My ML studies

Babak Poursartip

May 26, 2023

Contents

Chapter 1

Introduction

1.1 General

- Supervised learning: functional approximation from labeled data.
- Unsupervised learning: functional description. Learning from unlabeled data.
- Reinforcement learning: Learning from delayed reward.

1.2 Supervised learning

There are two types of supervised learning:

- classification: taking some inputs and mapping it to some discrete label. (true of false, male or female, etc.)
- regression: continuous valued function. Mapping from input to some **real number**. Something like age is more like discrete number, so, it is also classification.

1.2.1 Terms

- Instances: Inputs, such as pictures, pixels, etc.

- Concept: A set of functions that maps inputs to outputs.
- Target concept: The actual function that can map inputs to outputs. This is the actual answer.
- Hypothesis class: Set of all the functions that we are going to think about.
- Sample (Training set): A set of instances with correct labels.
- Candidate: The best approximation of the target concept.
- Testing set: A set of instances with correct labels that was not visible to the learning algorithm during the training phase. It's used to determine the algorithm performance on novel data. we can argue that we learned something by memorization, but indeed, we just memorized the concepts. What we want is generalization.

Chapter 2

Decision tree

2.1 Intro

First, we need to come up with a set of features for our decision tree.

2.2 Representation vs Algorithm:

A decision tree is a structure of nodes, edges and leaves that can be used to represent the data.

- Nodes represent attributes where you ask a question about it. (vehicle length, vehicle height, number of doors, etc.).
- Edges represent values. (A specific value for each attribute)
- Leaves represent the output/answer (vehicle's type).

A decision tree algorithm is a sequence of steps that will lead you to the desired output. To form a decision tree, we need to come up with the attributes that can split the space, roughly in half.

- Pick the best attribute (the one that can split the data roughly in half). If this attribute added no valuable information (not a good split), it might cause overfitting.
- Ask a question about this attribute.
- Follow the correct answer path.

- Loop back to (1) till you narrow down the possibilities to one answer (the output).

Note that the usefulness of a question depends upon the answers we got to previous questions.

2.3 Expressiveness

- Decision trees can basically express any function using the input attributes.
- For Boolean functions (AND, OR, XOR, etc.), the truth table row will be translated into a path to a leaf.
- How many decision trees we can generate for a specific function/problem? Decision tree hypothesis space is very huge ($2^{(2^n)}$), this why we need to design algorithms to efficiently search the hypothesis space for the best decision tree.

2.4 ID3 algorithm to create a decision tree

ID3 builds decision trees using a top-down, greedy approach. The greedy part of the approach comes from the fact that it will decide which attribute should be at the root of the tree by looking just one move ahead. It compares all available attributes to find which one classifies the data the best, but it doesn't look ahead (or behind) at other attributes to see which combinations of them classify the data the best.

Pseudocode:

1. Pick the best attribute A (The definitions of best attribute comes later). To select this attribute, a statistical test is used to determine for each attribute how well it alone classifies the training examples.
2. Split the data into subsets that correspond to the possible values of the best attribute.
3. Assign A as a decision attribute for a node.
4. For each value of A, create a descendant node.

5. Sort training examples to leaves.

6. If examples perfectly classified – Stop – else –Iterate over leaves

How to find the best attribute for the decision tree at each level:
There are a couple of options. The most common method is called information gain: a measure that expresses how well an attribute splits the data into groups based on classification. For example, an attribution returns *low information gain*, if it divides samples to two classes with an even yes and no, but the information gain is high, if each class has more of yeses or nos.

