Machine Learning

Decision Tree and Random Forest

Fabio Vandin

December 10, 2020

Decision Tree

Decision tree = hypothesis/model $h : \mathcal{X} \to \mathcal{Y}$ that predicts the target value $h(\mathbf{x})$ for an instance $\mathbf{x} \in \mathcal{X}$ using a *tree*.

- Prediction obtained starting from the root and traveling to a leaf of the tree.
- At each node, the successor (child) is chosen on the basis of a splitting of the input space
- Usually: splitting based on one of the features of $\mathbf{x}$

Example (for Classification)

$\mathcal{Y} = \{0, 1\}$; leaf $=$ predicted label

Geometric Interpretation

Expressiveness of Decision Trees

Assume $\mathcal{X} = \{0,1\}^d \Rightarrow$ rules in each node are of the type "$x_i = 1$" for some $i \in \{1,\ldots,d\}$

Proposition

Let $\mathcal{X} = \{0,1\}^d$, and let f be any function from $\mathcal{X}$ to $\{0,1\}$. Then the set of hypothesis $\mathcal{H} = \{\text{decision tree from } \{0,1\}^d \text{ to } \{0,1\}\}$ contains a tree h that for each $\mathbf{x} \in \mathcal{X}$: $h(\mathbf{x}) = f(\mathbf{x})$.

Sample Complexity of Decision Trees

Corollary

The VC-dimension of $\mathcal{H} = \{\text{decision tree from } \{0,1\}^d \text{ to } \{0,1\}\}$ is 2^d.

A large number of samples is required to learn decision trees!

In practice: need to limit the *complexity* of the tree.

How? Various options

- fix a maximum depth for the tree;
- fix a minimum number of training samples "corresponding" to a leaf
- instead of looking for the tree in $\mathcal{H}$ that minimizes the training error $L_S(h)$ (S=training set), look for the set that minimizes $L_S(h) + R(h)$, where $R(h)$ is some measure of complexity for the tree h (similar to regularization...)

Algorithm to Learn A Decision Tree

How do we find the best decision tree?

Informally: computing the "best" decision tree is NP-hard

$\Rightarrow$ in practice: greedy (non-optimal) approaches are used

Greedy Approach to Learn Decision Trees

Assume binary classification and binary features.

Training set S

Overall scheme:

- start with a tree with a single leaf (the root); label of this leaf = majority vote among all labels over the training set.
- perform a series of iterations, for each iteration
  - examine the effect of splitting a single leaf;
  - define some "gain" measure that quantifies the improvement due to the split
  - among all possible splits, either choose the one that maximizes the gain and perform it, or choose not to split the leaf at all.

Note: very easy to include constraints on the "complexity" of the tree (e.g.,build trees with depth $<$ some value)

$\texttt{ID3}(S, A)$

INPUT: training set S, feature subset $A \subseteq [d]$

if all examples in S are labeled by 1, return a leaf 1

if all examples in S are labeled by 0, return a leaf 0

if $A = \emptyset$, return a leaf whose value = majority of labels in S

else :

Let $j = \text{argmax}_{i \in A} \texttt{Gain}(S, i)$

if all examples in S have the same value of x_j:

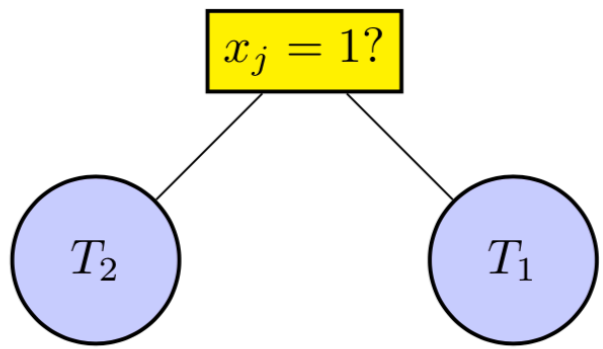
Return a leaf whose value = majority of labels in S

else

Let T_1 be the tree returned by $\texttt{ID3}(\{(\mathbf{x}, y) \in S : x_j = 1\}, A \setminus \{j\})$.

