Tree based methods

Alex Sanchez, Ferran Reverter and Esteban Vegas

Genetics Microbiology and Statistics Department. University of Barcelona

Outline

1. Introduction to decision trees
2. Data cleaning and preprocessing
3. Pruning and optimization
4. Classification trees
5. Regression trees
6. Ensemble methods and advanced topics
7. Practical examples and exercises
8. Conclusion and future directions

Motivation

- In many real-world applications, decisions need to be made based on complex, multi-dimensional data.
- One goal of statistical analysis is to provide insights and guidance to support these decisions.
- Decision trees provide a way to organize and summarize information in a way that is easy to understand and use in decision-making.

Examples

- A bank needs to have a way to decide if/when a customer can be granted a loan.
- A doctor may need to decide if a patient has to undergo a surgery or a less aggressive treatment.
- A company may need to decide about investing in new technologies or stay with the traditional ones.

In all those cases a decision tree may provide a structured approach to decision-making that is based on data and can be easily explained and justified.

An intuitive approach

Decisions are often based on asking several questions on available information whose answers induce binary splits on data that end up with some grouping or classification.

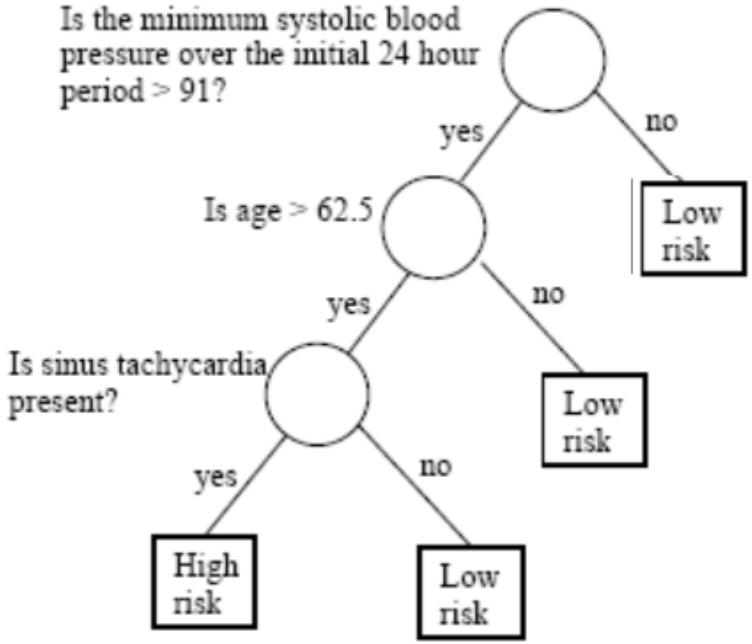


Figure 1: A doctor may classify patients at high or low cardiovascular risk using some type of decision tree

Introducing decision trees

- A decision tree is a graphical representation of a series of decisions and their potential outcomes.
- It is obtained by recursively *stratifying* or *segmenting* the *feature space* into a number of simple regions.
- Each region (decision) corresponds to a *node* in the tree, and each potential outcome to a *branch*.
- The tree structure can be used to guide decision-making based on data.

Types of decision trees

- Decision trees are simple yet reliable predictors that can be use for both classification or prediction
  - **Classification Trees** are *classifiers*, used when the response variable is categorical
  - **Regression Trees**, are *regression models* used to predict the values of a numerical response variable.

Tree building with R

Package	**Algorithm**	**Dataset size**	**Missing data handling**	**Visual repr**
`rpart`	RPART	Medium to large	Poor	Yes
`caret`	Various	Various	Depends on algorithm	Depends on algorithm
`tree`	CART	Small to medium	Poor	Yes

Tree building with Python

Package	Algorithm	Dataset size	Missing data handling	Visual repr
`scikit-learn`	CART, ID3, C4.5	Small to medium	Poor (requires preprocess-ing)	Yes (`plot_tree()`)
`xgboost`	Gradient Boosted Trees	Medium to large	Good (handles missing values natively)	Limited (requires external tools)
`lightgbm`	Gradient-based One-Side Sampling	Large	Good (handles missing values	Limited (requires external tools)

Building the trees

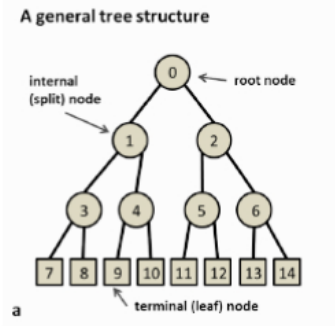
- As with any model, we aim not only at construting trees.
- We wish to build good trees and, if possible, optimal trees in some sense we decide.
- In order to **build good trees** we must decide
  - How to *construct* a tree?
  - How to *optimize* the tree?
  - How to *evaluate* it?

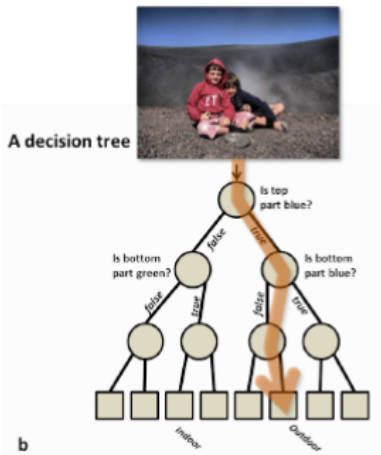
Decision Trees are *Supervised Learners*

- Classification / Regression: *Supervised Learning* tasks:
- There is a *learning set* $\mathcal{L} = \{(\mathbf{X_i}, \mathbf{Y_i})\}_{i=1}^{n}$
- And depending of $\mathbf{Y}$ we have:
  - Classification: $\mathbf{X} \in \mathbb{R}^d, \quad Y \in \{-1, +1\}$
  - Regression $\mathbf{X} \in \mathbb{R}^d, \quad Y \in \mathbb{R}$.

Trees vs Decision Trees

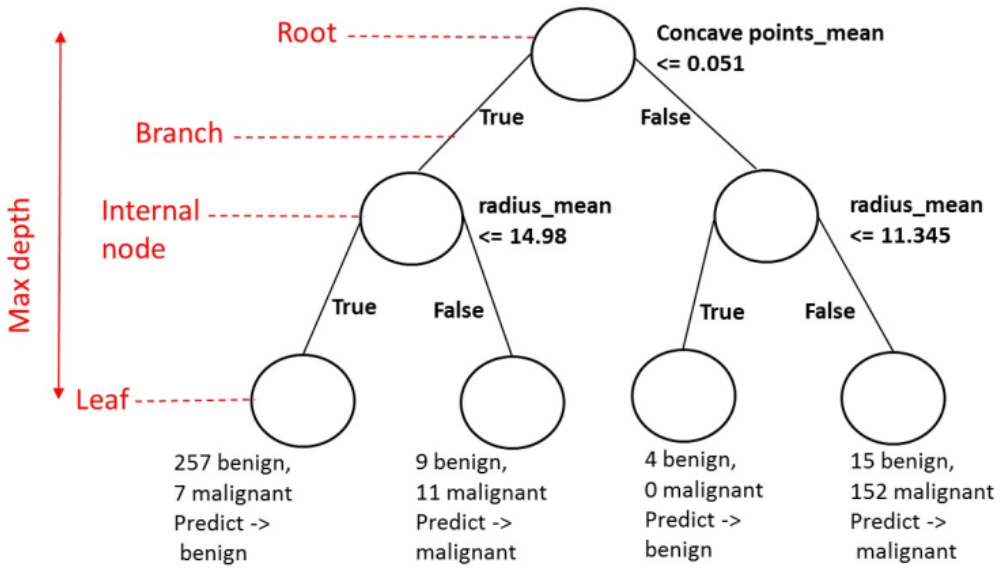
1. A tree is a set of nodes and edges organized in a hierarchical fashion. In contrast to a graph, in a tree there are no loops.
2. A decision tree is a tree where *each split node stores a boolean test function* to be applied to the incoming data. Each leaf stores the final answer (predictor)





Additional notation

- A node is denoted by t.
- The left and right child nodes are denoted by t_L and t_R
- The collection of all nodes in the tree is denoted T
- The collection of all the leaf nodes is denoted $\tilde{T}$
- A split will be denoted by s.
- The set of all splits is denoted by S.



