Classification

Logistic regression, CART (rpart) and Random Forest

COWIDUR

Torben Tvedebrink

tvede@math.aau.dk

Department of Mathematical Sciences

AALBORG UNIVERSITY
DENMARK

Terminology

- Supervised learning ("labelled" training data)
  - Classification
  - Regression
- Unsupervised learning (describe hidden structure from "unlabelled" data)
  - PCA
  - Clustering (K-means, ...)

Supervised learning

- Regression
  - Explain/predict a number Y from covariates/predictors/features/explanatory variables
- Classification
  - Now Y is not a number, but a qualitative variable
  - $Y = \text{Eye color} \in \{\text{green}, \text{blue}, \text{brown}\}$
  - $Y = \text{E-mail type} \in \{\text{Spam}, \text{Not spam}\}$
- Supervised: Training data is label-ed (we know Y!!)

Classification

- Given a feature vector x and a qualitative response Y taking values in the set C, the classification task is to build a function $f(x)$ that takes as input the feature vector x and predicts its value for Y; i.e. $f(x) \in C$
- Often: interested in estimating the probabilities that X belongs to each category in C

There are many methods for classification.

- Logistic regression
- Classification (and regression) trees
- Support Vector Machines
- (Artificial) Neural Networks
- k-Nearest Neighbours
- Discriminant analysis
- Naïve Bayes

Types of errors
Nomenclature

	Predicted class		
True Class	– or Null	+ or Non-null	Total
– or Null	True Neg. (TN)	False Pos. (FP)	N
+ or Non-Null	False Neg. (FN)	True Pos. (TP)	P
Total	N*	P*	

Types of errors

Nomenclature

	Predicted class		
True Class	– or Null	+ or Non-null	Total
– or Null	True Neg. (TN)	False Pos. (FP)	N
+ or Non-Null	False Neg. (FN)	True Pos. (TP)	P
Total	N*	P*	

Name	Definition	Synonyms
False pos. rate	FP/N	Type I error, 1 – specificity
True pos. rate	TP/N	1 – Type II error, power, sensitivity, recall
Pos. pred. value	TP/P*	Precision, 1 – false discovery proportion
Neg. pred.value	TN/N*	

ROC curves

Determining alternative threshold

The Receiver Operating Characteristic (ROC) curve is used to assess the accuracy of a continuous measurement for predicting a binary outcome.

The accuracy of a diagnostic test can be evaluated by considering the two possible types of errors: false positives, and false negatives.

For a continuous measurement that we denote as M, convention dictates that a test positive is defined as M exceeding some fixed threshold c: $M > c$.

In reference to the binary outcome that we denote as D, a good outcome of the test is when the test is positive among an individual who truly has a disease: $D = 1$. A bad outcome is when the test is positive among an individual who does not have the disease $D = 0$

ROC curves

Determining alternative threshold

Formally, for a fixed cutoff c, the true positive fraction is the probability of a test positive among the diseased population:

$$TPF(c) = P(M > c \mid D = 1)$$

and the false positive fraction is the probability of a test positive among the healthy population:

$$FPF(c) = P(M > c \mid D = 0)$$

Since the cutoff c is not usually fixed in advance, we can plot the TPF against the FPF for all possible values of c.

This is exactly what the ROC curve is, $FPF(c)$ on the x axis and $TPF(c)$ along the y axis.

ROC curves
Confidence regions

It is common to compute confidence regions for points on the ROC curve using the Clopper and Pearson (1934) exact method. Briefly, exact confidence intervals are calculated for the *FPF* and *TPF* separately, each at level $1-\sqrt{1-\alpha}$.

Based on result 2.4 from Pepe (2003), the cross-product of these intervals yields a $100\%(1-\alpha)$ rectangular confidence region for the pair.

ROC curves
Confidence regions

It is common to compute confidence regions for points on the ROC curve using the Clopper and Pearson (1934) exact method. Briefly, exact confidence intervals are calculated for the *FPF* and *TPF* separately, each at level $1 - \sqrt{1 - \alpha}$.

Based on result 2.4 from Pepe (2003), the cross-product of these intervals yields a $100\%(1 - \alpha)$ rectangular confidence region for the pair.

NB! The ROC curve is only defined for two-class problems but has been ex- tended to handle three or more classes. Hand and Till (2001), Lachiche and Flach (2003), and Li and Fine (2008) use different approaches extending the definition of the ROC curve with more than two classes.

