Cross-validation

Torben

August 21, 2018

```
library(tidyverse)
theme_set(theme_bw())
```

Generalisability of models

Let $y = f(x) + \varepsilon$ we our model, where ε is a stochastic error with zero mean and variance σ^2. Note, we don't assume anything about the distribution (e.g. not normality), only that the error is independent of $f(x)$.

Hence, we know that

$$E[y] = E[f(x) + \varepsilon] = E[f(x)] + E[\varepsilon] = f(x)$$

.

We do not necessarily know the *shape* of f(x) - but we wants to learn it. We may not even know which part of $x = (x_1, x_2, \ldots, x_p)$ that affects y. We have just collected data on the phenomenon y and hope the systematic components depends on the collected explanatory data, x.

Example: Linear models

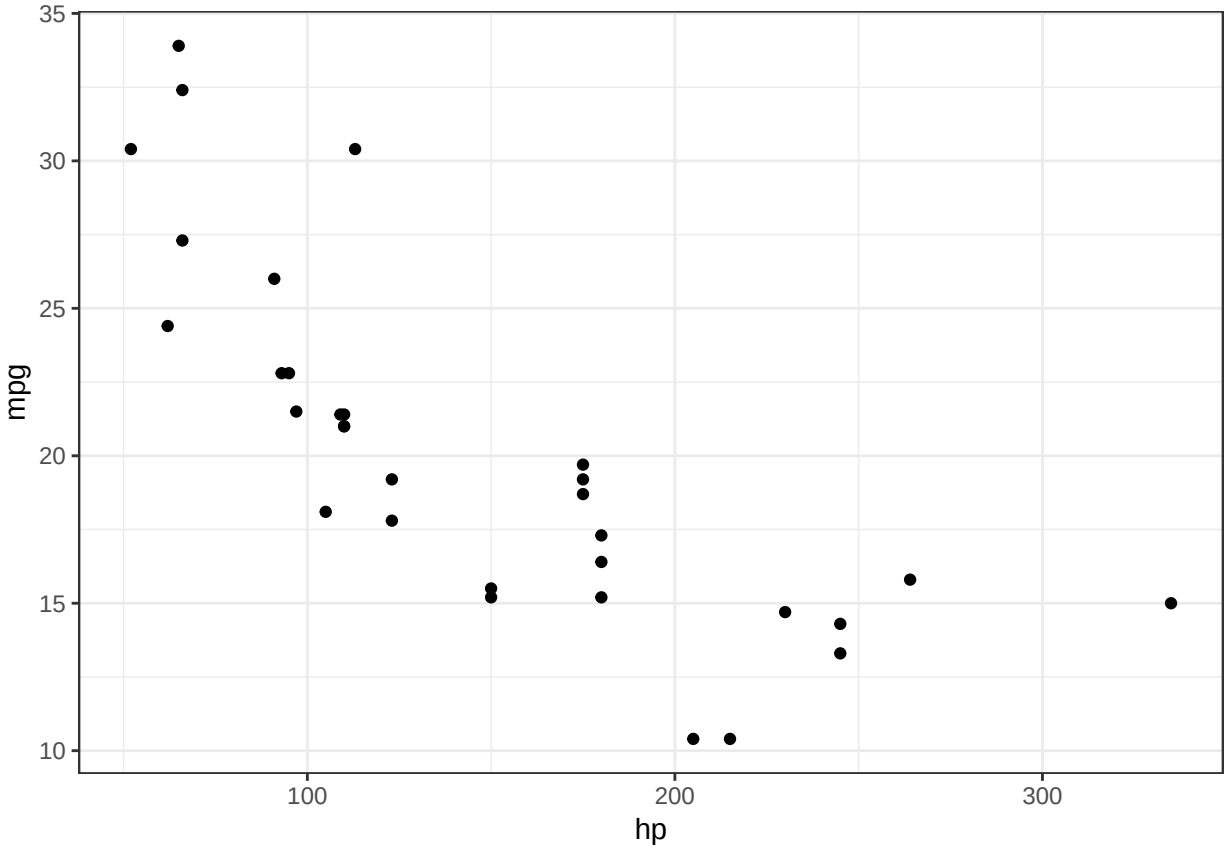
We know that linear models are models where $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \varepsilon$. Or in other words,

