Decision trees

A ***decision tree*** is a method for classification/regression that aims to ask a few relatively simple questions about an input $\mathbf{x} \in \mathbb{R}^d$ and then predicts the associated output y

Decision trees are useful to a large degree because of their ***simplicity*** and ***interpretability***

The resulting output rule is something that can easily be inspected and interpreted by hand… something that is not really the case with most algorithms

We will also see later that decision trees form a useful building block for other more sophisticated algorithms

Hypothetical example

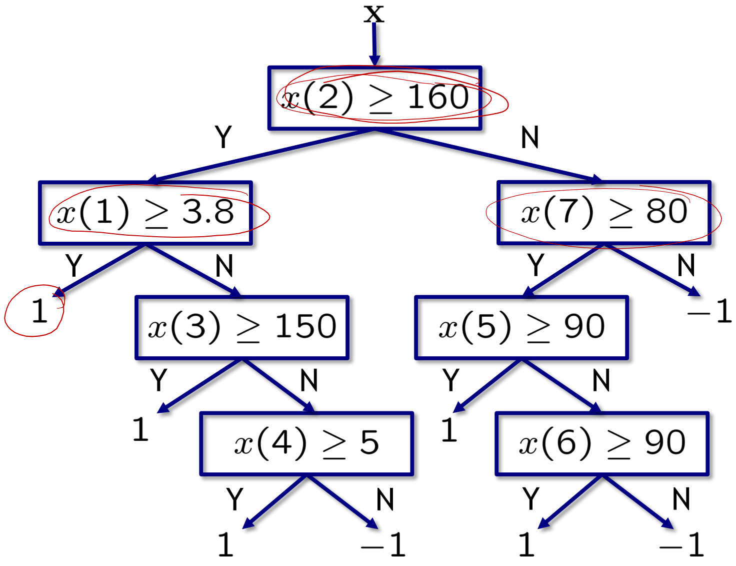
Suppose we are given a dataset where the $\mathbf{x}_i$ are 7-dimensional feature vectors with features

1. Undergraduate GPA (4.0 scale)
2. Quantitative GRE score (130-170)
3. Verbal GRE score (130-170)
4. Analytical Writing GRE score (0-6)
5. Undergraduate institution reputation (0-100)
6. Statement of purpose evaluation (0-100)
7. Letter of recommendation evaluation (0-100)

The labels for this dataset are:

$$y_i = \begin{cases} 1 & \text{student was accepted} \\ 0 & \text{student was denied} \end{cases}$$

Example decision tree

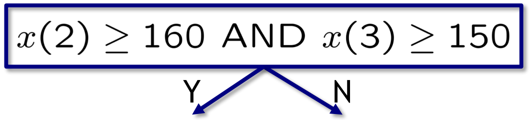


Generalizations

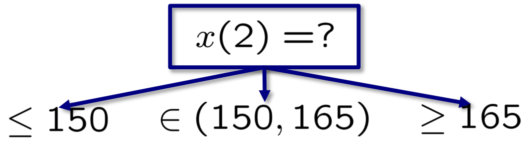
Decision trees can be applied to regression where the labels at the "leaf nodes" are real-valued (or real-valued functions)

Other generalizations include:

- splits that involve more than one feature



- Splits involving more than two outcomes

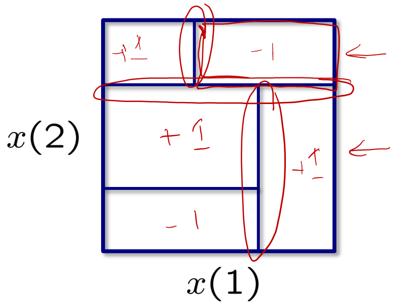


Binary decision trees

Using multiple features and allowing splits with more than two outcomes generally increases the risk of ***overfitting***

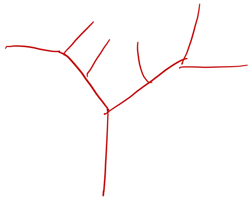
We will restrict our attention to binary, single-feature splits

In this case, every (binary) decision tree is associated with a ***partition*** of the feature space that looks something like this example in $\mathbb{R}^2$



Terminology

- The elements of the partition are called ***cells***
- Recall that a ***graph*** is a collection of ***nodes*** or ***vertices***, some of which are joined by ***edges***
- The ***degree*** of a vertex is the number of edges incident on that vertex
- A ***tree*** is a *connected* graph with *no cycles*

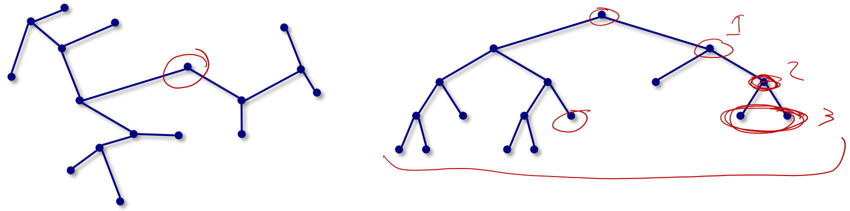


When is a tree not a tree?

