Decision Trees

Carlo Tomasi

November 4, 2018

1 Introduction

Linear regressors from $X = \mathbb{R}^d$ to $Y = \mathbb{R}^K$ have a limited expressive power, because they fit K *affine* functions to the training set. Functions that are more complex than that are approximated poorly. Similarly, K-class linear classifiers partition the data space X into K *convex* regions. If the true decision regions are not convex, performance suffers.

This limited expressive power has also a silver lining: The number m of parameters of linear predictors is relatively small, $m = K(d+1)$, so that a linear predictor requires fewer training samples to train, relative to a more expressive predictor, in order to achieve a certain value for the training risk. As a result, linear predictors generalize better than more expressive ones.

When we studied linear classifiers, we also saw the advantages of score-based classifiers: The K scores output by a K-class logistic-regression classifier can be interpreted as a probability distribution of the predicted label $\hat{y} = h(\mathbf{x})$ given the input $\mathbf{x}$, and this distribution can be used to compute some measure of the confidence of the answer.

Decision trees combine the advantages of a score-based predictor (for both classifiers and regressors!) with the expressiveness deriving from a very flexible partition of X. Specifically, they recursively split X with hyperplanes, and they assign a single value (real number or label, depending on whether the problem is one of regression or classification) to each subset of the partition. This recursive splitting leads to effectively arbitrary expressiveness, as long as the partition is fine enough.

Since the splitting is recursive, the resulting partition of the data space X can be represented by a strictly binary tree. As will be seen in more detail below, the root of the tree represents all of X, and contains the parameters of the hyper-plane used for the first split. The two children of the root represent the two half-spaces that that hyper-plane divides X into. Each of the two children in turn contains an additional hyper-plane that splits the child's half-space, and the structure continues recursively.

The tree can then be used for prediction: Given a data point $\mathbf{x}$, check which side of the root partition it belongs to, and send it to the corresponding child. This check is repeated at every internal tree node encountered in this way, until a leaf is reached. The leaf corresponds to a (possibly small) convex region of X, and a single value is associated to that region. That value is the prediction $y = h(\mathbf{x})$ returned by the tree. We will see in the next section how the hyper-planes are constructed, and how the prediction value at each leaf is computed.

Staying at a high level for now, decision trees determine each of the hyper-planes greedily, by choosing the hyper-plane (out of a set of choices that is typically restricted, as we will see) that maximizes the predictor's confidence, gauged through a measure called *purity*. Different definitions of purity have been proposed, but they are all based on the distribution $p(y|X_S)$ of the values y for data points $\mathbf{x} \in X_S$, where X_S is one of the regions of the partition. Loosely speaking, $p(y|X_S)$ is *pure* if it is heavily concentrated

around a small set of values, so that there is at least an approximate agreement as to what individual value should be assigned to data points $\mathbf{x}$ in X_S.

Of course, $p(y|X_S)$ is unknown, but it can be estimated from the training data:

$$p(y|X_S) \;\approx\; p(y|S)$$

where S is the subset of the training set T whose data points are in X_S:

$$S \stackrel{\mathsf{def}}{=} \{(\mathbf{x}, y) \in T \;:\; \mathbf{x} \in X_S\}$$

and $p(y|S)$ denotes an empirical estimate of a distribution, for instance, a histogram. With this approximation, the empirical probability distributions of values in subsets S of the training set T are interpreted as (estimates of the) statistical probability distributions of values for entire regions X_S of data space X.

In particular, the distributions $p(y|S)$ for the leaves of the tree are used to compute a prediction value through a statistical *summary* of $p(y|S)$:

$$\hat{y}(X_S) = \text{summary}(p(y|X_S)) \;.$$

Typically, the summary is a mode (majority value) for classification, and either a mean or a median for regression. These choices are akin to the ones used for k-nearest-neighbor classification when $k > 1$.

As should be clear from the discussion so far, decision trees also have limitations. First, the partition into regions is greedy, and therefore sub-optimal: It is possible (and indeed likely) that to achieve a good partition overall one may have to make partition decisions which, when seen individually and out of context, appear to be sub-optimal.

A second weakness of decision trees results from their very expressiveness, as we have by now come to expect: Flexibility leads to over-fitting. Because of this, a large body of literature has been devoted to ways of *pruning* decision trees, that is, curb their ability to partition X so as to improve generalization. We will not study pruning methods, because a better way to improve generalization is to build multiple trees with different views of the data (in a sense to be clarified later), and let them vote on a value. This is the basic idea of *random decision forests*, the subject of a later note.