Let T_2 be the tree returned by $\texttt{ID3}(\{(\mathbf{x}, y) \in S : x_j = 0\}, A \setminus \{j\})$.

Return the tree:



Gain Measures

There are several definitions, we consider the simplest one: **decrease in training error**.

Define $C(a) = \min\{a, 1 - a\}$.

Let $\Pr_S[y = 1]$ be the probability that a sample (uniformly at random) from S has label 1.

Then the training error *before splitting* the data S is $C(\Pr_S[y = 1])$.

Why? Because of the *majority of labels* rule.

The training error *after splitting* the data S using feature x_i is:

$$\Pr_S[x_i = 1]C(\Pr_S[y = 1|x_i = 1]) + \Pr_S[x_i = 0]C(\Pr_S[y = 1|x_i = 0])$$

Gain using feature x_i for splitting: difference in error before and after splitting

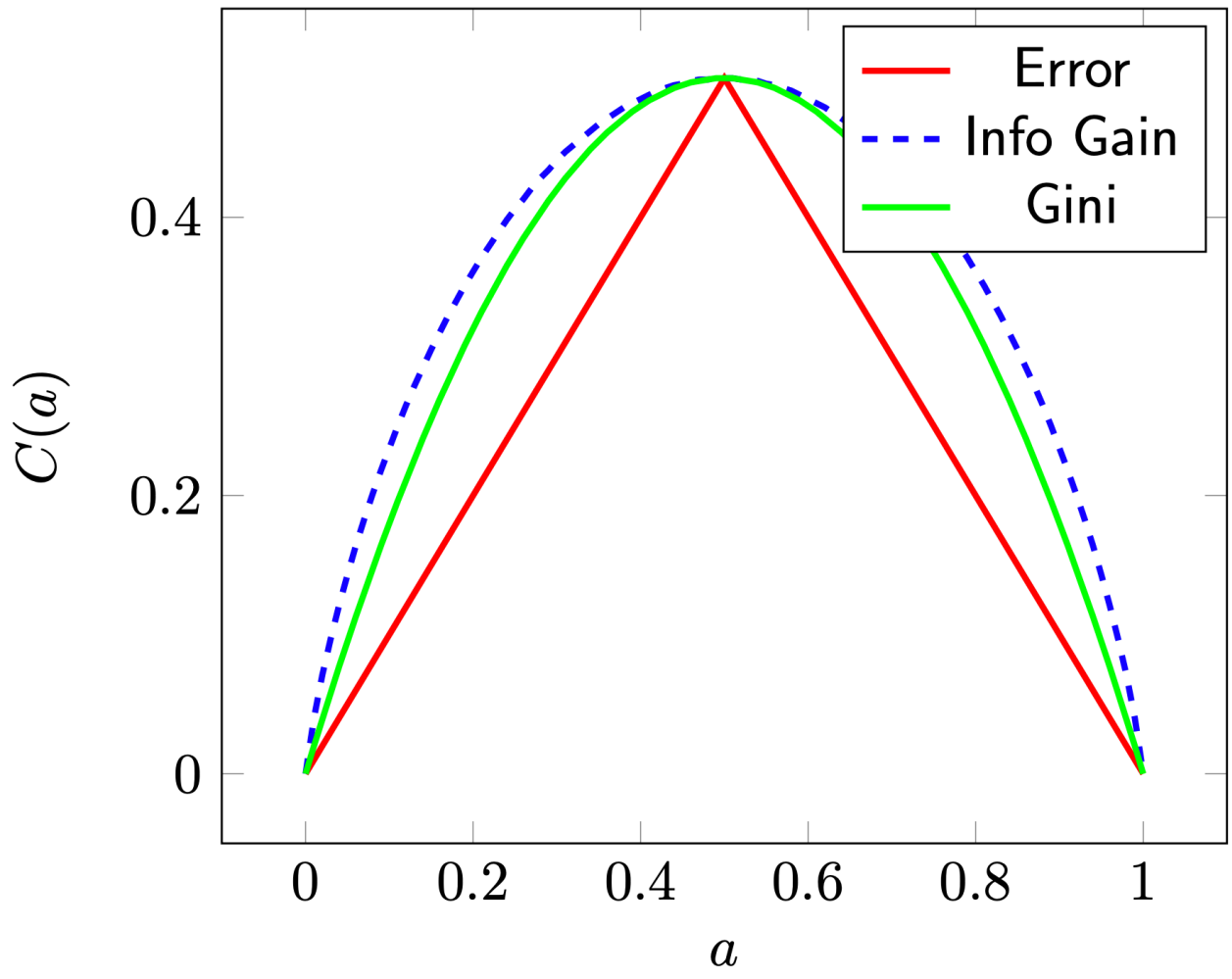
$$Gain(S, i) = C(\Pr_S[y = 1]) -$$

$$\left(\Pr_S[x_i = 1]C(\Pr_S[y = 1|x_i = 1]) + \Pr_S[x_i = 0]C(\Pr_S[y = 1|x_i = 0])\right)$$

Other Gain Measures

Obtained by different definition of $C(a)$:

- **Information gain**: $C(a) = -a \log a - (1-a) \log(1-a)$
- **Gini index**: $C(a) = 2a(1-a)$



What About Real-valued Features?

Consider the i feature that takes real value ($x_i \in \mathbb{R}$).

Assume we have "sorted" the training set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ so that for the i-th feature we have $x_{1,i} \leq x_{2,i} \leq \cdots \leq x_{m,i}$.

Define thresholds $\theta_{0,i}, \theta_{1,i}, \ldots, \theta_{m+1,i}$ with

- $\theta_{0,i} \in [-\infty, x_{1,i}]$
- $\theta_{j,i} \in [x_{j,i}, x_{j+1,i}]$ for all $j = 1, \ldots, m$
- $\theta_{m+1,i} \in [x_{m,i}, \infty]$