$$y = f(x) + \varepsilon = X\beta + \varepsilon,$$

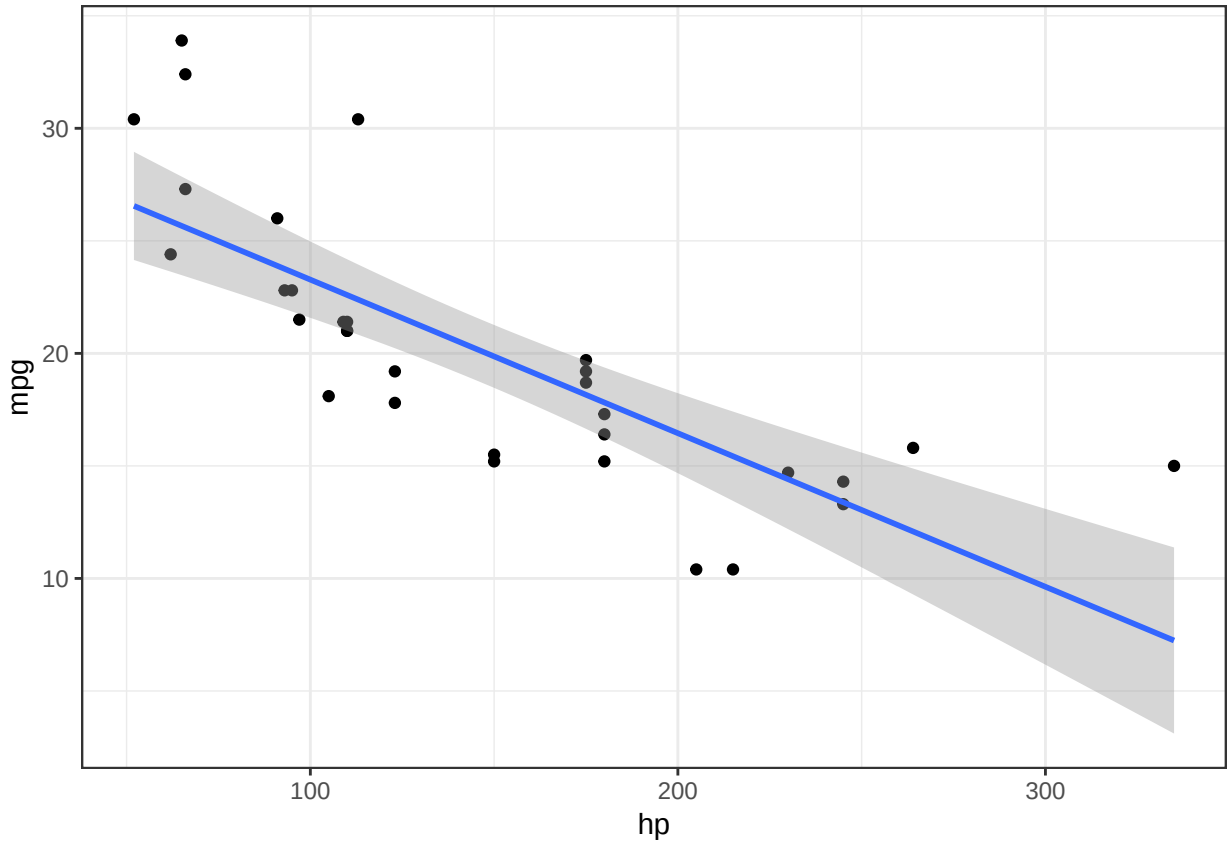
where $\varepsilon \sim N(0, \sigma^2 I)$

Example: `mtcars` and polynomial regression

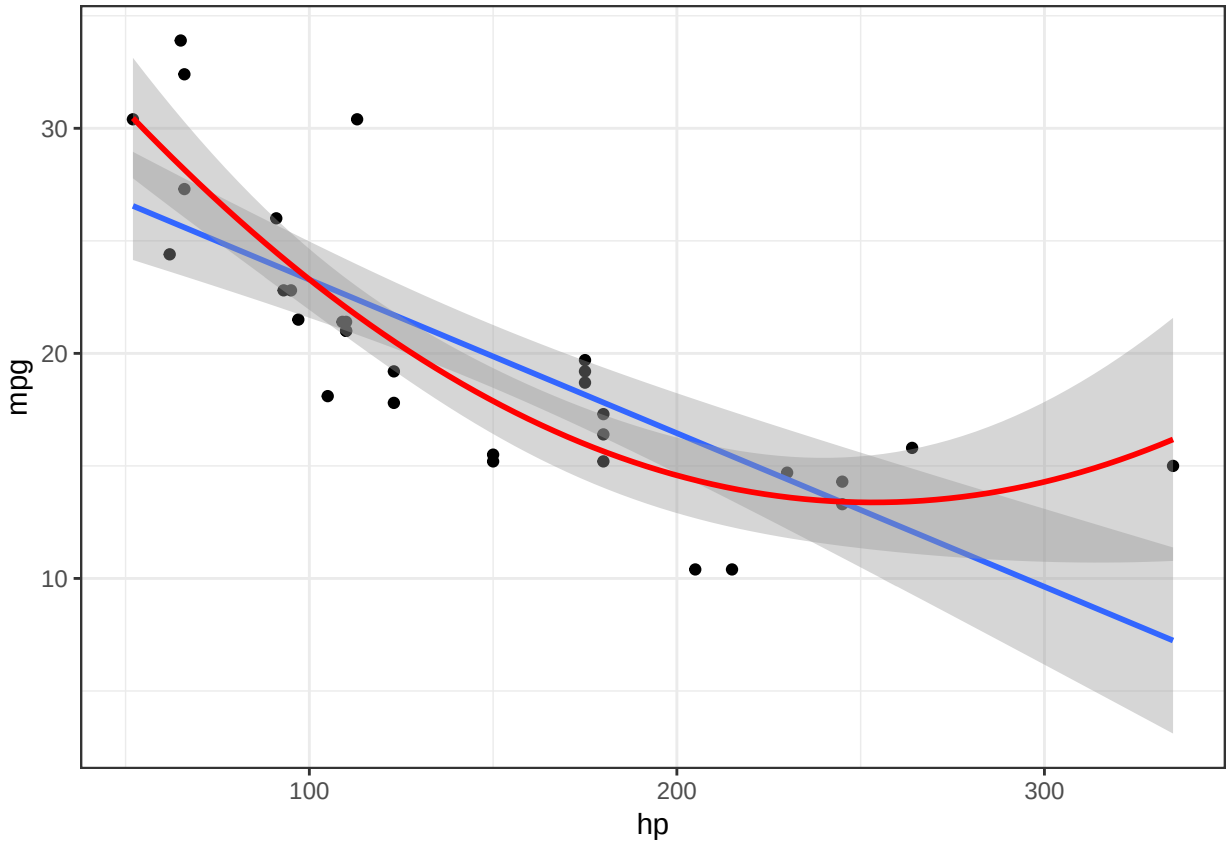
```
(mpg_hp_plot <- ggplot(mtcars, aes(y = mpg, x = hp)) + geom_point())
```