The present note first defines decision trees in general, and then considers how decision trees are trained. Choosing a training method involves defining a measure of purity, a way to infer a locally optimal partition of a set S into two subsets L and R so as to increase purity as much as possible, and a criterion for stopping the recursive partition process.

2 The Structure of Decision Trees and their Use as Predictors

A *decision tree* is a binary tree that defines a recursive partition of the data space X into subregions. Specifically, the root τ of the tree is associated to all of X, and contains a predicate $\mathcal{P}_\tau(\mathbf{x})$ called a *split rule*. The left child $\tau.L$ of τ is associated to the subset X_L of X of all the points in X that satisfy the predicate. The right child $\tau.R$ of τ is associated to the complement X_R of X_L. Each child is then expanded recursively in the same way, by its own predicate.

During training, the same predicates are applied to the training set T, rather than to the data space X, with the result of splitting T into subsets. Specifically, the set S at an internal node is split into those samples whose data points are in X_L and those whose data points are in X_R:

$$L \stackrel{\mathsf{def}}{=} \{(\mathbf{x}, y) \in S \mid \mathbf{x} \in X_L\} \quad \text{and} \quad R \stackrel{\mathsf{def}}{=} \{(\mathbf{x}, y) \in S \mid \mathbf{x} \in X_R\} \;.$$

Just as before, we can view $p(y|L)$ and $p(y|R)$ as estimates of the probability distributions of y in X_L and X_R, respectively:

$$p(y|L) \approx \mathbb{P}_T[y \mid \mathbf{x} \in X_L] \quad \text{and} \quad p(y|R) \approx \mathbb{P}_T[y \mid \mathbf{x} \in X_R] \ .$$

The split rules are learned by partitioning the training set T recursively in a way that increases the purity of the subsets formed by each split, in a sense to be made more precise later on.

A popular binary split rule called a *1-rule* partitions a subset $S \subseteq X \times Y$ into the two sets

$$L = \{(\mathbf{x}, y) \in S \,|\, x_j \leq t\} \quad \text{and} \quad R = \{(\mathbf{x}, y) \in S \,|\, x_j > t\} \tag{1}$$

where x_j is the j-th component of $\mathbf{x}$ and t is a real number. This, a 1-rule only allows for separating hyperplanes that are aligned with the coordinate axes: Pick a coordinate index j, and check if x_j is above or below a threshold t.

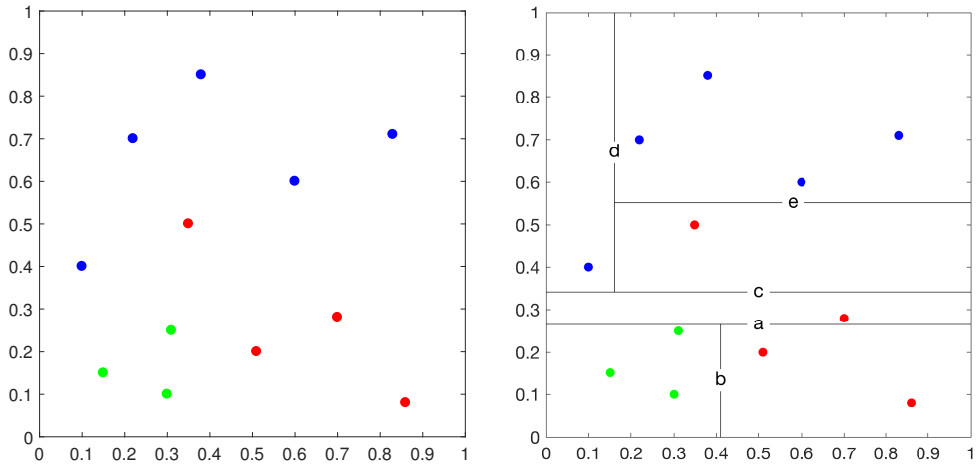
This note considers only binary decision trees[1] with 1-rule splits, and the word "binary" is omitted in what follows.

