# STAT 479: Machine Learning
# Lecture Notes

Sebastian Raschka
Department of Statistics
University of Wisconsin–Madison

http://stat.wisc.edu/~sraschka/teaching/stat479-fs2018/

Fall 2018

## Contents

# STAT 479: Machine Learning
# Lecture Notes

Sebastian Raschka
Department of Statistics
University of Wisconsin–Madison

http://stat.wisc.edu/~sraschka/teaching/stat479-fs2018/

Fall 2018

## 6.1 Introduction

- Decision tree algorithms can be considered as an iterative, top-down construction of the hypothesis; picture a hierarchy of decision, forking the dataset into subspaces.
- Can represent any Boolean (binary) function, and the hypothesis space being searched is the entire space of Boolean functions [1]; however, we need to keep in mind that a critical challenge in machine learning is whether an algorithm can learn/find the "right" function or a good approximation.
- Considering only binary (or Boolean) features, at each node, there $2^m$ potential splits to be evaluated given $m$ features.
- Hypothesis space is searched greedily (i.e., decision tree algorithms perform a greedy search[2] over all possible trees); an exhaustive search is not feasible because of its exponential nature of the problem ($2^m$ potential splits to evaluate given $m$ features).

### 6.1.1 Terminology

- Root node: no incoming edge, zero or more outgoing edges.
- Internal node: one incoming edge, two (or more) outgoing edges.
- Leaf node: each leaf node is an assigned a class label (if nodes are pure; otherwise majority vote).
- Parent and child nodes: If a node is split, we refer to that given node as the parent node, and the resulting nodes are called child nodes, respectively.

[1] the size of the hypothesis space determined by the dataset
[2] In greedy search, we make a series of locally optimal choices, which may not lead to the global optimum

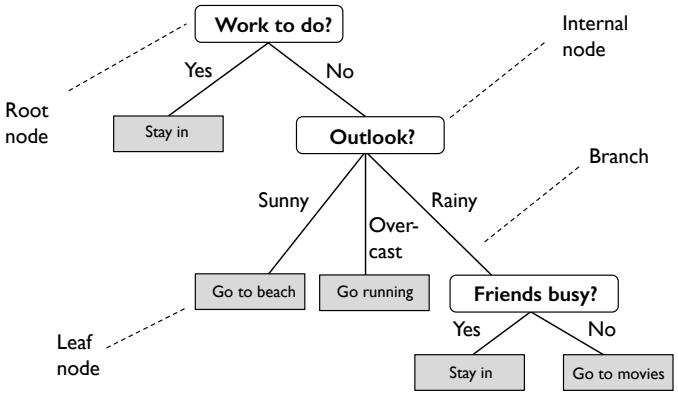


**Figure 1:** Example of a non-binary decision tree with categorical features.

### 6.1.2 Machine Learning Categories

In the context of the different categories of machine learning algorithms that we defined at the beginning of this course, we may categorize decision trees as follows:

- Supervised learning algorithm: classification and regression
- Optimization method: combinatorial -> greedy search
- Eager learning algorithm
- Batch learning algorithm
- Nonparametric model
- Deterministic (vs. stochastic)

### 6.1.3 Relationship Between Decision Trees and Rule-based Learning

Intuitively, we can also think of decision tree as nested "if-else" rules. And a rule is simply a conjunction of conditions. For example,

$$\text{Rule 1} = (\text{if } x = 1) \ \cap \ (\text{if } y = 2) \ \cap \ ... \tag{1}$$

Multiple rules can then be joined into a set of rules, which can be applied to predict the target value of a training example or test instance. For example,

$$\text{Class 1} = \text{if } \ (\text{Rule 1=True}) \ \cup \ (\text{Rule 2=True}) \ \cup ... \tag{2}$$

Each leaf node in a decision tree represents such a set of rules as illustrated in the following figure, which depicts the rule,

$$(\text{Work to do? = False}) \cap (\text{Outlook? = Rainy}) \cap (\text{Friends busy? = Yes}) \tag{3}$$