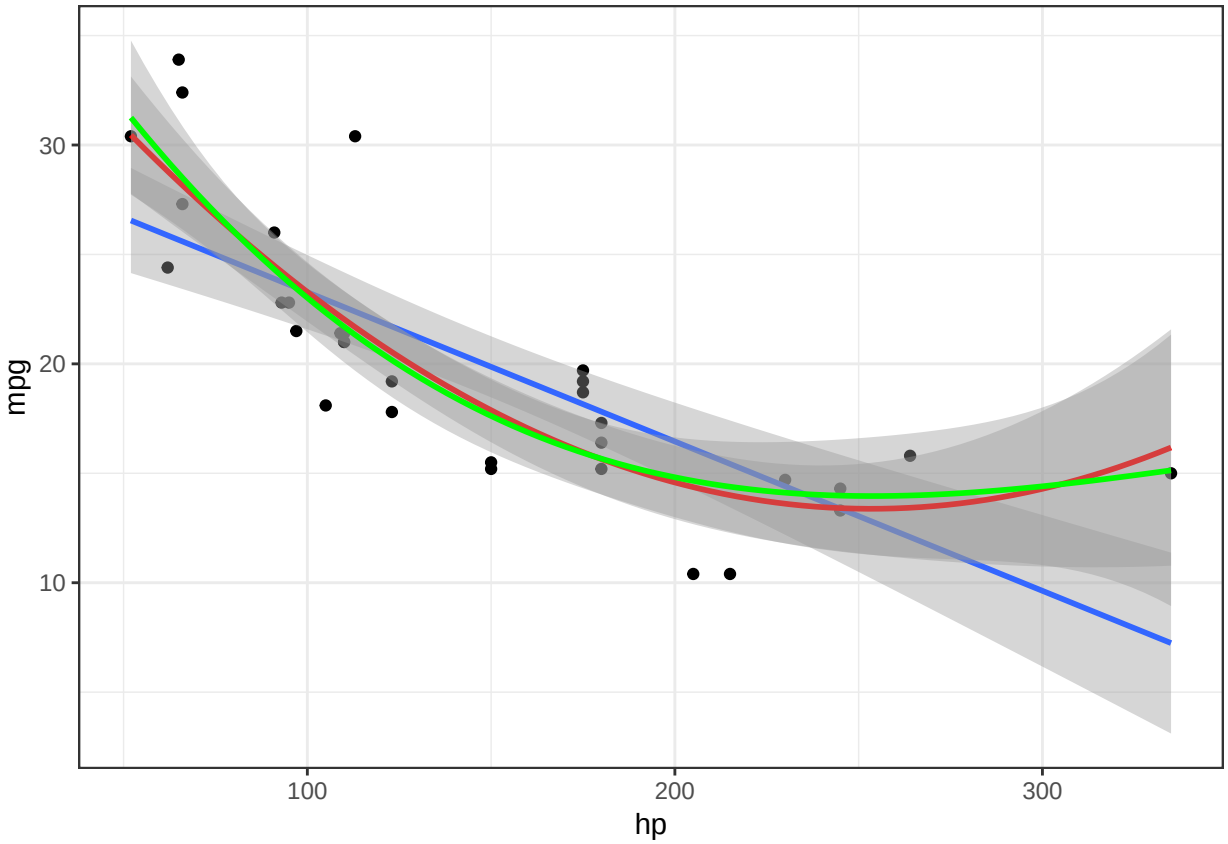

```
(mpg_hp_plot <- mpg_hp_plot + geom_smooth(method = "lm", formula = y ~ x))
```




```
(mpg_hp_plot <- mpg_hp_plot + geom_smooth(method = "lm", formula = y ~ x + I(x^2), colour = "red"))
```




```
(mpg_hp_plot <- mpg_hp_plot + geom_smooth(method = "lm", formula = y ~ x + I(x^2) + I(x^3), colour = "g
```



How many powers of `hp` is needed for a good fit?

What is a good fit? How is it measured?

The *root mean squared error* (RMSE) is one measure:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} r_i^2}$$

, where $\hat{y}_i = \hat{f}(x_i)$ and r_i is the i'th residual.

```
rmse <- function(lm_obj){
  sqrt(mean(residuals(lm_obj)^2))
}
```

How does this perform?

```
rmse(lm(mpg ~ hp, data = mtcars))
```

```
[1] 3.740297
```

```
rmse(lm(mpg ~ poly(hp, 2), data = mtcars))
```

```
[1] 2.929546
```

```
rmse(lm(mpg ~ poly(hp, 3), data = mtcars))
```

```
[1] 2.902615
```

```
rmse(lm(mpg ~ poly(hp, 4), data = mtcars))
```

Figure 1: 10-fold cross validation

```
[1] 2.902144
# and it goes on
```

How well does my model generalise? The above property only holds when *test* and *traning data* are the same.