Concretely, the split rules are placed on the interior nodes of a binary tree and the probability distributions are on the leaves. The tree τ can be defined recursively as either a single (leaf) node with values of the posterior probability $p(y|S)$ collected in a vector $\tau.\mathbf{p}$ or an (interior) node with a split function with parameters $\tau.j$ and $\tau.t$ that returns either the left or the right descendant ($\tau.L$ or $\tau.R$) of τ depending on the outcome of the split. A prediction tree τ then takes a new data point $\mathbf{x}$, looks up its posterior distribution in the tree by following splits in τ down to a leaf, and returns the value y obtained by summarizing $\tau.\mathbf{p}$. This algorithm is illustrated in Figure 1 for a three-class classifier (so that the summary is $\arg\max(\tau.\mathbf{p})$) and detailed in Algorithm 1.

Algorithm 1 Prediction with a decision tree

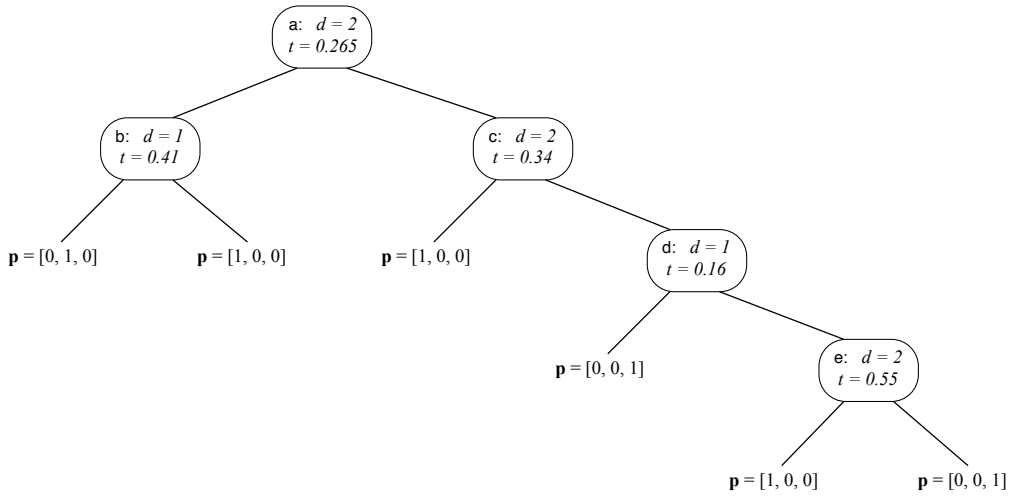
```
function y ← predict(x, τ, summary)
    if leaf?(τ) then
        return summary(τ.p)
    else
        return predict(x, split(x, τ), summary)
    end if
end function

function τ ← split(x, τ)
    if x_{τ.j} ≤ τ.t then
        return τ.L
    else
        return τ.R
    end if
end function
```

[1]The tree is binary, the predictor is not necessarily binary. For instance, a binary classification tree can handle more than two classes.



node	impurity
a	0.583
b	0.4
c	0.286
d	0.167
e	0.2

(c)



(d)

Figure 1: The training samples in (a) belong to three categories (1: red, 2: green, 3: blue). The partition in (b) is computed in the order a, b, c, d, e, and the impurities of the sets each line splits are shown in (c). The tree in (d) represents the same partition, and each interior node corresponds to one line. All the leaves of the tree contain zero-impurity posterior probabilities $\mathbf{p} = [\, p(\text{red}|\mathbf{x}),\ p(\text{green}|\mathbf{x}),\ p(\text{blue}|\mathbf{x})\,]$. This purity corresponds to the fact that all the points in each region of the partition in (b) have the same label.

3 Training Prediction Trees

Optimal training of a prediction tree would compute the partition of X that leads to the lowest possible generalization error. In addition, a good tree from a computational point of view would use the minimum possible number of splits. The second requirement, optimal efficiency, is unrealistic, since building the most efficient tree is NP-complete, as was proven by a reduction of the set cover problem [1].

Prediction trees are trained with a greedy procedure, and only ensure optimality—on the training set—at each node separately. The procedure is sketched in Algorithm 2, and is invoked with the call

$$\mathrm{trainTree}(T, 0) \ .$$

The algorithm first determines whether the set S it is given as input is worth splitting.[2] If so, it finds optimal parameters j and t that split S into sets L and R (equation (1)), stores those parameters at the root of a new tree τ, stores as the root's children $\tau.L$ and $\tau.R$ the result of calling itself on sets L and R, and returns τ.

If on the other hand S is not worth splitting, then the new tree τ is a single leaf node that contains an estimate of the posterior distribution of labels in S.