ROC curves in R

There are many packages for computing, plotting and manuípulating with ROC curves and other methods for classifier visualisations.

A nice recent review by Joe Ricket (RStudio): https://rviews.rstudio.com/2019/03/01/some-r-packages-for-roc-curves/

Focus on the ROCR package: https://rocr.bioinf.mpi-sb.mpg.de/

AUC
Area under the curve

The overall performance of a classifier, summarised over all possible thresholds, is given by the area under the (ROC) curve (AUC). An ideal ROC curve will hug the top left corner, so the larger the AUC the better the classifier.

To visually compare different models, their ROC curves can be superimposed on the same graph. Comparing ROC curves can be useful in contrasting two or more models with different predictor sets (for the same model), different tuning parameters (i.e., within model comparisons), or complete different classifiers (i.e., between models).

There is a considerable amount of research on methods to formally compare multiple ROC curves. See Hanley and McNeil (1982), DeLong et al. (1988), Venkatraman (2000), and Pepe et al. (2009) for more information.

Logistic regression

Logistic regression

Intuition

Linear regression (ignoring error term):

$$y = \beta_0 + \beta_1 x$$

Here, $y \in (-\infty, \infty)$, unless $\beta_1 = 0$.

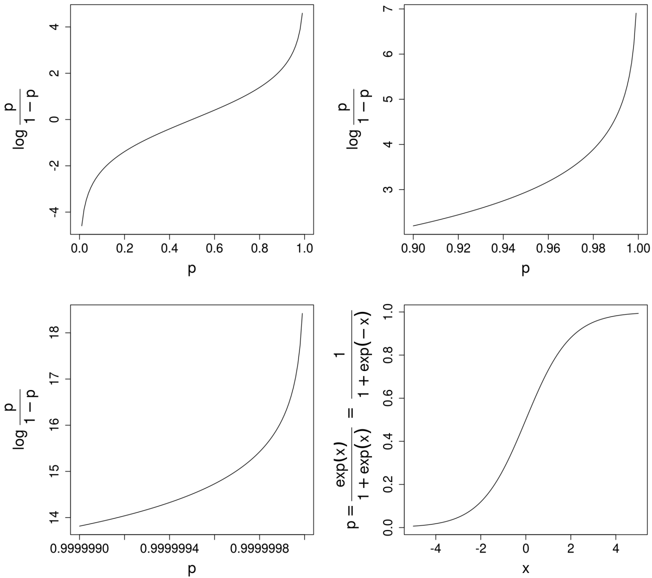
$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right),$$

$\text{logit}(p) \in (-\infty, \infty)$ for $p \in (0, 1)$.

Go from $(-\infty, \infty)$ to $(0, 1)$ (and back).

$$\text{logit}(p) = x \Leftrightarrow p = \frac{\exp(x)}{1 + \exp(x)} = \frac{1}{1 + \exp(-x)}$$

Logistic regression
Intuition



Intuition

$Y \in \{0,1\}$. Model $P(Y=1)$. Linear regression?

Logistic regression (ignoring error term):

$$\operatorname{logit}(P(Y=1)) = \beta_0 + \beta_1 x$$

Here, $\operatorname{logit}(P(Y=1)) \in (-\infty, \infty)$, unless $\beta_1 = 0$, and

$$\operatorname{logit}(P(Y=1)) = \log\left(\frac{P(Y=1)}{1-P(Y=1)}\right),$$

such that

$$P(Y=1) = \frac{\exp(\beta_0+\beta_1 x)}{1+\exp(\beta_0+\beta_1 x)} = \frac{1}{1+\exp\left(-(\beta_0+\beta_1 x)\right)}$$

and $P(Y=1) \in (0,1)$.

In R: `glm(y ~ x, family = binomial)`.