Test and traning data

We can split the data into *test* and *traning data* - however, a single split only gives us a single estimate of the generalisation error. The solution is to do this K times, by using K-fold cross validation

K-fold cross-validation

We divide the data into K equal sized chunks - the first $K-1$ serves as training and the last as test. We permute the role as test data K times (each time the training is the non-test data).

```
K <- 10
mtcars$cv_fold <- sample(K, size = nrow(mtcars), replace = TRUE)
head(mtcars)
```

```
                  mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
                  cv_fold
Mazda RX4               3
Mazda RX4 Wag           4
Datsun 710             10
Hornet 4 Drive         10
Hornet Sportabout       4
Valiant                 3
```

```r
cv_rmse <- function(power, K = 10){
  mtcars$pred <- NA
  for(k in 1:K){
    train_lm <- lm(mpg ~ poly(hp, degree = power), data = subset(mtcars, cv_fold != k))
    mtcars$pred[mtcars$cv_fold == k] <- predict(train_lm, newdata = subset(mtcars, cv_fold == k))
  }
  sqrt(mean((mtcars$mpg - mtcars$pred)^2))
}
```

So what happens when we run this?

```r
cv_rmse(power = 1)
```

```
[1] 4.219946
```

```r
cv_rmse(power = 2)
```

```
[1] 3.010637
```

```r
cv_rmse(power = 3)
```

```
[1] 3.210855
```

```r
cv_rmse(power = 4)
```

```
[1] 9.178502
```

```r
cv_rmse(power = 5)
```

```
[1] 59.95637
```

We see that the cross-validated RMSE starts to increase after `power = 2` or `power = 3`, suggesting that we start to overfit to the traning data.

CV is very powerful

No matter the type of model f we are fitting to data, we can always do cross-validation. For some model types it is not possible to do hypothesis tests with the usual distributional assumptions as for `lm`, hence we can utilise CV for the same type of questions..

Variance-bias trade-off

Torben

August 21, 2018

Variance-bias trade-off

The Root Mean Squared Error (RMSE) is a measure on the same scale of as the response of our model. The MSE (Mean Squared Error) also often used to assess the model performance.

When we discuss the generalisation of models, we consider *what happens on new unseen dataset* – that is conceptual data. To quantify what we mean by that, we look at the *expected* MSE, $E(MSE)$, where expectation is with respect to the average MSE when f is fitted on a large number of datasets.

Again, let $y = f(x) + \varepsilon$ as before. The σ^2 is an irreducible error that we can't get rid of - it is simply the nature of the phenomenon we model. However, we can attempt to learn f.

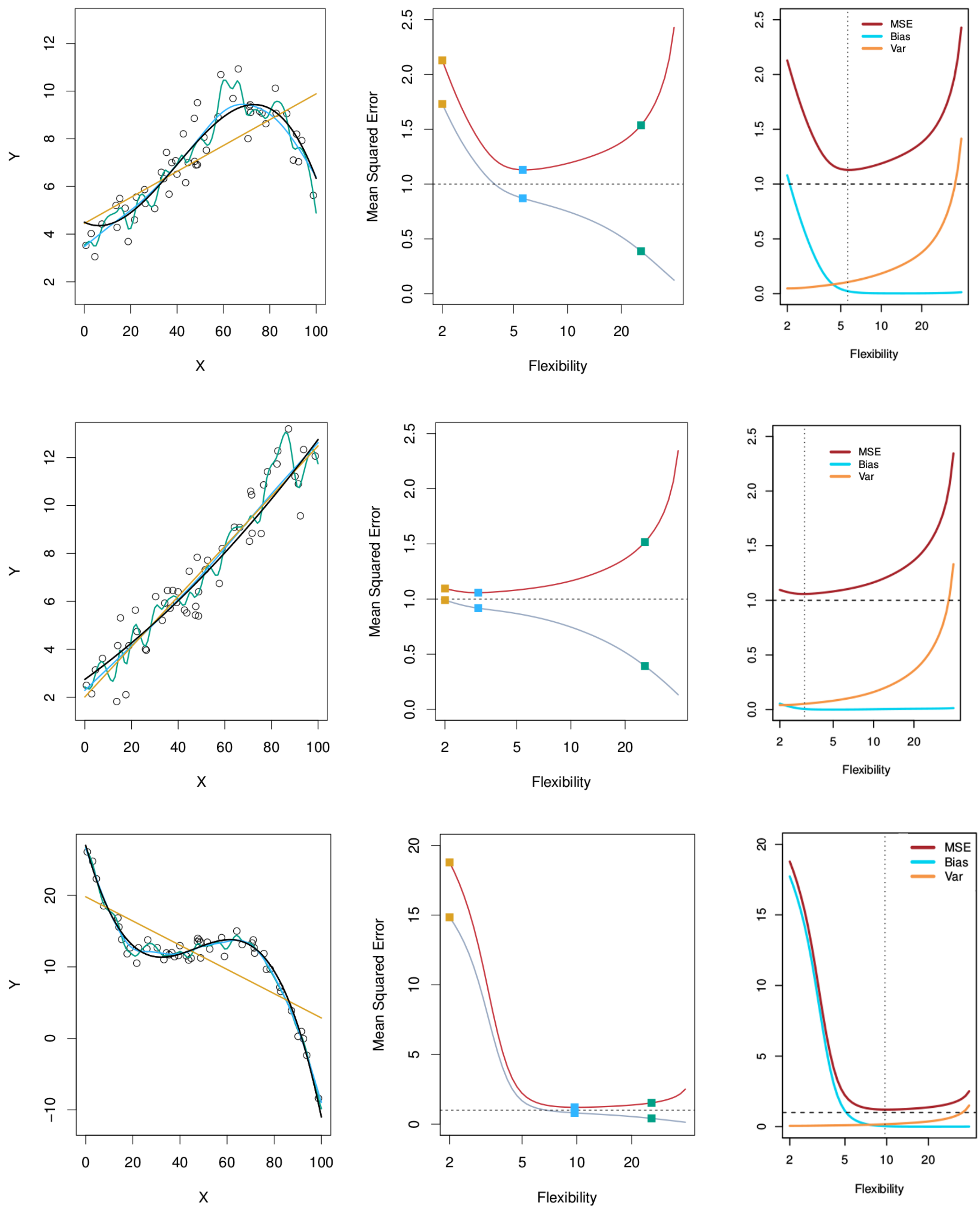
We let $\hat{y}_0 = \hat{f}(x_0)$, where x_0 is some instance of the explanatory variables and y_0 is the observed response with estimate $\hat{y}_0$. When,

$$E[\{y_0 - \hat{f}(x_0)\}^2] = Var[\hat{f}(x_0)] + [Bias\{\hat{f}(x_0)\}]^2 + Var(\varepsilon)$$