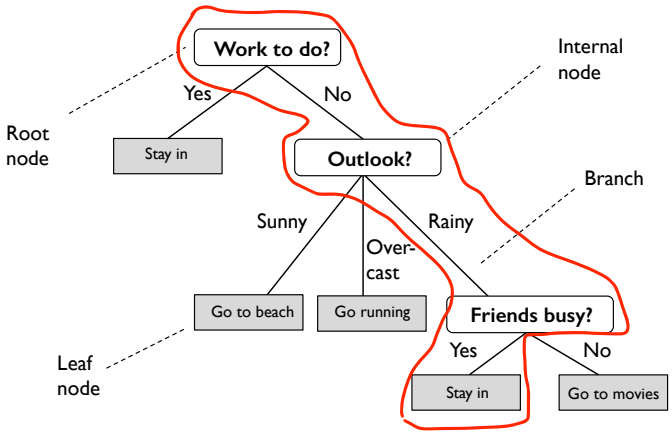


**Figure 2:** A rule for a given leaf node (circled): (Work to do? = False) ∩ (Outlook? = Rainy) ∩ (Friends busy? = Yes)

Considering the complete tree depicted in the previous figure, the decision rule for the class label "Stay In" can then be written as the following rule set:

$$(\text{Work to do? = False}) \cap (\text{Outlook = Rainy?}) \cap (\text{Friends busy? = Yes}) \cup (\text{Work to do? = True}) \tag{4}$$

- Rules can be constructed from decision trees easily: each leaf node is a rule.
- It is not possible to always build a decision tree from a set of rules, and in cases where it is obvious, it may not be immediately apparent how (especially if rules were pruned, e.g., where would you place the root of the tree?).
- Evaluating a rule set is much more expensive than evaluating a tree, where we only have to go to one single branch.
- Rulesets can have multiple answers if we are not careful.
- While rules are more expressive or flexible, they are more prone to overfitting, because they have a larger hypothesis space than decision trees.

## 6.2 Divide-and-Conquer and Recursion

- In the context of decision trees and how to implement them efficiently, it is helpful to visit the topic of divide-and-conquer algorithms and the concept of recursion.
- To understand the basic concept behind recursion, consider the simple algorithm (here: written as a Python function) to compute the length of an array:

```
def array_len(x):
    if x == []:
        return 0
    else:
        return 1 + array_len(x[1:])
```

- Divide-and-conquer is a concept in computer science where we divide a problem into subproblems of the same type.

- Usually, divide-and-conquer can be implemented using recursion, and an example is shown below, implementing a simple version of the quicksort algorithm (in Python):

```
def quicksort(array):
    if len(array) < 2:
        return array
    else:
        pivot = array[0]
        smaller, bigger = [], []
        for ele in array[1:]:
            if ele <= pivot:
                smaller.append(ele)
            else:
                bigger.append(ele)
        return quicksort(smaller) + [pivot] + quicksort(bigger)
```

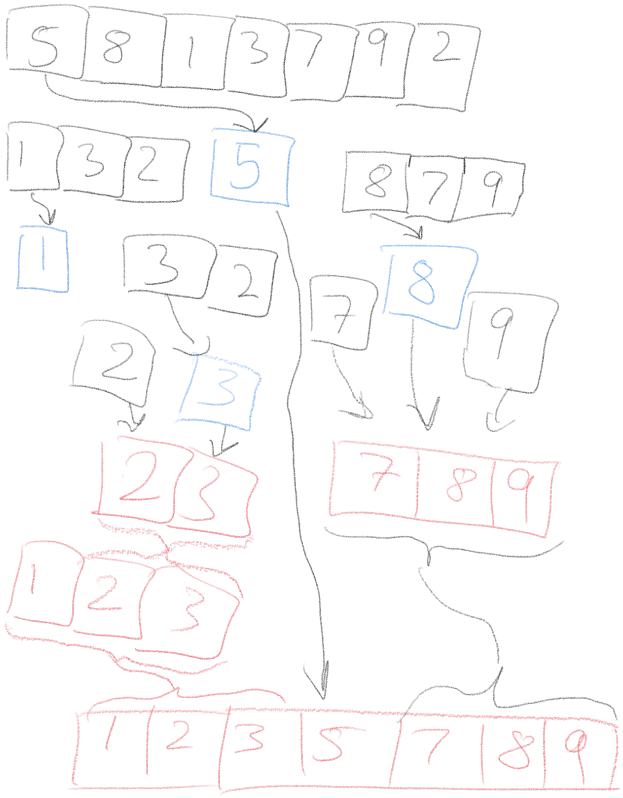

**Figure 3:** Illustration of the different steps of quicksort when applied to a simple toy dataset.

## 6.3 General Decision Tree Algorithm

In this section outlines a generic decision tree algorithm using the concept of recursion outlined in the previous section, which is the basis for most decision tree algorithms described in the literature.