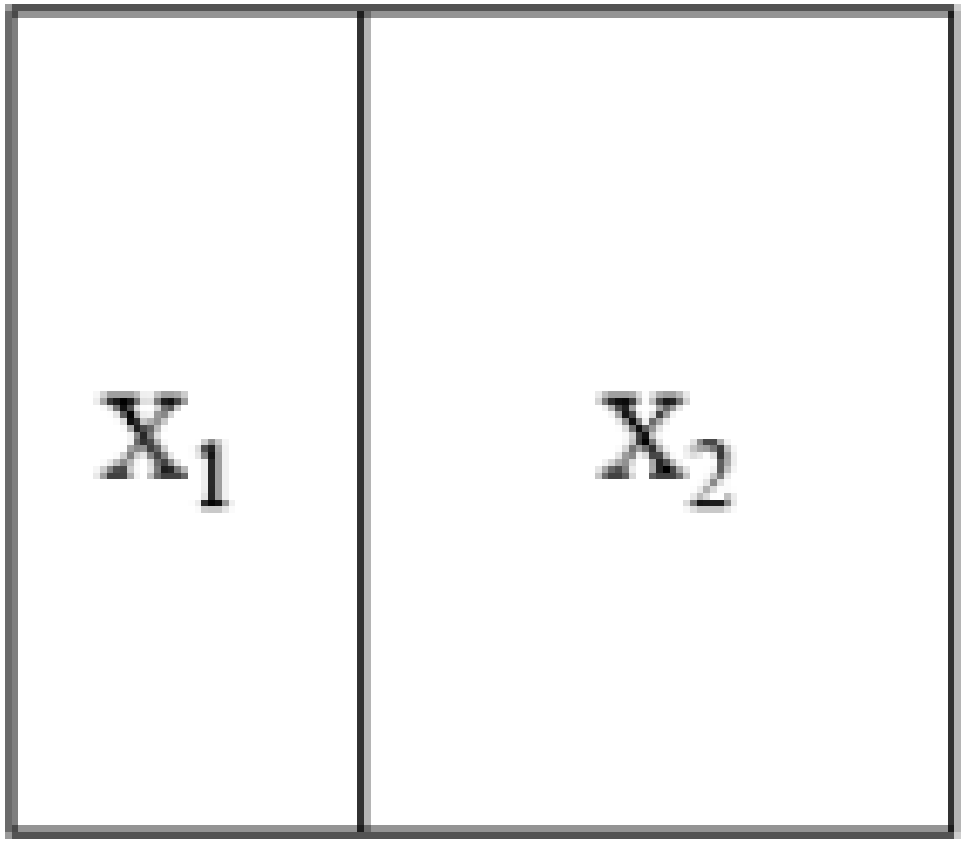
Building a tree

- A binary decision tree is built by defining a series of (recursive) splits on the feature space.
- The splits are decided in such a way that the associated learning task is attained
  - by setting thresholds on the variables values,
  - that induce paths in the tree,
- The ultimate goal of the tree is to be able to use a combination of the splits to accomplish the learning task with as small an error as possible.

Trees partition the space

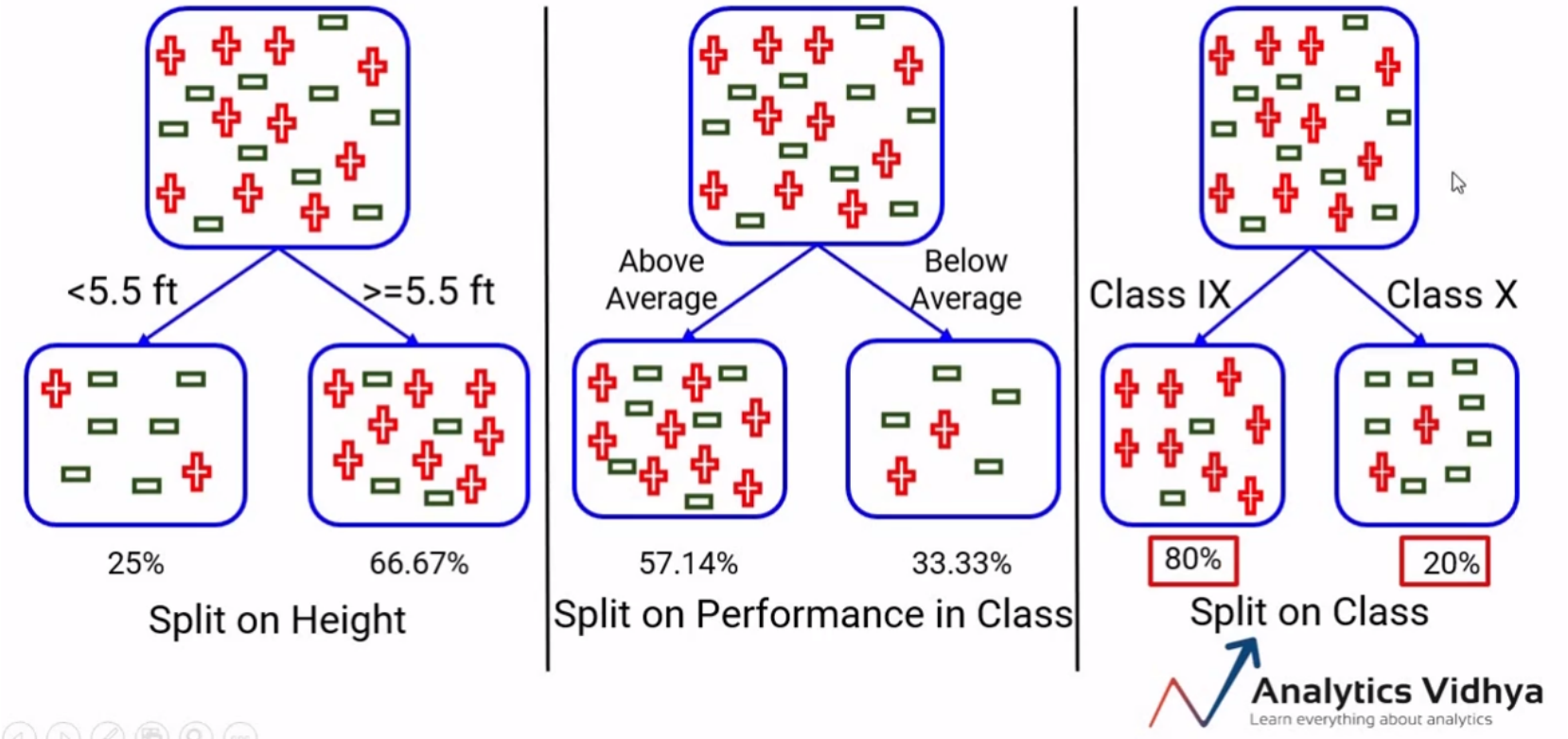
- *A tree represents a recursive splitting of the space.*
  - Every node of interest corresponds to one region in the original space.
  - Two child nodes occupy two different regions.
  - Together, they yield same region as that of the parent node.
- In the end, every leaf node is assigned with a class (or value) and a test point is assigned with the class (or value) of the leaf node it lands in.