Iris Flowers

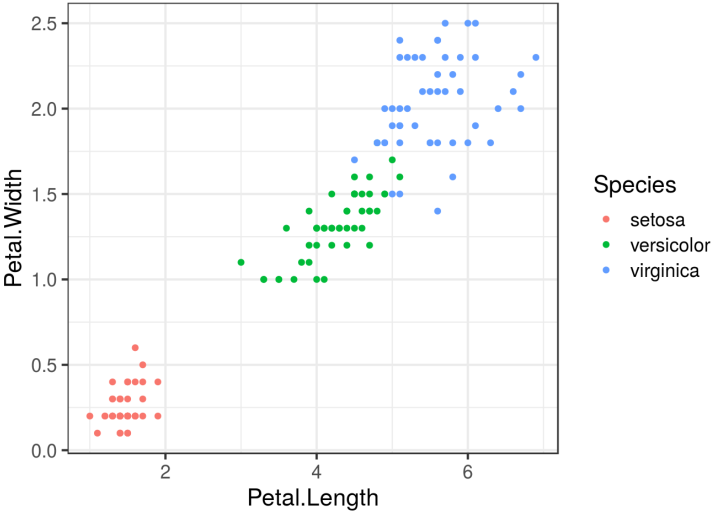
Iris Versicolor **Iris Setosa** **Iris Virginica**

> *This famous (Fisher's or Anderson's) iris data set gives the measurements in centimetres of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.*

Logistic regression only works for binary outcome (extensions exist: multinomial regression, `nnet::multinom`)

