27

TREE METHODS

27.1 Preliminaries: entropy concepts

For a more detailed treatment on entropy and information theory, see section 11.14.

27.1.1 Concept of entropy

Definition 27.1.1 (entropy of a random variable).

- *Let X be a discrete random variable taking values $x_k, k = 1, 2...$ with probability mass function*
$$Pr(X = x_k) = p_k, k = 1, 2, ...$$
Then the entropy of X is defined by
$$H(X) = -\sum_{k\geq 1} p_k \log p_k.$$
- *If X is a continuous random variable with pdf $f(x)$, then entropy of X is defined by*
$$H(X) = -\int_{-\infty}^{\infty} f(x) \log f(x) dx.$$

Remark 27.1.1 (entropy, information and probability distribution).

- Entropy is a measure of the uncertainty of a random variable: the larger the value, the uncertainty the random variable is.
- When the random variable is deterministic, the entropy is at the minimum.

Example 27.1.1.

- The entropy of the Gaussian density on $\mathbb{R}$ with mean μ and variance σ^2 is
$$\begin{aligned} H &= -\int_{\mathbb{R}} \frac{1}{\sqrt{2\pi}\sigma} \exp(-1/2((x-\mu)^2/\sigma^2))(-\log(\sqrt{2\pi}\sigma) - 1/2((x-\mu)^2/\sigma^2))dx \\ &= \frac{1}{2} + \log(\sqrt{2\pi}\sigma). \end{aligned}$$
Note that the mean μ does not enter the entropy; therefore the entropy for Gaussian distribution is translational invariant.
- The entropy of the exponential distribution with mean λ and pdf
$$f(x) = \frac{1}{\lambda} \exp(-x/\lambda)$$

is

$$H = -\int_0^\infty \frac{1}{\lambda}\exp(-x/\lambda)(-\log\lambda - x/\lambda)dx = \ln\lambda + 1.$$

Lemma 27.1.1 (basic properties of entropy).

- $H(X) \geq 0$.
- $H(X) = 0$ *if and only if there exists a* x_0 *such that* $P(X = x_0) = 1$.
- *If* X *can take on finite number* n *values, then* $H(X) \leq \log(n)$. $H(X) = \log(n)$ *if and only if* X *is uniformly distributed.*
- *Let* $X_1, X_2, ..., X_n$ *be discrete valued random variables on a common probability space. Then*
$$H(X_1, X_2, ..., X_n) = H(X_1) + \sum_{i=2}^{n} H(X_n|X_1, ..., X_{n-1}).$$
- $H(X) + H(Y) \geq H(X, Y)$*, with equality if and only if* X *and* Y *are independent.*

Proof. (1) Note that every term $\log(p)$ is non-positive, therefore $H(X) \geq 0$. (2) direct verification. (3) Direct verification. (4) It can be showed that $H(X, Y) = H(X|Y) + H(Y)$. (5) $H(X, Y) = H(X|Y) + H(Y) \leq H(X) + H(Y)$ (using chain rule and conditioning entropy). □

Definition 27.1.2 (entropy definition in classification, empirical entropy). *Let* D *be a training data set whose examples are labeled by* K *classes. Let* $C_1, ..., C_K$ *be the corresponding subsets that partitions* D *by the class labels. Then we define the* ***empirical entropy*** *of* D *as*

$$H(D) = H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|}\log\frac{|C_k|}{|D|}.$$

27.1.2 Conditional entropy and mutual information

Definition 27.1.3 (conditional entropy). *Let* X, Y *be two discrete random variables taking values in* $\mathcal{X}, \mathcal{Y}$.

- ***Specific conditional entropy*** $H(X|Y = y)$ *of* X *given* $Y = y$*:*
$$H(X|Y = y) = -\sum_{x\in\mathcal{X}} Pr(X = x|Y = y)\log Pr(X = x|Y = y).$$

- ***Conditional entropy*** $H(X|Y)$ *of* X *given* Y*:*

$$H(X|Y) = \sum_{y \in \mathcal{Y}} Pr(Y = y) H(X|Y = y).$$

Note that in general $H(X|Y) \neq H(Y|X)$.

Definition 27.1.4 (mutual information, information gain). *[1] Consider two discrete random variables* X *and* Y *taking values in* $\mathcal{X}, \mathcal{Y}$*. The* ***mutual information, or information gain)*** *of* X *and* Y *is given by:*

$$I(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X).$$

Lemma 27.1.2 (basic properties). *For two random variables* X *and* Y*, we have*

- $$I(X, Y) = I(Y, X).$$
- $$I(X, Y) \geq 0.$$
- *Particularly,* $I(X, Y) = 0$ *if* X *and* Y *are independent.*

Proof. (1)(2)

$$\begin{aligned} I(X, Y) &= H(X) - H(X|Y) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(p(x)) + \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x|y) p(y) \log(p(x|y)) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(p(x)) + \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(\frac{p(x, y)}{p(y)}) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(\frac{p(x, y)}{p(x) p(y)}) = D_{KL}(p(x, y) || p(x) p(y)) \geq 0. \end{aligned}$$

$$\begin{aligned} I(X, Y) &= H(Y) - H(Y|X) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(p(y)) + \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(y|x) p(x) \log(p(y|x)) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(p(y)) + \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(\frac{p(x, y)}{p(x)}) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(\frac{p(x, y)}{p(x) p(y)}) = D_{KL}(p(x, y) || p(x) p(y)) \geq 0. \end{aligned}$$

where we use the non-negativity property of KL divergence. Here is a brief proof:

$$D_{KL}(P||Q) = -\sum_{x\in\mathcal{X}} P(x)\log\frac{Q(x)}{P(x)} \geq -\log(\sum_{x\in\mathcal{X}}\frac{Q(x)}{P(x)}P(x)) = 0$$

where the fact that $-\log(x)$ is a convex function and Jensen's inequality has been used [Lemma 11.9.1].

