value f for feature F):

$$Gain(S, F) = Entropy(S) - \sum_{f \in values(F)} \frac{|S_f|}{S} Entropy(S_f) \tag{2}$$

As an example, suppose that we have data (with outcomes) $S = \{s_1 = true, s_2 = false, s_3 = false, s_4 = false\}$ and one feature F that can have values $\{f_1, f_2, f_3\}$.

The ID3 algorithm computes this information gain for each feature and chooses the one that produces the highest value. In essence, that is all there is to the algorithm. It searches the space of possible trees in a greedy way by choosing the feature with the highest information gain at each stage. The output of the algorithm is the tree, i.e., a list of nodes, edges, and leaves. As with any tree in computer science, it can be constructed recursively. At each stage the best feature is selected and then remove from the dataset, and the algorithm is recursively called on the rest. The recursion stops when either there is only one class remaining in the data (in which case a leaf is added with that class as its label), or there are no features left, when the most common label in the remaining data is used.

The ID3 Algorithm

- If all examples have the same label:

  - return a leaf with that label

- Else if there are no features left to test:

  - return a leaf with the most common label

- Else:

  - choose the feature $\hat{F}$ that maximizes the information gain of S to be the next

---

node using Equation (2)

- add a branch from the node for each possible value f in $\hat{F}$
- for each branch:
  - calculate S_f by removing $\hat{F}$ from the set of features
  - recursively call the algorithm with S_f, to compute the gain relative to the current set of examples

Owing to the focus on classification for real-world examples, trees are often used with text features rather than numeric values.

12.3 CLASSIFICATION AND REGRESSION TREES (CART)

There is another well-known tree-based algorithm, CART, whose name indicates that it can be used for both classification and regression. Classification is not wildly different in CART, although it is usually constrained to construct binary trees. This might seem odd at first, but there are sound computer science reasons why binary trees are good, as suggested in the computation cost discussion above, and it is not a real limitation. Even in the example that we started the chapter with, we can always turn questions into binary decisions by splitting the question up a little. Thus, a question that has three answers (say the question about when your nearest assignment deadline is, which is either 'urgent', 'near' or 'none') can be split into two questions: first, 'is the deadline urgent?', and then if the answer to that is 'no', second 'is the deadline near?' The only real difference with classification in CART is that a different information measure is commonly used.

12.3.1 Gini Impurity

The entropy that was used in ID3 as the information measure is not the only way to pick features. Another possibility is something known as the **Gini impurity**. The 'impurity' in the name suggests that the aim of the decision tree is to have each leaf node represent a set of datapoints that are in the same class, so that there are no mismatches. This is known as purity. If a leaf is pure then all of the training data within it have just one class. In which case, if we count the number of datapoints at the node (or better, the fraction of the number of datapoints) that belong to a class i (call it $N(i)$), then it should be 0 for all except one value of i. So suppose that you want to decide on which feature to choose

for a split. The algorithm loops over the different features and checks how many points belong to each class. If the node is pure, then $N(i) = 0$ for all values of i except one particular one. So for any particular feature k you can compute:

$$G_k = \sum_{i=1}^{c} \sum_{j \neq i} N(i)N(j) \tag{1}$$

where c is the number of classes. In fact, you can reduce the algorithmic effort required by noticing that $\sum_i N(i) = 1$ (since there has to be some output class) and so $\sum_{j \neq i} N((j) = 1 - N(i)$. Then equation (3) is equivalent to:

$$G_k = 1 - \sum_{i=1}^{c} N(i)^2 \tag{2}$$

Either way, the Gini impurity is equivalent to computing the expected error rate if the classification was picked according to the class distribution. The information gain can then be measured in the same way, subtracting each value G_i from the total Gini impurity.

The information measure can be changed in another way, which is to add a weight to the misclassificaitons. The idea is to consider the cost of misclassifying an instance of class i as class j (which we will call the risk) and add a weight that says how important each datapoint is. It is typically labelled as λ_{ij} and is presented as a matrix, with element λ_{ij} representing the cost of misclassifying i as j. Using it is simple, modifying the Gini impurity to be:

$$G_i = \sum_{j \neq i} \lambda_{ij} N(i)N(j) \tag{3}$$

12.3.2 Regression in Trees

The new part about CART is its application in regression. While it might seem strange to use trees for regression, it turns out to require only a simple modification to the algorithm. Suppose that the outputs are continuous, so that a regression model is appropriate. None of the node impurity measures that we have considered so far will work. Instead, we'll go back to our old favorite - the sum-of-squares error. To evaluate the choice of which feature to use next, we also need to find the value at which to split the database according to that feature. Remember that the output is a value at each leaf.

In general, this is just a constant value for the output, computed as the mean average of all the datapoints that are situated in that leaf. This is the optimal choice in order to minimize the sum-of-squares error, but it also means that we can choose the split point quickly for a given feature, by choosing it to minimize the sum-of-squares error. We can then pick the feature that has the split point that provides the best sum-of-squares error, and continue to use the algorithm as for classification.