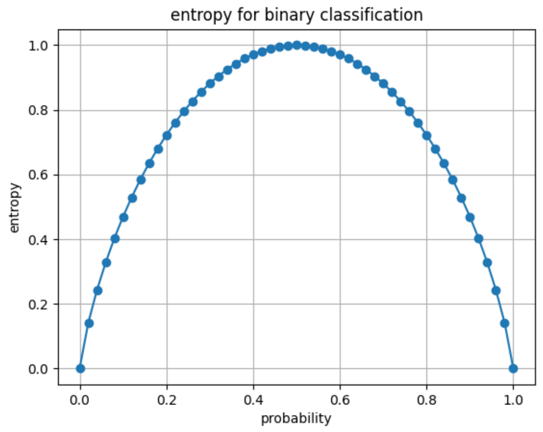
It quantifies the reduction in randomness (Entropy) over the labels we have with a set of data, based upon knowing the value of a particular attribute. A mathematical way to measure the gain, is entropy. It measures the homogeneity of a data set S's classifications. It ranges from 0, which means that all of the classifications in the data set are the same (either yes or no), to log_2 of the number of different classifications, which means that the classifications are equally distributed within the data set. For a binary classification the entropy is only $log_2 1 = 1$ (if yes and no samples are equally divided). entropy is calculated as follows:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \; log_2(p_i) \tag{2.1}$$

where:

- c corresponds to the number of different classifications
- p_i corresponds to the proportion of the data with the classification i

Here is the plot of entropy for a binary classifier, as the proportional of



yes and no samples change:

Information gain measures the reduction in entropy that results from partitioning the data on an attribute A, which is another way of saying that it represents how effective an attribute is at classifying the data. Given a set of training data S and an attribute A, the formula for information gain is:

$$Gain(S, A) = Entropy(S) - \sum_{v} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.2}$$

where v is all the possible values of attribute A (for example, sunny, cloudy, rainy), and S_v is the total number of samples (regardless of the value of the sample) corresponding to this value of attribute (for example sunny). We want to maximize information gain, so we want the entropies of the partitioned data to be as low as possible, which explains why attributes that exhibit high information gain split training data into relatively heterogeneous groups.

As the size of the training data and the number of attributes increase, it becomes likelier that running ID3 on it will return a suboptimal decision tree.

2.5 Extending ID3 algorithm

Small tweaks allow ID3 to handle continuous attributes, missing attribute values, data that doesn't work particularly well with information gain, and more.

For continuous attributes, we can classify the attributes based on thresholds.

If the data for some attributes is missing, we can choose the most popular data for the missing item. Two options here: does not take the classification into account, or choose the most popular value for the same classification. Another method to deal with the missing data, is assigning the probability of each value of the attribute to the missing entries.

2.6 over-fitting

There are two popular approaches to avoid this in decision trees: stop growing the tree before it becomes too large or prune the tree after it becomes too large. Typically, a limit to a decision tree's growth will be specified in terms of the maximum number of layers, or depth, it's allowed to have.

The data available to train the decision tree will be split into a training set and test set and trees with various maximum depths will be created based on the training set and tested against the test set.

Pruning the tree, on the other hand, involves testing the original tree against pruned versions of it. Leaf nodes are taken away from the tree as long as the pruned tree performs better against test data than the larger tree.

Changing the information gain formula: The information gain formula used by the ID3 algorithm treats all of the variables the same, regardless of their distribution and their importance. This is a problem when it comes to continuous variables or discrete variables with many possible values because training examples may be few and far between for each possible value, which leads to low entropy and high information gain by virtue of splitting the data into small subsets, but results in a decision tree that might not generalize well.

One successful approach to deal with this is using a formula called GainRatio in the place of information gain. GainRatio tries to correct for information gain's natural bias toward attributes with many possible values by adding a denominator to information gain called SplitInformation. SplitInformation attempts to measure how broadly partitioned an attribute is and how evenly distributed those partitions are. In general, the SplitInformation of an attribute with n equallydistributed values is log 2 n . These relatively large denominators significantly affect an attribute's chances of being the best attribute after an iteration of the ID3 algorithm and help to avoid choices that perform particularly well on the training data but not so well outside of it.

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInformation(S, A)}$$
$$SplitInformation(S, A) = -\sum_{i=1}^{c} \frac{|S_i|}{S} \, log_2 \frac{|S_i|}{|S|}$$

2.7 BIAS (Inductive bias)

Two types of bias: restriction bias (hypophysis), and preference bias (prefer a subset of hypophisys, this is inductive).

Bias of ID3 are:

- it prefers the decision tree with good splits at top (even if a bad split generates the same outcome).
- it prefers correct outcome over incorrect.
- it prefers shorter trees (comes from that fact that we use good splits from the top).