(3) When X and Y are independent, we have

$$H(X|Y) = \sum_{v\in\mathcal{Y}}(Y=v)H(X|Y=v) = \sum_{v\in\mathcal{Y}} P(Y=v)H(X) = H(x).$$

□

Definition 27.1.5 (entropy definition in classification, empirical entropy). *Let D be a training data set whose examples are labeled by K classes. Let $C_1, ..., C_K$ be the corresponding subsets that partitions D by the class labels. Then we define the **empirical entropy** of D as*

$$H(D) = H(D) = -\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log\frac{|C_k|}{|D|}.$$

Corollary 27.1.0.1 (conditioning reduce entropy). *Given discrete random variables X and Y, we have*

$$H(X|Y) \leq H(X),$$

which is also known as conditioning reduces entropy (i.e., conditioning provides information); and this equality holds if and only if X and Y are independent.

Definition 27.1.6 (empirical conditional entropy).

- *Let D be a training data set whose examples are labeled by K classes. Let $C_1, ..., C_K$ be the corresponding subsets that partitions D by the class labels.*
- *Let A be one feature that takes n different values $a_1, ..., a_n$. We partition D into n subsets $D_1, ..., D_n$ such that D_i is the subset of D that feature A is taking value a_i.*
- *Further denote D_{ik} as the subset of D_i that belongs to class k.*

*Then we define the **empirical conditional entropy** of D conditioned on A as*

$$H(D|A) = \sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i) = -\sum_{i=1}^{n}\frac{|D_i|}{D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log\frac{|D_{ik}|}{|D_i|}.$$

27.2 Classification tree

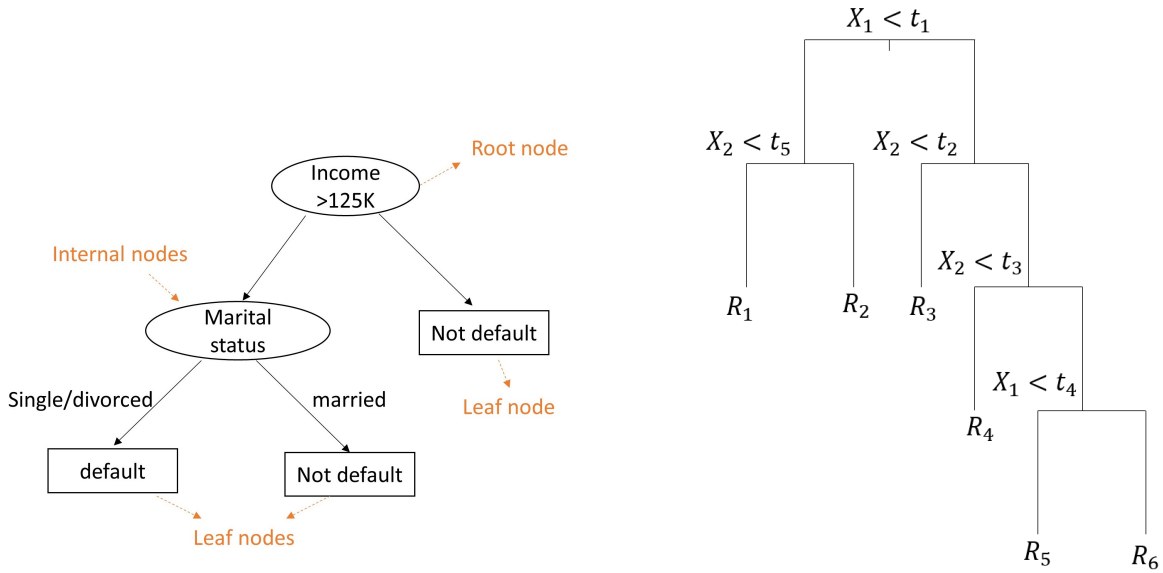
27.2.1 Basic concepts of decision tree learning

Decision tree algorithms generally employ a top-down, iterative approach to construct the desired hypothesis or model. During the construction process, the decision tree algorithm performs a **greedy** search in the hypothesis space, based on some metric on each greedy step (e.g., maximum mutual information gain), to find a hypothesis, i.e., a decision tree that best captures the relationship between features and labels. Particularly, for classification problems, if there is no contradictory examples (i.e. two examples with the same x yet produce different y), there will exist a decision tree that perfectly predicts y based on x.