In order to minimize the expected test error, we need to select a statistical learning method that simultaneously achieves low variance and low bias. Note that variance is inherently a nonnegative quantity, and squared bias is also onnegative. Hence, we see that the expected test MSE can never lie below σ^2, the irreducible error.

Examples

Three different situations. The data is generated from the black curves. The fitted functions (orange: `lm`, blue: low-degree spline, and green: higher-order spline) has varying flexibility (degrees of freedom).


Y
X
Mean Squared Error
Flexibility
MSE
Bias
Var
Y
X
Mean Squared Error
Flexibility
MSE
Bias
Var
Y
X
Mean Squared Error
Flexibility
MSE
Bias
Var

Pay a little bias to get a reduction in variance?

Why use an unbiased estimator when we have an unbiased one from OLS? One reason is that the estimator has a lower variance. Hence, by introducing a little bias, we are able to reduce the variance of the estimated coefficients.

`glmnet`: Elastic net – Penalised regression methods

Torben

August 21, 2018

Introduction and (one) motivation

In analysis of high-dimensional data, we often face the situation of having more predictors, p, than observations, n, i.e $n < p$. This often the case in genomics, where the number of genetic markers by far exceeds the number of samples. We denote X the $n \times (p)$-design matrix.

For the ordinary least squares this causes a problem, as $\hat{\beta}^{ols} = (X^\top X)^{-1} X^\top y$ implies that $(X^\top X)$ needs to be invertible, i.e. X needs to have full rank, which it does not when $n < p$.

Ridge regression

One way to deal with this is to add a full-rank matrix, λI_p to $X^\top X$, where I_p is the identity matrix. The constant $\lambda \geq 0$ is a tuning parameter, that needs to be specified, e.g. by cross-validation.

A little more maths show that in fact we have

$$\hat{\beta}^{ridge} = (X^\top X + \lambda I_p)^{-1} X^\top y,$$

implying that $\hat{\beta}^{ridge} = \hat{\beta}^{ols}$ for $\lambda = 0$.

One property that is relevant to discuss is bias and variance of the estimators in OLS and Ridge Regression. Hence, let us recall that $\hat{\beta}^{ols}$ is unbiased:

$$\begin{aligned}
\mathbb{E}(\hat{\beta}^{ols}) &= \mathbb{E}\{(X^\top X)^{-1} X^\top y\} \\
&= (X^\top X)^{-1} X^\top \mathbb{E}\{y\} \\
&= (X^\top X)^{-1} X^\top X \beta \\
&= \beta,
\end{aligned}$$

where we used that $y \sim (X\beta, \sigma^2 I_n)$ by assumption.

Bias

We could repeat this argument for $\hat{\beta}^{ridge}$, however it is more instructive to rewrite $\hat{\beta}^{ridge}$ in terms of $\hat{\beta}^{ols}$:

$$\begin{aligned}
\hat{\beta}^{ridge} &= [X^\top X + \lambda I_p]^{-1} X^\top y \\
&= [(X^\top X)\{I_p + \lambda (X^\top X)^{-1}\}]^{-1} X^\top y \\
&= \{I_p + \lambda (X^\top X)^{-1}\}^{-1} (X^\top X)^{-1} X^\top y \\
&= \{I_p + \lambda (X^\top X)^{-1}\}^{-1} \hat{\beta}^{ols},
\end{aligned}$$

where we assumed that $X^\top X$ is invertible.

From this we find that $\mathbb{E}(\hat{\beta}^{ridge}) = \{I_p + \lambda (X^\top X)^{-1}\}^{-1} \beta$, i.e. an biased estimator when $\lambda > 0$.

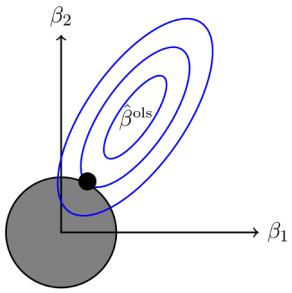


Figure 1: Ridge regression and OLS solutions. The intersection between the ellipsoid contour lines and the disc represents the Ridge solution relative to the OLS solution

However, we can also give a direct calculation of the variance for Ridge Regression and OLS, showing that

$$\text{trace}(\mathbb{V}[\hat{\beta}^{ols}]) = \sigma^2 \sum_{j=1}^{p} \frac{1}{d_j^2} \qquad \text{trace}(\mathbb{V}[\hat{\beta}^{ridge}]) = \sigma^2 \sum_{j=1}^{p} \frac{d_j^2}{(\lambda + d_j^2)^2},$$

where d_j is proportional to the sample variance in the j'th principal component.

A different view on Ridge Regression

One can also look at Ridge Regression differently, namely in terms of penalised regression, where a penalty term is applied to the squared sum of coefficient estimates:

$$\hat{\beta}^{ridge} = \arg\min_{\beta} \|y - X\beta\|_2^2 \quad \text{subject to} \|\beta\|_2^2 \leq t,$$

for some positive constant t and $\|x\|_2^2 = \sum_{j=1}^{p} x_j^2$ is the ℓ_2-norm.