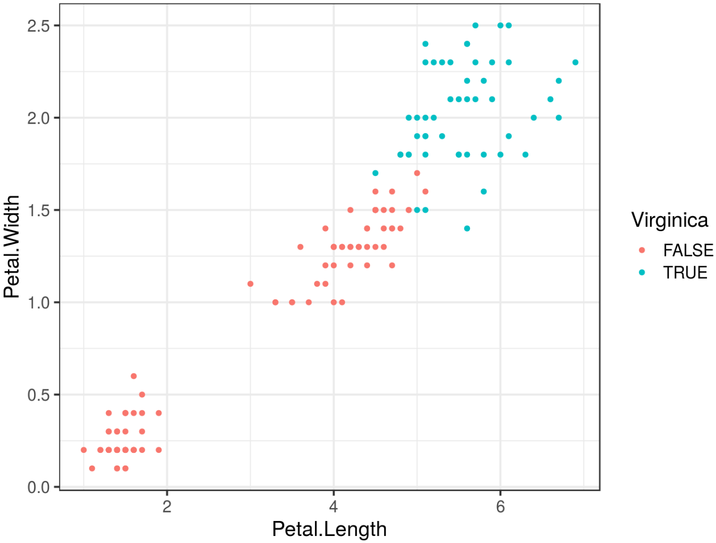
Iris flowers

Example



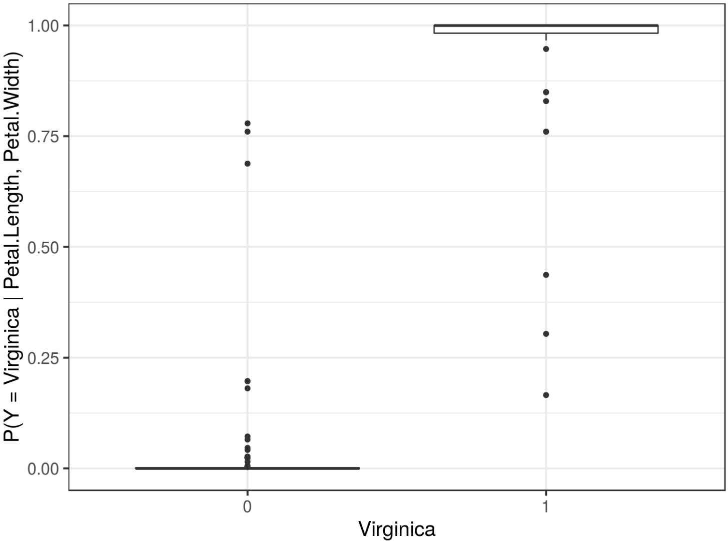
Iris flowers

Example

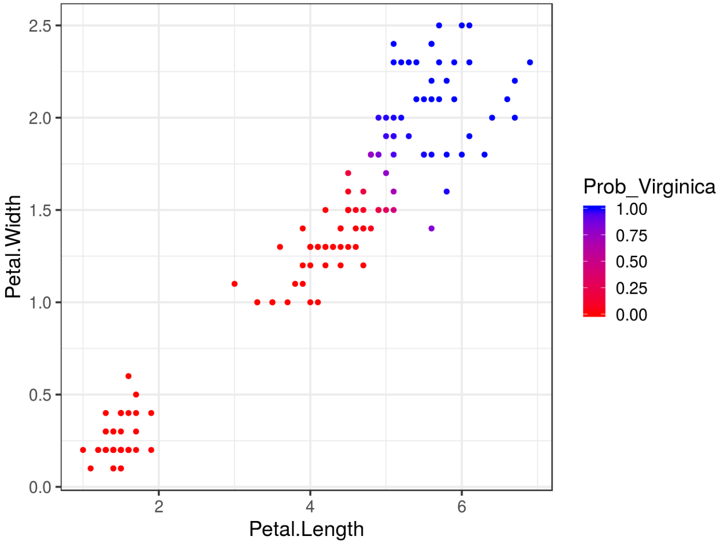


Iris flowers

Example

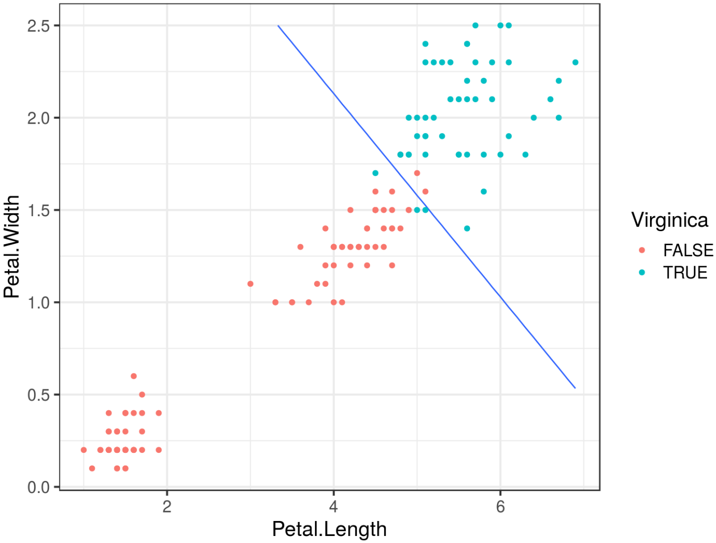


Iris flowers
Example



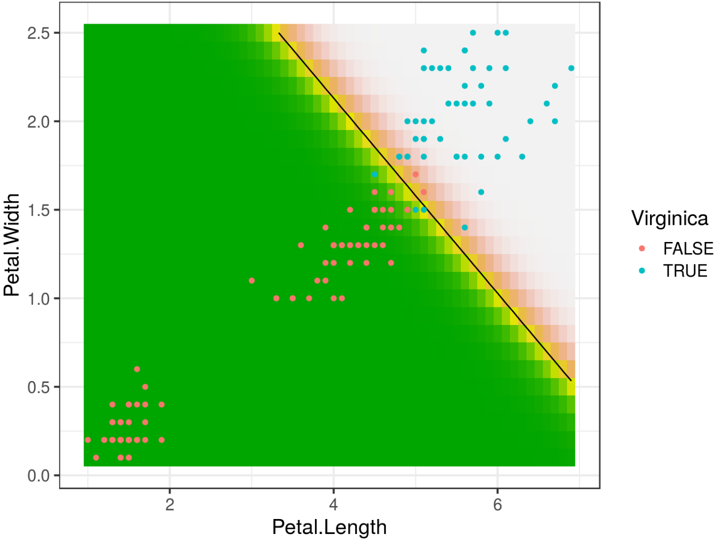
Iris flowers

Example



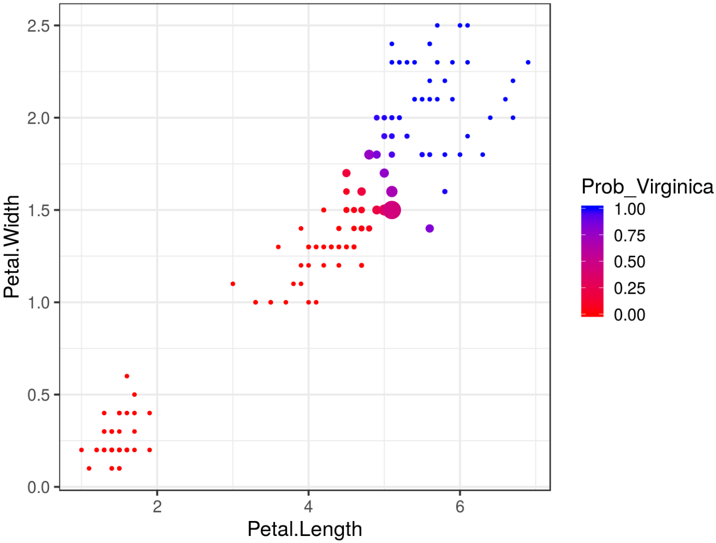
Iris flowers

Example

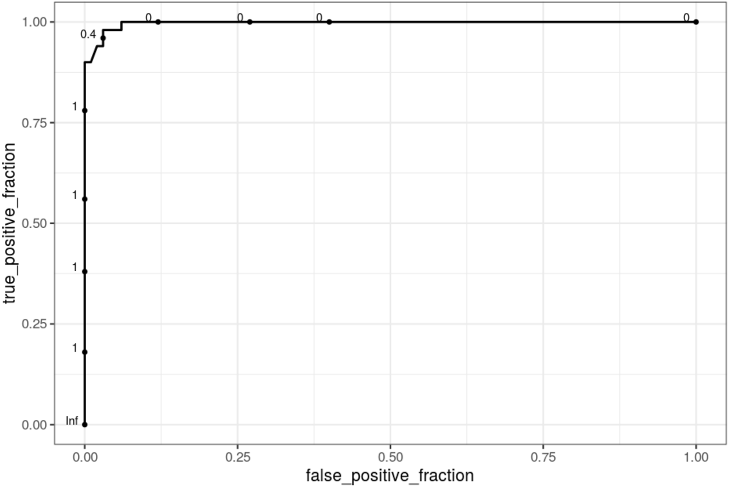


Iris flowers

Example



Iris flowers
Example

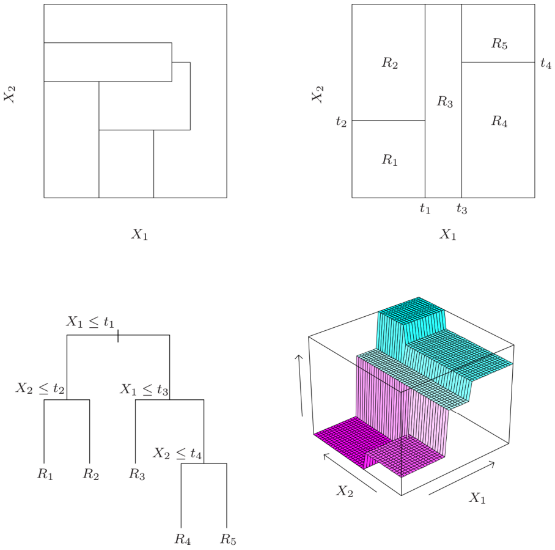


AUC: 0.997

Classification and Regression Trees

CART: Classification And Regression Trees

Link: Introduction to rpart



CART: Regression

For regression the CART methodology fits a piece-wise constant prediction for each region R_j,

$$\hat{Y}_{\text{CART}}(\boldsymbol{x}) = \sum_{j=1}^{R} \beta_j \mathbb{I}(\boldsymbol{x} \in R_j),$$

where β_j is the constant level for region R_j.