Figure 27.2.1(a) shows a simple decision tree classifier structure for a customer default behavior classification problem. Input features include income and marital status and labels are *default* vs. *not default*. The decision process starts with the **root node** in the tree, proceeds along edges to children nodes, where data are split at each parent node according to a **splitting rule**. The splitting process proceeds until maximum depth are reached or class labels in the current node are sufficiently pure. These final nodes called **leaf nodes**, where a prediction is made based on the majority of labels in this leaf node.

Figure 27.2.1(b) shows another example of decision tree for regression applications. The input space is 2-dimensional space denoted by x_1 and x_2. The splitting rule is realized performing $x_i < v, i = 1, 2$, where i and v are determined based on some criterion. Samples are split into different children node according to the splitting rule. As the splitting proceeds, the tree will continue to grow until the splitting stops. Finally, leaf nodes will partition the input space, and the predicted value for each partitioned input space are the average of samples in that leaf node.

Decision tree algorithms enjoy marked popularity in industry and academia due to its simplicity and flexibility for mixed data types. Further, resulting decision tree structures often elucidate the classification process, offering substantial model interpretability.



(a) A simple decision tree classifier structure for a customer default behavior classification problem.

(b) An example decision tree for application. The input space is 2 dimensional represented by (x_1, x_2).

Figure 27.2.1: Demonstration of decision trees.

27.2.2 A generic tree-growth algorithm

Here we introduce a generic algorithm [algorithm 38] to recursively construct a tree by splitting, which is known as Hunt's algorithm. The algorithm grows a decision tree in a recursive fashion by partitioning the training examples into successively purer subsets. Let $\mathcal{D}_t$ be the set of training examples that are associated with node t and $y = y1, y2, ..., yc$ be the class labels. The following is a recursive definition of Hunt's algorithm.

- If all the records in $\mathcal{D}_t$ belong to the same class y_t, then t is a leaf node labeled as y_t.
- If $\mathcal{D}_t$ contains records that belong to more than one class, an attribute test condition is selected to partition the records into smaller subsets. A child node is created for each outcome of the test condition and the records in $\mathcal{D}_t$ are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

This recursive tree generation algorithm is summarized in algorithm 38.

Algorithm 38: A generic decision tree generation algorithm

Input: training data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, threshold ϵ, feature set $\mathcal{A}$. We assume y has being K classes and therefore the samples can be divided into $C_1, ..., C_k$.

1 If all samples in D belongs to the same class C_i, label current node as C_i. Stop splitting.

2 If $A = \emptyset$, label label current node as the dominant class in D. Stop splitting.

3 Select one feature $A \in \mathcal{A}$ with some desired splitting properties.

4 Partition D into $D_1, ..., D_n$ based on each possible value a_i of feature A. Label current node with the dominant class in D.

5 For each non-empty set D_i, create a child node.For each i child node Using D_i as the training data and $\mathcal{A} - A$ as the feature set, recursively continue the tree generation process.

Output: the tree model

Remark 27.2.1 (predicting probability). In general, obtaining predicted class probability from decision tree is not as straight-forward as other methods like logistic regression. One simple approach is to proxy the probability via computing class fraction in a leaf node. If we do not limit the maximum depth of a decision tree, there could exist only one class in the leaf node, giving class prediction probability of 1.

27.2.3 Splitting criterion

Virtually all decision tree algorithms involves splitting to growth a tree. Given an input with multiple features and each features taking either multiple discrete values or continuous values, there are countless ways to carry out the splitting and consequently producing the different trees with varying classification performance.

The guiding principle on splitting is that after splitting, samples sitting in left and right children nodes are more homogeneous or pure than when they sit on the parent node. In the following, we introduce several mathematical metrics used to measure the purity or homogeneity of samples. Then we will look at some classification splitting criterion and a typical tree growth procedure employing these metrics and criteria.

Definition 27.2.1 (impurity measures). *Consider a set of items of K classes. Let p_i be the fraction of items labeled with class i. Common **impurity measures** for examples in node are given by*

- $$Entropy = -\sum_{i=1}^{K} p_i \ln p_i.$$
- $$Gini = 1 - \sum_{i=1}^{K} p_i^2.$$
- $$Classifying\ error = 1 - \max(p_1, p_2, ..., p_K).$$

Remark 27.2.2 (interpret Gini impurity). If all examples belong to one class, then $Gini = 0$. And $Gini$ takes the maximum value when $p_i = 1/K$, that is, when the class labels are most impure.

Example 27.2.1. Consider the following computation example:

- A node contains 4 examples with label 1 and 6 examples with label 0. Then

$$\begin{aligned} Gini &= 1 - 0.4^2 - 0.6^2 = 0.48 \\ Entropy &= -0.4\ln(0.4) - 0.6\ln(0.6) = 0.673 \\ Error &= 1 - \max(0.4, 0.6) = 0.4 \end{aligned}$$

- A node contains 9 examples with label 1 and 1 example with label 0. Then

$$\begin{aligned} Gini &= 1 - 0.1^2 - 0.9^2 = 0.18 \\ Entropy &= -0.1\ln(0.1) - 0.9\ln(0.9) = 0.326 \\ Error &= 1 - \max(0.1, 0.9) = 0.1 \end{aligned}$$

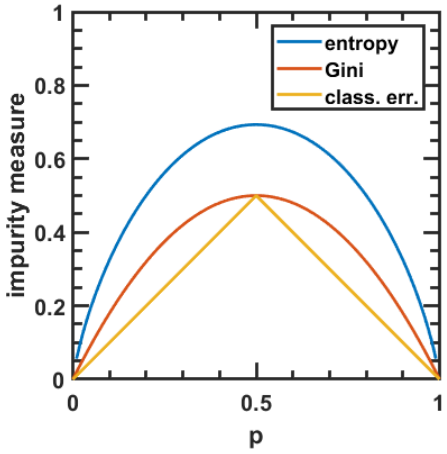


Figure 27.2.2: Different types of impurity measure. Entropy function, Gini function and classification error function.