The tree represents the splitting



Different splits are possible

- It is always possible to split a space in distinct ways



- Some ways perform better than other for a given task, but rarely will they be perfect.
- So we aim at combining splits to find a better rule.

Building a decision tree

Tree building involves the following three elements:

1. The selection of the splits, i.e., how do we decide which node (region) to split and how to split it?
   - How to select from the pool of candidate splits?
   - What are appropriate *goodness of split* criteria?
   - when to declare a node terminal and *stop splitting*?
2. How to assign each terminal node to a class
3. How to ensure the Tree is the *best possible* we can build.

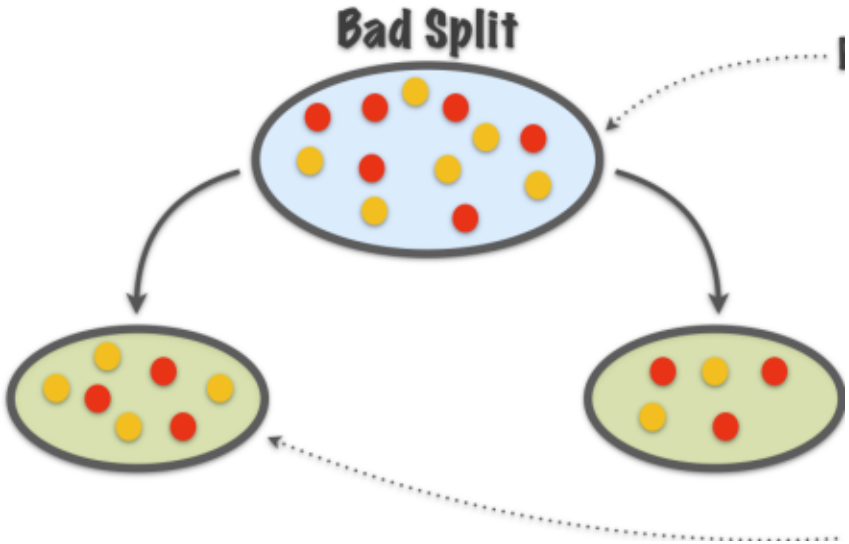
TB 1.1 - Split selection

- To build a Tree, questions have to be generated that induce splits based on the value of a single variable.
- Ordered variable X_j:
  - Is $X_j \leq c$? for all possible thresholds c.
  - Split lines: parallel to the coordinates.
- Categorical variables, $X_j \in \{1, 2, ..., M\}$:
  - Is $X_j \in A$?, where $A \subseteq M$.
- The pool of *candidate splits* for all p variables is formed by combining all the generated questions.
- With the pool of candidate splits, next step is to decide *which one to use* when constructing the decision tree.

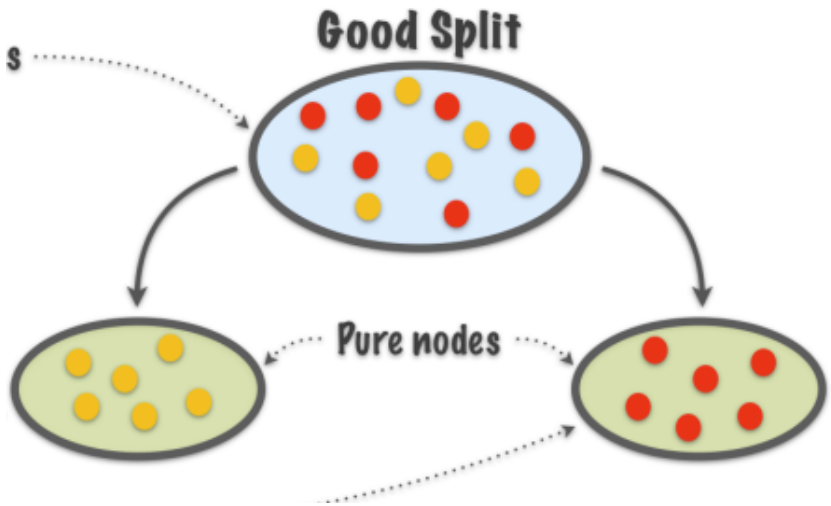
TB 1.2.1 - Goodness of Split

- Intuitively, when we split the points we want the region corresponding to each leaf node to be "pure".
- That is, we aim at regions where most points belong to the same class.
- With this goal in mind we select splits by measuring their "goodness of split" using some of the available *Impurity fuctions* introduced later.

TB 1.2.2 - Good splits vs bad splits



Purity not increased



Purity increased

TB 1.2.3 - Measuring homogeneity

- In order to measure homogeneity,or as called here, *purity*, of splits we rely on different *Impurity functions*
- These functions allow us to quantify the extent of *homogeneity* for a region containing data points from possibly different classes.
  - A region will be more pure or more homogeneous the less variable is the set of points it contains.
  - In the image in TB 1.2.2 regions on the right of the image are homogeneous that is *purer* than the heterogeneous regions on the left.

TB 1.2.4 - Impurity functions

An **impurity function** is a function Φ defined on the set of all K-tuples of numbers $\mathbf{p} = (p_1, \cdots, p_K)$ s.t. $p_j \geq 0$, $\sum_{j=1}^{K} p_j = 1$,

$$\Phi : (p_1, \cdots, p_K) \rightarrow [0, 1]$$

with the properties:

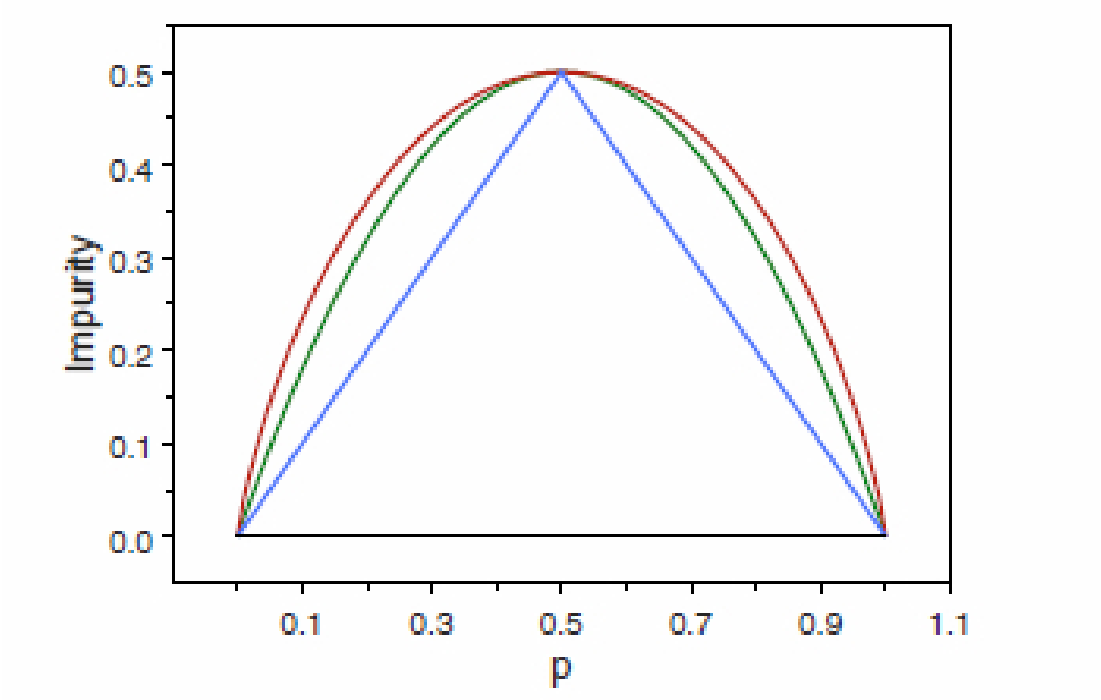
1. Φ is maximum only for the uniform distribution, that is all the p_j are equal.
2. Φ is minimum only at points $(1, 0, \ldots, 0)$,$(0, 1, 0, \ldots, 0)$, $\ldots, (0, 0, \ldots, 0, 1)$, i.e., when the probability of being in a certain class is 1 and 0 for all the other classes.
3. Φ is a symmetric function of $p_1, \cdots, p_K$, i.e., if we permute p_j, Φ remains constant.

