

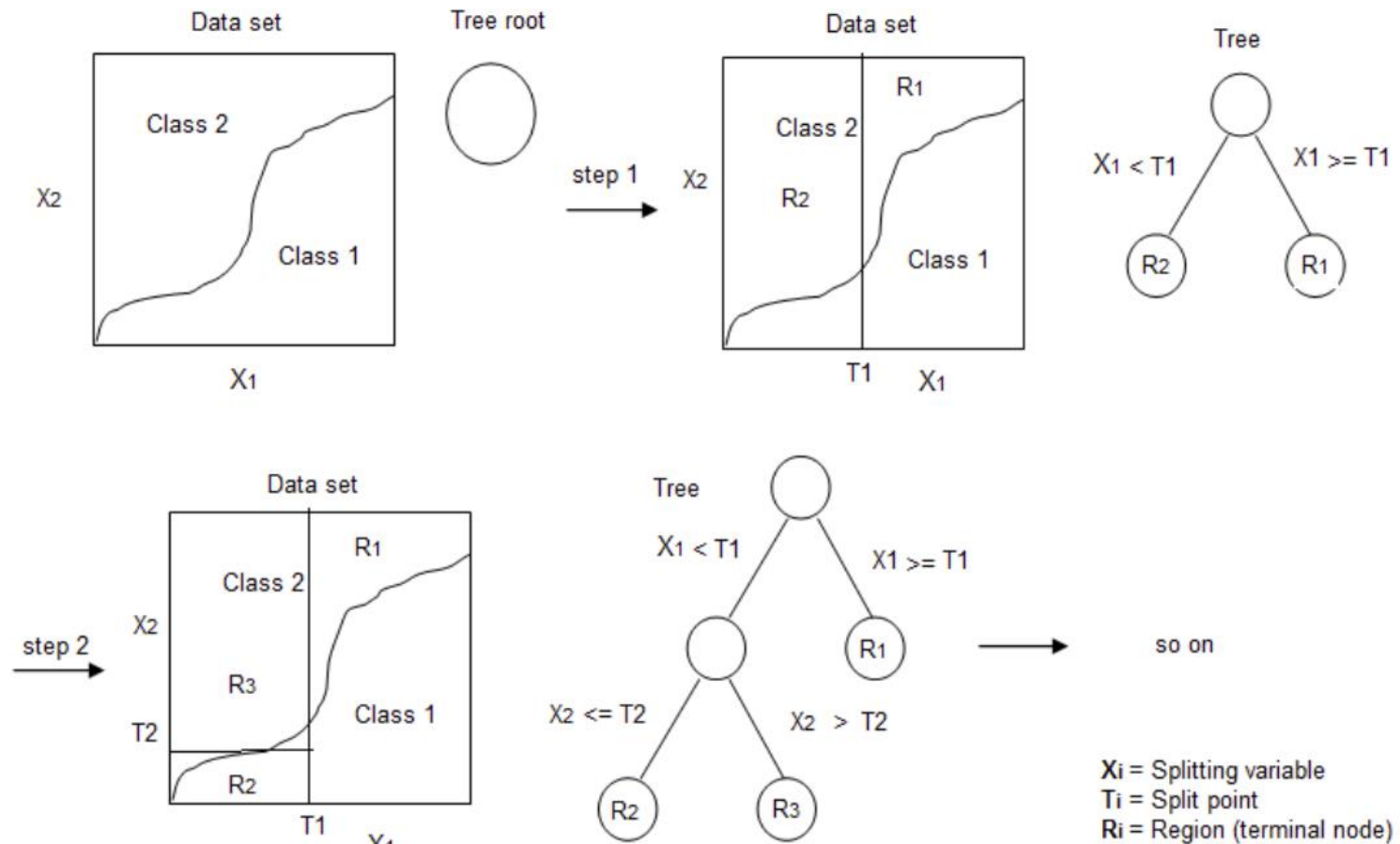
Tree Based Methods

Machine Learning II

Master in Business Analytics and Big Data

acastellanos@faculty.ie.edu

Overview



Prediction via Stratification

1. Divide the set of possible values for the different **predictors** (X_1, X_2, \dots, X_p) into **J non-overlapping regions** (R_1, R_2, \dots, R_J).
2. For every observation that falls into region R_J , the **same prediction is associated** (the mean of the response values for the observations in that region).
3. Repeat the process within each of the resulting regions.

Why partition the data space?

Linear regression is global model

$$f(x) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p$$

where a **single formula applies to the entire data space**.

It is possible to incorporate interaction

$$f(x) = \beta_0 + \beta_1 X_1 \cdot X_2 + \dots$$

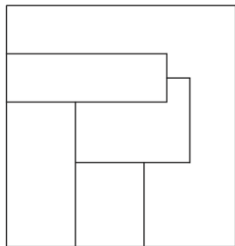
...but the number of possibilities is very large, adding complicated non-linear interactions difficult to anticipate.

Alternative:

- Sub-divide (**partition**) the space into **smaller regions** again and again (**recursively**) where interactions are easier to manage, and a **simple model** that fits can be obtained.

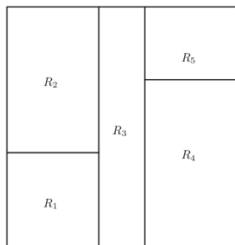
Why *top-down-greedy-recursive-binary-splitting*?

NOT Binary splitting



Computed using *non-binary* approach. Computationally expensive (unfeasible?)

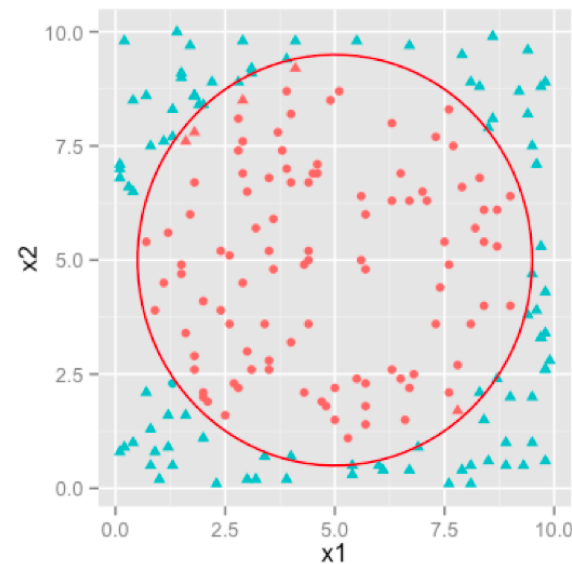
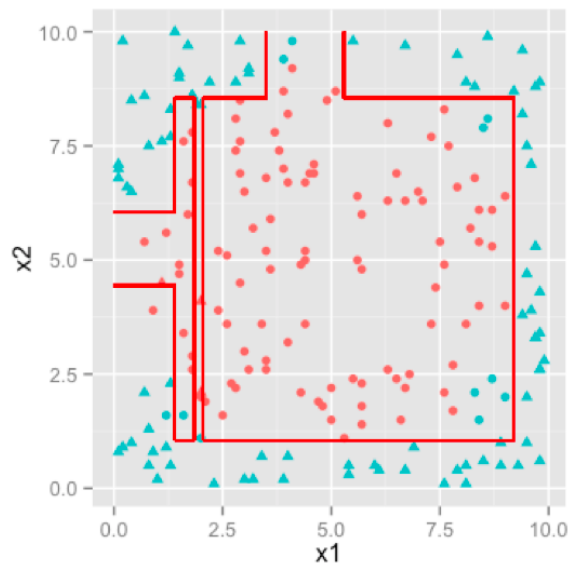
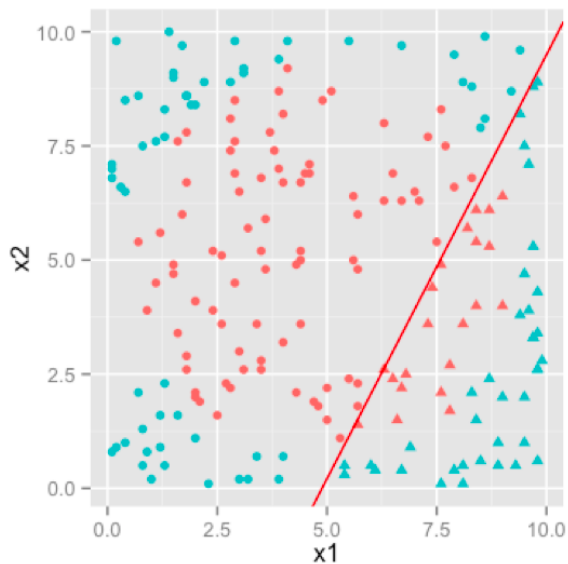
Binary splitting



Computed using *top down, greedy, recursive binary* approach

- **Top down**, because we start with a single partition where all observations belong to (root of tree)
- **Greedy** because, instead of looking ahead and picking a split that lead to the better tree, the method selects the best split at each step.
- **Recursive** because the method works on every region defined at each step.
- **Binary**, because the cost function separates the space into two half-planes.