Hence, the expression for $\hat{Y}$ can be determined if

a) the partition (i.e. the regions $R_1, \ldots, R_R$) are known

b) the estimated parameters β_j are known

These are chosen such that they minimises the expected squared loss for future observations $(\boldsymbol{x}, y)$,

$$\mathbb{E}[(Y - \hat{Y})^2]$$

CART: Classification

Assume that $y \in \{0, 1\}$ and CART once again constructs a piece-wise constant function

$$\hat{Y}_{\text{CART}}(\boldsymbol{x}) = \sum_{j=1}^{R} \beta_j \mathbb{I}(\boldsymbol{x} \in R_j),$$

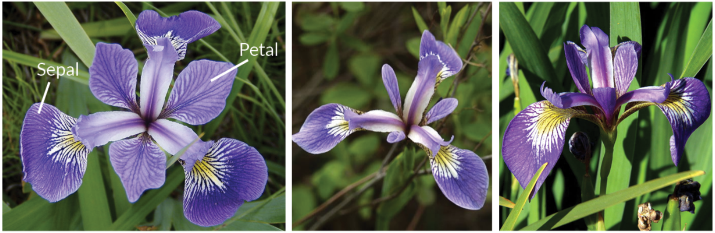
where $\beta_j \in [0, 1]$. Standard classification uses

$$Y_{\text{CART}}(\boldsymbol{x}) = \begin{cases} 0, & \text{if } \hat{Y}_{\text{CART}} \leq 0.5 \\ 1, & \text{if } \hat{Y}_{\text{CART}} > 0.5 \end{cases}$$

A good choice of $\hat{Y}_{\text{CART}}$ leads to a small mis-classification rate, $P(Y_{\text{CART}}(\boldsymbol{x}) \neq y)$.