Methodology 27.2.1 (decision tree classifier splitting methods). *Consider a set of items of K classes. Let p_i be the fraction of items labeled with class i.*

- *Gini impurity decrement :*

$$\Delta G = \frac{N_t}{N}(G_0 - \frac{NR_t}{WNT} \times G_R - \frac{NL_t}{N_t} \times G_L)$$

  where N is the total number of samples, W_t is the total number of samples at the current node, NL_t and NR_t are the total weight of samples in the left child and right child respectively, G_0 is the impurity before splitting, and G_L and G_R are the impurity measure in the left and right child respectively.
- *Information gain*

$$IG = H(T) - H(p|a) = -\sum_{i=1}^{K} p_i \log(p_i) - \sum_a p(a) \sum_{i=1}^{K} (-Pr(i|a))$$

- *Information gain ratio*

$$IGR = \frac{IG}{H(T)}.$$

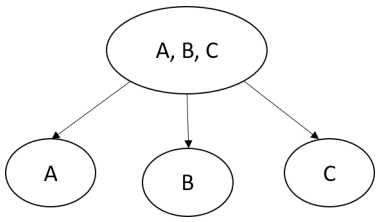
Remark 27.2.3 (the need for information gain ratio)**.** Information gain ratio biases the decision tree against considering attributes with a large number of distinct values. So it solves the drawback of information gain—namely, information gain applied to attributes that can take on a large number of distinct values might learn the training set too well.

For example, suppose that we are building a decision tree for some data describing a business's customers. Information gain is often used to decide which of the attributes are the most relevant, so they can be tested near the root of the tree. One of the input attributes might be the customer's credit card number. This attribute has a high information gain, because it uniquely identifies each customer, but we do not want to include it in the decision tree: deciding how to treat a customer based on their credit card number is unlikely to generalize to customers we haven't seen before.

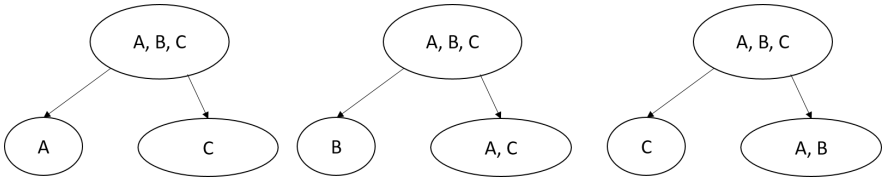
Remark 27.2.4 (classification tree growing stopping criterion). A classification tree can *grow until all the leave node are pure*[1]. Additional parameters controlling the stopping criterion are

- maximum depth: the maximum depth of the tree.
- minimum sample size for splitting: the minimum sample number in a node for splitting.
- information gain threshold: the minimum information gain when splitting a node.
- Gini impurity decreasing threshold: the minimum Gini impurity decreasing when splitting a node.

An addition note on the splitting regarding variables taking more than two values. There could be either a multi-way splitting or two-way splitting. Most algorithms are two-way splitting due to its simplicity[Figure 27.2.3].



(a) Multi-way split in a node on one attribute with three types values.



(b) A recursive tree representation of the partition.

Figure 27.2.3: Different splitting strategy when variables taking more than two discrete values.

1 If the input variables in a classification problems are all categorical and data are consistent, then there exist a classification tree perfectly classify all the training examples.

27.2.4 Tree pruning

Decision tree algorithms usually generate a tree that is too large and overfit the training data. Besides applying tree growth stopping criterion to stop tree growth, we can also prune a tree after its creation, which involves removing tree branches that contribute little to classification and replacing them with leaf nodes.

The following summarizes a generic procedure of pruning a tree.

Methodology 27.2.2 (a generic procedure of pruning a classification/regression tree).

- *Classify examples in the validation set.*
- *From root to leaf for each node*
  - *Sum the errors over entire subtree.*
  - *Calculate the error if the subtree is replaced by a leaf node with major class label (classification) or the mean of the subtree (regression).*
  - *Record the error reduction and replace subtree with leaf node with lowest reduction in error.*
  - *Repeat until reduced error is above a threshold.*

27.2.5 Practical algorithms

Here we briefly discuss several classical algorithms, including ID3 (Iterative Dichotomiser 3)[2], ID4.5 (an improved version of ID3)[3] and CART(Classification And Regression Tree)[4, 5].

Primary difference among these algorithms include splitting criterion, tree growth, stopping criterion, tree pruning, handling of continuous variables, and handling of missing values.

Below we present ID3 algorithm [algorithm 39], which uses information gain as splitting criterion.