TB 1.2.5 - Some Impurity Functions

The functions below are commonly used to measure impurity.

- **Entropy**: $\Phi_E(\mathbf{p}) = -\sum_{j=1}^{K} p_j \log(p_j)$.
- **Gini Index**: $\Phi_G(\mathbf{p}) = 1 - \sum_{j=1}^{K} p_j^2$.
- **Misclassification rate**: $\Phi_M(\mathbf{p}) = \sum_{i=1}^{K} p_j(1 - p_j)$.
- In practice, only the first two are recommended because misclassification rate is not sensitive enough to differences in class probabilies.

TB 1.2.5 Impurity functions behavior



Node impurity functions for the two-class case. The entropy function (rescaled) is the red curve, the Gini index is the green curve, and the resubstitution estimate of the misclassification rate is the blue curve.

TB 1.2.6 - Impurity measure of a split

- Given an impurity function Φ, a node t, and given $p(j \mid t)$, the estimated posterior probability of class j given node t, the *impurity measure of* t, $i(t)$, is:

$$i(t) = \phi\left(p(1 \mid t), p(2 \mid t), \ldots, p(K \mid t)\right)$$

- That is, the *impurity measure* of a split (or a node) is the impurity function computed on probabilities associated (conditional) with a node.

TB 1.2.7 - Goodness of a split

- Once we have defined $i(t)$, we define the goodness of split s for node t, denoted by $\Phi(s,t)$:

$$\Phi(s,t) = \Delta i(s,t) = i(t) - p_R i\left(t_R\right) - p_L i\left(t_L\right)$$

- The best split for the single variable X_j is the one that has the largest value of $\Phi(s,t)$ over all $s \in \mathcal{S}_j$, the set of possible distinct splits for X_j.

TB 1.2.8 - Impurity score for a node

- The impurity, $i(t)$, of a node is based solely on the estimated posterior probabilities of the classes
  - That is, *it doesn't account for the size of* t.
- This is done by the *impurity score* of t, defined as $I(t) = i(t) \cdot p(t)$, a *weighted impurity measure* of node t that takes into account:
  - The estimated posterior probabilities of the classes,
  - The estimated proportion of data that go to node t.

TB 1.2.9 - Applications of $I(t)$

- $I(t)$ can be used to:
  - Define the aggregated impurity of a tree, by adding the impurity scores of all terminal leaves.
  - Provide a weighted measure of impurity decrease for a split: $\Delta I(s,t) = p(t)\Delta i(s,t)$.
  - Define a criteria for stop splitting a tree (see below).

TB 1.2.10 - Entropy as an impurity measure

- The entropy of a node, t, that is split in n child nodes t_1, t_2, ..., t_n, is:

$$\Phi_E(\mathbf{p}) = H(\mathbf{t}) = -\sum_{i=1}^{n} \underbrace{P(t_i)}_{p_i} \log_2 P(t_i)$$

TB 1.2.11 - Goodness of split based on entropy

- From here, an information gain (that is impurity decrease) measure can be introduced.
- Information theoretic approach that compares
  - the entropy of the parent node before the split to
  - that of a weighted sum of the child nodes after the split where the weights are proportional to the number of observations in each node.

TB 1.2.12 - Information gain

- For a split s and a set of observations (a node) t, information gain is defined as:

$$IG(t,s) = \text{ (original entr.) } - (\text{ entr. after split})$$

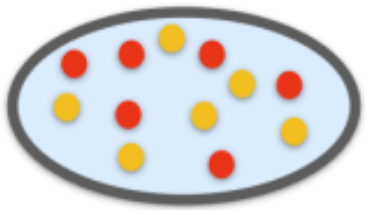
$$IG(t,s) = H(t) - \sum_{i=1}^{n} \frac{|t_i|}{t} H(x_i)$$

Example 1: Pears vs Apples

Consider the problem of designing an algorithm to automatically differentiate between apples and pears (class labels) given only their width and height measurements (features).

Width	Height	Fruit
7.1	7.3	Apple
7.9	7.5	Apple
7.4	7.0	Apple
8.2	7.3	Apple
7.6	6.9	Apple
7.8	8.0	Apple
7.0	7.5	Pear
7.1	7.9	Pear
6.8	8.0	Pear
6.6	7.7	Pear
7.3	8.2	Pear
7.2	7.9	Pear

Example 1. Entropy Calculation



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

$$= -\frac{6}{12} \cdot log_2(\frac{6}{12}) - \frac{6}{12} \cdot log_2(\frac{6}{12})$$

$$= \frac{1}{2} + \frac{1}{2} = 1$$

$$H(X) = -\overbrace{\frac{4}{7} \cdot log_2(\frac{4}{7})}^{\text{●}} - \overbrace{\frac{3}{7} \cdot log_2(\frac{3}{7})}^{\text{●}}$$

$$= 0.4613 + 0.5239 = 0.9852$$

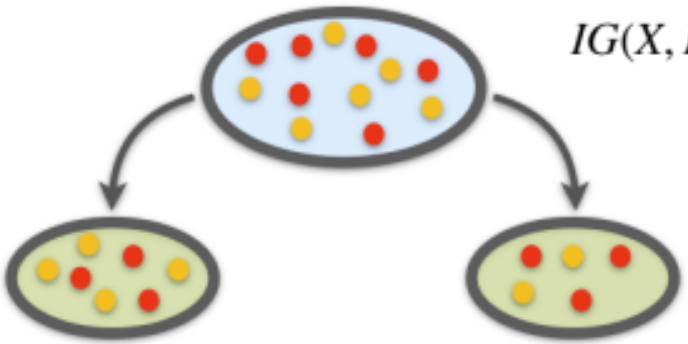
$$H(X) = -\overbrace{\frac{3}{5} \cdot log_2(\frac{3}{5})}^{\text{●}} - \overbrace{\frac{2}{5} \cdot log_2(\frac{2}{5})}^{\text{●}}$$

$$= 0.4422 + 0.5288 = 0.9710$$

$$H(X) = -\frac{6}{6} \cdot log_2(\frac{6}{6}) = 0$$

$$H(X) = -\frac{6}{6} \cdot log_2(\frac{6}{6}) = 0$$

Example 1. Information Gain

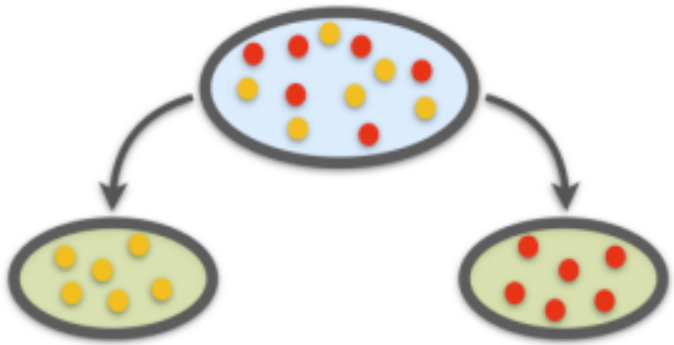


$$IG(X,F) = (original\ entropy) - (entropy\ after\ split)$$

$$= H(X) - \sum_{i=1}^{n} \frac{|x_i|}{X} H(x_i)$$

$$= 1 - \frac{7}{12} \cdot (0.9852) - \frac{5}{12} \cdot (0.9710) = 0.0213$$

$$IG(X,F) = 1 - \frac{6}{12} \cdot (0) - \frac{6}{12} \cdot (0) = 1$$



TB. 1.3.1 - When to stop growing

- Maximizing information gain is one possible criteria to choose among splits.
- In order to avoid excessive complexity it is usually decided to stop splitting when *information gain does not compensate for increase in complexity*.