The process of growing a decision tree can be expressed as a recursive algorithm as follows:

1) Pick the feature that when parent node[3] is split, it results in the largest information gain[4]
2) Stop if child nodes are pure or no improvement in class purity can be made
3) Go back to step 1 for each of the two child nodes

Below is a more formal expression of the algorithm outlined above:

GenerateTree($\mathcal{D}$):

- if $y = 1 \; \forall \; \langle \mathbf{x}, \mathbf{y} \rangle \in \mathcal{D}$ or $y = 0 \; \forall \; \langle \mathbf{x}, y \rangle \in \mathcal{D}$ :
  - return Tree
- else:
  - Pick best feature $x_j$:
    - $\mathcal{D}_0$ at Child$_0$ : $x_j = 0 \; \forall \; \langle \mathbf{x}, y \rangle \in \mathcal{D}$
    - $\mathcal{D}_1$ at Child$_1$ : $x_j = 1 \; \forall \; \langle \mathbf{x}, y \rangle \in \mathcal{D}$

    return Node($x_j$, GenerateTree($\mathcal{D}_0$), GenerateTree($\mathcal{D}_1$))

Now, while the algorithm above appears to be a viable approach to construct a decision tree, in practice, we may face several edge cases and issues that we need to think of when implementing decision tree algorithms.

For instance, some of the design choices and considerations we have to make

- How do we decide which feature to select for splitting a parent node into child nodes? I.e., what is a criterion to measure the goodness of the split?
- Since a multi-category splitting can be expressed as a series of binary splits, which approach is to be preferred?
- While splitting categorical features is intuitive, how can we deal with continuous inputs?
- When do we stop growing a tree (because complete separation can easily lead to overfitting)?
- How do we make predictions if no attributes exist to perfectly separate non-pure nodes further?[5]

## 6.4 Time Complexity

Measuring the time complexity of decision tree algorithms can be complicated and is not immediately apparent.

However, under the assumption a decision tree is a balanced binary decision tree, the final tree will have a depth of $\log_2 n$, where $n$ is the number of examples in the training set.

[3] in first iteration, this is the root node
[4] Stop if information gain is not positive.
[5] Majority voting for classification trees and the sample mean for regression trees is typically a good choice.

Hence, it should be immediately obvious that the time complexity for the prediction step is $\mathcal{O}(\log n)$.[6]

Determining the runtime complexity of decision tree training is less straightforward and varies wildly based on the algorithm choice and implementation. Assuming we have continuous features and perform binary splits, the runtime of the decision tree construction is generally $\mathcal{O}(m \cdot n^2 \log n)$. It can be shown that optimal binary split on continuous features is on the boundary between adjacent examples[7] with different class labels[8]. This means that sorting the values of continuous features helps with determining a decision threshold efficiently. If we have $n$ examples, the sorting has time complexity $\mathcal{O}(n \log n)$. If we have to compare sort $m$ features, this becomes $\mathcal{O}(m \cdot n \log n)$.

To see why the time complexity of decision tree construction is typically quoted at $\mathcal{O}(m \cdot n^2 \log n)$, keep in mind that we earlier determined the depth of a decision tree at $\log_2 n$. It follows that the number of "splitting" nodes in the tree is then $2^{\log_2 n} - 1 = n - 1$[9]. Hence, if we are not efficient and re-sort the features prior to each split we have to perform the $\mathcal{O}(m \cdot n \log n)$ sorting step up to $n - 1$ times – once for each splitting node in the tree – which results in a time complexity of $\mathcal{O}(m \cdot n^2 \log n)$.

(Many implementations such as scikit-learn use efficient caching tricks to keep track of the general order of indices at each node such that the features do not need to be re-sorted at each node; hence, the time complexity of these implementations merely is $\mathcal{O}(m \cdot n \log(n))$.)

## 6.5 ID3, C4.5, CART

There exists a relatively large variety of decision tree algorithms. This section lists some of the most influential/popular once.

