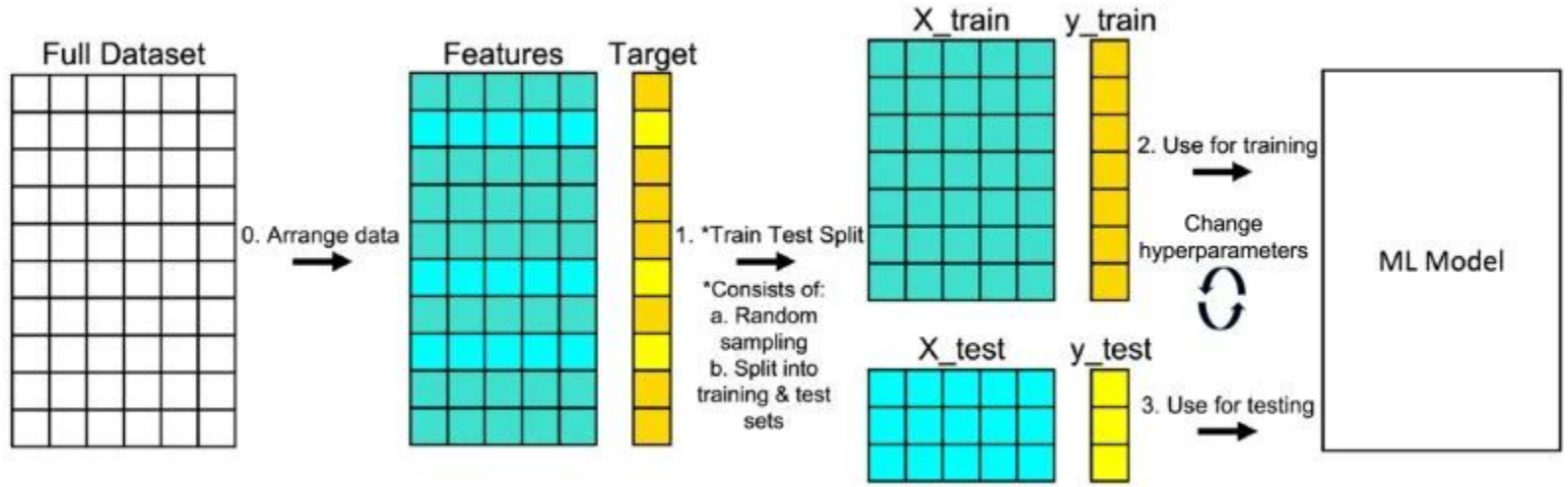
Decision Trees, Data Splitting and Overfitting/Underfitting

Hrachya Asatryan

Train-Test Split

Why do we split the data?

When building a machine learning model, accurate evaluation depends on the problem type. To get unbiased evaluation, you must not test the model on the same data it was trained on. The simplest way to ensure this is to split the data into a training set and a test set before training. The model is trained only on the training set and evaluated on the test set, which contains data the model hasn't seen before.



Train-Test Split

Methods of splitting:

Different kinds of datasets require different methods for splitting the data into training and testing sets. Here's some common methods of splitting data in a train test split:

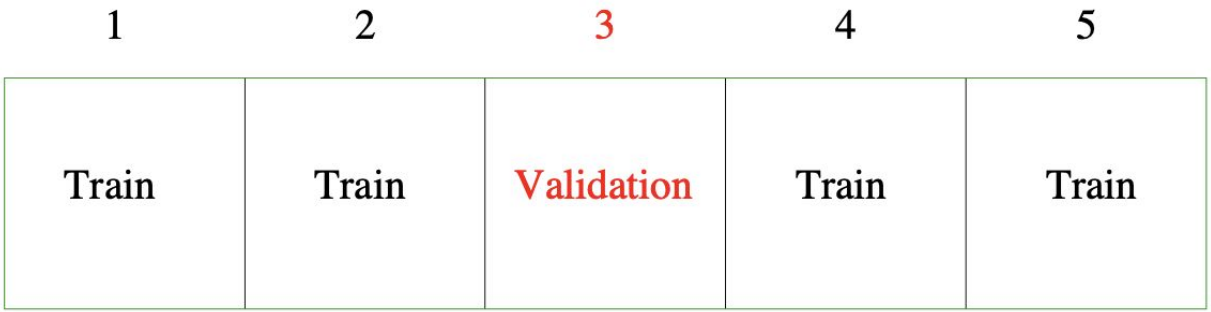
- **Random Splitting**
  - Randomly shuffle and split data (e.g., 75% train, 25% test). Simple and widely used, especially for large, balanced datasets. This is the **default in scikit-learn's train_test_split()**.
- **Stratified Splitting**
  - Split data so class proportions are preserved in both sets. Useful for imbalanced datasets to avoid bias. In scikit-learn, use the **stratify parameter in train_test_split()**.
- **Time-Based Splitting**
  - Split data based on time order, training on past data and testing on future data. Ideal for time series and forecasting tasks. In scikit-learn, use **TimeSeriesSplit()** for this purpose. Beware it may struggle with non-stationary data.

Cross-Validation

Key points:

Cross-validation is a straightforward and popular way to estimate **prediction error**, directly approximating the expected **generalization error** $Err=E[L(Y,f(X))]$ when applying f to new data.

With limited data, setting aside a separate validation set isn't always feasible. **K-fold cross-validation** solves this by splitting the data into K equal parts—training the model on $K-1$ folds and testing it on the remaining fold, rotating until each part has been used for testing. For instance, with $K=5$, you repeat this process five times.



Cross-Validation

In **K-fold cross-validation**, for each fold *k*, we train the model on the other *K−1* parts and test it on the *kth* part. We repeat this for all *k=1,2,...,K* and average the resulting errors.
More formally, let *κ(i)* indicate which fold observation *i* belongs to $f^{\hat{}-k}(x)$ be the model trained without the *kth* fold. The overall cross-validation error is then the average loss over all observations using their corresponding fold-trained model:

$$\mathrm{CV}(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i))$$

Given a set of models *f(x,α)* indexed by a tuning parameter *α*, denote by $f^{\hat{}-k}(x,\alpha)$ the *αth* model fit with the *kth* part of the data removed. Then for this set of models we define:

$$\mathrm{CV}(\hat{f}, \alpha) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \alpha))$$

The function *CV(f, α)* provides an estimate of the test error curve, and we find the tuning parameter $\alpha^{\hat{}}$ that minimizes it. Our final chosen model is $f(x,\alpha^{\hat{}})$, which we then fit to all the data.