Logistic Reg. vs. Trees vs. SVM



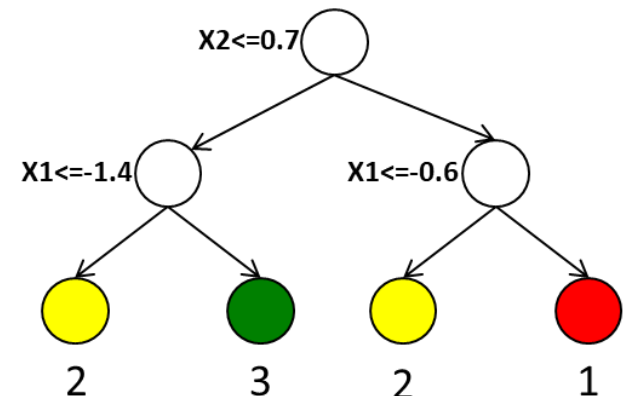
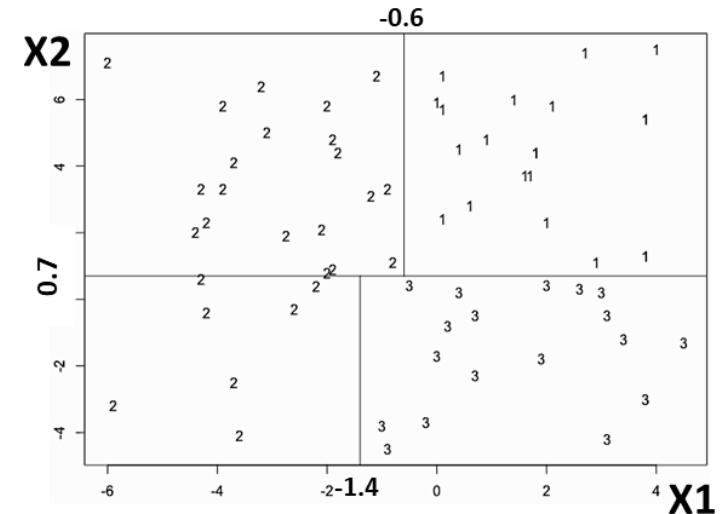
The tree-based model

The result is:

- The **tree**, which represents the recursive partition method
- The **leaves** of the tree, which represent the simple model that applies to each final smaller data region

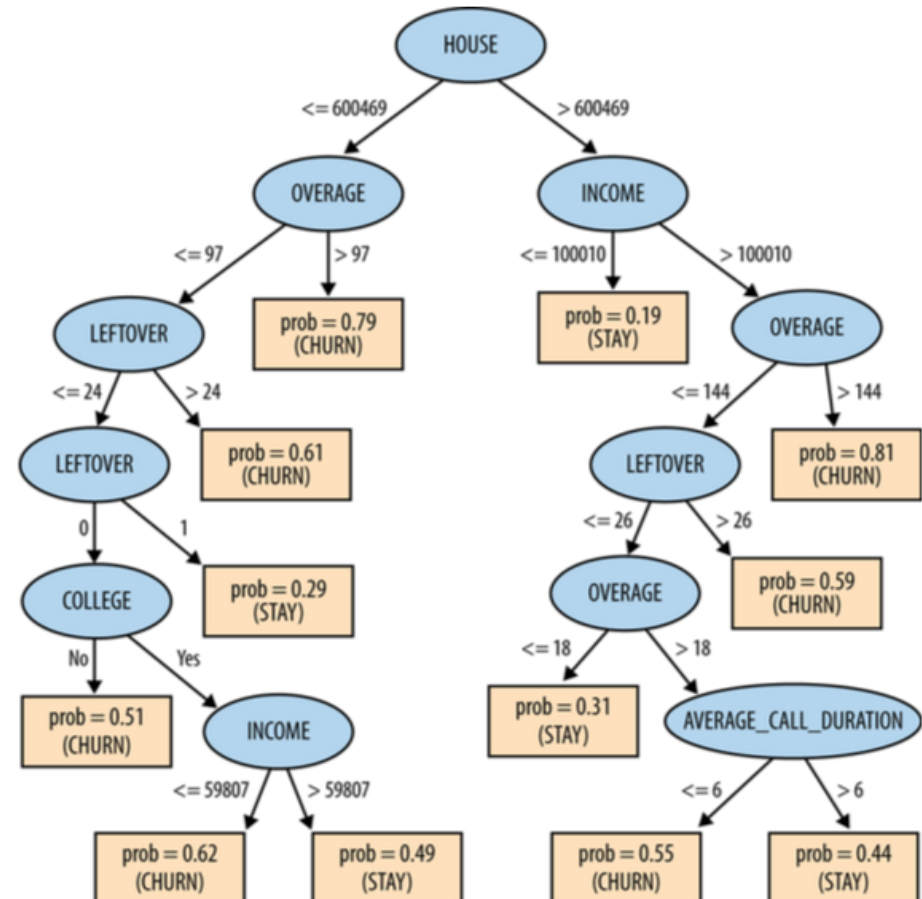
Interpretation

- Starting at the root node, a sequence of **questions** about the predictors are made (all questions are labeled within tree nodes)
- The **answers** are labeled in the branches between nodes



Real application example

Variable	Explanation
COLLEGE	Is the customer college educated?
INCOME	Annual income
OVERAGE	Average overcharges per month
LEFTOVER	Average number of leftover minutes per month
HOUSE	Estimated value of dwelling (from census tract)
HANDSET_PRICE	Cost of phone
LONG_CALLS_PER_MONTH	Average number of long calls (15 mins or over) per month
AVERAGE_CALL_DURATION	Average duration of a call
REPORTED_SATISFACTION	Reported level of satisfaction
REPORTED_USAGE_LEVEL	Self-reported usage level
LEAVE (Target variable)	Did the customer stay or leave (churn)?




Guess what users will churn



Regression Trees

- Let's use an example:

- Predict the annual spending on Fresh products
- The data set includes the annual spending in monetary units on diverse product categories.




Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
2	3	12669	9656	7561	214	2674	1338
2	3	7057	9810	9568	1762	3293	1776
2	3	6353	8808	7684	2405	3516	7844
1	3	13265	1196	4221	6404	507	1788
2	3	22615	5410	7198	3915	1777	5185
2	3	9413	8259	5126	666	1795	1451
2	3	12126	3199	6975	480	3140	545
2	3	7570	4956	9426	1669	3321	2566

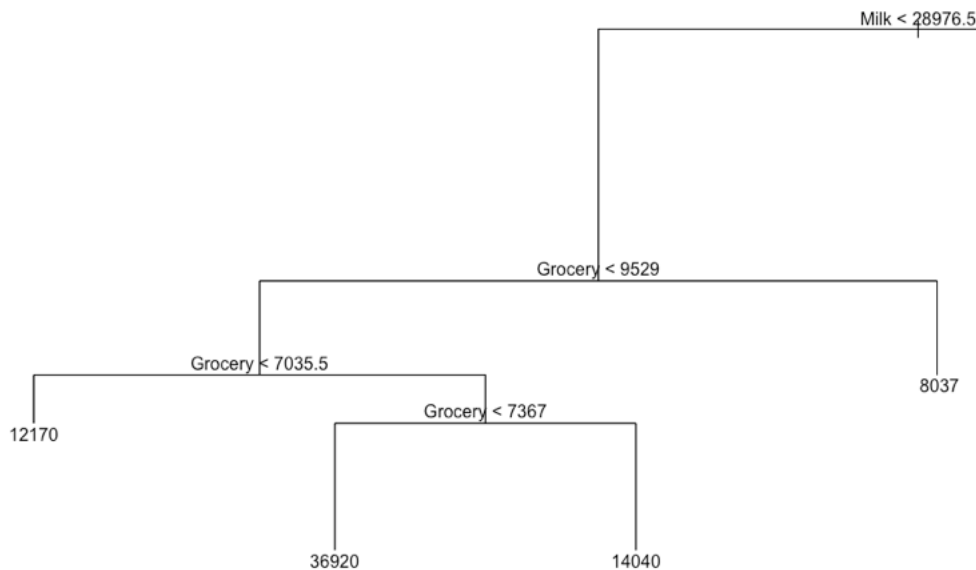
- We will try to **predict Fresh** from the annual spending on Grocery and Milk
- We will later **predict Channel** (categorical variable!!) from the annual spending on Grocery and Milk

Regression Trees

```
t <- tree(Fresh ~ Grocery+Milk)
plot(t);
text(t)
```



Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
2	3	12669	9656	7561	214	2674	1338
2	3	7057	9810	9568	1762	3293	1776
2	3	6353	8808	7684	2405	3516	7844
1	3	13265	1196	4221	6404	507	1788
2	3	22615	5410	7198	3915	1777	5185
2	3	9413	8259	5126	666	1795	1451
2	3	12126	3199	6975	480	3140	545
2	3	7570	4056	9476	1660	3371	2566