Example

Iris data – three species



Iris Versicolor **Iris Setosa** **Iris Virginica**

```
> iris[c(1:2,51:52,101:102),]
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
51           7.0         3.2          4.7         1.4 versicolor
52           6.4         3.2          4.5         1.5 versicolor
101          6.3         3.3          6.0         2.5  virginica
102          5.8         2.7          5.1         1.9  virginica
```

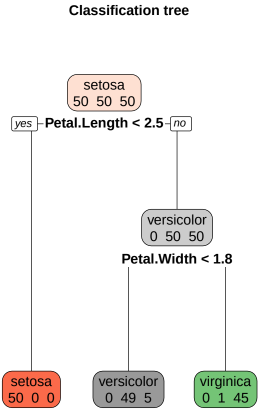
Example
Iris data

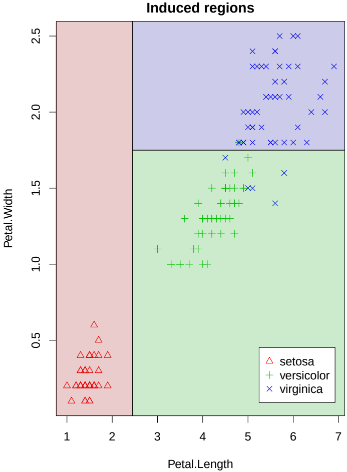
We can classify the species in the Iris dataset using CART classification.

```
library(rpart)

data(iris)

(cart.iris <- rpart(Species˜.,data=iris))

n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 100 setosa (0.33 0.33 0.33)
 2) Petal.Length< 2.45 50   0 setosa (1.00 0.00 0.00) *
 3) Petal.Length>=2.45 100  50 versicolor (0.00 0.50 0.50)
  6) Petal.Width< 1.75 54   5 versicolor (0.00 0.91 0.09) *
  7) Petal.Width>=1.75 46   1 virginica (0.00 0.02 0.98) *
```

Example

Iris data – Cont'd





Parameter estimation

From the model

$$\hat{Y}_{\text{CART}}(\boldsymbol{x}) = \sum_{j=1}^{R} \beta_j \mathbb{I}(\boldsymbol{x} \in R_j),$$

we have that when the partitions/regions R_j are given, the MLE for β_j is given by

$$\hat{\beta}_j = \frac{\sum_{i=1}^{n} y_i \mathbb{I}(\boldsymbol{x}_i \in R_j)}{\sum_{i=1}^{n} \mathbb{I}(\boldsymbol{x}_i \in R_j)} = \bar{y}_{R_j}.$$

where $\hat{\beta}_j$ for regression just is the average of the ys with $\boldsymbol{x} \in R_j$ and for classification the fraction of "$y = 1$"-samples.

Partitioning

Ideally we wants a partitioning which given the smallest expected loss (regression: sum of squares, classification: error rate).

The number of partitions is to vast, why an exhaustive search is infeasible.

Hence, we use a greedy algorithm to search for partitions with good splits.

Note! The `r` in `rpart` stands for *recursive*. Hence, what applies to the root is used recursively down the tree.

Method to generate splits

In the training data we have $\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i = (x_{i1}, \dots, x_{ip})$ is p-dimensional.

For a numeric predictor vector $\boldsymbol{x}$ we search for the partition:

1. Start by $R_1 = \mathbb{R}^p$
2. Given $R_1, \dots, R_r$, split each R_j into R_{j_1} and R_{j_2} where

   $$\begin{aligned} R_{j_1} &= \{\boldsymbol{x} \in \mathbb{R}^p : \boldsymbol{x} \in R_j \text{ and } x_k \leq c\} \\ R_{j_2} &= \{\boldsymbol{x} \in \mathbb{R}^p : \boldsymbol{x} \in R_j \text{ and } x_k > c\}, \end{aligned}$$

   and the variable x_k with splitting points c is chosen such

   $$\arg\min_{k,c} \min_{\beta_1,\beta_2} \left(\sum_{i:\boldsymbol{x}_i \in R_{j_1}} (y_i - \beta_1)^2 + \sum_{i:\boldsymbol{x}_i \in R_{j_2}} (y_i - \beta_2)^2 \right)$$

   Let $R_{1_1}, R_{1_2}, \dots, R_{r_1}, R_{r_2}$ be new partitions.
3. Repeat step 2. d times to get a tree of depth d.