TB 1.3.2 Stop splitting criteria

- In practice, stop splitting is decided when:

$$\max_{s \in S} \Delta I(s,t) < \beta,$$

where:

- ΔI represents the information gain associated with an optimal split s and a node t,
- and β is a pre-determined threshold.

TB 1. Summary

- Decision trees are built by iteratively partitioning data into smaller regions based on feature values.
- Splits are aimed at producing purer nodes, that contains mostly data from one class.
- Homogeneity is measured by impurity functions such as *Entropy*, *Gini Index* or *Misclassification rate*
- The best split is chosen from candidate splits as the one that maximizes impurity reduction for example through *Information Gain*
- Tree growth continues until impurity reduction is minimal, or a predefined depth or node size threshold is reached.

Example 2. The PIMA database

- The Pima Indian Diabetes dataset contains 768 individuals (female) and 9 clinical variables.

```
Rows: 768
Columns: 9
$ pregnant <dbl> 6, 1, 8, 1, 0, 5, 3, 10, 2, 8, 4, 10, 10,
$ glucose  <dbl> 148, 85, 183, 89, 137, 116, 78, 115, 197,
$ pressure <dbl> 72, 66, 64, 66, 40, 74, 50, NA, 70, 96, 92
$ triceps  <dbl> 35, 29, NA, 23, 35, NA, 32, NA, 45, NA, NA
$ insulin  <dbl> NA, NA, NA, 94, 168, NA, 88, NA, 543, NA,
$ mass     <dbl> 33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31.0,
$ pedigree <dbl> 0.627, 0.351, 0.672, 0.167, 2.288, 0.201,
$ age      <dbl> 50, 31, 32, 21, 33, 30, 26, 29, 53, 54, 30
$ diabetes <fct> pos, neg, pos, neg, pos, neg, pos, neg, po
```

Example 2. Looking at the data

- These Variables are known to be related with cardiovascular diseases.
- It seems intuitive to use these variables to decide if a person is affected by diabetes

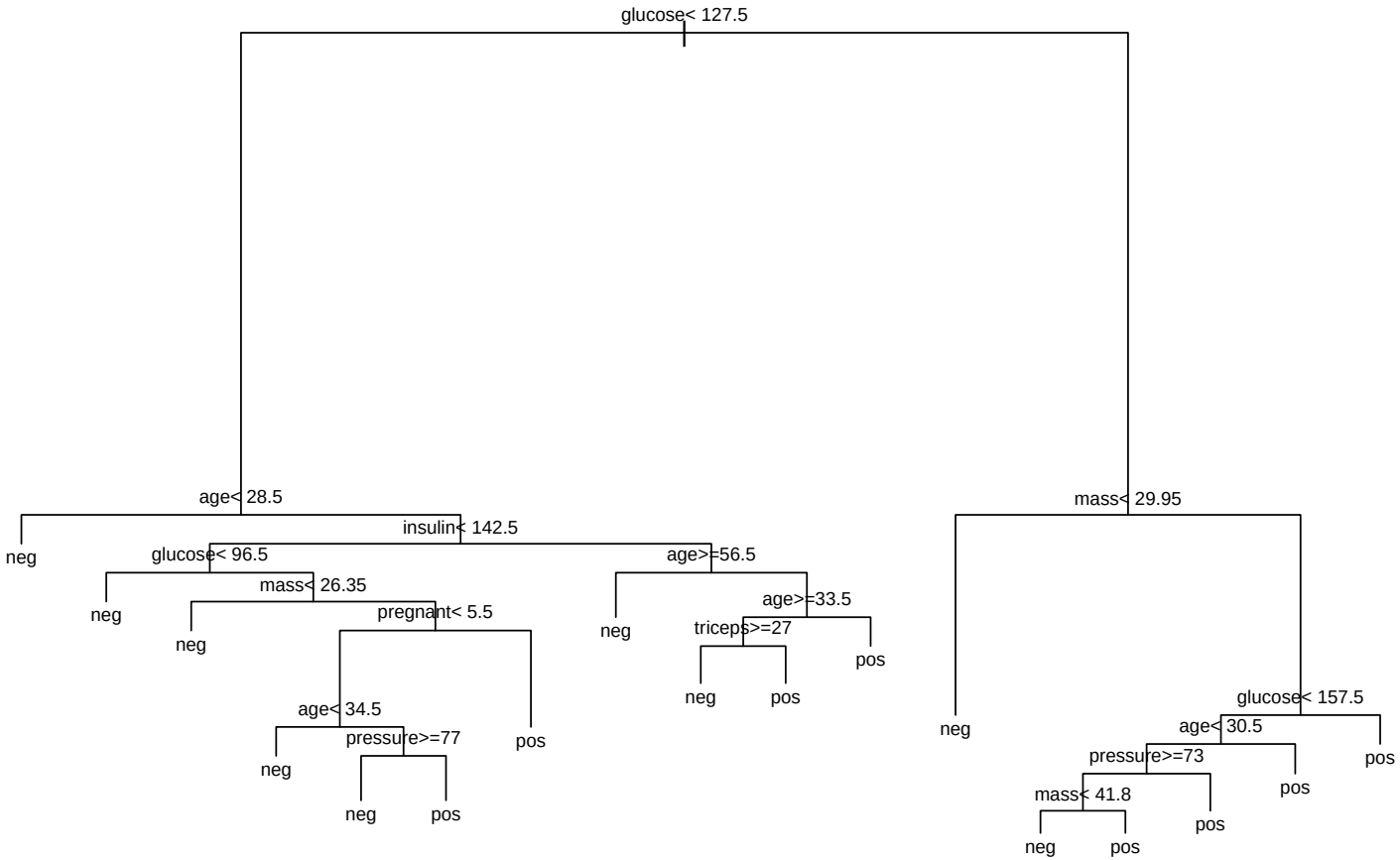
	p0	p25	p50	p75	p100	hist
diabetes	NA	NA	NA	NA	NA	<NA>
pregnant	0.000	1.00000	3.0000	6.00000	17.00	
glucose	44.000	99.00000	117.0000	141.00000	199.00	
pressure	24.000	64.00000	72.0000	80.00000	122.00	
triceps	7.000	22.00000	29.0000	36.00000	99.00	
insulin	14.000	76.25000	125.0000	190.00000	846.00	
mass	18.200	27.50000	32.3000	36.60000	67.10	
pedigree	0.078	0.24375	0.3725	0.62625	2.42	
age	21.000	24.00000	29.0000	41.00000	81.00	