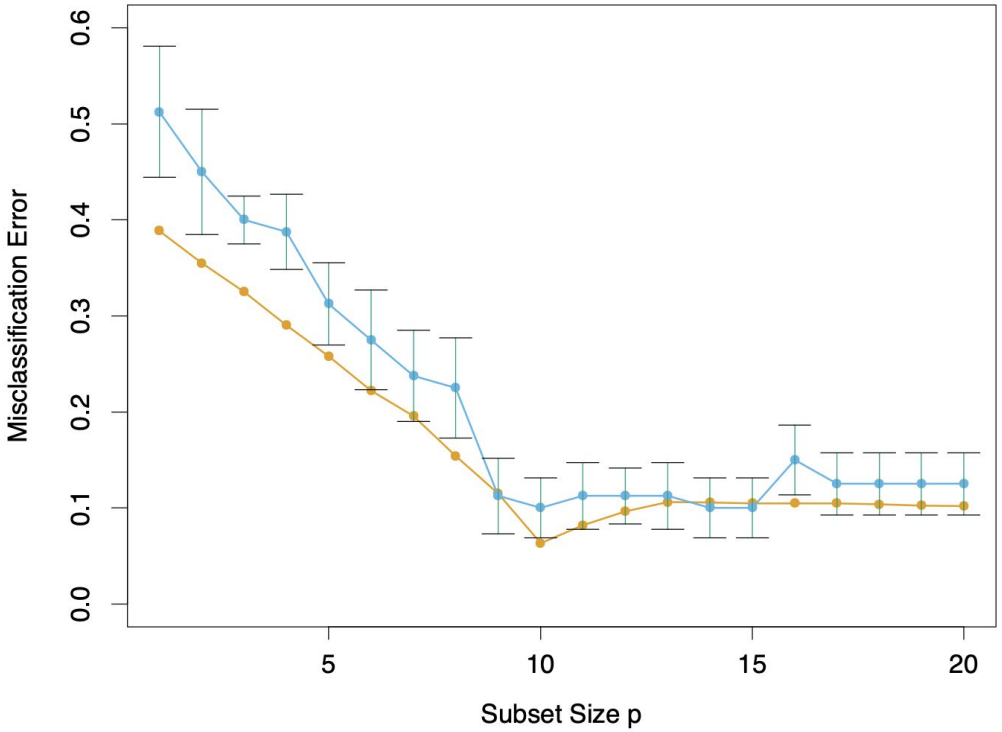
Cross-Validation

Value of K:

What value should we choose for **K**? With ***K = N***, the cross-validation estimator is approximately **unbiased** for the true (expected) prediction error, but can have **high variance** because the *N* "training sets" are so similar to one another. The computational burden is also considerable, requiring *N* applications of the learning method. On the other hand, with *K = 5* say, cross-validation has **lower variance**. But **bias** could be a problem, depending on how the performance of the learning method varies with the size of the training set.

The illustration below shows prediction error and tenfold cross-validation curves for a two-class classification problem using linear models with subsets of size *p*. Both curves reach a minimum at *p = 10,* but the CV curve is flat beyond that. Using the "one-standard error" rule, we select the simplest model with error within one standard error of the minimum—here, a model with about *p=9* predictors, slightly smaller than the true model with 10 predictors.

Illustration



Prediction error (orange) and tenfold cross-validation curve (blue) estimated from a single training set

Decision Trees

Problem Formulation

Main intuition

Create a tree based structure that splits based on every feature.

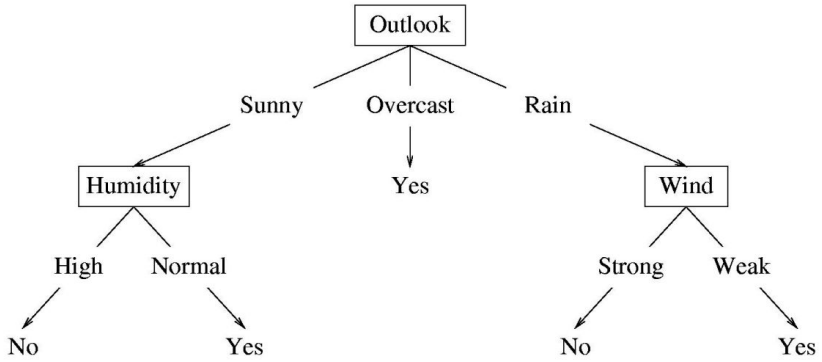
What is our goal?

Our goal is to fit the data the best we can, while we make no assumption about the underlying function.

How do we do that?

We split the data into branches in a way, that maximizes our information gain.

Predictors				Response
Outlook	**Temperature**	**Humidity**	**Wind**	**Class**
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No



Problem Formulation

Main assumptions

1. Data can be split into subsets based on the input variables from the data features.
2. The split can stand alone and is able to maximize information gain.
3. These splits are binary and this means each split is formed into two subsets from a single input feature.

Advantages

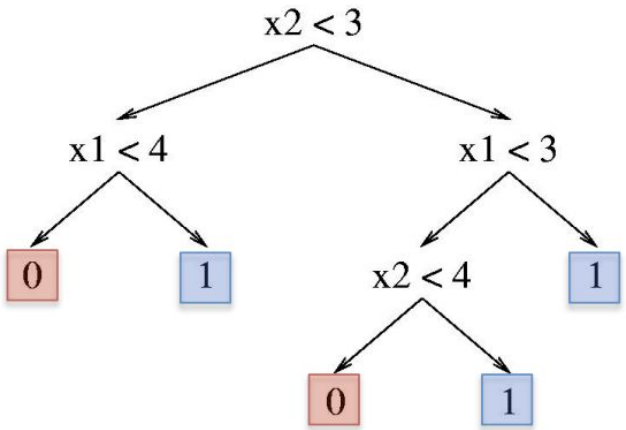
- Can perform functions on both categorical and numerical data, making it versatile.
- Adding new features is straightforward.
- it requires minimal data preprocessing since it's robust against outliers.
- In conjunction with ensemble methods, it can be used to build more larger classifiers.
- It doesn't require scaling or normalization of the data.

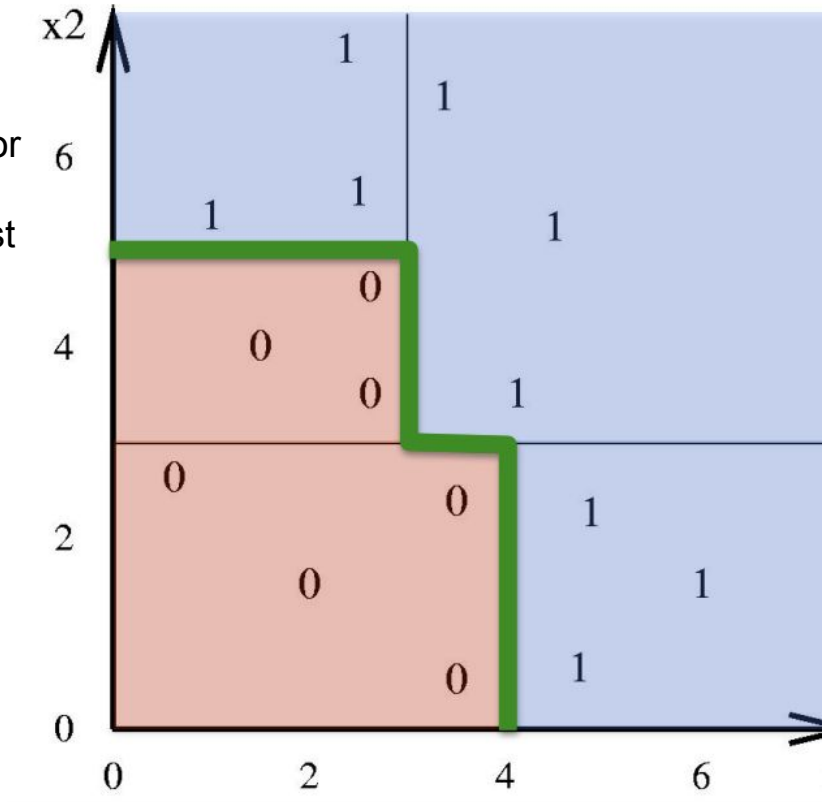
Disadvantages

- A small change in the data can cause a large change in the decision tree structure causing instability.
- The calculation can be far more complex compared to other algorithms, taking more time to train the models.
- Trees are prone to overfitting.
- Require some kind of measurement as to how well they are doing.
- Can be biased, if some classes dominate.

Decision Boundary

Decision trees divide the feature space into axis-parallel rectangles.
Each rectangular region is labeled with one label or a probability distribution over labels.
Trees can encapsulate any boolean function, worst case they will need exponentially many nodes.