Algorithm 39: ID3 classification decision tree algorithm

Input: training data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, threshold ϵ, feature set $\mathcal{A}$. We assume y has being K classes and therefore the samples can be divided into $C_1, ..., C_k$.

1 If all samples in D belongs to the same class C_i, label current node as C_i. Stop splitting.

2 If $A = \emptyset$, label label current node as the dominant class in D. Stop splitting.

3 **foreach** *feature* $A \in \mathcal{A}$ **do**

4 Compute empirical entropy for D [Definition 27.1.2) via

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log \frac{|C_k|}{|D|}$$

where C_k is the count of samples labeled as class k.

5 Compute emprical condtional entropy for D conditioned on A [Definition 27.1.6]:

$$H(D|A) = -\sum_{i=1}^{n} \frac{|D_i|}{D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log \frac{|D_{ik}|}{|D_i|},$$

6 Compute information gain

$$IG(D, A) = H(D) - H(D|A)$$

7 **end**

8 Select the feature A with the maximum information gain.

9 **if** $IG(D, A) < \epsilon$ **then**

10 Stop splitting. Label current node with the dominant class in D.

11 **else**

12 Partition D into $D_1, ..., D_n$ based on each possible value a_i of feature A. Lebel current node with the dominant class in D.

13 For each non-empty set D_i, create a child node.For each i child node Using D_i as the training data and $\mathcal{A} - A$ as the feature set

Output: the tree model

Because ID3 algorithm only has tree growth but not tree pruning, ID3 algorithm can easily lead to overfitting. Hyperparameters controlling the tree growth process are listed as follows.

Remark 27.2.5 (hyper-parameter tuning).

- **max_depth** specifies the depth of each tree in the forest. The deeper the trees, the more splits they have, and more complex models will be generated to achieve better fitting to the training data. However, increasing the depth of tree will also be likely to cause over-fitting.
- **min_samples_split** specifies the minimum number of samples required in a node before splitting. Increasing the number will limit the depth, and thus the complexity, of the tree; decreasing the number might lead to over-fitting.
- **min_samples_leaf** specifies the minimum number of samples required to construct a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. Increasing the number will limit the depth, and thus the complexity, of the tree.
- **max_feature** specifies the number of features to be consider during the search of best splits. The best practice for classification problem is $\sqrt(2)$ of the total number of features.

Improvement in ID4.5 includes the usage of information gain ratio, handling missing values, and tree pruning. Besides ID3 and ID4.5, CART is another very popular decision tree algorithm. CART uses the Gini impurity measure to specify splitting rules. Particularly, a weighted version of impurity reduction is used as follows:

$$\frac{W_t}{W}(G_0 - \frac{WR_t}{W_T} \times G_R - \frac{WL_t}{W_t} \times G_L)$$

where W is the total weight of samples, W_t is the total weight of samples at the current node, WL_t and WR_t are the total weight of samples in the left child and right child respectively, G_0 is the impurity before splitting, and G_L and G_R are the impurity measure in the left and right child respectively.

The weighted splitting rule is particularly useful in ensemble learning via Adaboost or learning tasks with imbalanced data, where we usually want to associate weight to different samples to make the learning algorithm place more effort on some classes.

Remark 27.2.6 (computational time complexity). Suppose there are N training examples, d features, and the final tree has H levels. For each level, the algorithm has to go through N examples to look for the split point. Since each split point requires going through a subset of examples and d features, each level will cost $O(Nd)$ in total (assuming binary features). Since there are H levels, total cost will be $O(NdH)$.

In the optimal case where we get a balanced tree, the level $H = O(\log N)$; in the worst case where we get a extremely skewed tree, the level $H = O(n)$.

On the prediction stage, the computational time cost for each example is $O(H)$ since we need to evaluate H splits.

Finally, note that if features are continuous variables, we need to sort examples by the feature values and then perform a left to right linear scan to determine the split point. For each level, the total cost is $O(dN \log N)$ for each level, where $O(N \log N)$ is the sorting cost.

27.2.6 Examples

27.2.6.1 *Tree structures in Iris data classification*

To have more concrete understanding on the tree growth algorithm [algorithm 39], here we present examples when hyperparameters and splitting rules are used differently.

- Figure 27.2.4 shows a tree is grown until all examples are classified for the Iris data set. We use Gini impurity decrement splitting rule.
- Figure 27.2.5 shows a tree is grown until until examples in each node is smaller than 10. We use Gini impurity decrement splitting rule.
- Figure 27.2.6 shows a tree is grown until until examples in each node is smaller than 10. We use information gain splitting rule.