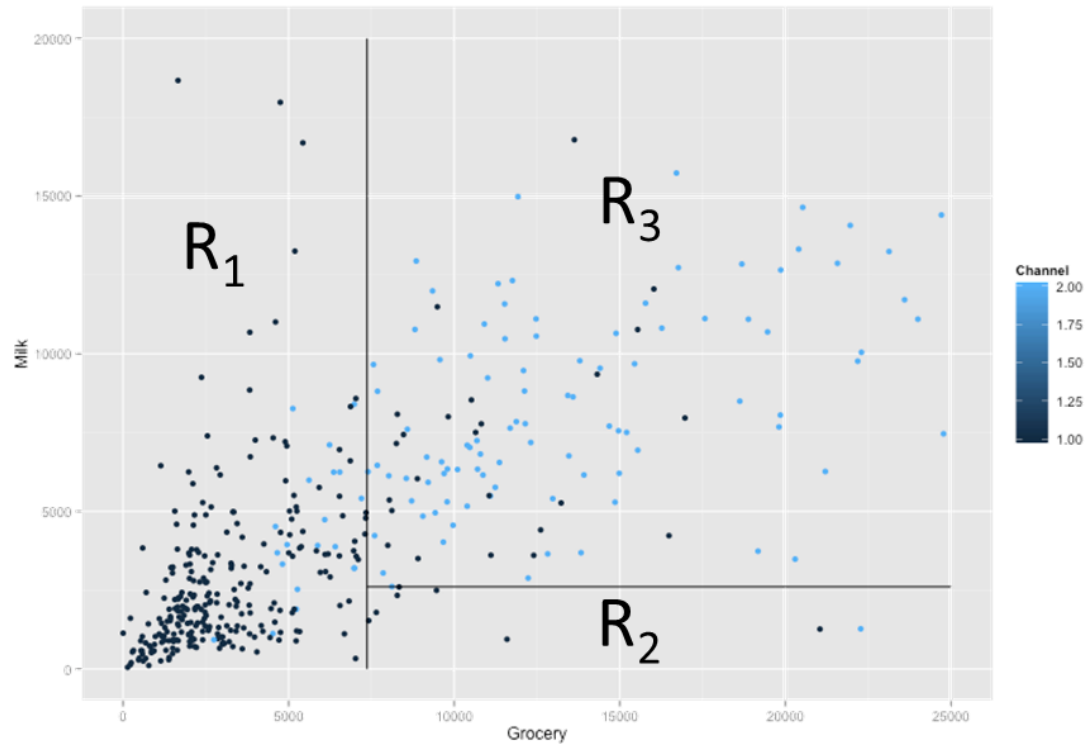
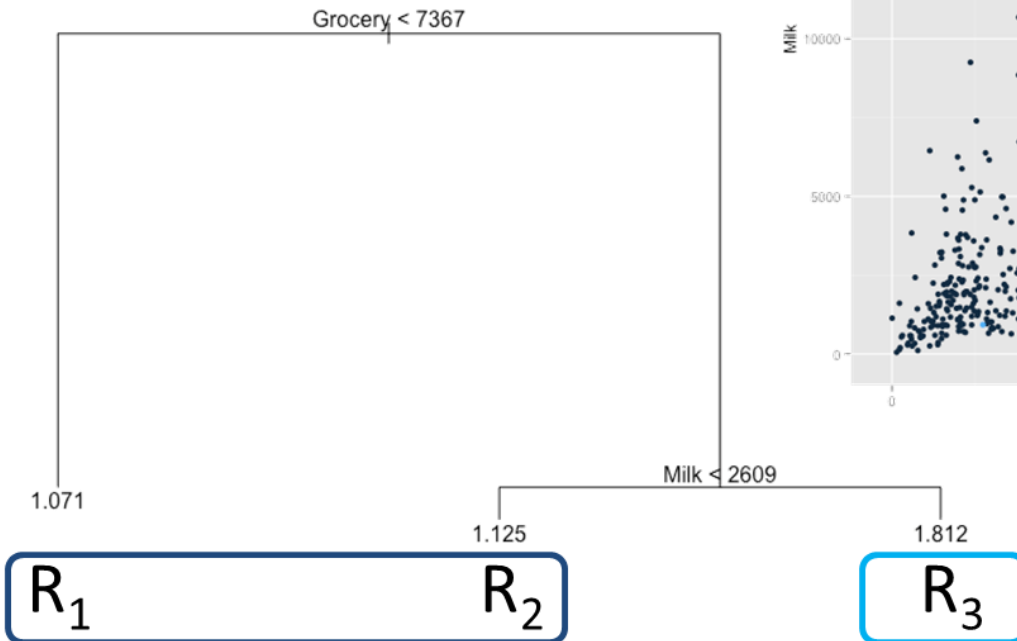
The **mean response value** for the annual spend on Fresh products in the data set with **Milk \geq 28976.5**

Classification Trees

- Very similar to a regression tree, except that it predicts **qualitative** response.
- Predict that each observation belong to the ***most commonly occurring class*** of training observations in the region to which it belongs.
- From classification trees, we're interested in
 - Class **Prediction**
 - Class **Proportion**

Classification Trees

```
t <- tree(Channel~Grocery+Milk)
plot(t);
text(t)
```



Cost function

The goal is to find boxes that minimize the RSS:

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

Where \hat{y}_{R_j} is the **mean** response of the training observations that belong to the same terminal node, and that is the uniform **prediction** made for that class

At each step, ALL possible predictors “**j**” and all possible cutpoint values “**s**”, such as we found the pair of half-planes

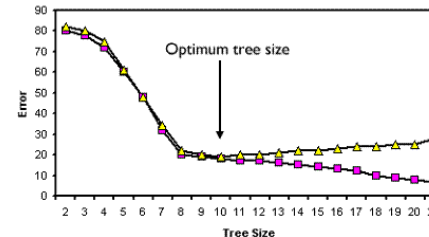
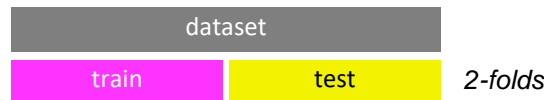
$$R_1(j, s) = \{X \mid X_j < s\} \text{ and } R_2(j, s) = \{X \mid X_j \geq s\}$$

We seek the values of “**j**” and “**s**” that **minimize the expression**

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

Tree pruning

- Trees **normally overfit the data**, and result in too complex interpretations. Simply **making the tree a bit smaller** lead to **lower variance** and better interpretability, though at the cost of a little **bias**.
- Best strategy to **prune** the tree?
 - Estimate **cross-validation** error for every possible sub-tree.

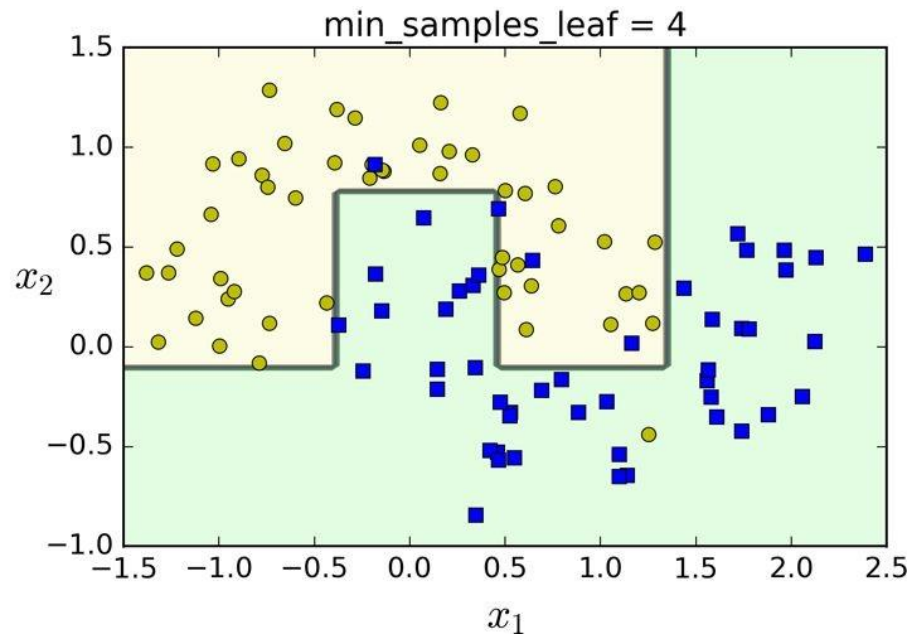
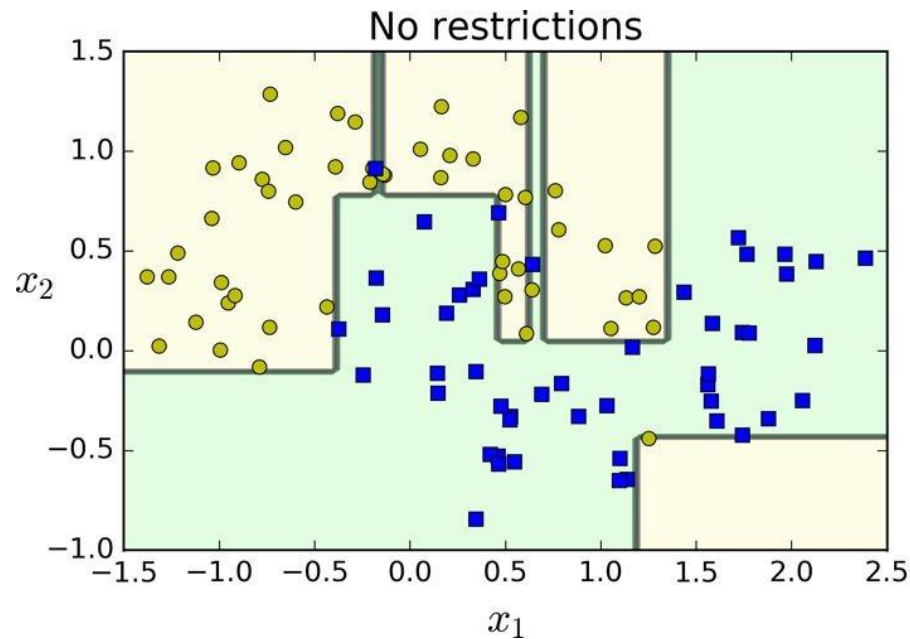


- Use **cost-complexity pruning**: consider sub-trees, sorted by a cost-complexity factor, instead of considering every possible sub-tree.