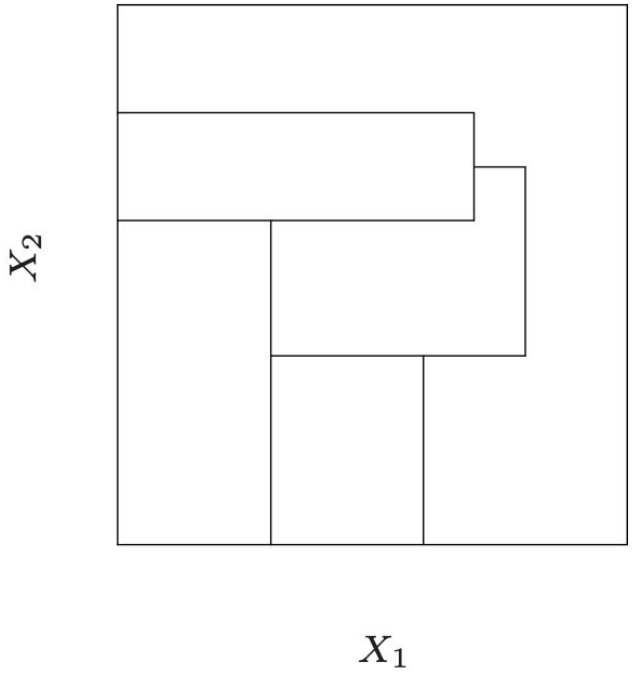
Mathematical Formulation

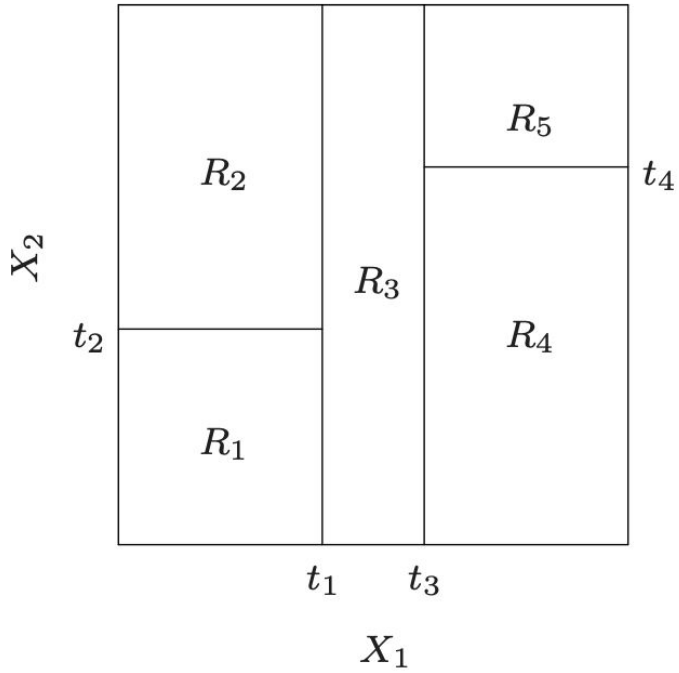
- Tree-based methods partition the feature space into a set of rectangles,
- Then fit a simple model (like a constant) in each one.
- They are conceptually simple yet powerful.

Example:

Let’s consider a regression problem with continuous response Y and inputs X_1 and X_2, each taking values in the unit interval.

Mathematical Formulation

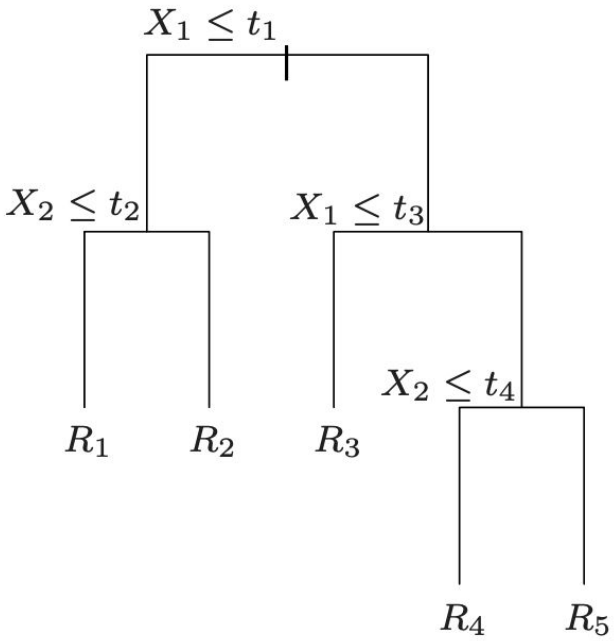




Mathematical Formulation

- In each partition element we can model Y with a different constant.
- However, there is a problem: although each partitioning line has a simple description like $X_1 = c$, some of the resulting regions are complicated to describe.
- To simplify matters, we restrict attention to recursive binary partitions

Mathematical Formulation



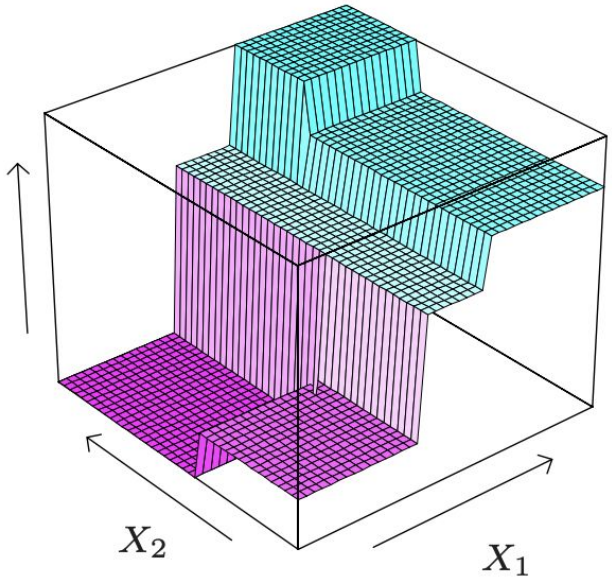


Figure is taken from the book: Elements of Statistical Learning

Mathematical Formulation

- Our data consists of p inputs and a response,
- For each of N observations: That is, (x_i, y_i) for i = 1, 2,... ,N, with $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$.
- The algorithm needs to automatically decide on the splitting variables and split points.
- Suppose first that we have a partition into M regions $R_1, R_2, ... , R_M$, and we model the response as a constant c_m in each region:

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

Regression Trees

- If we adopt as our criterion minimization of the sum of squares $\sum(y_i - f(x_i))^2$ it is easy to see that the best c_m is just the average of y_i in region R_m:

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m).$$

- Finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible.

- We use greedy algorithm. Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j, s) = \{X | X_j \leq s\} \text{ and } R_2(j, s) = \{X | X_j > s\}.$$

Regression Trees

- Then we seek the splitting variable j and split point s that solve

$$\min_{j,\,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right].$$

- For any choice j and s, the inner minimization is solved by

$$\hat{c}_1=\text{ave}(y_i|x_i\in R_1(j,s))\quad\text{and}\quad\hat{c}_2=\text{ave}(y_i|x_i\in R_2(j,s)).$$

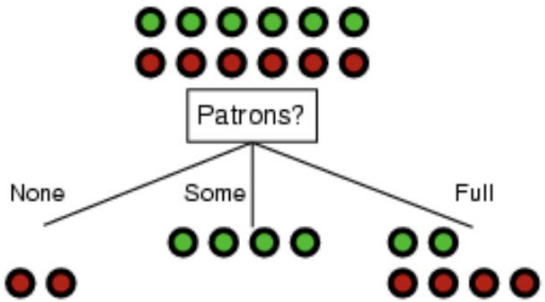
Algorithm

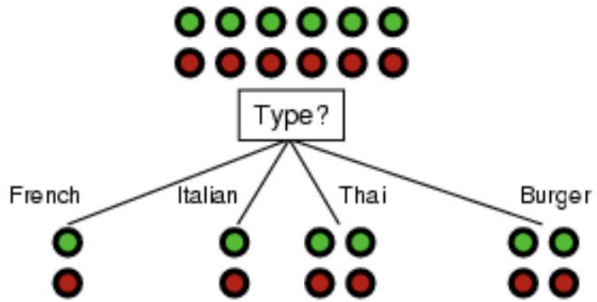
How do I construct a tree?