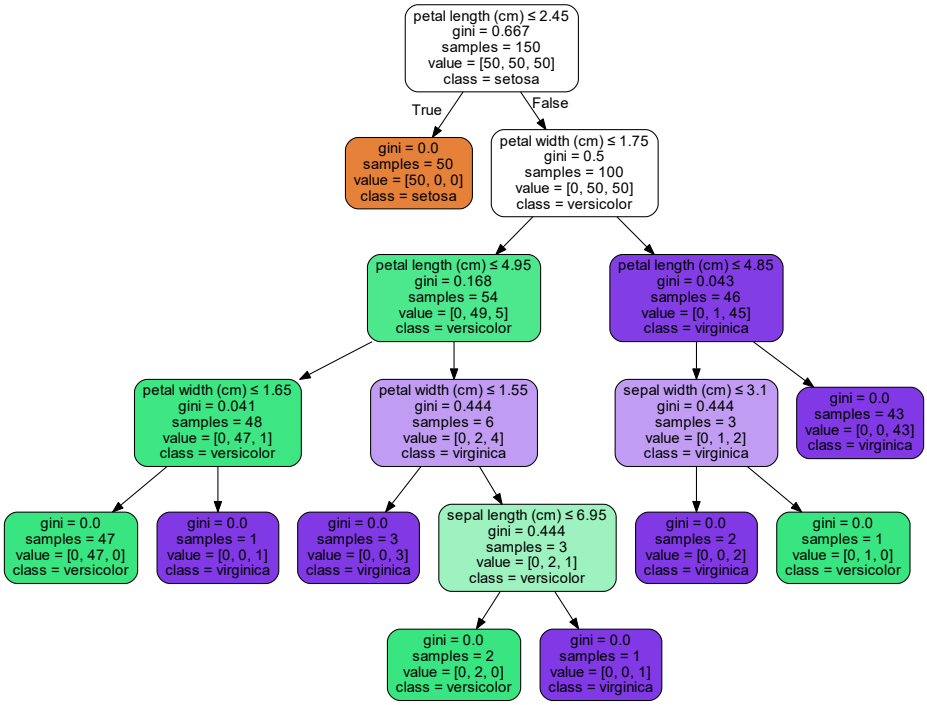


Figure 27.2.4: The visualization of the decision tree classifier for Iris data set. The tree grows until all examples are classified correctly. The splitting criterion is Gini impurity.

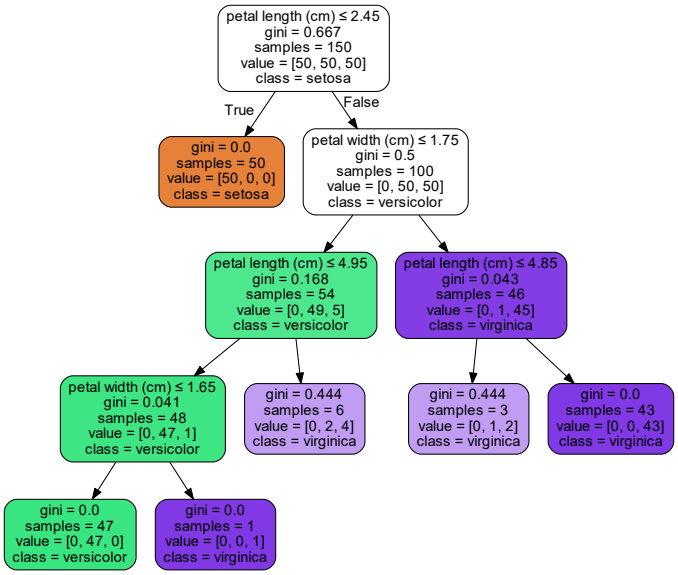


Figure 27.2.5: The visualization of the decision tree classifier for Iris data set. The tree grows until examples in each node is smaller than 10. The splitting criterion is Gini impurity.

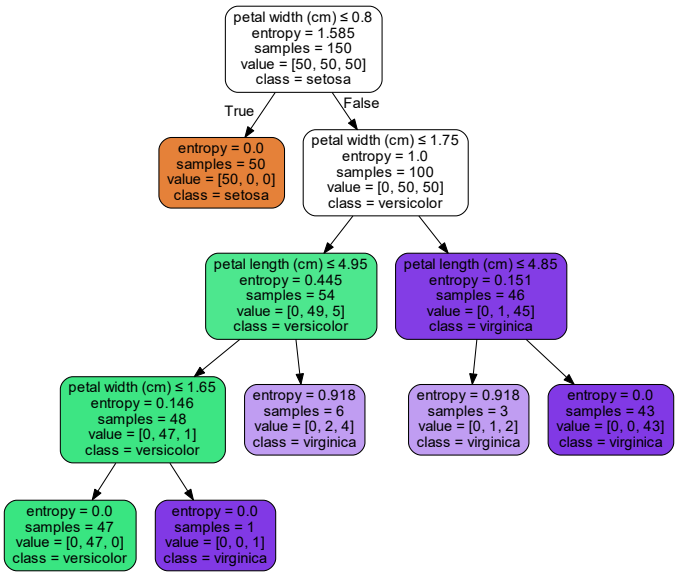


Figure 27.2.6: The visualization of the decision tree classifier for Iris data set. The tree grows until examples in each node is smaller than 10. The splitting criterion is entropy and information gain.

27.3 Regression tree

27.3.1 Basics

Regression trees differ from classification trees only in their output. Regression trees have numerical values in the leaf nodes while classification trees have class label in their leaf nodes. This diction also requires a different splitting rule at each node. Regression trees generally use variance reduction as the splitting rule, as stated as follows.