This procedure leads to large trees that overfit and therefore generalize poorly, so a second step of training *prunes* the tree to improve the generalization error. However, random forest classifiers address generalization in a different way, by combining the predictions made by several trees. Because of this, pruning is not performed in random forest predictors, and is not discussed here. Instead, we now show how to split a set (findSplit), how to decide whether to continue splitting (split?), and how to estimate the distribution of labels in a set (distribution).

Splitting. The optimal single split of training data in set S into two sets L and R maximizes the decrease

$$\Delta \mathrm{i}(S, L, R) = \mathrm{i}(S) - \frac{|L|}{|S|}\mathrm{i}(L) - \frac{|R|}{|S|}\mathrm{i}(R) \tag{2}$$

where $\mathrm{i}(S)$ is the *impurity* of S. The so-called *Gini index*

$$\mathrm{i}(S) = 1 - \sum_{y \in Y} p^2(y|S)$$

is often used for either classifiers or regressors, where

$$p(y|S) = \frac{1}{|S|} \sum_{(\mathbf{x}_i, y_i) \in S} I(y_i \approx y)$$

is the fraction of training values in set S that fall into the same bin as (or are equal to, for classifiers) value y. The Gini index minimizes the training error for the stochastic decision rule

$$y = h_{\mathrm{Gini}}(\mathbf{x}) = y \text{ with probability } p(y|S(\mathbf{x}))$$

where $S(\mathbf{x})$ is the region of data space that data point $\mathbf{x}$ falls into. When this predictor returns value y as the answer, it contributes to the training error with probability that is approximately $1 - p(y|S)$, because that is the fraction of samples in S whose values do not fall into the same bin as y. So the training error for the

[2] Read on to find out when a set is worth splitting, how the optimal split parameters are found, and how the posterior distribution is estimated.

Algorithm 2 Training a prediction tree

function $\tau \leftarrow \text{trainTree}(S, \text{depth})$
  if $\text{split?}(S, \text{depth})$ **then**
    $[L, R, \tau.j, \tau.t] \leftarrow \text{findSplit}(S)$
    $\tau.L \leftarrow \text{trainTree}(L, \text{depth} + 1)$
    $\tau.R \leftarrow \text{trainTree}(R, \text{depth} + 1)$
  else
    $\tau.\mathbf{p} \leftarrow \text{distribution}(S)$
  end if
  return τ
end function

function $[L, R, j, t] \leftarrow \text{findSplit}(S)$
  $i_S \leftarrow \text{i}(S)$ ▷ $\text{i}(S)$ is the impurity of S. See text.
  $\Delta_{\text{opt}} \leftarrow -1$ ▷ At the end, Δ_{opt} will be the greatest decrease in impurity.
  for $j = 1, \ldots, d$ **do** ▷ Loop on all data dimensions.
    for $\ell = 1, \ldots, u_j$ **do** ▷ Loop on all thresholds for dimension j.
      $L \leftarrow \{(\mathbf{x}, y) \in S \,|\, x_j \leq t_j^{(\ell)}\}$ ▷ The thresholds $t_j^{(\ell)}$ for $j = 1, \ldots, N$ and $\ell = 1, \ldots, u_j$
      $R \leftarrow S \setminus L$ ▷ are assumed to have been precomputed (see text).
      $\Delta \leftarrow i_S - \frac{|L|}{|S|}\text{i}(L) - \frac{|R|}{|S|}\text{i}(R)$ ▷ See text for a faster way to compute Δ
      if $\Delta > \Delta_{\text{opt}}$ **then**
        $[\Delta_{\text{opt}}, L_{\text{opt}}, R_{\text{opt}}, d_{\text{opt}}, t_{\text{opt}}] \leftarrow [\Delta, L, R, j, t]$
      end if
    end for
  end for
  return $[L_{\text{opt}}, R_{\text{opt}}, d_{\text{opt}}, t_{\text{opt}}]$
end function

function $\text{answer} \leftarrow \text{split?}(S, \text{depth})$
  return $\text{i}(S) > 0$ and $|S| > s_{\min}$ and $\text{depth} < d_{\max}$ ▷ $s_{\min}$ and $d_{\max}$ are predefined thresholds
end function

function $\mathbf{p} \leftarrow \text{distribution}(S)$
  $\mathbf{p} \leftarrow [0, \ldots, 0]$ ▷ A vector of K zeros
  $n \leftarrow 0$
  for $(\mathbf{x}, y) \in S$ **do**
    $p(y) \leftarrow p(y) + 1$
    $n \leftarrow n + 1$
  end for
  return $\mathbf{p}/n$
end function