Given that **A** is our best decision attribute for the next node:

1. Assign **A** as a decision attribute.
2. For each value of A create a new descendant of node.
3. Sort training examples to the leaf nodes.
4. If the training examples are perfectly classified, stop. Else recurse over new leaf nodes.

How do we choose the best decision attribute?
A good attribute splits the examples into (ideally) uniform subsets:





Classification Trees

- The only difference is that the squared-error node that we used for regression is not applicable.

- In a node m, representing a region R_m with N_m observations, let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k),$$

the proportion of class k observations in node m.

- We classify the observations in node m to class

$$k(m) = \arg\max_k \hat{p}_{mk},$$

the majority class in node m.

Classification Trees

Different measures of node impurity include the following:

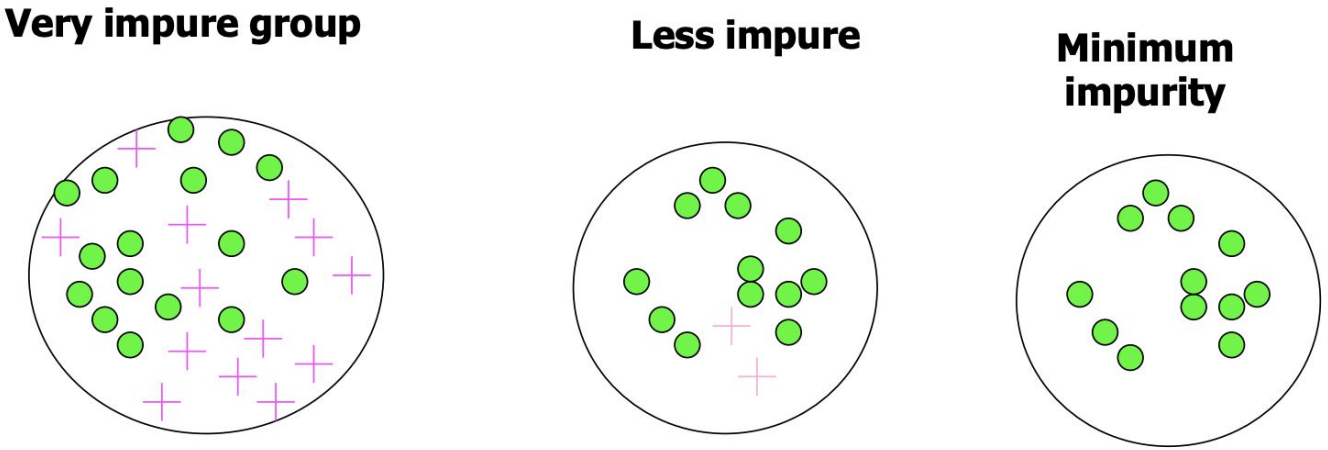
Misclassification error:	$\frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}.$
Gini index:	$\sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}).$
Cross-entropy or deviance:	$-\sum_{k=1}^{K} \hat{p}_{mk}\log\hat{p}_{mk}.$

Figure is taken from the book: Elements of Statistical Learning

Algorithm

What is impurity?

As the name suggests, we measure how pure our data is, given the split, which in turn shows how much information we have gained after the split.



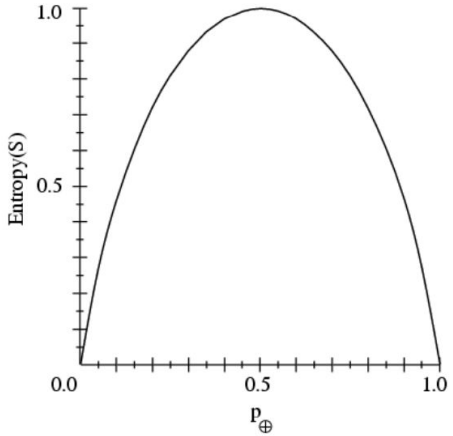
Algorithm

What is impurity?

We measure impurity using different methods, one of them is Entropy:

$$H(X) = -\sum_{i=1}^{n} P(X = i) \log_2 P(X = i)$$

Thus given the formulation, we get a maximum entropy, When our split contains about the same number of elements from each class.



Algorithm

From Entropy to Information Gain

We want to determine which attribute in a given set of training feature vectors is the most useful for discriminating between the classes to be learned. InformatIon gain tells us how important a given attribute of the feature vectors is. We will use it to decide the ordering of attributes the nodes of a decision tree.
Thus for **Gain** we will have:

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

$$H(X) = -\sum_{i=1}^{n} P(X=i)\log_2 P(X=i)$$

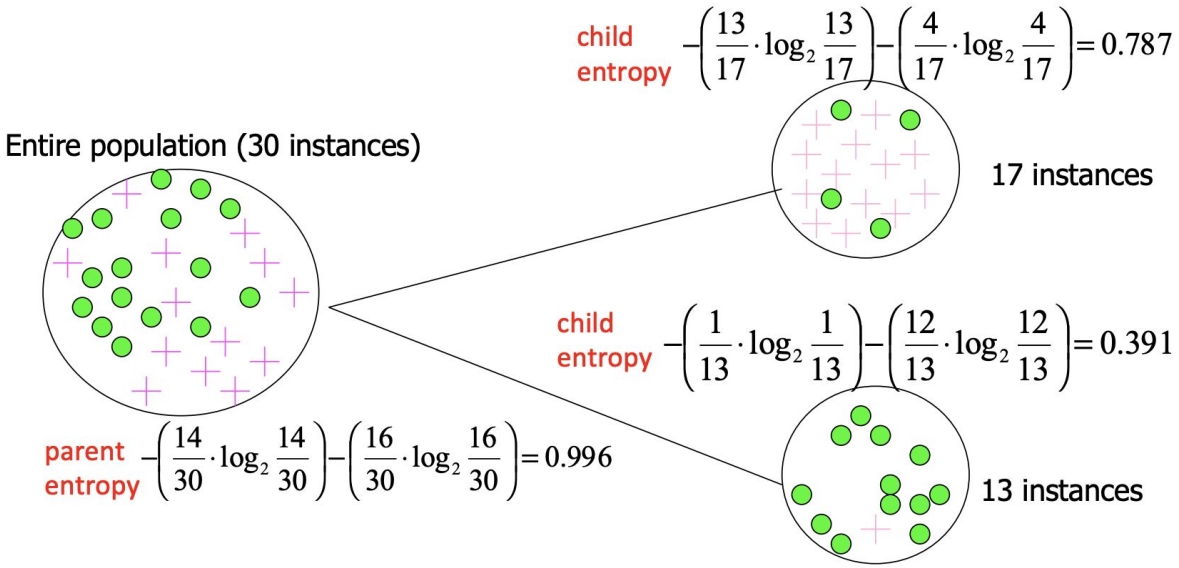
$$H(X|Y=v) = -\sum_{i=1}^{n} P(X=i|Y=v)\log_2 P(X=i|Y=v)$$

$$H(X|Y) = \sum_{v \in values(Y)} P(Y=v)H(X|Y=v)$$

Algorithm

Calculating Information Gain

Information Gain = entropy(parent) – [average entropy(children)]



(Weighted) Average Entropy of Children = $\left(\frac{17}{30}\cdot 0.787\right)+\left(\frac{13}{30}\cdot 0.391\right)=0.615$

Information Gain= 0.996 - 0.615 = 0.38

Algorithm

Is our tree any good?