Methodology 27.3.1 (tree splitting via variance reduction). *Consider examples with input features of $X_1, ..., X_p$ and output $Y \in \mathbb{R}$ We want to select the j feature and a value s, such that for the pair of half-planes:*

$$R_1(j,s) = \{X|X_j < s\}, R_2(j,s) = \{X|X_j \geq s\},$$

the equation

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

is minimized. In here, $\hat{y}_{R_1}$ is the mean response for the training examples in R_1 and $\hat{y}_{R_2}$ is the mean response for the training examples in R_2.

Example 27.3.1. Consider the following training samples

x_i	1	2	3	4	5	6	7	8	9	10
y_i	5.56	5.70	5.91	6.40	6.80	7.05	8.90	8.70	9.00	9.05

To get the split point s such that we have partitions on $\mathbb{R}$ given by

$$R_1 \in \{x|x \geq s\}, R_2 = \{x > s\},$$

we iterate over split point candidates of $\{1.5, 2.5, 3.5, ..., 9.5\}$, such that

$$m(s) = \sum_{x_i \in R_1} (y_i - c_1)^2 + \sum_{x_i \in R_2} (y_i - c_2)^2,$$

where

$$c_1 = \frac{1}{N_1} \sum_{x_i \in R_1} y_i, c_2 = \frac{1}{N_2} \sum_{x_i \in R_2} y_i.$$

Then we can find when $s = 6.5$, $m(s)$ has a minimum value; that is $s = 6.5$ is selected as the first split point.

In every node, we loop through every feature X_i, to choose optimal s based on two situations:

- If X_i is discrete variable(e.g., 0 and 1), then we simply loop through the discrete values to select the best one.
- If X_i is a continuous variable, then we can first sort the training examples based on X_i, and test all possible splits using all values for variable X_i to select the optimal value.[2]

The constructed regression tree amounts to a model of the form

$$f(X) = \sum_{m=1}^{M} c_m 1(X \in R_m)$$

where $R_1, ..., R_m$ represent a partition of the feature space $\mathcal{X}$[Figure 27.3.1b].

To present the tree from growing too big and leading to overfitting. Typical stopping criteria [Remark 27.2.4] include maximum depth, minimum sample size in a leaf node, decrement amount of variance reduction, etc.

In addition, there are interesting relationship between splits, depths, and the number of output levels:

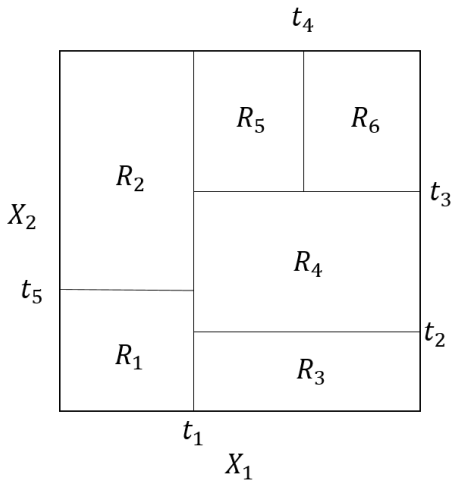
- Let the number of splits be S, then the number of output levels M is given by

$$M = S + 1.$$

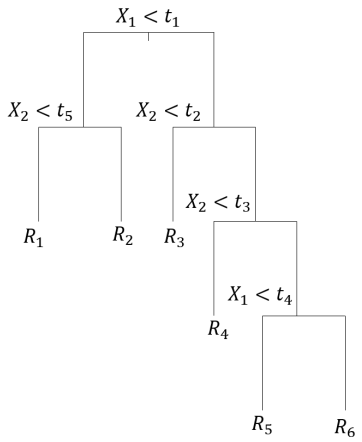
- Let the number of depth be M, then the number of output levels M is bounded by

$$D + 1 \leq M \leq 2^D.$$

2 Depending on the number of examples, more efficient splitting based on sorting, binary search, etc. will be considered.



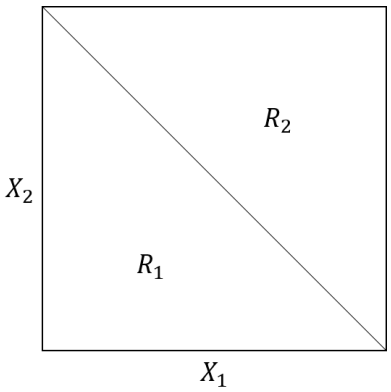
(a) Demonstration of a partition of a 2D input space.



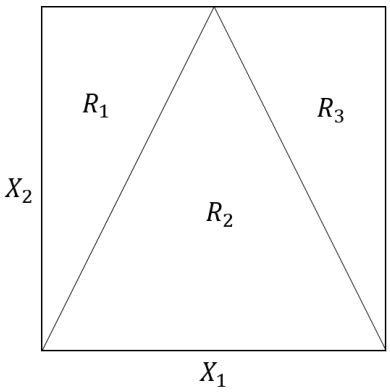
(b) A recursive tree representation of the partition.

Figure 27.3.1: Demonstration of a tree and input space partitioning.

Because input space partitioning is carried out in an orthogonal manner, regression trees will have poorer performance if a more desired partitioning is non-orthogonal, as showed in Figure 27.3.2b. One quick way to fix is to transform the original input space by rotation.



(a) A partition of 2D input space that cannot be represented by a regression tree. However, it can be represented after a linear transformation of the input space.