### 6.5.1 General Differences

Most decision tree algorithms differ in the following ways:

- Splitting criterion: information gain (Shannon Entropy, Gini impurity, misclassification error), use of statistical tests, objective function, etc.
- Binary split vs. multi-way splits
- Discrete vs. continuous variables
- Pre- vs. post-pruning

### 6.5.2 ID3 – Iterative Dichotomizer 3

- Described in Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81-106.
- One of the earlier/earliest decision tree algorithms
- Discrete features, cannot handle numeric features
- Multi-category splits

[6] Remember, we write just $\log n$ because the basis of the log is just a scaling factor, $\log_2 n = \frac{\log_b n}{\log_b 2}$.

[7] Training examples with similar feature values.

[8] Usama Mohammad Fayyad. "On the induction of decision trees for multiple concept learning". In: (1992).

[9] This count includes the root node but excludes the terminal leaf nodes.

- No pruning, prone to overfitting
- Short and wide trees (compared to CART)
- Maximizes information gain/minimizes entropy
- Discrete features, binary and multi-category features

#### 6.5.3 C4.5

- Described in Quinlan, J. R. (1993). C4. 5: Programming for machine learning. *Morgan Kauffmann*, *38*, 48.
- Continuous and discrete features (continuous feature splitting is very expensive because must consider all possible ranges)
- Splitting criterion is computed via the gain ratio (explained later)
- Handles missing attributes (ignores them in information gain computation)
- Performs post-pruning (bottom-up pruning)

#### 6.5.4 CART

- Described in Breiman, L. (1984). *Classification and regression trees*. Belmont, Calif: Wadsworth International Group.
- Continuous and discrete features
- Strictly binary splits (resulting trees are taller compared to ID3 and C4.5)
- Binary splits can generate better trees than C4.5, but tend to be larger and harder to interpret; i.e., for $k$ attributes, we have $2^{k-1} - 1$ ways to create a binary partitioning
- Variance reduction in regression trees
- Uses Gini impurity (or "twoing criteria") in classification trees
- Performs cost-complexity pruning (more on that later)

#### 6.5.5 Others

- CHAID (CHi-squared Automatic Interaction Detector); Kass, G. V. (1980). "An exploratory technique for investigating large quantities of categorical data." *Applied Statistics*. 29 (2): 119–127.
- MARS (Multivariate adaptive regression splines); Friedman, J. H. (1991). "Multivariate Adaptive Regression Splines." *The Annals of Statistics*. 19: 1
- C5.0 (patented)

## 6.6 Information Gain

The standard criterion that is chosen for splitting in decision trees is information gain (the better the split, the higher the information gain).

- Information gain relies on the concept of mutual information: The reduction of the entropy of one variable by knowing the other.[10]
- We want to maximize mutual information when defining splitting criteria.
- I.e., we define the criterion at a note such that it maximizes information gain

$$GAIN(\mathcal{D}, x_j) = H(\mathcal{D}) - \sum_{v \in Values(x_j)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} H(\mathcal{D}_v). \tag{5}$$

Where $\mathcal{D}$ is the training set at the parent node, and $\mathcal{D}_{\sqsubseteq}$ is a dataset at a child node upon splitting.

## 6.7 Information Theory and Entropy

This section briefly summarizes the concept of Entropy as it was coined by Claude Shannon in the context of information theory.[11]

- In ID3, we use Shannon Entropy to measure improvement in a decision tree (instead of misclassification error); i.e., we use it as a optimization metric (or impurity measure)
- This entropy measure was originally proposed in the context of encoding digital information in the form of Bits (0s or 1s).
- Consider entropy as a measure of the amount of information of a discrete random variable (two outcomes, Bernoulli distribution)
- Shannon information:
  - Shannon defined information as the number of bits to encode a number $\frac{1}{p}$, where $p$ is the probability that an event is true (i.e., $\frac{1}{1-p}$ is the uncertainty)
  - The number of bits for encoding $\frac{1}{p}$is $\log_2(1/p)$
  - Note: $\log_2(1/p) = \log_2(1) - \log_2(p) = -\log_2(p)$
  - $-\log_2(p) \rightarrow [\infty, 0]$ ; i.e., if we are 100% certain about an event, we gain 0 information
  - E.g., assume 2 soccer teams, team 1 and team 2, both with a win probability 50%
    - If the information "team 1 wins" is transmitted, we transmitted 1 bit: $\log_2(1/0.5) = -\log_2(2) = 1$
- Shannon entropy is then the "average information"
  - $H(p) = \sum_i p_i \log_2(1/p_i)$

[10] **In this context; knowing class label by knowing feature value.**.