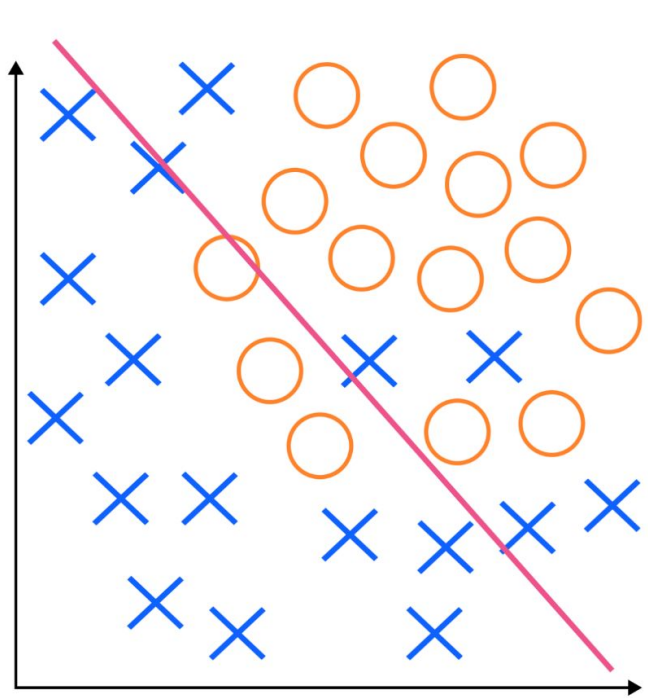
Our ideal tree should be:

- ➔ Not too small: need to handle important but possibly subtle distinctions in data
- ➔ Not too big:
  - ◆ Avoid **overfitting** training examples.
    - ● We need enough samples in each region to confidently determine the output.
  - ◆ Computational efficiency (avoid redundant, spurious attributes)
  - ◆ Human interpretability
- ➔ Occam's Razor: find the simplest hypothesis that fits the observations
- ➔ We desire small trees with informative nodes near the root

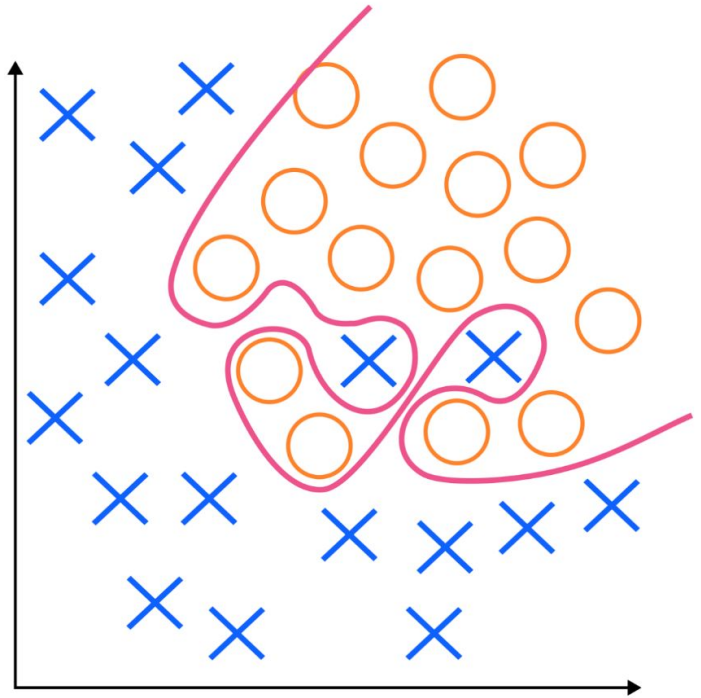
We might encounter problems such as:

- ➔ Having exponentially less data at lower levels
- ➔ A large tree can overfit the data
- ➔ Greedy algorithms do not necessarily yield the global optimum
- ➔ Mistakes at top-level propagate down

Overfitting and Underfitting



Underfitting



Overfitting

Overfitting

Main Principle:

In case of **Overfitting**:

- The **training** error is **low**, but the **testing** error is significantly **higher**.
- The model ends up memorizing the training dataset, as it has too many parameters of layers, making it highly adaptable to data - think you can fit n data points with a *(n-1)th degree* polynomial.
- When training on a small and noisy dataset, the model risks memorizing the noise and specific data points, instead of the general patterns. If the data has inconsistencies, the model risks learning them, instead of the meaningful patterns.
- In case of high dimensional data, the data points become sparse and finding a meaningful pattern across them becomes harder: **this increases variance** and the risk of overfitting.

Overfitting

Examples:

- A machine learning model trained on a small medical image dataset achieves high accuracy by memorizing training data but performs poorly on new images due to overfitting to noise and artifacts instead of learning general disease features.
- A complex financial model overfits by learning random fluctuations in historical stock data, resulting in high training accuracy but poor future predictions.
- A customer retention model overfits by using overly specific demographic features, making it struggle to generalize across a broader customer base.

Overfitting

How to avoid?

- Regularization, such as L1, L2, or dropout, prevents models from overfitting by discouraging reliance on specific features or noise. L1 promotes sparsity by setting some coefficients to zero, while L2 shrinks all coefficients to simplify the model and improve generalization.
- Data augmentation, like flipping or rotating images, helps models generalize in tasks like computer vision. Simplifying models by reducing parameters or layers also reduces overfitting.
- Engineers use techniques like k-fold cross-validation and holdout sets to assess model generalization. These methods split data into training and testing portions, with results averaged to evaluate overall performance.
- Ensemble methods like bagging and boosting combine models to improve generalization. Random forests, for example, reduce overfitting by averaging predictions from many decision trees, balancing bias and variance.

Underfitting

Main Principle:

In case of **underfitting**:

- The errors are **consistently high**, across both training and testing sets,
- The models tend to show high errors in learning curves,
- Return suboptimal evaluation metrics,
- Exhibit systematic residual patterns.

Underfitting usually occurs due to:

- Usage of simplistic models,
- Poor feature engineering and selection,
- Excessive regularization,
- Inadequate preprocessing,
- Insufficient training time,
- Lack of sufficient data.

Underfitting

Examples:

- A linear regression model predicts house prices using only square footage, ignoring key features like location and age, resulting in poor performance.
- A rainfall prediction model using only basic features like temperature and humidity misses complex patterns, leading to consistently low accuracy.
- A shallow decision tree struggles to classify cat and dog images due to its simplicity, resulting in poor performance on both training and unseen data.

Underfitting

How to avoid?

- To fix underfitting, engineers increase model complexity, like using polynomial instead of linear regression for nonlinear patterns. However, more complex models risk overfitting if not properly regularized.
- Reducing regularization penalties lets the model fit data more flexibly. L1 (lasso) encourages selecting key features by adding penalties, while L2 (ridge) spreads importance more evenly across features.
- Feature engineering and selection create or transform features—like interaction terms, polynomials, or encoded categories—to give the model more relevant information.
- Increasing epochs gives the model more chances to learn from the data, improving pattern recognition. Expanding the training dataset also helps capture diverse patterns, reducing oversimplification and boosting generalization.
- Engineers should ensure training data is accurate, complete, and consistent, using normalization or standardization to balance feature scales. Since data distributions can shift over time (data drift), regular monitoring and retraining are needed. Removing outliers also helps improve model robustness.