We can think of t as a "budget" for the regression, as we have to "spend" the regression parameter budget on the variables best explaining y from X. For this reason the data is also *scaled* before fitting to have zero mean and variance one (per column - `glmnet` does that automatically).

Using some Lagrange multipliers we can show that this is equivalent to

$$\hat{\beta}^{ridge} = \arg\min_{\beta} \|y - X\beta\|_2^2 + \lambda\|\beta\|_2^2,$$

where λ is the same constant as before and determined through cross-validation.

Relationship between solutions to OLS and Ridge Regression

We can try to inspect the solutions graphically in two dimensions

The Lasso Regression

In the figure above we saw that the Ridge Regression contracts the solution, $\hat{\beta}^{ridge}$, towards the disc defined by the "budget"-parameter t. As a disc/sphere don't have pointy edges, is it rarely the case that any of the parameters in $\hat{\beta}^{ridge}$ are set to zero.

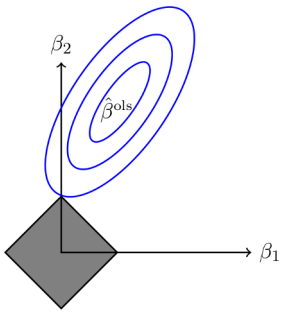


Figure 2: Lasso regression and OLS solutions. The intersection between the ellipsoid contour lines and the square represents the Lasso solution relative to the OLS solution.

In the linear inference that Søren talked about yesterday, the elimination of insignificant terms is important. Typically we do so by successive removing terms from the model – either by some information criterion or hypothesis tests (for nested linear models).

However, the Lasso Regression makes variable selection while estimating the parameters. This is done by solving

$$\hat{\beta}^{lasso} = \arg\min_{\beta} \|y - X\beta\|_2^2 + \lambda\|\beta\|_1,$$

where the penalty is the ℓ_1 norm, $\|x\|_1 = \sum_{j=1}^{p} |x_j|$, i.e. the sum of the absolute values.

The Lasso penalty generally sets more parameter values to zero than the Ridge Regression, where we seldom see any terms fixed to zero.

It is not as easy to express bias and variance for the Lasso since we generally don't have closed forms solutions to the likelihood equations. However, the general picture is that the larger λ, the more bias and consequently lower variance.

As for the Ridge Regression we can visualise the Lasso solution together with the OLS solution for two dimensions. Since, the Lasso penalty can be viewed as

$$\sum_{j=1}^{p} |\beta_j| \leq t,$$

we have a 45°-rotated square centered in origo where the Lasso solution exists.

A hand-waving argument for more sparse solutions comes from the intuition that it is more likely that the corners of the hyper-cube will intersect the ellipsoid. Corners result in one or more zero-parameters.

The Baysian perspective

In Bayesian statistics, we think of the data as fixed and the parameters being random. This is different from the frequentistic approach, where we think of the parameters having some *true* value.

In the Bayesian context, the Ridge Regression results from assigning a normal distributed prior on each component in the β-vector, with zero mean and some variance, τ^2. This implies that we *a priori* assumes many of the parameters to be close to zero.

The Lasso also assigns a zero-mean distribution, but with a Laplacian distribution that decays more rapidly towards zero implying that less terms are expected to be non-zero.