(b) A partition of 2D input space that cannot be represented by a regression tree. No linear transformation can make it perfectly represented by a regression tree.

Figure 27.3.2: 2D input space partitions cannot be represented by a regression tree.

27.3.2 Practical algorithms

We now present a complete algorithms for regression tree growth [algorithm 40]. Note that typical stopping criteria [Remark 27.2.4] include maximum depth, minimum sample size in a leaf node, decrement amount of variance reduction, etc.

Algorithm 40: A regression tree growth algorithm

Input: training data set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, threshold ϵ, feature set $\mathcal{A}$. We assume $y \in \mathbb{R}$.

repeat
  foreach *feature* $X_j \in \mathcal{A}$ **do**
    select the best splitting point s in X_j that minimizes

$$R_1 = \{X | X_j < s\}, R_2 = \{X | X_j \geq s\},$$

    the equation

$$\sum_{i:x_i \in R_1} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2} (y_i - \hat{y}_{R_2})^2$$

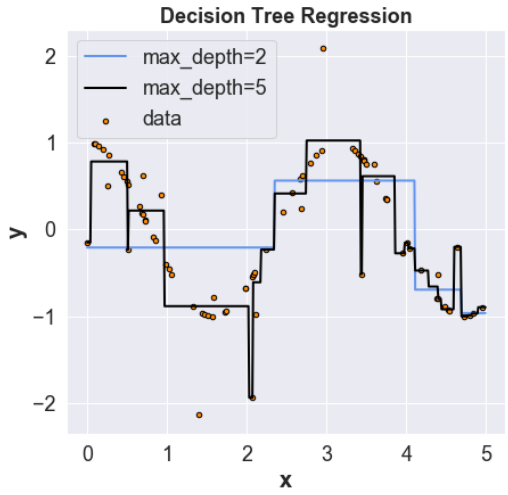
    is minimized. In here, $\hat{y}_{R_1}$ is the mean response for the training observations in R_1 and $\hat{y}_{R_2}$ is the mean response for the training observations in R_2.
  end
  Select the feature X_j with the minimum variance and perform splitting.
  Move the children node.
until *stopping condition satisfied*;
Label all leaf node with the mean value of y.

Output: the tree model

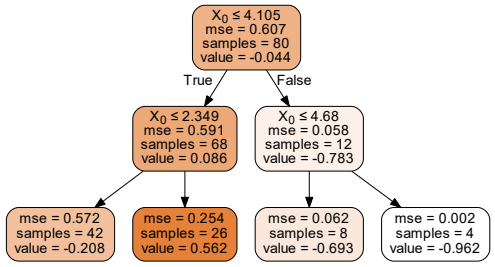
27.3.3 Examples

27.3.3.1 *A toy example*

We first consider a toy example where we use a regression tree to fit samples generated by a simple function $y = cos(2x)$ plus large deviations at some points. A decision tree with maximum depth of 2 tends to underfit the data, whereas a tree with maximum depth of 5 will overfit to noisy points [Figure 27.3.3(a)]. The diagram with decision logic is showed in Figure 27.3.3(b)



(a) Decision tree regression for training examples generated by $y = \cos(2x)$ plus large deviations at some points. Decision trees used have maximum depths of 2 and 5.



(b) Rendering of the regression tree with maximum depth of 2.

Figure 27.3.3: Regression tree demonstration in a toy example.

27.3.3.2 *Boston Housing prices*

For another application, we applied regression tree to Boston housing pricing prediction problem and achieved RMSE of 6.82. We can further examine feature importance by the extent of variance reduction [Figure 27.3.4].

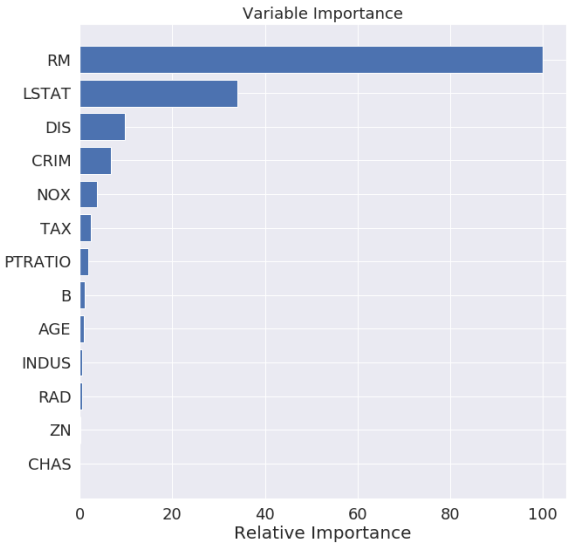


Figure 27.3.4: Variable importance from regression tree in the Boston Housing Pricing problem.

BIBLIOGRAPHY

1. Cover, T. M. & Thomas, J. A. *Elements of information theory* (John Wiley & Sons, 2012).
2. Quinlan, J. R. Induction of decision trees. *Machine learning* **1,** 81–106 (1986).
3. Quinlan, J. R. *C4. 5: programs for machine learning* (Elsevier, 2014).
4. Loh, W.-Y. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **1,** 14–23 (2011).
5. Breiman, L. *Classification and regression trees* (Routledge, 2017).