Model complexity

What size of tree is optimal?

We can grow the tree until each observations has its own leaf (terminal node). This gives an error rate of null, but not very enlightening!.

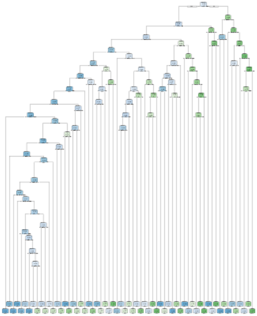
Hence, stop before that, but when?

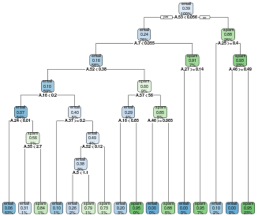
Example
Spam

Can be predict which email are spam and which are not?

```
library(ElemStatLearn)
data(spam, package = "ElemStatLearn")
```

We have 57 explanatory variables, two classes (spam/ham) on 4601 observations.





Bias vs. variance

Which of the two previous trees for the spam data was better? The difference is controlled by a *tuning parameter* that decides the size of the tree (its complexity).

The larger the tree, the less bias but also a higher variance for the test data. Conversely, smaller trees gives larger bias, but little variance for test data.

In general, a bigger tree gives a better prediction for *training data*. However, an increased model complexity may result in a the model too specific for the training data (over-fitting!), which makes it less applicable for test data and prediction for new data. It has a poor *generalisation* ability.

Choosing the *optimal* tree

Tuning parameter α

We wants to search for the *optimal* tree T^*, that minimises the *true* test error, $\text{Error}_{\text{Test}}$. This quantity is unknown, but may be approximated using cross-validation.

The estimate/approximation is used to identify T^*, such that

$$T^* = \arg\min_{T} \text{Error}_{\text{Test}}(T)$$

Choosing the *optimal* tree

Tuning parameter α

We wants to search for the *optimal* tree T^*, that minimises the *true* test error, $\text{Error}_{\text{Test}}$. This quantity is unknown, but may be approximated using cross-validation.

The estimate/approximation is used to identify T^*, such that

$$T^* = \arg\min_T \text{Error}_{\text{Test}}(T)$$

This, however, would require an exhaustive search over all possible trees T – which obviously is infeasible.

Using a tuning parameter α the problem can be translated into a one-dimensional problem.

Pruning

The tuning parameter α penalises large trees,

$$\text{Error}_{\text{Train}}(T) + \alpha|T|, \tag{1}$$

where $|T|$ is the number of leafs in the tree.

Pruning

The tuning parameter α penalises large trees,

$$\text{Error}_{\text{Train}}(T) + \alpha|T|, \tag{1}$$

where $|T|$ is the number of leafs in the tree.

Two approaches:

- Grow the tree until (1) increases.
- Grow a full tree and prune it until (1) increases.

Selecting α

What value of α should be used? Given $\alpha \in \mathbb{R}_+$, let T_α be the tree that minimises

$$T_\alpha = \arg\min_T \mathrm{Error}_{\mathsf{Train}}(T) + \alpha |T|$$

Selecting α

What value of α should be used? Given $\alpha \in \mathbb{R}_+$, let T_α be the tree that minimises

$$T_\alpha = \arg\min_T \text{Error}_{\text{Train}}(T) + \alpha|T|$$

We wants α^* such that the resulting tree has the minimal test error

$$T_{\alpha^*} = \arg \min_{T_\alpha,\ \alpha \in \mathbb{R}_+} \hat{\text{Error}}_{\text{Test}}(T_\alpha),$$

where $\hat{\text{Error}}_{\text{Test}}$ is the estimate of the test error.

Selecting α

Cont'd

We may plot the generalisation error $\hat{\text{Error}}_{\text{Test}}$ for the optimal tree using the criterion

$$\text{Error}_{\text{Train}}(T) + \alpha|T|$$

as a function of α.