Overfitting and Underfitting

How to choose a good model?

A good model fit lies at the optimal balance between underfitting and overfitting. It describes a model that accurately captures the underlying patterns in the data without being overly sensitive to noise or random fluctuations.

- The tradeoff between model complexity and generalization is about finding the right balance between a model being too simple or too complex.
- One must balance bias and variance to achieve optimal model performance.
- One way to do this is by tracking learning curves, which will show training and validation errors over time.
- Analyzing validation metrics such as accuracy, precision, recall or mean squared error helps evaluate how well the model generalizes to unseen data.
- A good fit model carefully balances model complexity, training data and regularization techniques to generalize well to new data and provide accurate predictions.

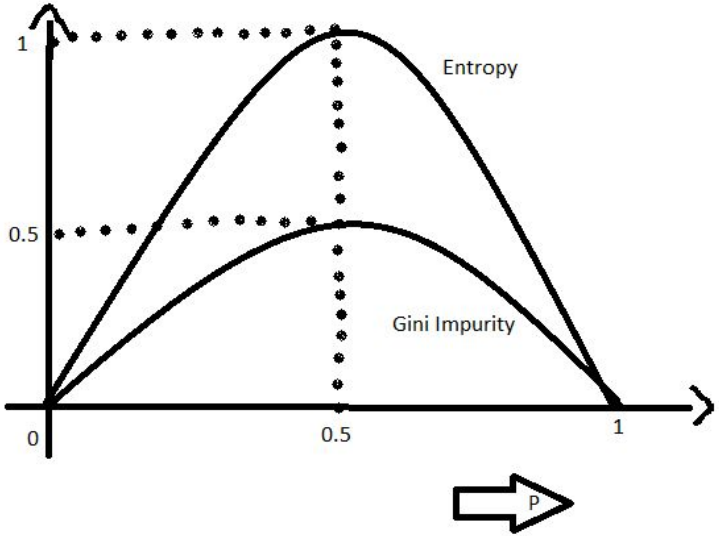
Gini Impurity

Are there other ways to split?

As Entropy can be computationally cost heavy, we can devise other functions that behave similarly and take less resources. One of them is the Gini impurity.

The Gini Index is the probability of misclassifying a randomly chosen element in a set, while entropy measures the amount of uncertainty or randomness in a set. It is a linear measure and can be interpreted as the expected error rate in a classifier.

However it is less robust and more sensitive than entropy.



$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

Cost-Complexity Pruning

- The strategy is to grow a large tree T_0 by stopping the splitting process only when some minimum node size is reached.
- Then this large tree is pruned using cost-complexity pruning,
- We index terminal nodes by m, with node m representing region R_m.
- Let |T| denote the number of terminal nodes in T.
- We define the cost complexity criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|.$$

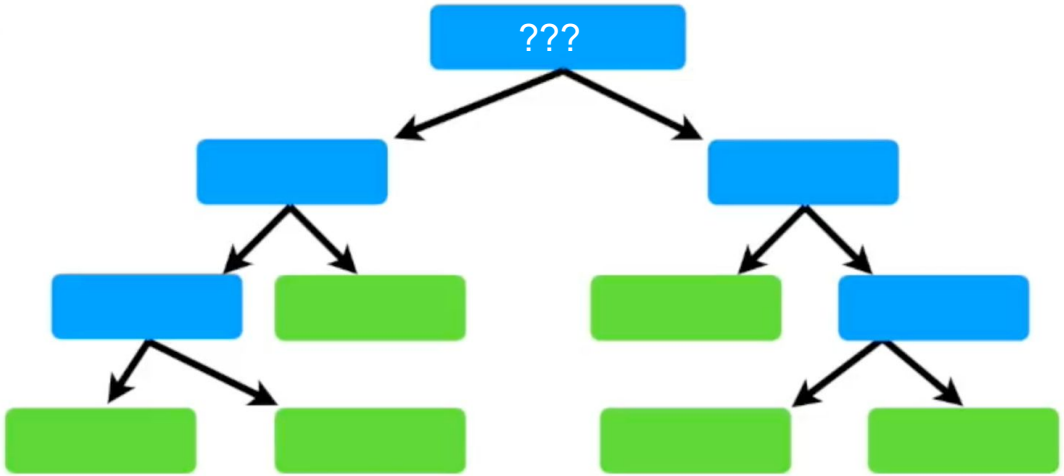
  where Q_m is the node impurity
- The idea is to find, for each α, the subtree $T_\alpha \subseteq T_0$ to minimize $C_\alpha(T)$.
- The tuning parameter $\alpha \geq 0$ governs the tradeoff between tree size and its goodness of fit to the data.
- Large values of α result in smaller trees T_α, and conversely for smaller values of α.

Pruning and the Estimation of α

- For each α one can show that there is a unique smallest subtree T_α that minimizes $C_\alpha(T)$.
- To find Tα we use **weakest link pruning**: we successively collapse the internal node that produces the smallest per-node increase in $\Sigma_m N_m Q_m(T)$,and continue until we produce the single-node tree.
- This gives a (finite) sequence of subtrees, and one can show this sequence must contain T_α.
- Estimation of α is achieved by five- or tenfold cross-validation: we choose the value α* to minimize the cross-validated sum of squares. Our final tree is T_{α^*}.