Ex. 2. Building a classification tree

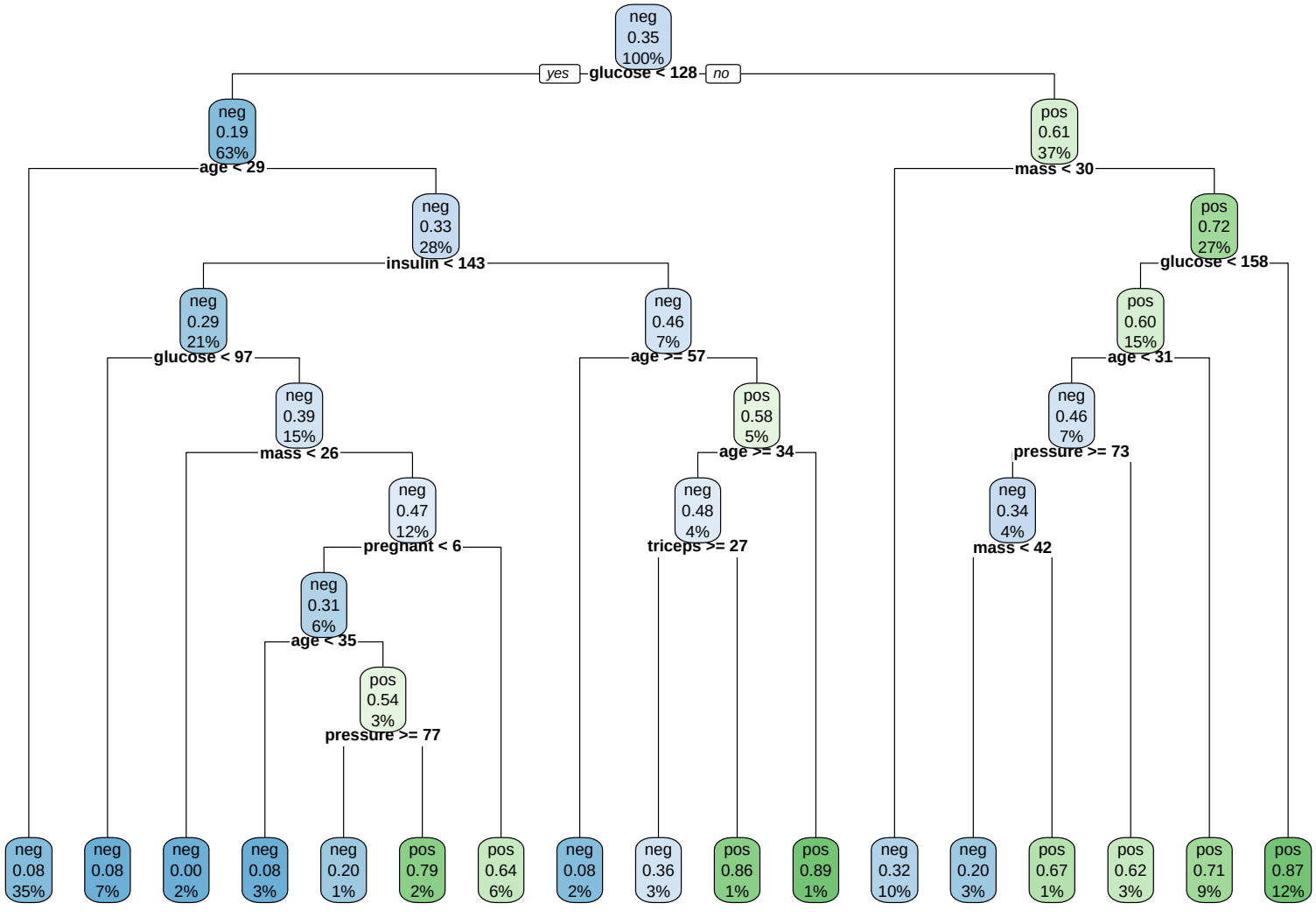
- We wish to predict the probability of individuals in being diabete-positive or negative.
    - We start building a tree with all the variables

```
library(rpart)
model1 <- rpart(diabetes ~., data = PimaIndiansDiabetes2)
```

Ex.2. Plotting the tree (1)



Ex. 2. Plotting the tree (Nicer)



TB 2 - Class Assignment

- The decision tree classifies new data points as follows.
  - We let a data point pass down the tree and see which leaf node it lands in.
  - The class of the leaf node is assigned to the new data point. Basically, all the points that land in the same leaf node will be given the same class.
  - This is similar to k-means or any prototype method.

TB 2.1 - Class Assignment Rules

- A class assignment rule assigns a class $j = 1, \cdots, K$ to every terminal (leaf) node $t \in \tilde{T}$.
- The class is assigned to node t is denoted by $\kappa(t)$,
  - E.g., if $\kappa(t) = 2$, all the points in node t would be assigned to class 2.
- If we use 0-1 loss, the class assignment rule picks the class with maximum posterior probability:

$$\kappa(t) = \arg\max_j p(j \mid t)$$

TB 2.2. Estimating the error rate (1)

- Let's assume we have built a tree and have the classes assigned for the leaf nodes.
- Goal: estimate *the classification error rate* for this tree.
- We use the *resubstitution estimate* $r(t)$ for the probability of misclassification, given that a case falls into node t. This is:

$$r(t) = 1 - \max_j p(j \mid t) = 1 - p(\kappa(t) \mid t)$$

TB 2.3. Estimating the error rate (2)

- Denote $R(t) = r(t)p(t)$, that is the miscclassification error rate weighted by the probability of the node.
- The resubstitution estimation for the overall misclassification rate $R(T)$ of the tree classifier T is:

$$R(T) = \sum_{t \in \tilde{T}} R(t)$$

Example 2: Individual prediction

Consider individuals 521 and 562

	pregnant	glucose	pressure	triceps	insulin	mass	pedigree
521	2	68	70	32	66	25.0	0.187
562	0	198	66	32	274	41.3	0.502

```
521 562
neg pos
Levels: neg pos
```

- If we follow individuals 521 and 562 along the tree, we reach the same prediction.
- The tree provides not only a classification but also an explanation.

Example 2: How accurate is the model?

- It is straightforward to obtain a simple performance measure.

```
predicted.classes<- predict(model1, PimaIndiansDiabetes2, '
mean(predicted.classes == PimaIndiansDiabetes2$diabetes)
```

```
[1] 0.8294271
```

Example 2: Is the Tree optimal?

- The question becomes harder when we go back and ask if *we obtained the best possible tree*.
- In order to answer this question we must study tree construction in more detail.

TB 3.1 Optimizing the Tree

- Trees obtained by looking for optimal splits tend to overfit: good for the data in the tree, but generalize badly and tend to fail more in predictions.
- In order to reduce complexity and overfitting, while keeping the tree as good as possible, tree *pruning* may be applied.
- Pruning works *removing branches that are unlikely to improve the accuracy* of the model on new data.

TB 3.2 Pruning methods

- There are different pruning methods, but the most common one is the *cost-complexity* pruning algorithm, also known as the *weakest link pruning*.
- The algorithm works by adding a penalty term to the misclassification rate of the terminal nodes:

$$R_\alpha(T) = R(T) + \alpha|T|$$

where α is the parameter that controls the trade-off between tree complexity and accuracy.

TB 3.3 Cost complexity pruning

- Start by building a large tree that overfits the data.
- Then, use cross-validation to estimate the optimal value of alpha that minimizes the generalization error.
- Finally, prune the tree by removing the branches that have a smaller improvement in impurity than the penalty term multiplied by alpha.
- Iterate the process until no more branches can be pruned, or until a minimum tree size is reached.