Gini classifier is the sum of these error probabilities, weighted by the probability that the predictor returns y:

$$\overline{\mathrm{err}}_{\mathrm{Gini}}(S) = \sum_{y \in Y} p(y|S)(1 - p(y|S)) = 1 - \sum_{y \in Y} p^2(y|S) = \mathrm{i}(S) \ .$$

Note that in this interpretation the number of possible values for y is finite even for regressors, because the true distribution $p(y|X_S)$ is replaced with the empirical histogram $p(y|S)$. This limitation could be overcome by other representations for $p(y|X_S)$ that do not resort to bins.

For classifiers, a simpler alternative measure of impurity is

$$\mathrm{i}(S) = \overline{\mathrm{err}}(S) \ ,$$

the training error accrued for the elements in S if this set were no longer split. This measure of impurity is therefore the fraction of labels in S that are different from the label that occurs most frequently in S, since all these labels would be misclassified:

$$\overline{\mathrm{err}}(S) = 1 - \max_y p(y|S) \ .$$

Both the Gini index and the training error are empirical measures of the impurity of the distribution of the training data in set S, in the sense that when and only when all training data in S have the same value (S is "pure") one obtains

$$\overline{\mathrm{err}}(S) = \overline{\mathrm{err}}_{\mathrm{Gini}}(S) = 0 \ ,$$

and the two measures are otherwise positive. The choice of impurity measure depends on the application domain, and it is difficult to give general rules of thumb for which one is better.

With either measure of impurity, the best split is found in practice by cycling over all values of the component index $j \in 1, \ldots, d$ and all possible choices of threshold t in equation (1). The number of thresholds to be tried is finite because the number of training samples and therefore values of x_j is finite as well: If x_j and x'_j are consecutive values for the j-th component of $\mathbf{x}$ among all the samples in T, there is no need to evaluate thresholds that are between x_j and x'_j. As a refinement, one can build a sorted list

$$x_j^{(0)}, \ldots, x_j^{(u_j)}$$

of the u_j unique values of x_j in T and set the thresholds to be tested as

$$t = t_j^{(1)}, \ldots, t_j^{(u_j)} \quad \text{where} \quad t_j^{(\ell)} = \frac{x_j^{(\ell-1)} + x_j^{(\ell)}}{2} \quad \text{for} \quad \ell = 1, \ldots, u_j$$

to maximize the prediction margin.

The function $\mathrm{findSplit}$ in Algorithm 2 summarizes the computation of the optimal split. Efficiency can be improved by sorting the values of x_j and updating $|L|$, $|R|$, $\mathrm{i}(R)$, $\mathrm{i}(L)$ and Δ while traversing the list from left to right, rather than computing Δ from scratch at every iteration.

Stopping Criterion. It is dangerous to stop splits when the change in training error falls below some threshold, because a split that seems useless now might lead to good splits later on. Consider for instance the data space

$$X = \{\mathbf{x} \in \mathbb{R}^2 \mid -1 \leq x_1 \leq 1 \text{ and } -1 \leq x_2 \leq 1\}$$

for a classification problem with $K = 2$ classes and true labels

$$y = c_1 \text{ for } x_1 x_2 > 0 \quad \text{and} \quad y = c_2 \text{ for } x_1 x_2 < 0 \ .$$

Splitting on either x_1 or x_2 once does not change the misclassification rate, but splitting twice leads to a good classifier. In other words, neither data point is predictive by itself, but the two of them together are.

Instead, one typically stops when the impurity of a set is zero, or when splitting a set would result in too few samples in the resulting subsets, or when the tree has reached a maximum depth. See Algorithm 2.

Label Distribution. The training algorithm places an estimate of the posterior distribution of labels given the data point at each leaf of the prediction tree. This estimate is simply the empirical estimate from the training set. Specifically, for a classifier the set of possible values is Y. For a regressor, the possible values in Y are binned into a predetermined number of bins, indexed by an index y. Either way, if the leaf set S contains N_y samples with index y, the distribution is

$$p(y|S) = \frac{N_y}{|S|} \ .$$

References

[1] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees in NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.