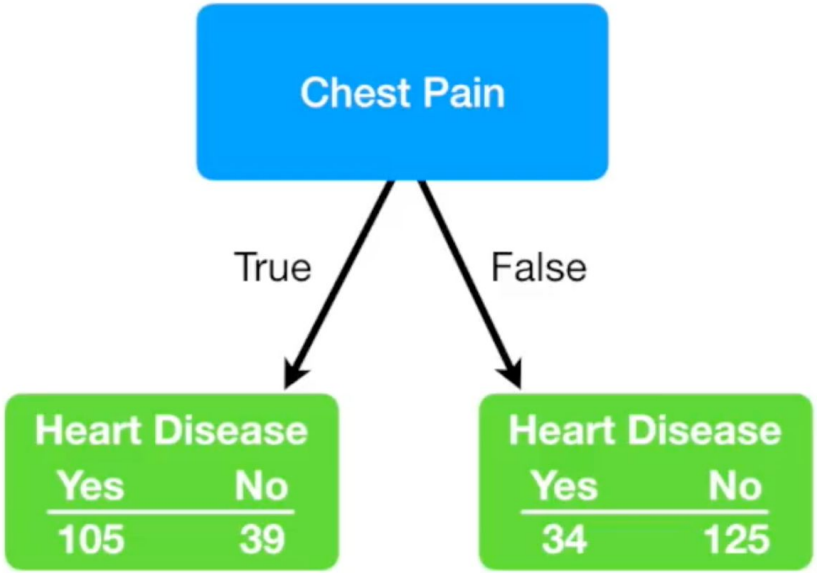
Example

Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
etc…	etc…	etc…	etc…



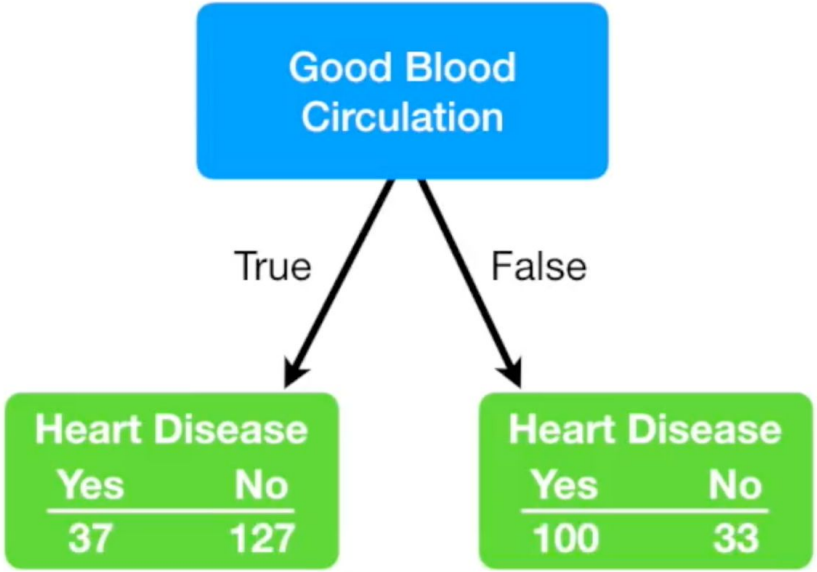
Example

Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
etc…	etc…	etc…	etc…



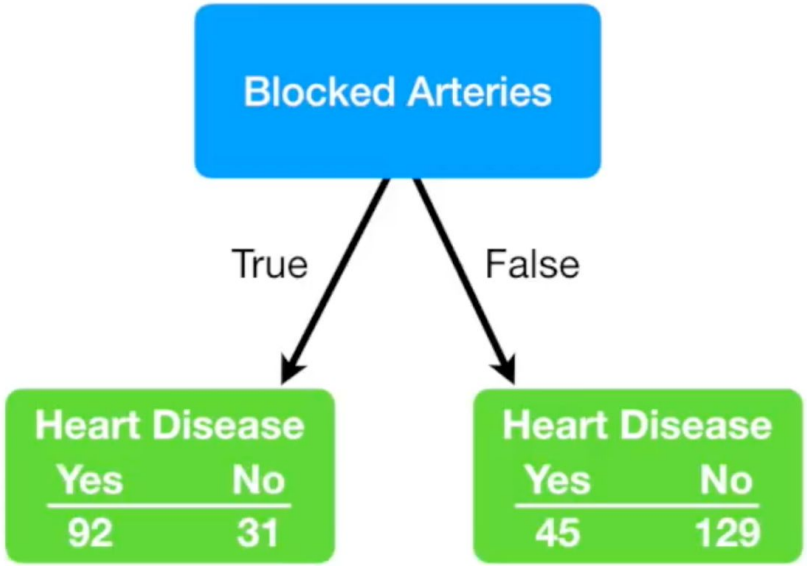
Example

Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
etc…	etc…	etc…	etc…



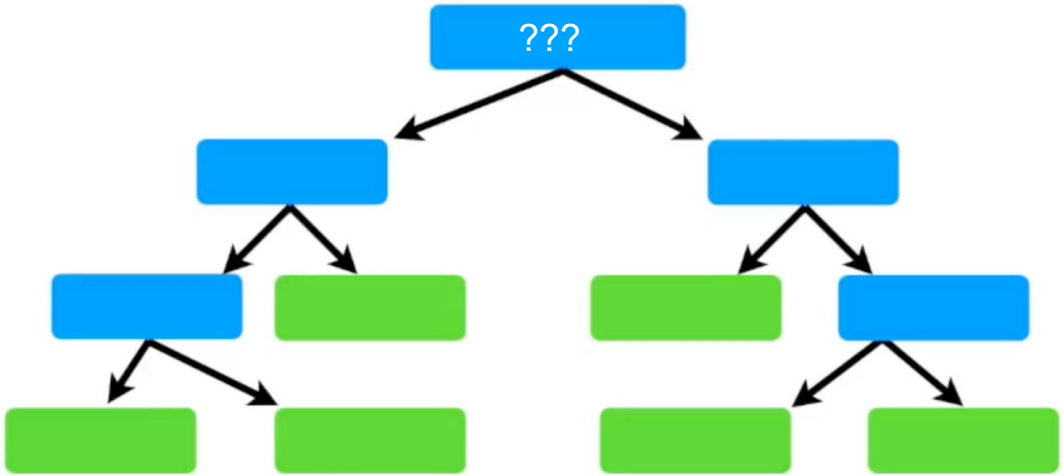
Example

Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
etc…	etc…	etc…	etc…

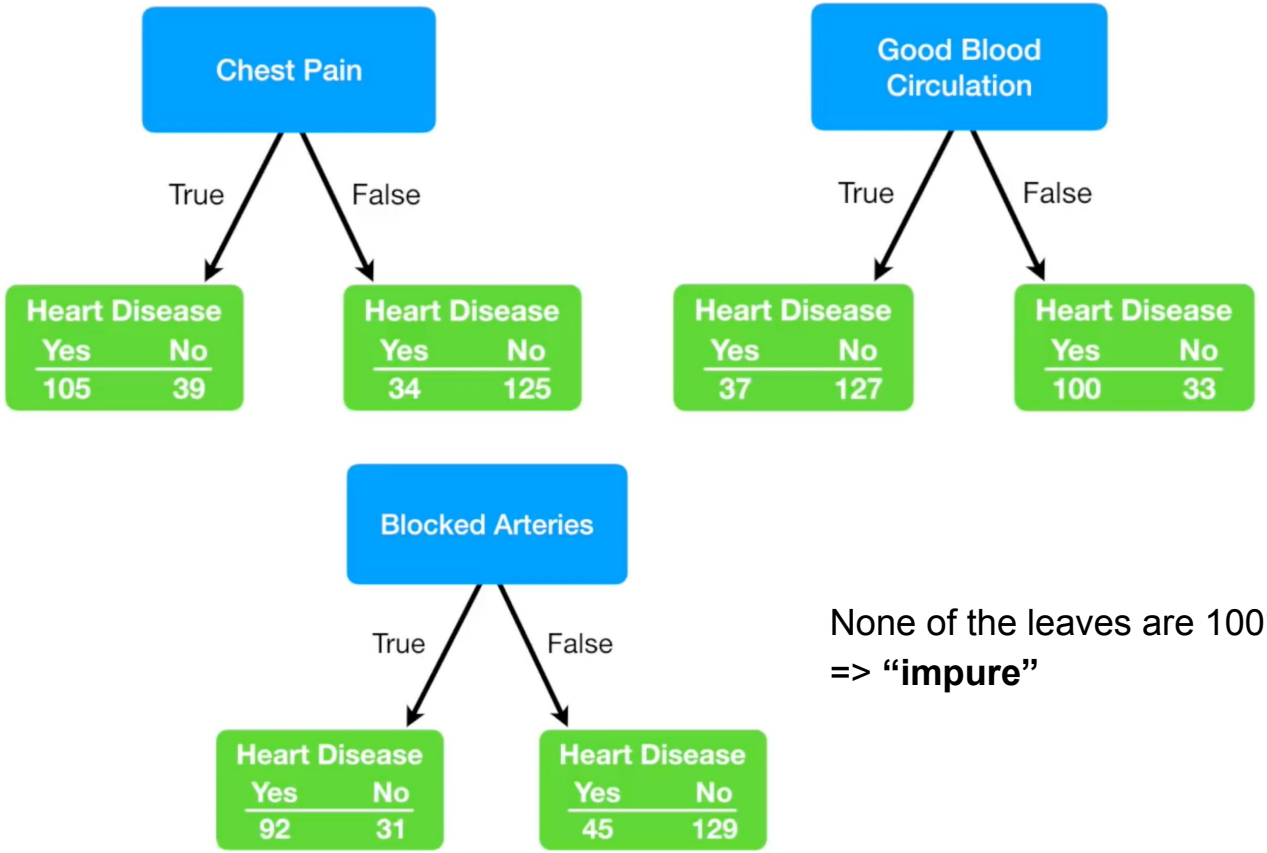


Example

Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
etc…	etc…	etc…	etc…

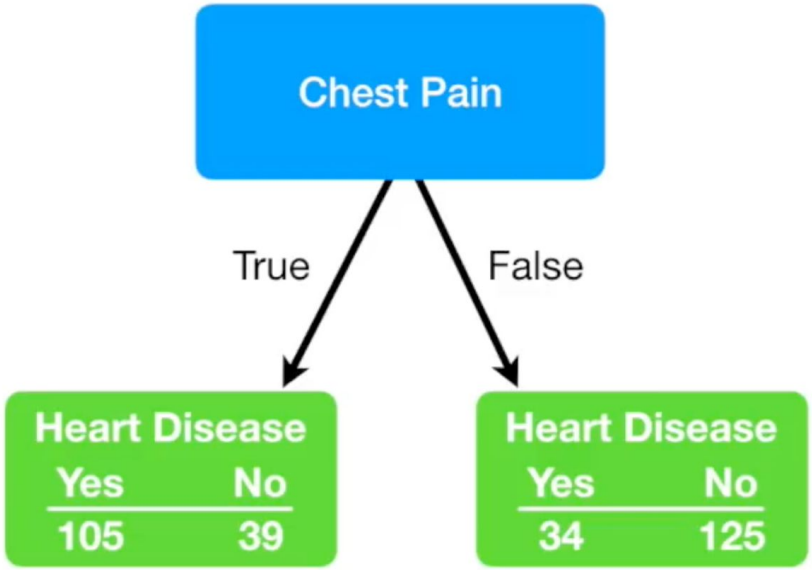


Example



None of the leaves are 100% yes or 100% no
=> **“impure”**

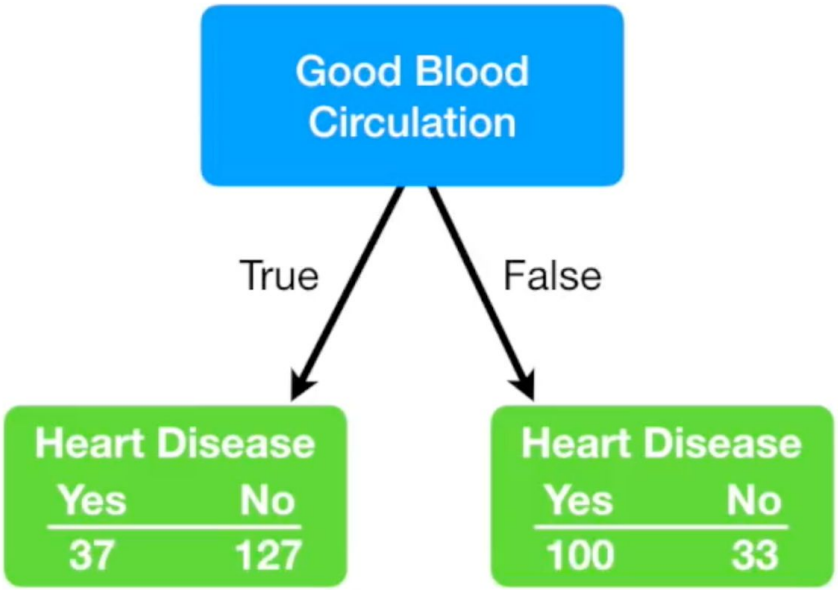
Example



$$Group1\_class0 = \frac{39}{105+39} = 0.27$$
$$Group1\_class1 = \frac{105}{105+39} = 0.73$$
$$Group0\_class0 = \frac{125}{34+125} = 0.79$$
$$Group0\_class1 = \frac{34}{34+125} = 0.21$$

$$Group1 = (1 - (0.27^2 + 0.73^2)) \times \frac{144}{303} = 0.187$$
$$Group0 = (1 - (0.79^2 + 0.21^2)) \times \frac{159}{303} = 0.174$$
$$Gini\ score = 0.187 + 0.174 = 0.361$$

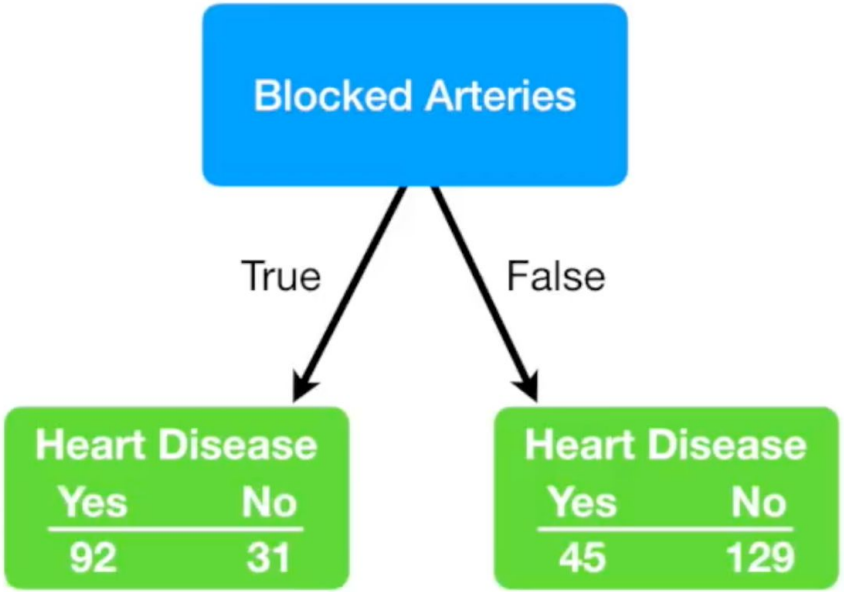
Example



$$Group1\_class0 = \frac{127}{127+37} = 0.77$$
$$Group1\_class1 = \frac{37}{127+37} = 0.23$$