[11] **In this lecture; the use of the term Entropy refers to the Shannon Entropy is not to be confused with other definitions of e such as the one from thermodynamics; although; it's conceptually related if we think about order and disorder.**.

- $H(p) = -\sum_i p_i \log_2(p_i)$
- E.g., assume soccer team 1 and team 2 have win probabilities 75% and 25%, respectively, we get an average information content of 0.81 bits:

$$\begin{aligned} H(p) &= \frac{1}{2}\Big( -0.75 \times \log_2(0.75) - 0.25 \times \log_2(0.25)\Big) \\ &= (-0.75 \times 0.41 - 0.25 \times 2)/2 \\ &= 0.81 \end{aligned} \tag{6}$$

- That means, since we have some information about the problem if the distribution is non-uniform, and we could technically encode this message more efficiently then (consider in a real application, we cannot send fractional bits, but this is just a toy example)
- The key idea behind Shannon entropy is that we can use entropy as a way to create messages of different lengths to transmit different information contents send information more efficiently (saving bits) depending on how confident we are, however, this is beyond the scope of this course.

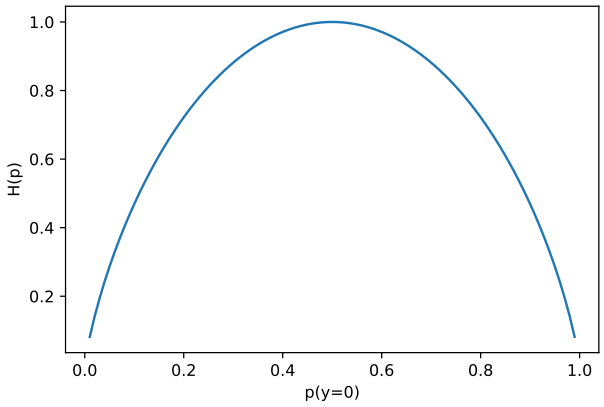


**Figure 4:** Entropy function for a binary classification problem

- If we have $i$ class labels (i.e., $i$ different values for $y$), then the entropy can be as large as $\log_2 i$.
  - E.g., for 10 classes: $10 \times (-(0.1 \times \log_2(0.1))) = \log_2(10) = 3.32$
- In the context of decision trees, think of entropy as the minimum number of bits that are required to encode the classification of data points. For instance, if we have $p = 0.5$ (uniform class distribution in a binary classification problem) we need to send 1 bit on average (most expensive) to classify a data point.

## 6.8 Why Growing Decision Trees via Entropy or Gini Impurity instead of Misclassification Error?

Consider the more general formula for measuring information gain,

$$GAIN(\mathcal{D}, x_j) = I(\mathcal{D}) - \sum_{v \in Values(x_j)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} I(\mathcal{D}_v), \tag{7}$$

where $I$ is a function that measures the impurity of a given node. If $I$ defined as the Entropy measure ($H$) we defined earlier, this equation is equal to the information gain equation used in classification trees.

Instead of using Entropy as an impurity measure, the misclassification error ERR seems to be another reasonable choice, where

$$ERR(\mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} L(\hat{y}^{[i]}, y^{[i]}), \tag{8}$$

with the 0-1 Loss,

$$L(\hat{y}, y) = \begin{cases} 0 \text{ if } \hat{y} = y, \\ 1 \text{ otherwise.} \end{cases} \tag{9}$$

This, in case of the training set, is equal to

$$ERR(p) = 1 - \max_i(p(i|x_j)) \tag{10}$$

for a given node if we use majority voting at this node.

Now, to see the difference between using the misclassification error as opposed to the entropy measure for computing the information gain upon growing a decision tree, we will take a look at an example. Consider the following binary decision tree for a binary classification problem with binary features:

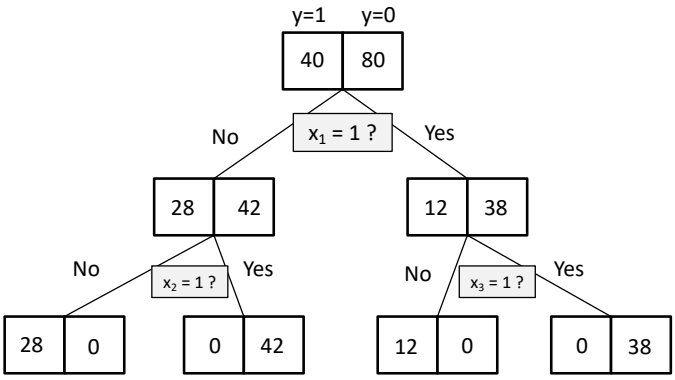


**Figure 5:** decisiontree-error-vs-entropy-1

Note that this is a toy example where we assume that there exist three features $x_1, x_2, x_3 \in \{0, 1\}$ that result in a perfect class separation if split as illustrated in the preceding figure.