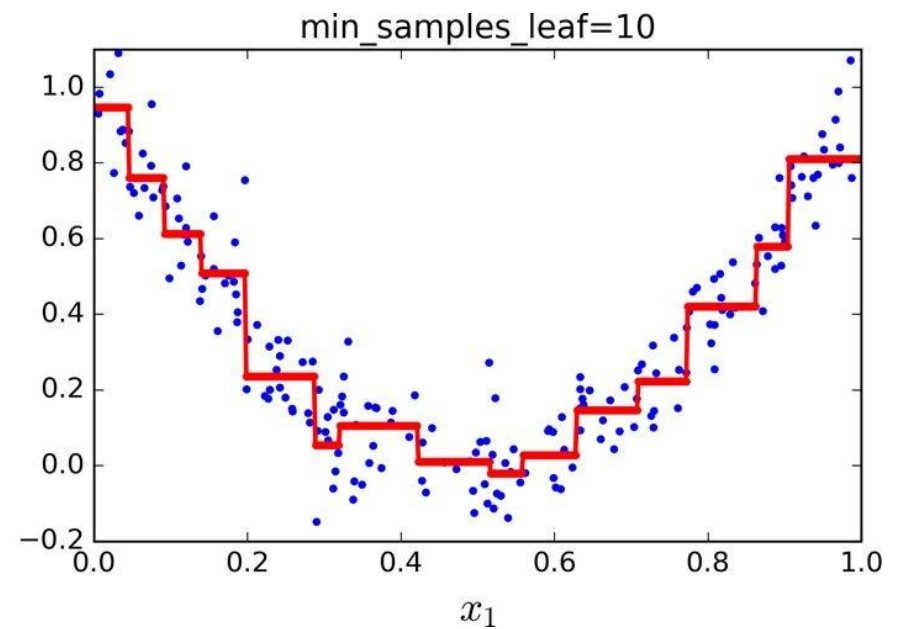
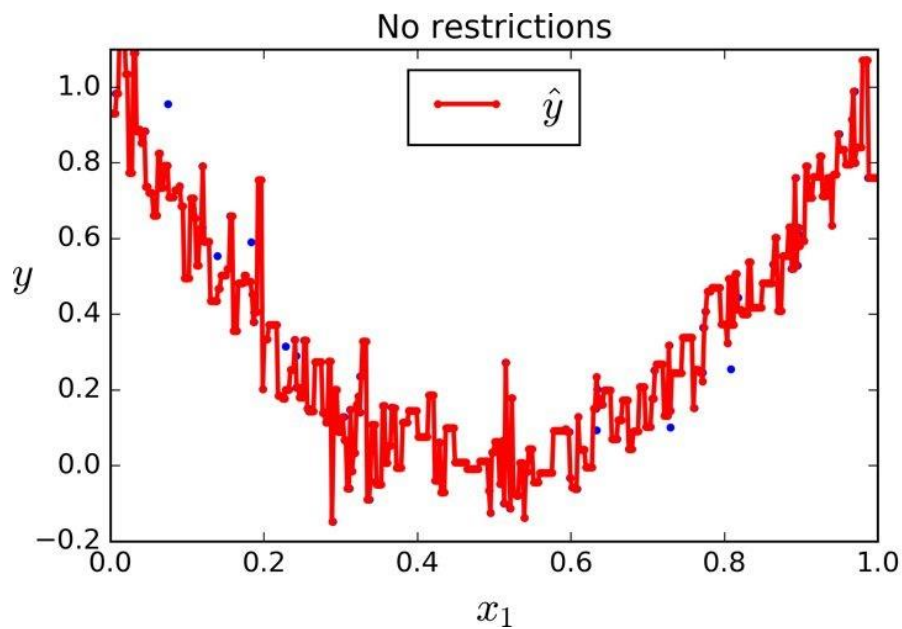
$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + a|T|$$

Tuning parameter
of leaves

Classification Tree pruning



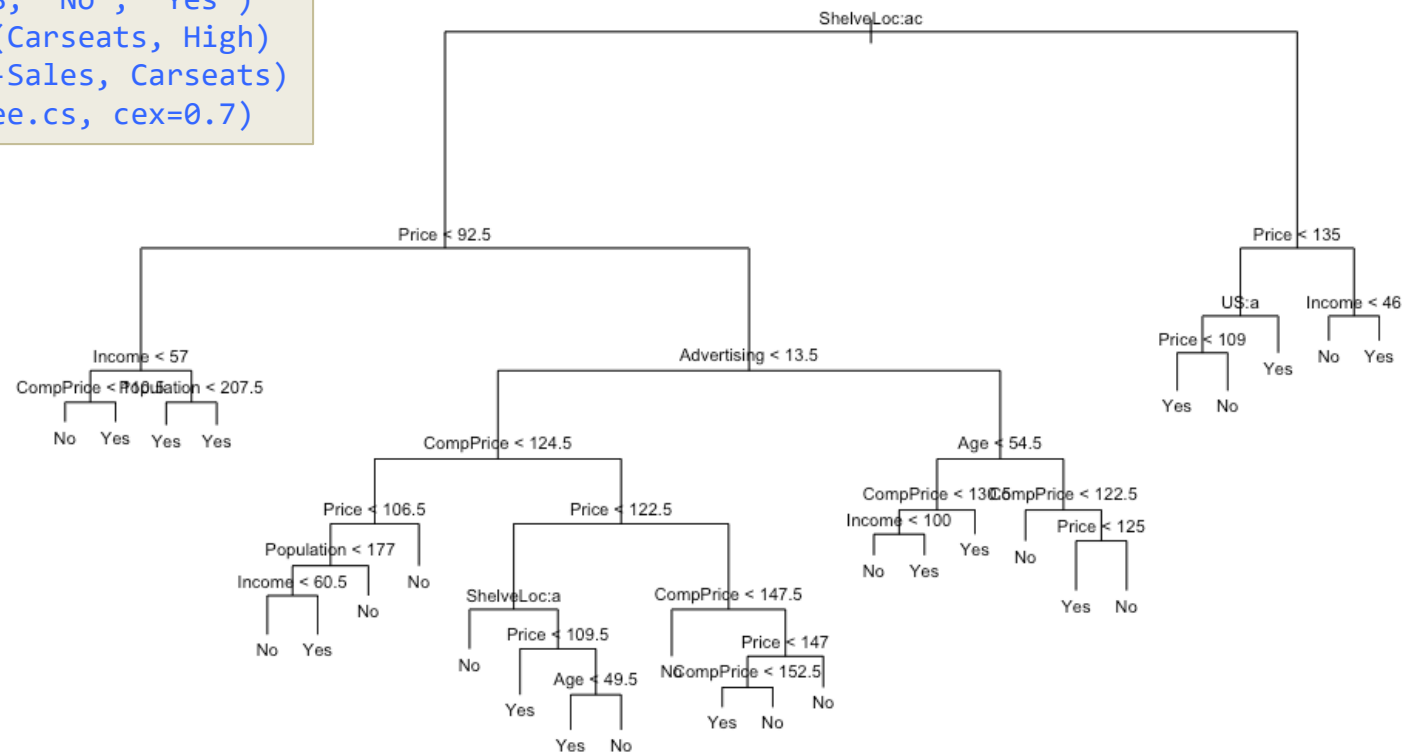
Regression Tree Pruning



Predict if Sales are High (> 8) or Not.