Rooted binary trees

A ***rooted binary tree*** is a tree where

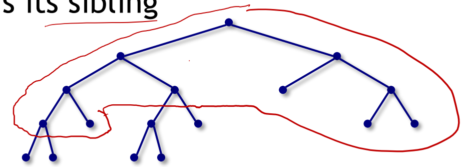
- one vertex, called the ***root***, has degree 2
- and all other vertices have degree 1 or 3



- vertices with degree 1 are called ***leaf*** or ***terminal*** vertices
- vertices with degree 2 or 3 are called ***internal*** vertices

More terminology

- The ***depth*** of a vertex is the length to the root
- The ***parent*** of a vertex is the neighbor with depth one less
- Two vertices are ***siblings*** if they have the same parent
- A ***subtree*** is a subgraph that is also a tree
- A ***rooted binary subtree*** is a subtree that contains the root and is such that if any non-root vertex is in the subtree, so is its sibling



A ***binary decision tree*** is a rooted binary tree where each internal vertex is associated with a binary classifier, and each leaf with a label

Learning decision trees

Let $\mathcal{T}$ denote the set of all binary decision trees

Given a dataset $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, it might be tempting to pose the learning problem as an optimization problem of the form

$$\min_{T \in \mathcal{T}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, T(\mathbf{x}_i))$$

where ℓ is an appropriate loss function, e.g.,

- 0/1 loss for classification
- squared error loss for regression

Unfortunately, this will lead to massive overfitting

If we grow the tree deep enough, we can usually fit the training data perfectly!

Penalized empirical risk minimization

Since deep/complex decision trees tend to overfit, we would like to avoid such trees

A natural way to quantify the complexity of a tree is the number of leaf nodes, which we denote by $|T|$

Note that if $|T| = n$, we would very likely be overfitting

The penalized ERM approach is to consider the optimization problem

$$\min_{T \in \mathcal{T}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, T(\mathbf{x}_i)) + \lambda |T|$$

Learning a tree in practice

Unfortunately, this optimization problem is intractable

A two-stage procedure is typically employed in practice

1. Grow a very large tree T_0 in a greedy fashion
2. Prune T_0 by solving

$$\min_{T \in \mathcal{T}_0} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, T(\mathbf{x}_i)) + \lambda |T|$$

where $\mathcal{T}_0$ is the set of all decision trees based on rooted binary subtrees of T_0

Growing a decision tree

The construction of T_0 follows a simple greedy (looking just one step ahead) strategy

1. Start at the root vertex (i.e., the entire feature space)
2. Decide whether to stop growing tree at current vertex
   - If yes, assign a label and stop
3. If no, consider all possible ways to split the data in the cell corresponding to the current vertex, and select the best one
4. For each branch of the split, create a new vertex and go to step 2

Implementation

To implement this strategy, we need:

- A list of possible splits

  When considering splits based on a single real-valued feature, splits have the form
  $$x(j) \leq t?$$
  Since there are only n data points, only at most $n-1$ values of t need to be considered for each j

  For discrete or categorical features, other simple splits can be used, such as
  $$x(j) = \text{“blue”}\,?$$

Implementation

To implement this strategy, we need:

- A labeling rule

  For classification, we can just assign labels by majority vote over data in the cell corresponding to the current vertex

  For regression, we can just take the average of the y_i in the cell for a piecewise constant approximation
  (or least squares fit for piecewise polynomial or other approximation)

Implementation

To implement this strategy, we need:

- A rule for stopping splitting

  The most common strategy is to just split until each leaf vertex contains a single data point

- A rule for selecting the best split

  This is the hard part!