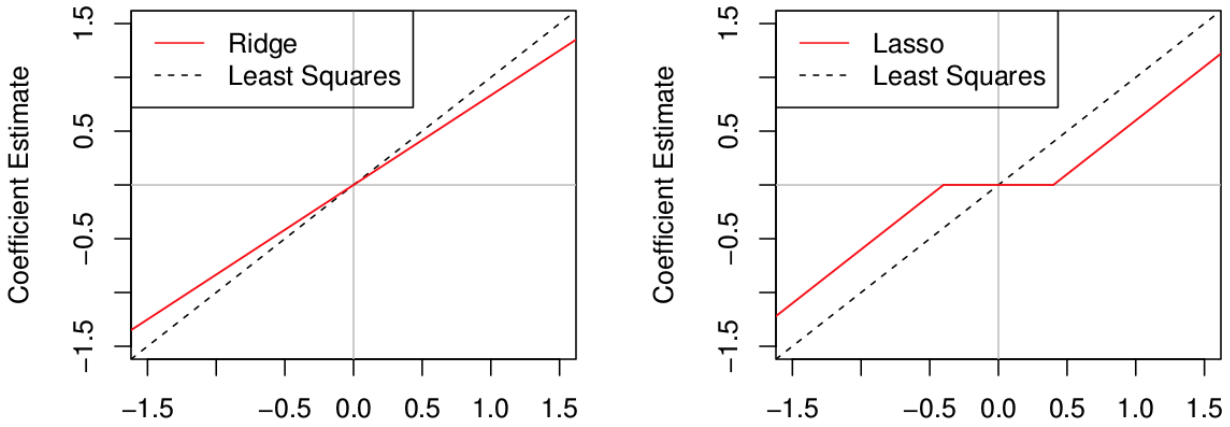


Figure 3: Ridge regession and LASSO - soft thresholding

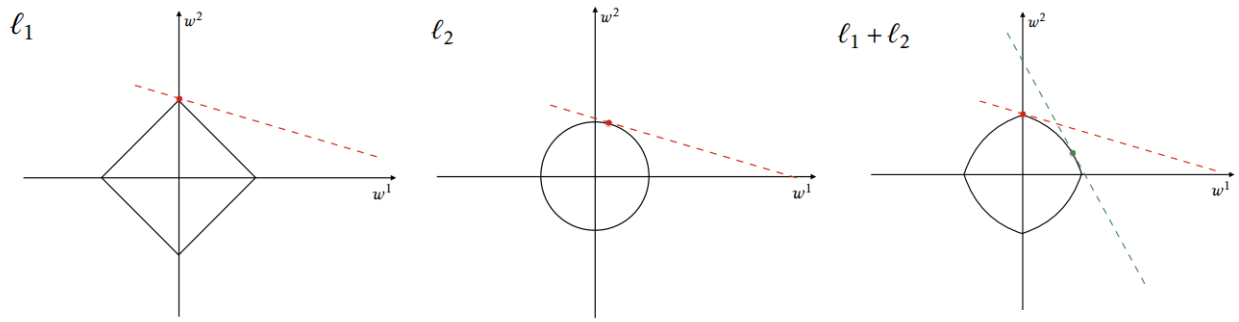


Figure 4: Lasso, Ridge and Elastic Net regularisation. Source: Wikipedia

Elastic Net – the best from two worlds?

A downside with the Lasso is that it may have difficulties when several variables are collinear, such that linear combinations of them are hard to distinguish. In such a case the Ridge Regression is better as it will typically form an average of the variables. Hence, for stable selection of variables in this case Ridge Regression may be preferred. However, Ridge Regression seldom sets any parameters to zero, i.e. no variable selection which is what we would like in the end...

The solution to the problem is Elastic Net, which incorporates both the ℓ_1 (Lasso) and ℓ_2 (Ridge) penalties in a convex way:

$$\hat{\beta}^{en} = \arg\min_{\beta} \|y - X\beta\|_2^2 + \lambda \left(\alpha \|\beta\|_1 + \frac{1-\alpha}{2} \|\beta\|_2^2 \right),$$

where α is yet another tuning parameter deciding the amount of Lasso ($\alpha = 1$) and Ridge ($\alpha = 0$) penalty that goes into the solution.

Both α and λ are selected based on cross-validation.

In the Figure below we see the three types of regularisation discussed above. The shape of the Elastic Net solution area depends on α - the closer to 1 the more square it is, and the closer to 0 the more spherical.

Further reading:

https://web.stanford.edu/~hastie/StatLearnSparsity/

glmnet in R

Torben

August 21, 2018

Load `glmnet` to make the functionalities available.

```
library(glmnetUtils)
library(tidyverse)
```

Ressources on `glmnet`

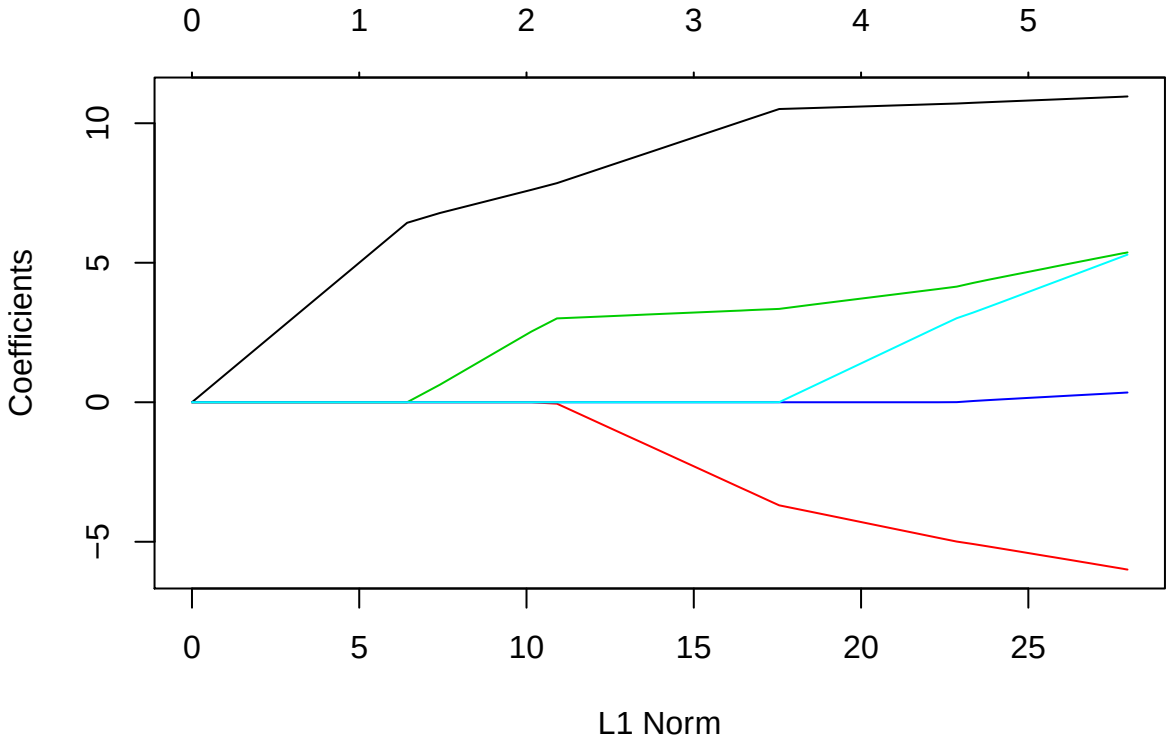
- The package vignette `vignette("glmnet_beta")` is highly recommendable.
- Many questions have already been asked *and* answered at https://stackoverflow.com/questions/tagged/glmnet.