We split the tree using entropy as an information or impurity measure first, and the entropy values of the child nodes of the root node are provided in the figure below.

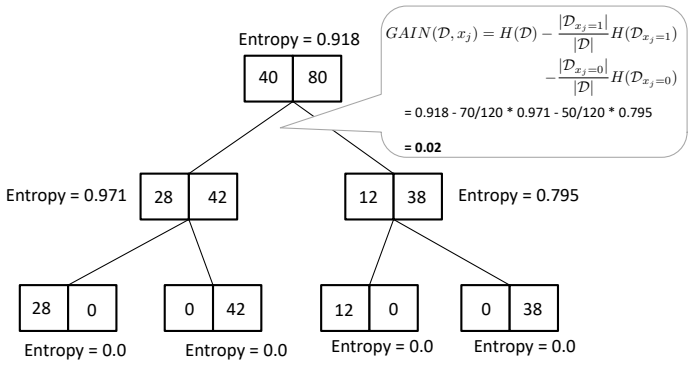


**Figure 6:** decisiontree-error-vs-entropy-3

We note that splitting the root node into the two children nodes results in entropy values 0.971 and 0.795, respectively. The information gain of this split is 0.02. In the toy example, the splits that follow this first split will separate the classes correctly, as indicated in the figure.

Next, consider the same decision tree in a scenario where the misclassification error was used as an impurity metric, as shown in the next figure.

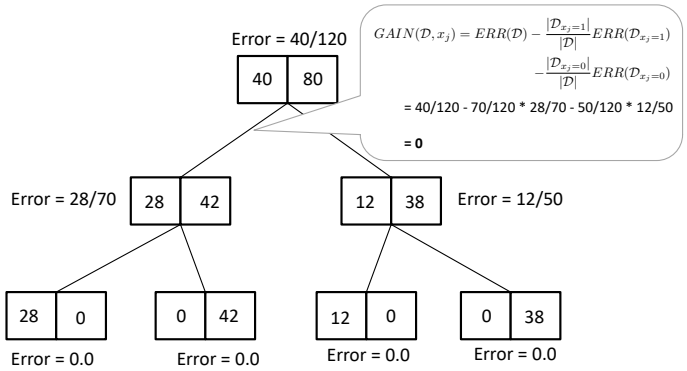


**Figure 7:** decisiontree-error-vs-entropy-2

As shown in the previous figure, the information gain upon splitting the root node using the misclassification error as impurity metric is 0, which means that performing this split actually did not result in an improvement compared to the original state (i.e., the root node). Hence, according to the decision tree algorithm(s) we defined earlier, we would stop growing the tree if we do not make further improvement(s) as measured via information gain.

To provide an intuitive explanation as to why this happens, consider the next figure, which shows the entropy values of the root node plotted along with the two child nodes.

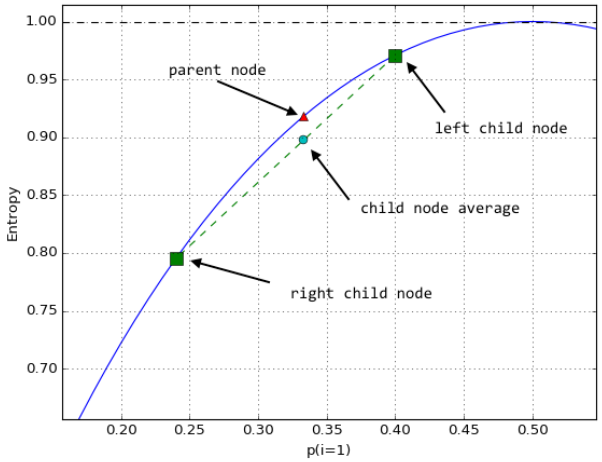


**Figure 8:** entropy_annotated

As it can be seen in the plot above, the weighted entropies of the child nodes is always more substantial than the average of the entropy values due to the convex shape of the entropy function. This means that we can always find a splitting criterion to continue growing the tree where the tree growth might get stuck because the error doesn't improve.