Split selection

Let's focus on binary classification

Suppose V is a leaf vertex at some stage in the growing process

We can think of V as also defining a cell in the feature space, allowing us to consider $\{\mathbf{x}_i : \mathbf{x}_i \in V\}$

Intuitively, a good split of V will lead to children that are more ***homogenous*** or ***pure*** than V

To define this, we will define a notion of ***impurity***

Impurity measure

Assume the class labels are $\{+1, -1\}$

Let

$$q := \frac{|\{i : \mathbf{x}_i \in V, y_i = 1\}|}{|\{i : \mathbf{x}_i \in V\}|}$$

Note that $(q, (1-q))$ defines a probability distribution on the labels (i.e., on the probability of getting each label on a randomly selected point in V)

An ***impurity measure*** is a function $i(V)$ such that

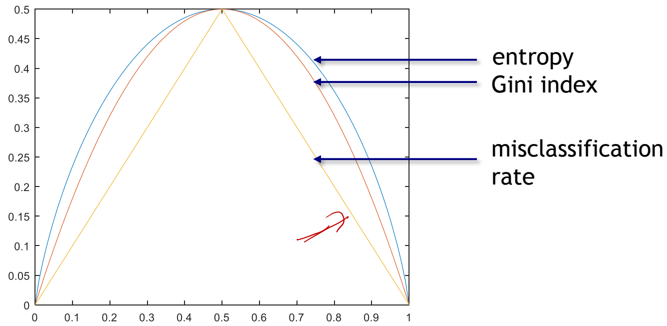
- $i(V) \geq 0$, with $i(V) = 0$ iff V consists of a single class
- a larger value of $i(V)$ indicates that the distribution defined by $(q, (1-q))$ is closer to the uniform distribution

Examples

Entropy: $i(V) = -\big(q \log q + (1-q) \log(1-q)\big)$

Gini index: $i(V) = 2q(1-q)$

Misclassification rate: $i(V) = \min(q, 1-q)$



Using the impurity measure

To select the best split, we aim to ***maximize*** the ***decrease*** in impurity

Let V_1 and V_2 be two children of V

Define

$$p(V_1) = \frac{|\{\mathbf{x}_i : \mathbf{x}_i \in V_1\}|}{|\{\mathbf{x}_i : \mathbf{x}_i \in V\}|} \qquad p(V_2) = \frac{|\{\mathbf{x}_i : \mathbf{x}_i \in V_2\}|}{|\{\mathbf{x}_i : \mathbf{x}_i \in V\}|}$$

The decrease in impurity is

$$i(V) - \Big(p(V_1) i(V_1) + p(V_2) i(V_2)\Big)$$

When i is entropy, this is called the ***information gain***

Nonnegativity of decrease in impurity

Proposition

If i is concave as a function of q, then

$$i(V) - \Big(p(V_1)i(V_1) + p(V_2)i(V_2)\Big) \geq 0$$

for all V_1 and V_2

If i is *strictly* concave and $V_1 \neq V_2$ and $V_1, V_2 \neq \emptyset$, then equality holds if and only if $q = q_1 = q_2$, where q_i is the proportion of class 1 in V_i

Because of this fact, *strictly* concave impurity measures are generally preferred

Note that this generalizes easily to multiclass classification

Proof

Want to show

$$\boxed{i(V) - \Big(p(V_1)i(V_1) + p(V_2)i(V_2)\Big) \geq 0}$$

If we write $i(V) = \phi(q)$, then we have

$$\begin{aligned} i(V) = \phi(q) &= \phi\big(p(V_1)q_1 + p(V_2)q_2\big) \\ &\geq p(V_1)\phi(q_1) + p(V_2)\phi(q_2) \\ &= p(V_1)i(V_1) + p(V_2)i(V_2) \end{aligned}$$

Jensen's inequality

Since V_1 and V_2 are assumed to be nonempty, $p(V_1) > 0$ and $p(V_2) > 0$, and therefore equality holds if and only if $q_1 = q_2$, in which case $q = q_1 = q_2$.

Pruning

Pruning the model involves solving the optimization problem:

$$\min_{T \in \mathcal{T}_0} \underbrace{\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, T(\mathbf{x}_i)) + \lambda|T|}_{f(T)}$$

$f(T)$ is additive in the following sense:

For any tree T, let $\Pi(T) = \{T_1, T_2\}$ be the partition of the tree corresponding to the children of the root vertex

Then $f(T) = f(T_1) + f(T_2)$

Applying this idea recursively suggests a natural algorithm of ***weakest link pruning***. One can also attack this via an efficient “bottom up” dynamic programming approach.

Why grow then prune?

Why should we go to the trouble of growing the tree and then pruning?

A simpler approach would be to just grow the tree and then stop when the decrease in impurity is neglibible

Answer: Ancillary splits

These are splits that have no value by themselves, but enable useful splits later on

Remarks

Advantages

- interpretable
- rapid evaluation
- easily handles
  - categorical/mixed data
  - missing data
  - multiple classes

Disadvantages

- unstable: slight perturbations of training data can drastically alter the learned tree
- jagged decision boundaries