It holds that T_α is constant in intervals $I_1 = [0, \alpha_1]$, $I_2 = (\alpha_1, \alpha_2]$, $\dots$, $I_m = (\alpha_{m-1}, \infty]$. Hence, all values $\alpha' \in I_j$ gives the same tree, i.e. α_j, $T_{\alpha'} \equiv T_{\alpha_j}$

Note, T_0 og T_∞ are special cases – T_0 receives no penalty for its size (the full tree), T_∞ gives the empty tree $T_\emptyset$.

How in `rpart`

To decide on α, in `rpart` we use `printcp` or `plotcp`.

These functions use a rewritten version of the above:

$$\begin{aligned}\frac{\text{Error}_\alpha(T)}{\text{Error}_\infty(T)} &= \frac{\text{Error}(T) + \alpha|T|}{\text{Error}(T_\emptyset)} \\ &= \frac{\text{Error}(T)}{\text{Error}(T_\emptyset)} + \frac{\alpha}{\text{Error}(T_\emptyset)}|T| \\ &= \texttt{rel error} + \text{cp}|T|,\end{aligned}$$

where the error is relative to $T_\infty = T_\emptyset$ – i.e. the 'total' variance as we don't have any splits in T_∞

The variable cp is short for 'complexity parameter'.

Choice of `cp`

There are (at least) two criteria to select α^* that decides the complexity of T_{α^*}:

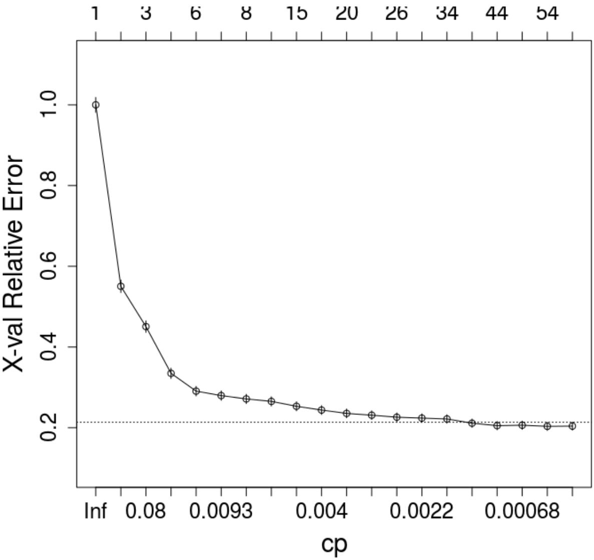
1. Choose `cp` where `xerror` (CV estimate of `rel error`) is smallest,
2. Choose `cp` giving `xerror` within one standard deviation of the smallest `xerror`.

In the `plotcp`-plot the dotted line shows `xerror`+`xstd` relative to the `cp`-value with smallest `xerror`.

Note! `xerror` and `xstd` changes with the CV and is recomputed for each run of `rpart`.

In practice we use 2. since this gives the more parsimonious model (and we consider models within one standard deviation as equally good).

Example
Spam emails – Cont'd



Example

Spam emails – Cont'd

```
library(ElemStatLearn)

data(spam, package = "ElemStatLearn")

spam_rpart <- rpart(spam ~ ., data = spam, cp = 0)

rpart.plot(spam_rpart)

plotcp(spam_rpart)
printcp(spam_rpart)

spam_rpart_prune <- prune(spam_rpart, cp = 0.004)

rpart.plot(spam_rpart_prune)
```

Surrogates

A nice feature of the CART methodology are the so called *surrogates*. These are variables in the data that are not chosen as primary splitting variables, but assemples the splitting properties of the primary split.

They are in particularly important when *missing* observations exists in the primary split variables.

Random Forests

Random forests

An “extension” of CART (or any tree algorithm) are Random Forests.

Random Forests are a relatively simple, but efficient application of classification trees.

Random Forests use “bagging”, which is short for “bootstrap” and “aggregation”. That is, take average (or majority decision) over many trees based on different bootstrap samples.

Random forests

To construct a Random Forest:

1. Make a bootstrap sample of the data and use it as training data.
2. Of the p covariates, select randomly m variables and find the best splitting variable.
   - Default for classification: $m = \sqrt{p}$
   - Default for regression: $m = \left\lfloor \frac{p}{3} \right\rfloor$
3. Grow each tree to maximal size (no pruning)

To *classify* a new observation we use majority voting among the trees in the Random Forest – for *regression* we take the average.