## 6.9 Gini Impurity

- Gini impurity is a measure used in in CART as opposed to entropy:

$$Gini(t) = 1 - \sum_i \left(p(c = i)^2\right) \tag{11}$$

- In practice, whether we use entropy or Gini impurity does not really matter, because both have the same convex/bell shape which is essential.
- Gini is computationally more efficient to compute than entropy (due to the lack of the log), which could make code negligibly more efficient.

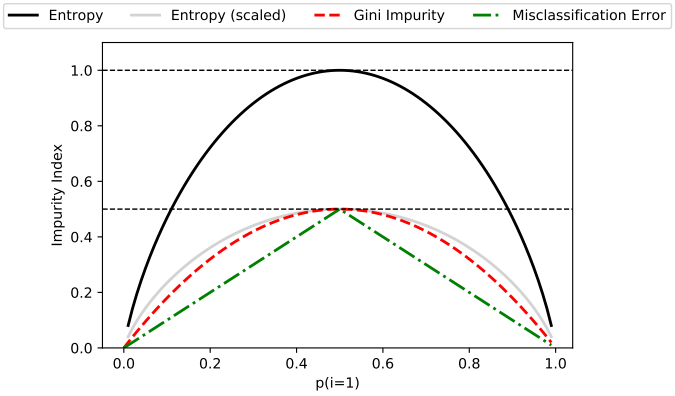


**Figure 9:** Comparison of different impurity measures.

## 6.10 Improvements

This section introduces several ideas to improve the decision tree algorithm.

### 6.10.1 Grain Ratio

The gain ration was introduced by Quinlan penalizes splitting categorical attributes with many values (e.g., think date column, or really bad: row ID) via the split information:

$$GainRatio(\mathcal{D}, x_j) = \frac{Gain(\mathcal{D}, x_j)}{SplitInfo(\mathcal{D}, x_j)}, \tag{12}$$

where SplitInfo measures the entropy of the attribute itself:

$$SplitInfo(\mathcal{D}, x_j) = -\sum_{v \in x_j} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} \log_2 \frac{|\mathcal{D}_v|}{|\mathcal{D}|}. \tag{13}$$

## 6.11 Overfitting

If decision trees are not pruned, they have a high risk to overfit the training data to a high degree.

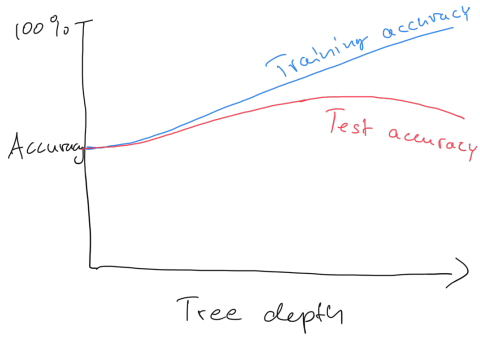


**Figure 10:** Relationship between tree depth and overfitting (gap between training and test accuracy). Note that the plot starts at 50% Accuracy, because of majority voting assuming a binary classification problem.

- Overfitting occurs if models pick up noise or errors in the training dataset; hence, overfitting can be seen as a performance gap between training and test data

  Given a hypothesis space $H$, a hypothesis $h \in H$is said to overfit the training data if there exists some alternative hypothesis $h' \in H$, such that $h$ has smaller error than $h'$ over the training examples, but $h'$ has a smaller error than $h$ over the entire distribution of instances. – Tom Mitchell

- Occam's razor: Favor a simpler hypothesis, because a simpler hypothesis that fits the data equally well (let's say the same accuracy) is more likely or plausible than a complex one.

- A general approach for minimizing overfitting in decision trees is decision tree pruning. There are generally two approaches: post- and pre-pruning.

### 6.11.1 Pre-Pruning

- Set a depth cut-off (maximum tree depth) *a priori*
- Cost-complexity pruning: $I + \alpha|N|$, where $I$ is an impurity measure, $\alpha$ is a tuning parameter, and $|N|$ is the total number of nodes.