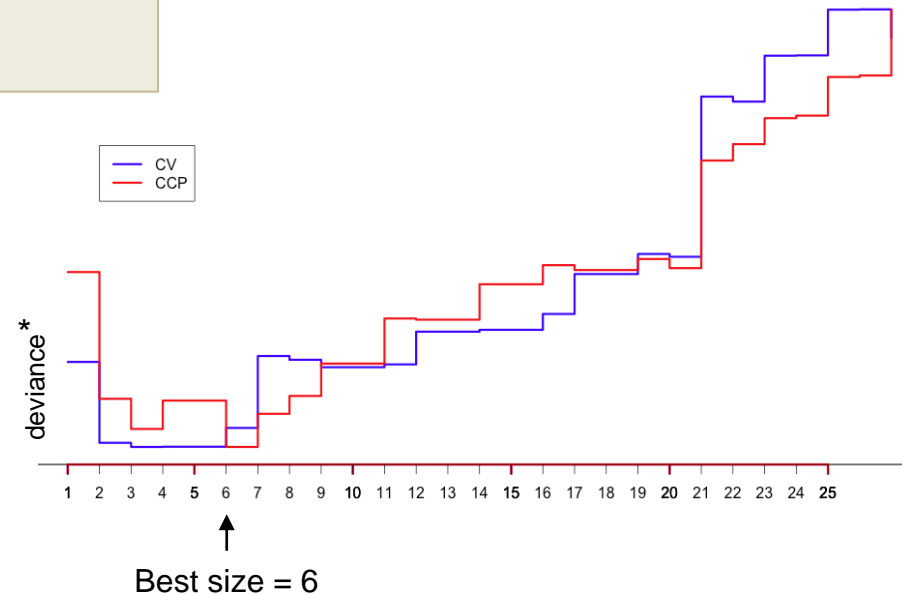
Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US	High
9.50	138	73	11	276	120	Bad	42	17	Yes	Yes	Yes
11.22	111	48	16	260	83	Good	65	10	Yes	Yes	Yes
11.22	113	35	10	269	80	Medium	59	12	Yes	Yes	Yes
11.22	117	100	4	466	97	Medium	55	14	Yes	Yes	No
11.22	141	64	3	340	128	Bad	38	13	Yes	No	No

```
library(MASS)
library(tree)
High <- ifelse(Sales<=8, "No", "Yes")
Carseats <- data.frame(Carseats, High)
tree.cs <- tree(High~.-Sales, Carseats)
plot(tree.cs); text(tree.cs, cex=0.7)
```



Prune strategy (R)

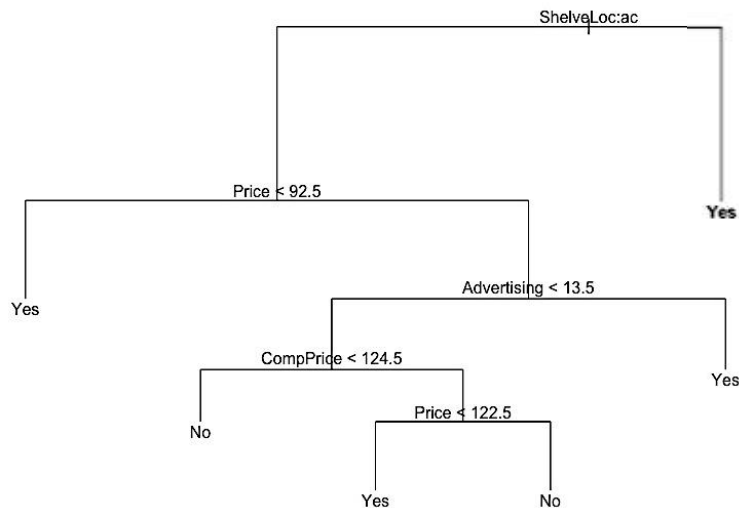
```
# Cross validation vs. Cost complexity pruning
CV <- cv.tree(tree.cs)
CCP <- cv.tree(tree.cs, prune.tree)
plot(CV, col="blue", lwd=3)
par(new=TRUE)
plot(CCP, col="red", lwd=3)
plot(prune.tree(tree.cs, best=6))
```



(*) Consider the deviance as a goodness-of-fit statistic that attempts to measure how well the tree is at reproducing the conditional distribution of the response variable for each possible profile. *Computing and using the deviance with classification trees*. Gilbert Ritschard.

http://mephisto.unige.ch/pub/publications/gr/ritschard_compstat06.pdf

Prune strategy (R)



```

# Measure trees performance. Create train & test sets
trainset <- sample(1:nrow(Carseats), 200)
testset <- Carseats[-trainset,]
High.test <- High[-trainset]
tree.train <- tree(High~.-Sales, Carseats, subset=trainset)
tree.pred <- predict(tree.train, testset, type="class")
table(tree.pred, High.test) # 0.73
# Now the pruned tree.
tree.pruned <- prune.tree(tree.cs, best=6)
prunedtree.pred <- predict(tree.pruned, testset, type="class")
table(prunedtree.pred, High.test) # 0.78

```

tree.pred	No	Yes
No	91	26
Yes	28	55

prunedtree.pred	No	Yes
No	81	6
Yes	38	75

Gini Index, Node *Purity* and *Cross Entropy*

- Binary splitting is also used, though instead of using RSS, **Classification error rate** is used. It is the fraction of observations that do not belong to the most common class in a given region m .

$$E = 1 - \max_k (\hat{p}_{mk})$$

- \hat{p}_{mk} represents the proportion of observations in m^{th} region, that are from k^{th} class.

- **Gini** index measures the total variance across the K classes.

$$G = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$$

- Also known as **node purity**. A small value indicates that a node contains predominantly observations from a single class.

- As an alternative to Gini index, **cross entropy** is also used, and are quite similar, numerically.

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log(\hat{p}_{mk})$$

Gini index in practice

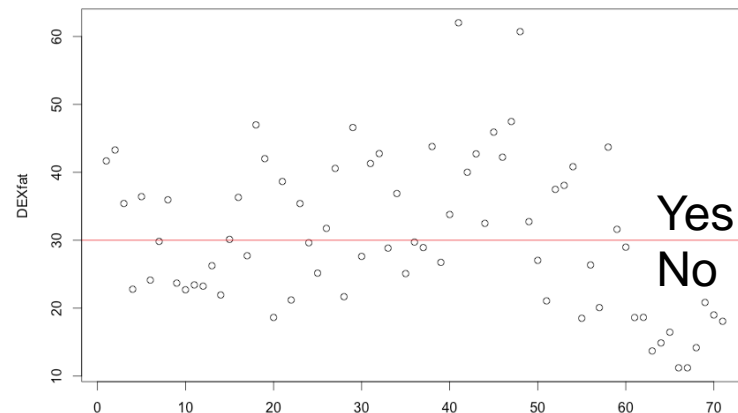
Prediction of Body Fat by Skinfold Thickness, Circumferences, and Bone Breadths

Description

For 71 healthy female subjects, body fat measurements and several anthropometric measurements are available for predictive modeling of body fat.

- Age - age in years.
- DEXfat - body fat measured by DXA, response variable.
- Waistcirc - waist circumference.
- Hipcirc - hip circumference.
- Elbowbreadth - breadth of the elbow.
- Kneebreadth - breadth of the knee.
- Anthro3a - sum of logarithm of three anthropometric measurements.
- Anthro3b - sum of logarithm of three anthropometric measurements.
- Anthro3c - sum of logarithm of three anthropometric measurements.
- Anthro4 - sum of logarithm of three anthropometric measurements.

Let's predict whether
DEXfat is >30 or not.



```
library(tree)
library("TH.data")
data("bodyfat", package = "TH.data")
attach(bodyfat)

# We create a qualitative variable from DEXfat variable.
High=ifelse(DEXfat<=30, "no", "yes")

# We merge High with the rest of the data frame.
bodyfat=data.frame(bodyfat, High)

# We fit a classification tree to predict High
# using all variables but DEXfat
tree.bodyfat = tree(High~.-DEXfat, bodyfat)
```

```
> summary(tree.bodyfat)

Classification tree:
tree(formula = High ~ . - DEXfat, data = bodyfat)
Variables actually used in tree construction:
[1] "waistcirc" "hipcirc"  "anthro3a"
Number of terminal nodes: 4
Residual mean deviance: 0.1751 = 11.73 / 67
Misclassification error rate: 0.04225 = 3 / 71
```

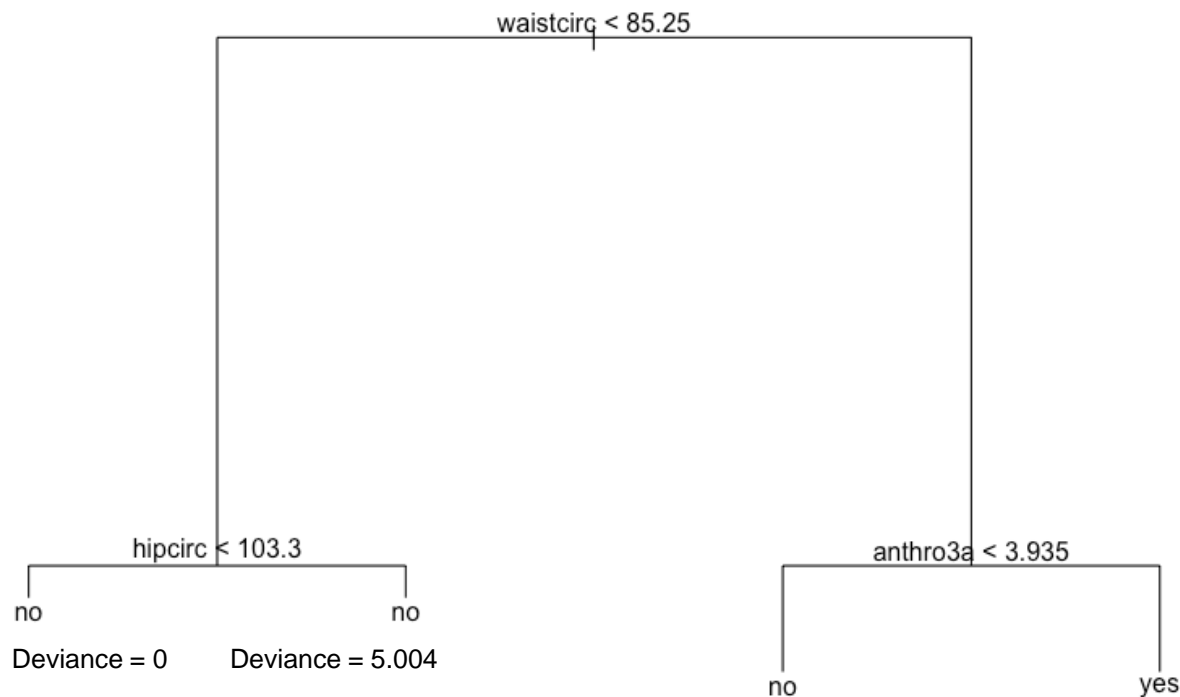
Small deviance = good fit.