$$Group0\_class0 = \frac{33}{33+100} = 0.25$$
$$Group0\_class1 = \frac{100}{33+100} = 0.75$$

$$Group1 = (1 - (0.77^2 + 0.23^2)) \times \frac{164}{297} = 0.196$$
$$Group0 = (1 - (0.25^2 + 0.75^2)) \times \frac{133}{297} = 0.168$$
$$Gini\ score = 0.196 + 0.168 = 0.364$$

Example



$$Group1\_class0 = \frac{31}{31+92} = 0.25$$
$$Group1\_class1 = \frac{92}{31+92} = 0.75$$
$$Group0\_class0 = \frac{129}{129+45} = 0.74$$
$$Group0\_class1 = \frac{45}{129+45} = 0.26$$

$$Group1 = (1 - (0.25^2 + 0.75^2)) \times \frac{123}{297} = 0.155$$
$$Group0 = (1 - (0.74^2 + 0.26^2)) \times \frac{174}{297} = 0.225$$
$$Gini\ score = 0.155 + 0.225 = 0.38$$

Regression Tree Limitations

- Instability of Trees

  One major problem with trees is their high variance. Often a small change in the data can result in a very different series of splits, making interpretation somewhat precarious. The major reason for this instability is the hierarchical nature of the process: the effect of an error in the top split is propagated down to all of the splits below it.

- Lack of Smoothness

  In classification with 0/1 loss, this doesn't hurt much, since bias in estimation of the class probabilities has a limited effect. However, this can degrade performance in the regression setting.