- Stop growing if a split is not statistically significant (e.g., $\chi^2$ test)
- Set a minimum number of data points for each node

### 6.11.2 Post-Pruning

- Grow full tree first, then remove nodes (e.g., done in C4.5)
- Reduced-error pruning, remove nodes via validation set evaluation (problematic for limited data)
- Reduced-error pruning: Greedily remove nodes based on validation set performance; generally improves performance but can be problematic for limited data set sizes
- We can also convert trees to rules first and then prune the rules
  - There is one rule per leaf node
  - If rules are not sorted, rule sets are costly to evaluate but more expressive
  - In contrast to pruned rule sets, rules from decision trees are mutually exclusive
  - here we can prune rules independently of others (means we do not remove both child nodes if we remove the root node)

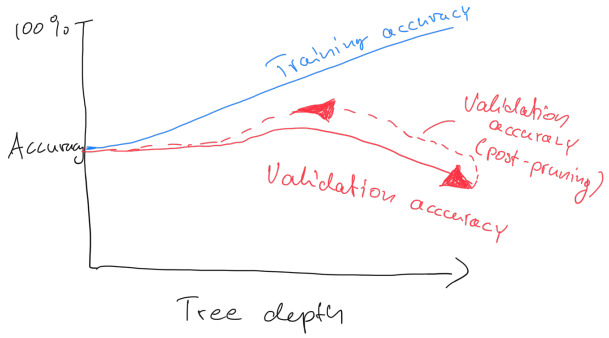


**Figure 11:** Illustration of reduced-error pruning.

## 6.12 Decision Tree for Regression

Decision trees can also be used for regression analysis, which was introduced via CART – as you remember from the previous sections, CART stands for Classification And *Regression* Trees.

If we use decision trees for regression, we grow the tree (that is, deciding upon splitting criteria at each node) through variance reduction at each node. Here, the variance refers to the variance among the target variables at the parent node and its child nodes.

Earlier, in the context of classification, we defined information gain as follows:

$$GAIN(\mathcal{D}, x_j) = I(\mathcal{D}) - \sum_{v \in Values(x_j)} \frac{|\mathcal{D}_v|}{|\mathcal{D}|} I(\mathcal{D}_v), \tag{14}$$

where $I$ was defined as either Entropy, Gini impurity, or the misclassification error. For regression, we can simply use a metric for comparing continuous target variables to the predictions using a metric such as the mean squared error at a given node $t$:

$$MSE = \frac{1}{n_t} \sum_{i=1, i \in \mathcal{D}_t}^{n} \left( y_t^{[i]} - h(\mathbf{x}^{[i]})_t \right)^2. \tag{15}$$

Note that the predicted target value at a node $t$, $h(\mathbf{x})_t$,is computed as the sample mean of the training subset at that node:

$$h(\mathbf{x})_t = \frac{1}{n_t} \sum_{i \in D_t} y^{[i]}. \tag{16}$$

The MSE at a given node is hence also often referred to as "within-node variance," and the splitting criterion is thus called "variance reduction."

Note that decision trees suffer from the same problem as classification trees in that they are not good at approximating diagonal hyperplanes.

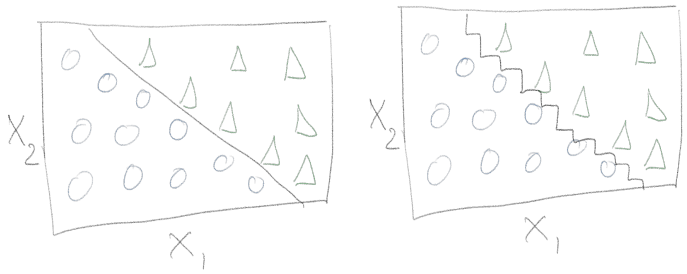


**Figure 12:** Classification tree approximating a diagonal decision boundary. Splits are always perpendicular to the feature axes.

## 6.13 Summary

### 6.13.1 Pros and Cons of Decision Trees

Listed below are some of the pros and cons of using decision trees as a predictive model.

- (+) Easy to interpret and communicate
- (+) Independent of feature scaling
- (-) Easy to overfit
- (-) Elaborate pruning required
- (-) Expensive to just fit a "diagonal line"
- (-) Output range is bounded (dep. on training examples) in regression trees

In the next lecture, we will talk about several ensemble methods, some of which are traditionally focused on using decision trees, e.g., bagging and random forests, which help with making decision tree models more robust against overfitting by creating an ensemble that reduces the variance (here: in terms of the variance of the model with respect to the loss function [12]) compared to the individual trees.

[12] More details on the bias-variance decomposition and trade-off will be provided in the model evaluation lectures