For every pair i, j, $i = 1, \ldots, m$, $j = 0, \ldots, m+1$ and every sample $\mathbf{x} \in S$, consider the sample $\mathbf{x}'$ having features $x'_{i,j} = 1$ if $x_i < \theta_{j,i}$, and $x'_{i,j} = 0$ otherwise.

Run the algorithm for binary features on the new samples $\mathbf{x}'$.

Additional Notes

- in practice, *pruning* procedures are often applied after a tree is built to simplify the tree
- there are other algorithms to learn trees, with similar approaches (e.g., scikit-learn uses an implementation of CART)
- what about regression?
  - squared loss is often used
  - prediction of a leaf = average value of target for associated samples
  - error on a set of samples = mean squared error
- decision trees are *interpretable* models

Random Forest

A different approach to prevent overfitting for decision trees.

It is an *ensemble method*: builds a *collection* of decision trees from the same data

Prediction:

- classification: majority vote over the predictions of the individual trees.
- regression: average of predictions of the individual trees.

How are the trees generated?

Tree Generation in Random Forest

Let S be the training set of m samples, with d features.

To build one of the trees in the forest:

1. sample a new training set S' of size m using the uniform distribution over S
2. construct a sequence $\mathcal{I}_1, \mathcal{I}_2, \ldots$, where each $\mathcal{I}_t$ is a subset of size k of the d features, generated by sampling uniformly at random the features.
3. use an algorithm to *grow* a decision tree (e.g., `ID3`) using S' where at splitting stage t of the algorithm, the algorithm is restricted to choosing a feature that maximizes Gain from the set $\mathcal{I}_t$

Intution: if k is small, it will be difficult to overfit