Select your quantitative variable. Remember we're in a classification problem. In our case it is "High".

Training error is 4,2%
Deviance computed from:

$$-2 \sum_m \sum_k n_{mk} \log \hat{p}_{mk}$$

n_{mk} is the number of observations in the m^{th} terminal node that belong to the k^{th} class.



Most important indicator in our case is “waistcirc”.

We have TWO terminal nodes having the same predicted value. Why? Increased **node purity**.

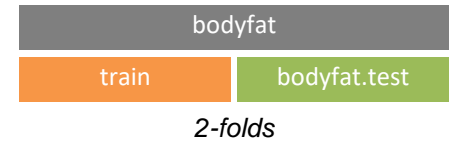
Why is node purity important? Suppose that we have a test observation that belongs to the region given by that left-hand leaf (100%-0%). Then we can be pretty certain that its response value for “High” is Yes. In contrast, if a test observation belongs to the region given by the right-hand leaf, then its response value is probably Yes, but we are much less certain (80% High-20% Low).

```

> tree.bodyfat
node), split, n, deviance, yval, (yprob)
  * denotes terminal node

1) root 71 98.070 no ( 0.53521 0.46479 )
 2) waistcirc < 85.25 36 9.139 no ( 0.97222 0.02778 )
   4) hipcirc < 103.3 31 0.000 no ( 1.00000 0.00000 ) *
   5) hipcirc > 103.3 5 5.004 no ( 0.80000 0.20000 ) *
 3) waistcirc > 85.25 35 20.480 yes ( 0.08571 0.91429 )
   6) anthro3a < 3.935 5 6.730 no ( 0.60000 0.40000 ) *
   7) anthro3a > 3.935 30 0.000 yes ( 0.00000 1.00000 ) *
  
```

```
# NOW split into training and test sets.
set.seed(2)
nrow(bodyfat)
train = sample(1:nrow(bodyfat), 35) # half of samples
bodyfat.test=bodyfat[-train,]
High.test=High[-train]
```



And now, build a tree using the training set and evaluate its performance on the test data.

```
tree.bodyfat = tree(High~.-DEXfat, bodyfat, subset=train)
tree.pred = predict(tree.bodyfat, bodyfat.test, type="class")

> table(tree.pred, High.test)
      High.test
tree.pred no  yes
      no  14   2
      yes   3  17

> (14+17)/36
[1] 0.8611111
```

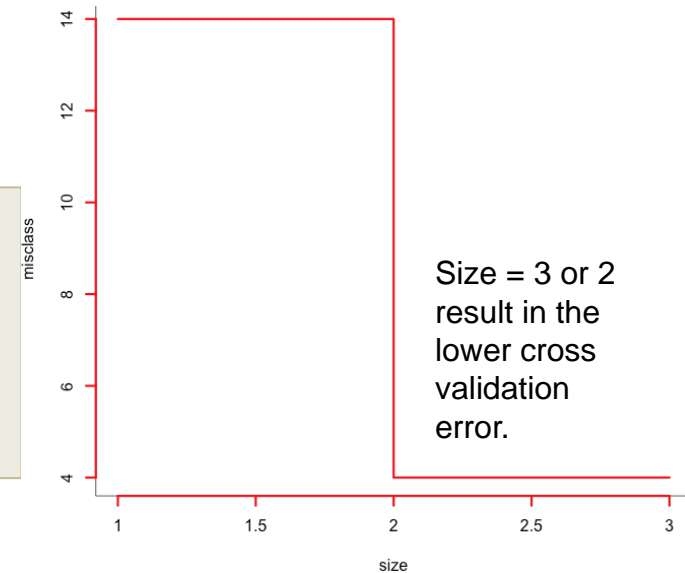
86%

Do we need to prune? Let's apply `cv.tree` to determine the optimal level of tree complexity.

```
cv.bodyfat = cv.tree(tree.bodyfat, FUN=prune.misclass)
> cv.bodyfat
$size
[1] 3 2 1

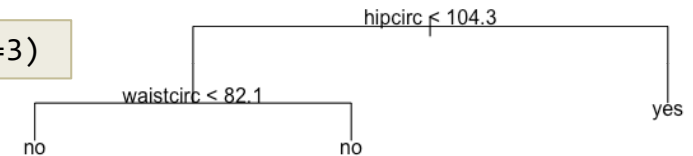
$dev
[1] 4 4 14
```

<- dev here is cross validation error



Use `prune.tree` to obtain the three-node tree.

```
prune.bodyfat=prune.tree(tree.bodyfat, method="misclass", best=3)
```



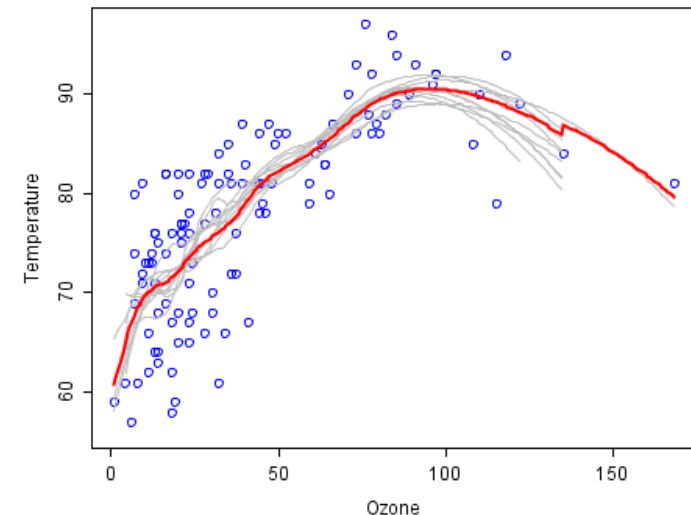
And now measure how well this tree performs.

```
tree.pred = predict(prune.bodyfat, bodyfat.test, type="class")
> table(tree.pred, High.test)
      High.test
tree.pred no  yes
      no   14   2
      yes   3  17
```

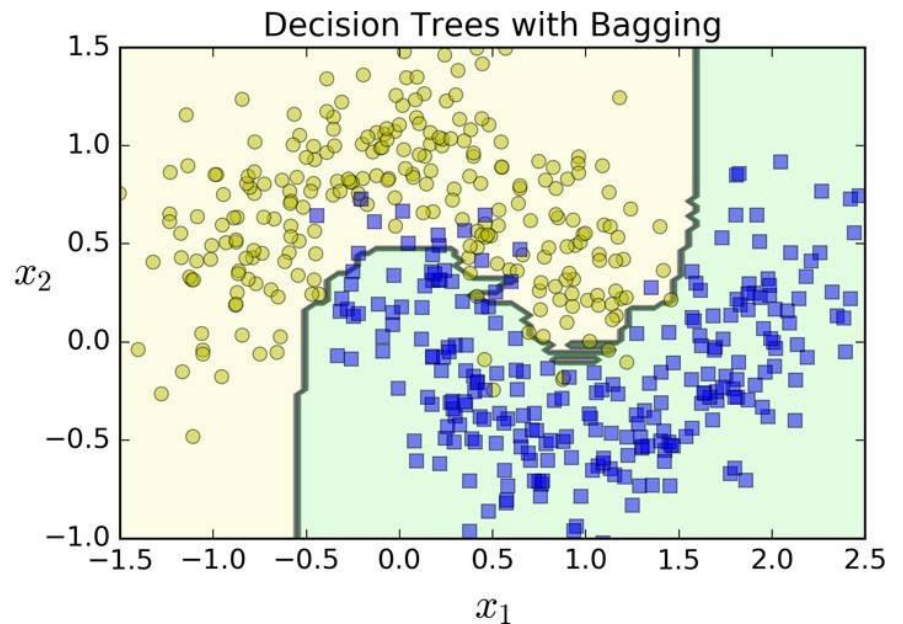
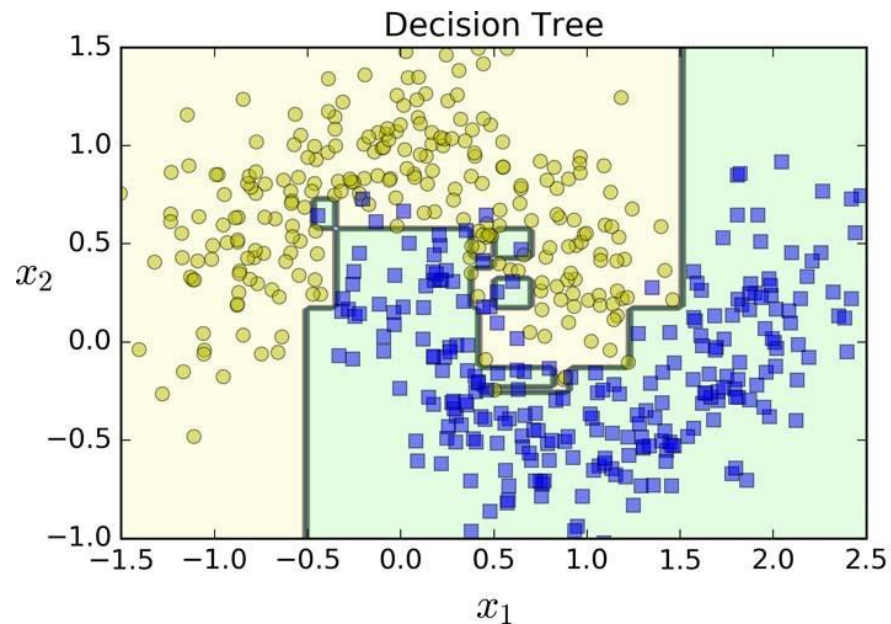
86%