Regression modelling with trees

- When the response variable is numeric, decision trees are *regression trees*.
- Option of choice for distinct reasons
  - The relation between response and potential explanatory variables is not linear.
  - Perform automatic variable selection.
  - Easy to interpret, visualize, explain.
  - Robust to outliers and can handle missing data

Classification vs Regression Trees

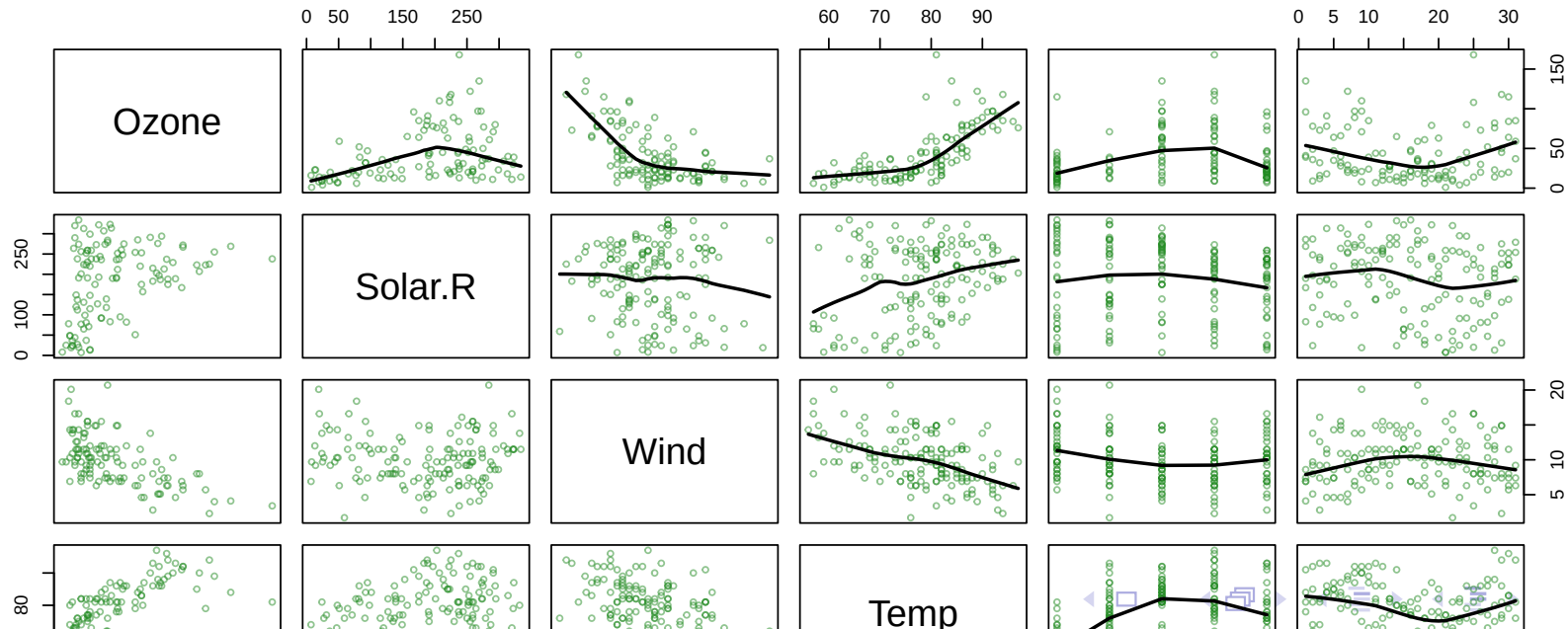
Aspect	**Regression Trees**	**Classification Trees**
Outcome var. type	Continuous	Categorical
Goal	To predict a numerical value	To predict a class label
Splitting criteria	Mean Squared Error, Mean Abs. Error	Gini Impurity, Entropy, etc.
Leaf node prediction	Mean or median of the target variable in that region	Mode or majority class of the target variable ...
Examples of use cases	Predicting housing prices, predicting stock prices	Predicting customer churn, predicting high/low risk in diease
Evaluation metric	Mean Squared Error, Mean Absolute Error, R-square	Accuracy, Precision, Recall, F1-score, etc.

Regression tree example

- The `airquality` dataset from the `datasets` package contains daily air quality measurements in New York from May through September of 1973 (153 days).
- The main variables include:
  - Ozone: the mean ozone (in parts per billion) ...
  - Solar.R: the solar radiation (in Langleys) ...
  - Wind: the average wind speed (in mph) ...
  - Temp: the maximum daily temperature (ºF) ...
- Main goal : Predict ozone concentration.

Non linear relationships!

```
aq <- datasets::airquality
color <- adjustcolor("forestgreen", alpha.f = 0.5)
ps <- function(x, y, ...) {  # custom panel function
  panel.smooth(x, y, col = color, col.smooth = "black", cex
}
pairs(aq, cex = 0.7, upper.panel = ps, col = color)
```



Building the tree (1): Splitting

- Consider:
  - all predictors $X_1, \ldots, X_n$, and
  - all values of cutpoint s for each predictor and
- For each predictor find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by:

$$\sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \hat{y}_{R_j} \right)^2$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the j th box.

Building the tree (2): Splitting

- To do this, define the pair of half-planes

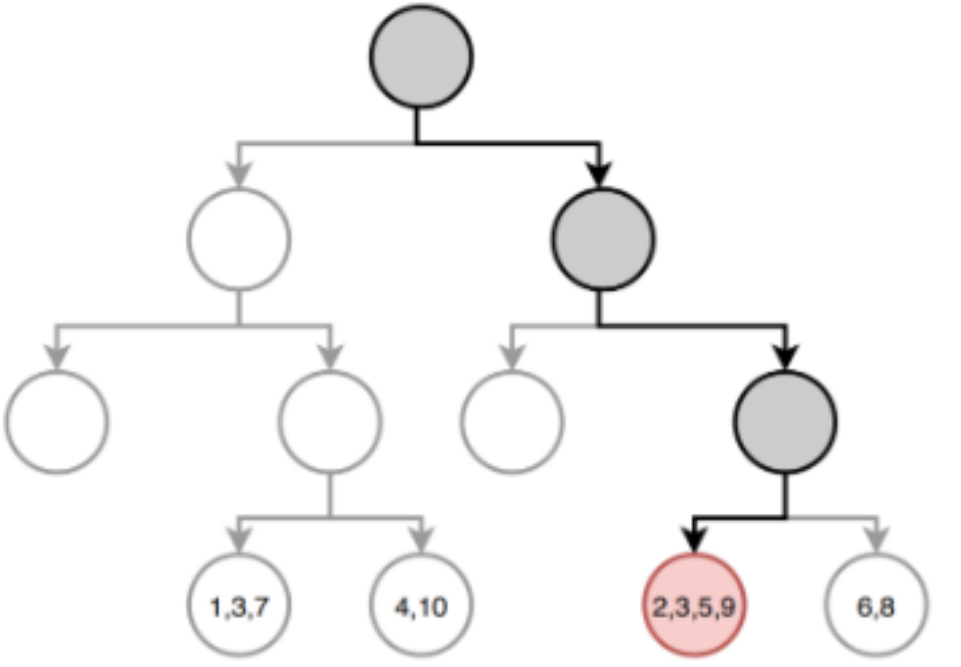
$$R_1(j,s) = \{X \mid X_j < s\} \text{ and } R_2(j,s) = \{X \mid X_j \geq s\}$$

and seek the value of j and s that minimize the equation:

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

Building the tree (3): Prediction

- Once the regions have been created we predict the response using the mean of the trainig observations *in the region to which that observation belongs*.
- In the example, for an observation belonging to the shaded region, the prediction would be:



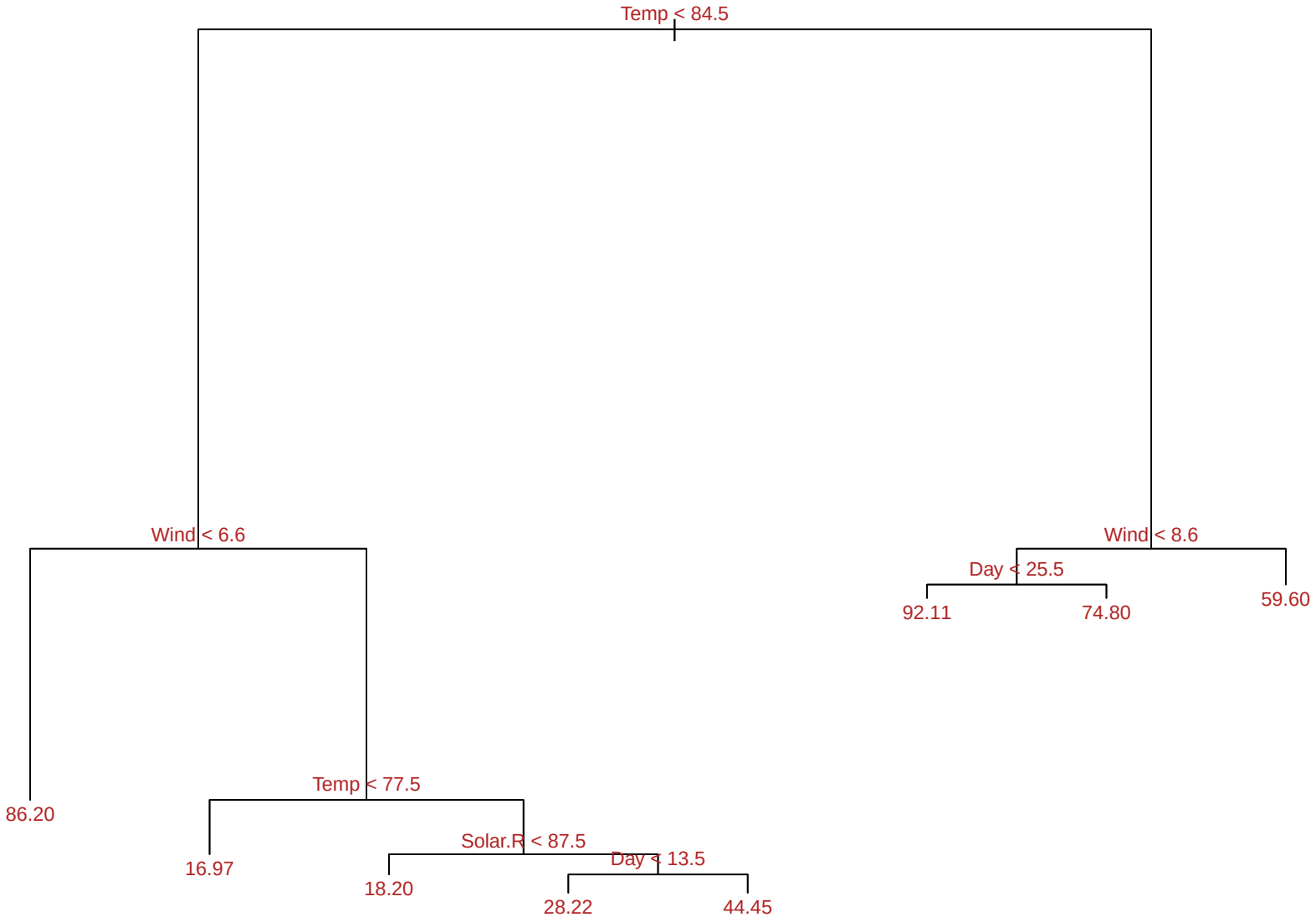
$$\hat{y} = \frac{1}{4}(y_2 + y_3 + y_5 + y_9)$$

Example: A regression tree

```
set.seed(123)
train <- sample(1:nrow(aq), size = nrow(aq)*0.7)
aq_train <- aq[train,]
aq_test  <- aq[-train,]
aq_regresion <- tree::tree(formula = Ozone ~ .,
                           data = aq_train, split = "devian
summary(aq_regresion)
```

```
Regression tree:
tree::tree(formula = Ozone ~ ., data = aq_train, split = "d
Variables actually used in tree construction:
[1] "Temp"    "Wind"    "Solar.R" "Day"
Number of terminal nodes:  8
Residual mean deviance:  285.6 = 21420 / 75
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
```

Example: Plot the tree



Prunning the tree (1)

- As before, *cost-complexity prunning* can be applied
- We consider a sequence of trees indexed by a nonnegative tuning parameter α.
- For each value of α there corresponds a subtree $T \subset T_0$ such that:

$$\sum_{m=1}^{|T|} \sum_{y_i \in R_m} \left(y_i - \hat{y}_{R_m} \right)^2 + \alpha |T| \quad (*)$$

is as small as possible.

Tuning parameter α

- α controls a trade-off between the subtree's complexity and its fit to the training data.
- When $\alpha = 0$, then the subtree T will simply equal T_0.
- As α increases, there is a price to pay for having a tree with many terminal nodes, and so (*) will tend to be minimized for a smaller subtree.
- Equation (*1) is reminiscent of the lasso.
- α can be chosen by cross-validation .

Optimizing the tree (α)

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:
    1. Repeat Steps 1 and 2 on all but the k th fold of the training data.
    2. Evaluate the mean squared prediction error on the data in the left-out k th fold, as a function of α.

Average the results for each value of α. Pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α

Example: Prune the tree

```
cv_aq <- tree::cv.tree(aq_regresion, K = 5)
optimal_size <-  rev(cv_aq$size)[which.min(rev(cv_aq$dev))]
aq_final_tree <- tree::prune.tree(
                  tree = aq_regresion,
                  best = optimal_size
                )
summary(aq_final_tree)
```

```
Regression tree:
tree::tree(formula = Ozone ~ ., data = aq_train, split = "d
Variables actually used in tree construction:
[1] "Temp"    "Wind"    "Solar.R" "Day"
Number of terminal nodes:  8
Residual mean deviance:  285.6 = 21420 / 75
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
```

Trees have many advantages

- Trees are very easy to explain to people.
- Decision trees may be seen as good mirrors of human decision-making.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert.
- Trees can easily handle qualitative predictors without the need to create dummy variables.

But they come at a price

- Trees generally do not have the same level of predictive accuracy as sorne of the other regression and classification approaches.
- Additionally, trees can be very non-robust: a small change in the data can cause a large change in the final estimated tree.

References

- A. Criminisi, J. Shotton and E. Konukoglu (2011) Decision Forests for Classifcation, Regression ... Microsoft Research technical report TR-2011-114
- Efron, B., Hastie T. (2016) Computer Age Statistical Inference. Cambridge University Press. Web site
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction. Springer.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112). Springer. Web site

Complementary references

- Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). Classification and regression trees. CRC press.
- Brandon M. Greenwell (202) Tree-Based Methods for Statistical Learning in R. 1st Edition. Chapman and Hall/CRC DOI: https://doi.org/10.1201/9781003089032
- Genuer R., Poggi, J.M. (2020) Random Forests with R. Springer ed. (UseR!)

Resources

- Applied Data Mining and Statistical Learning (Penn Statte-University)
- R for statistical learning
- CART Model: Decision Tree Essentials
- An Introduction to Recursive Partitioning Using the RPART Routines