Bagging

- Decision trees suffer from **high variance**
 - Variance** refers to the amount by which our estimate would change if we use a different data set.
 - Bias** refers to the error that is introduced by approximating a real-life problem, by a much simpler model.
- One way of reducing the variance is:
 - Take many training sets from the population (**bootstrapping**),
 - Build separate prediction models using each training set, and average the resulting predictions

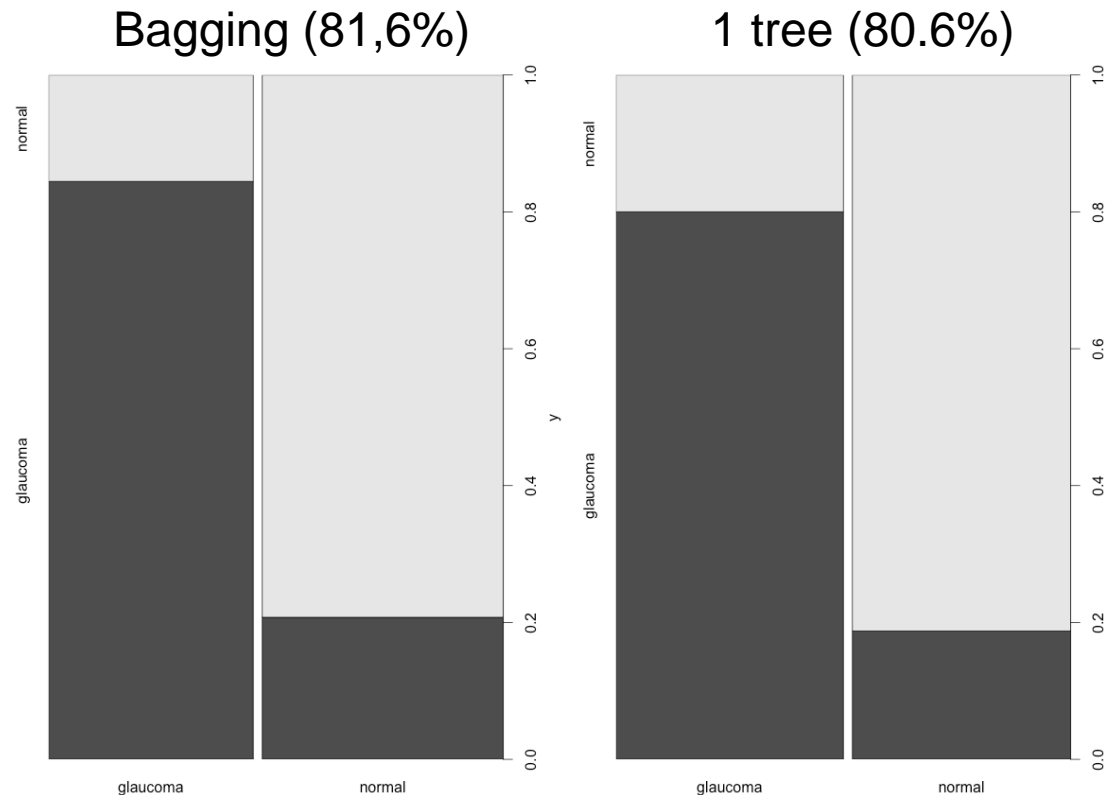


Bagging



This example shows a slight improvement from bagging over trying a quantitative regression using a single tree.

Bagging uses 500 trees, and tries **all** variables (62) at each split.



```
data("GlaucomaM", package="TH.data")

train.set <- sample(1:nrow(GlaucomaM), nrow(GlaucomaM)/2)
test.set <- GlaucomaM[-train.set,]
test.class <- GlaucomaM[-train.set,"Class"]

bag.glaucoma <- randomForest(Class~., data=GlaucomaM, mtry=62, subset=train.set)
estimate <- predict(bag.glaucoma, test.set)

plot(estimate, test.class); table(estimate, test.class)
```

Random Forests

- Differ from bagging in the number of predictors used at each split.
 - Instead of using all predictors, **the algorithm is not allowed to consider a majority of the available predictors.**
- Why? Strong predictors cause most of the trees look similar.
- This technique ***decorrelate*** the trees.

```
# Random forest:: sqrt(62) ~ 8
seed(3)
train.set <- sample(1:nrow(GlaucomaM), nrow(GlaucomaM)/2)
test.set <- GlaucomaM[-train.set,]
test.class <- GlaucomaM[-train.set,"Class"]
bag.glaucoma=randomForest(Class~., data=GlaucomaM, mtry=8, subset=train.set, importance=TRUE)
estimate <- predict(bag.glaucoma, test.set)
table(estimate, test.class) # 85,7%
```

	test.class		
estimate	glaucoma	normal	
glaucoma	41	5	85.7%
normal	9	43	

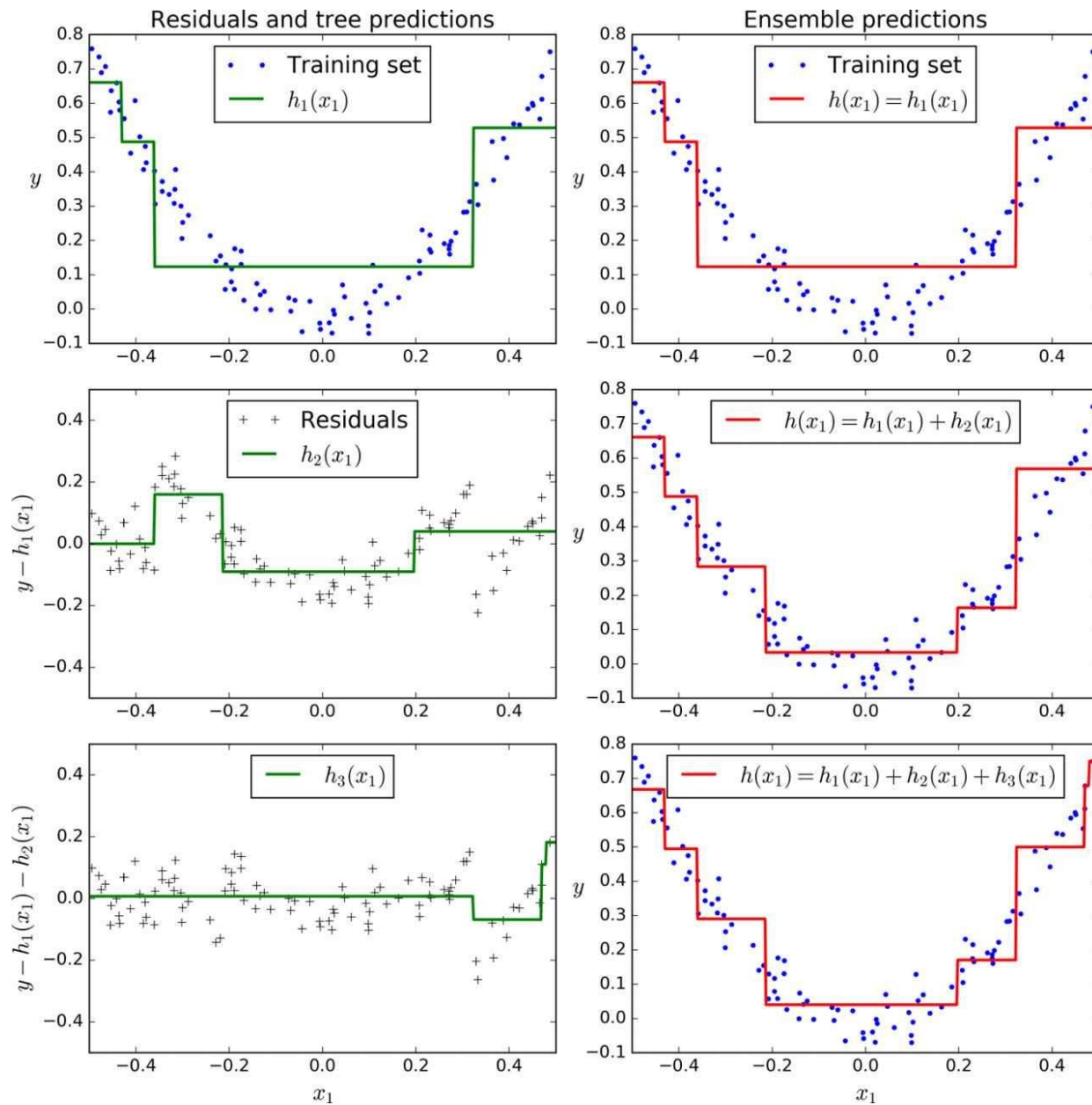
Boosting

- Instead of fitting a large decision tree to the data, **boosting tries to learn slowly from the errors.**
 - **Regression:** The model fits a regression tree from the residuals
 - **Classification:** Gradient Boosting or AdaBoost to build additive tree models.
 - These small trees are added into the fitted function, slowly improving our estimate in areas where it does not perform well.
- Tuning parameters:
 - **Number of trees (B)**
 - **Shrinkage parameter (λ):** rate at which boosting learns (typically: 0.01, 0.001)
 - The **number of splits** in each tree

```
library(gbm)
boost.model = gbm(Class~., data=GlaucomaM[train,], distribution="gaussian",
                  n.trees=5000, interaction.depth=4, shrinkage = 0.001)
summary(boost.model)
boost.predicted = predict(boost.model, test.set, n.trees=5000)
```

83.6%

Test.class		
Pred	glaucoma	normal
FALSE	38	4
TRUE	12	44

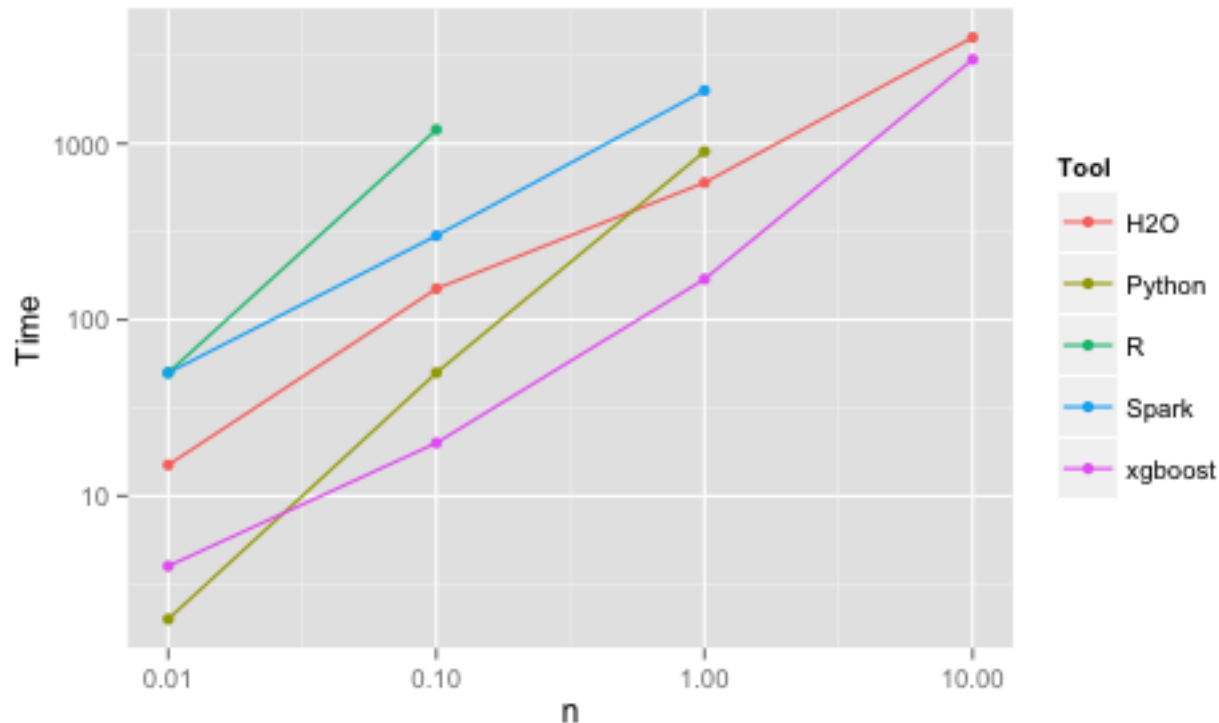


XGBoost (eXtreme Gradient Boosting)



XGBoost

Just an **optimization of GBM** (Gradient Boosting Machines)



XGBoost

Pros:

- Extremely fast (parallel computation) and Highly efficient.
- Versatile (Can be used for classification, regression or ranking).
- Can be used to extract variable importance.
- Do not require feature transformation (missing values imputation, scaling and normalization)

Cons:

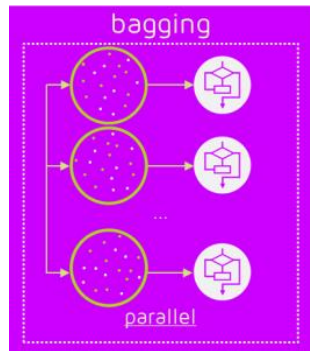
- Leads to **overfitting** if hyperparameters are not tuned properly.

XGBoost

Additional Resources:

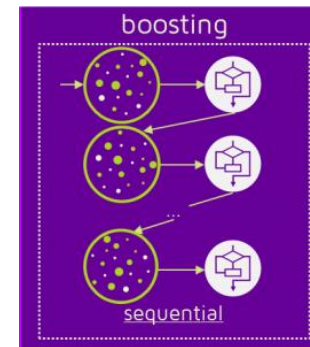
- Deeper explanation of the Gradient Boosting idea: [link](#)
- Examples in R:
 - XGBoost in R in easy steps: [link](#)
 - XGboost Tutorial: [link](#)
- Talk of Tianqi Chen, the creator of the library
 - <https://www.youtube.com/watch?v=Vly8xGnNiWs>

Bagging vs. Boosting



Random Forest

- Independent Classifiers
- Reduce Variance
- Handles Overfitting



XGBoost

- Sequential Classifiers
- Reduce Bias
- Can (**and do**) overfit

Advantages of Decision Trees

- Recursive Partitioning is **relatively fast**, even with large data sets (10^6) and many attributes (10^3)
 - advantage of recursive partitioning: only process all cases at root
- Recursive Partitioning does **feature selection**
- Small-medium size trees usually **intelligible**
- Can be converted to **rules**

Weaknesses of Decision Trees

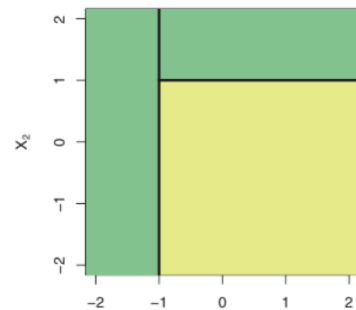
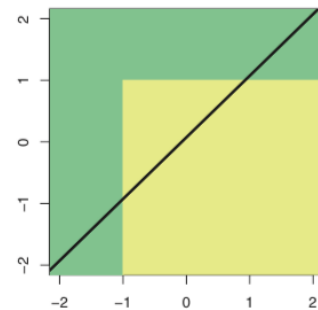
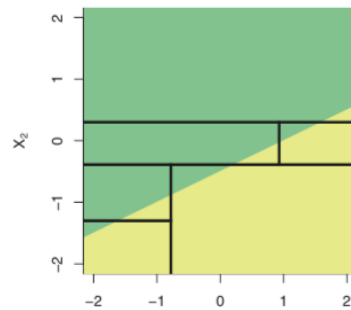
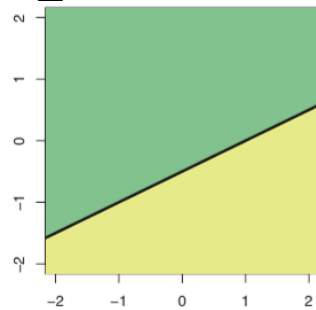
- Large or complex trees can be just as unintelligible as other models
- DTs that look very different can be same/similar
- **Don't handle real-valued parameters as well as Booleans**
- If model depends on summing contribution of many different attributes, DTs probably won't do well
- Usually **poor for predicting continuous values** (regression)

Popular Decision Tree Packages

- **ID3 (ID4, ID5, ...) [Quinlan]**
 - research code with many variations introduced to test new ideas
- **CART: Classification and Regression Trees [Breiman]**
 - best known package to people outside machine learning
 - 1st chapter of CART book is a good introduction to basic issues
- **C4.5 (C5.0) [Quinlan]**
 - most popular package in machine learning community
 - both decision trees and rules

When to Use Decision Trees

- Regression doesn't work



- Model intelligibility is important
- Problem does not depend on many features
- Linear combinations of features not critical
- Speed of learning is important
- Medium to large training sets

Further Readings

- Decision Trees Visually

- <http://www.r2d3.us/visual-intro-to-machine-learning-part-1/>

- R tutorial on Tree Based Modeling

- <https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/>

- Python Tutorial

- <https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.08-Random-Forests.ipynb>