Example

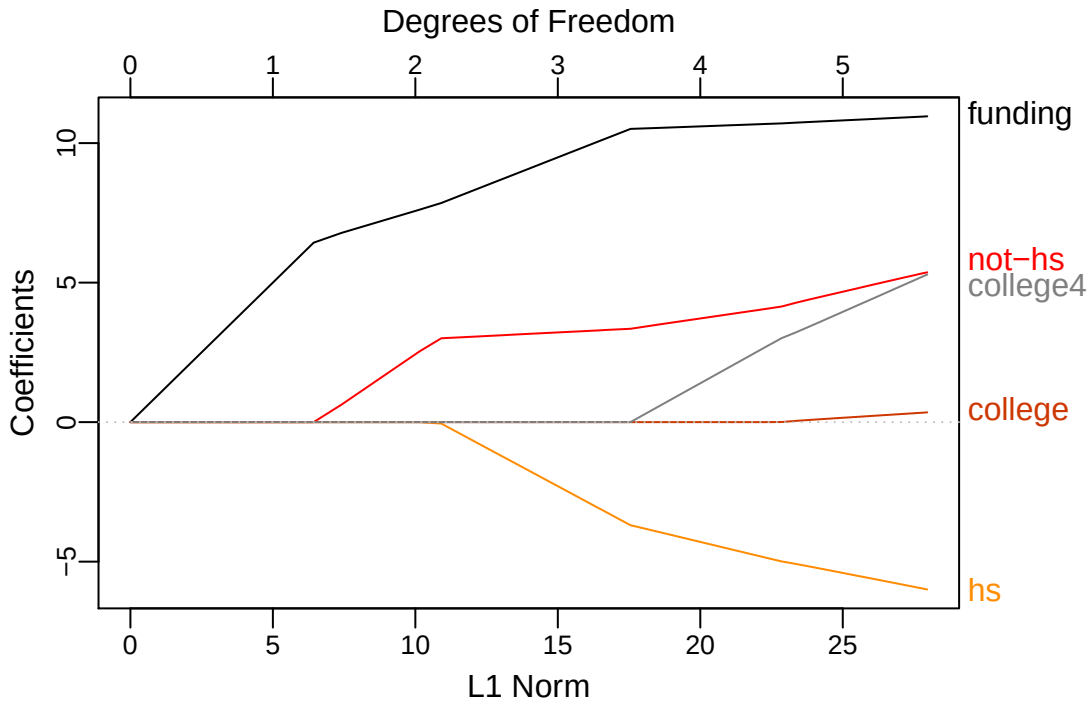
```
crime <- read_csv("crime.csv", col_types = cols())
```

Note: Due to different scales of the variables, the estimated parameters may be different in size simply due to different units. Hence, The `glmnet` function automatically standardises both the response (for `family = "gaussian"`) `y` and the covariates `x`.

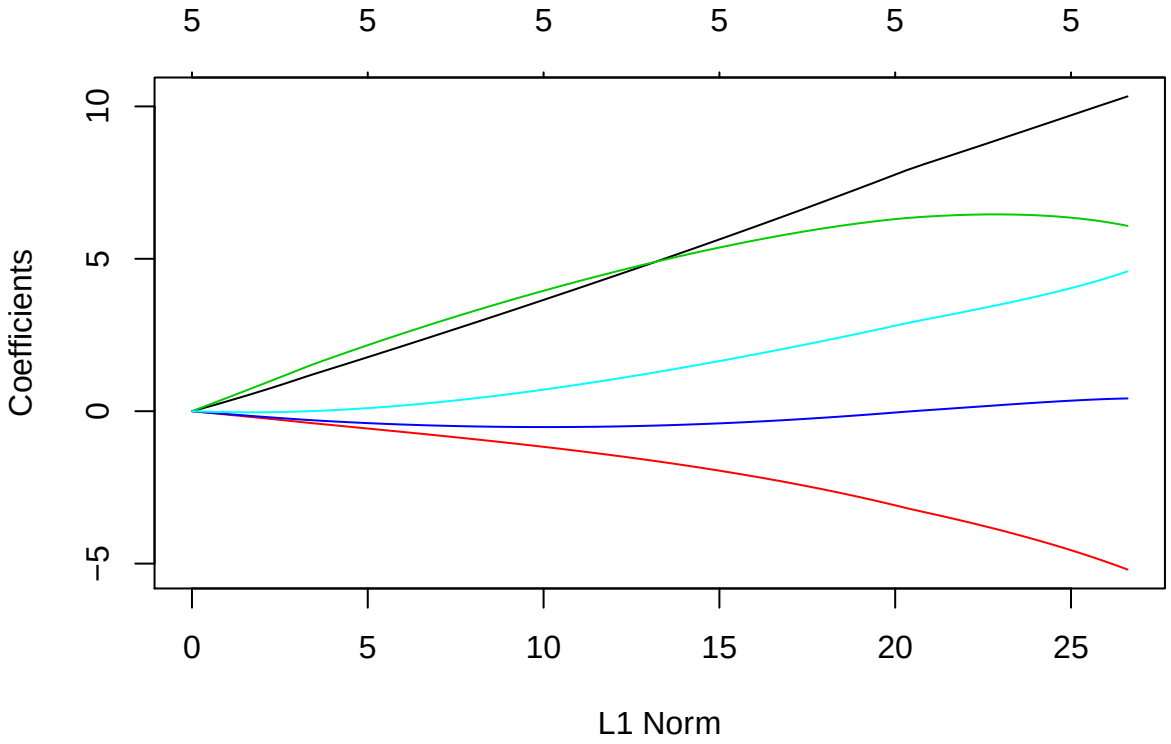
```
crime_lasso <- glmnet(`crime rate` ~ ., alpha = 1, data = crime) ## alpha = 1: LASSO
plot(crime_lasso)
```



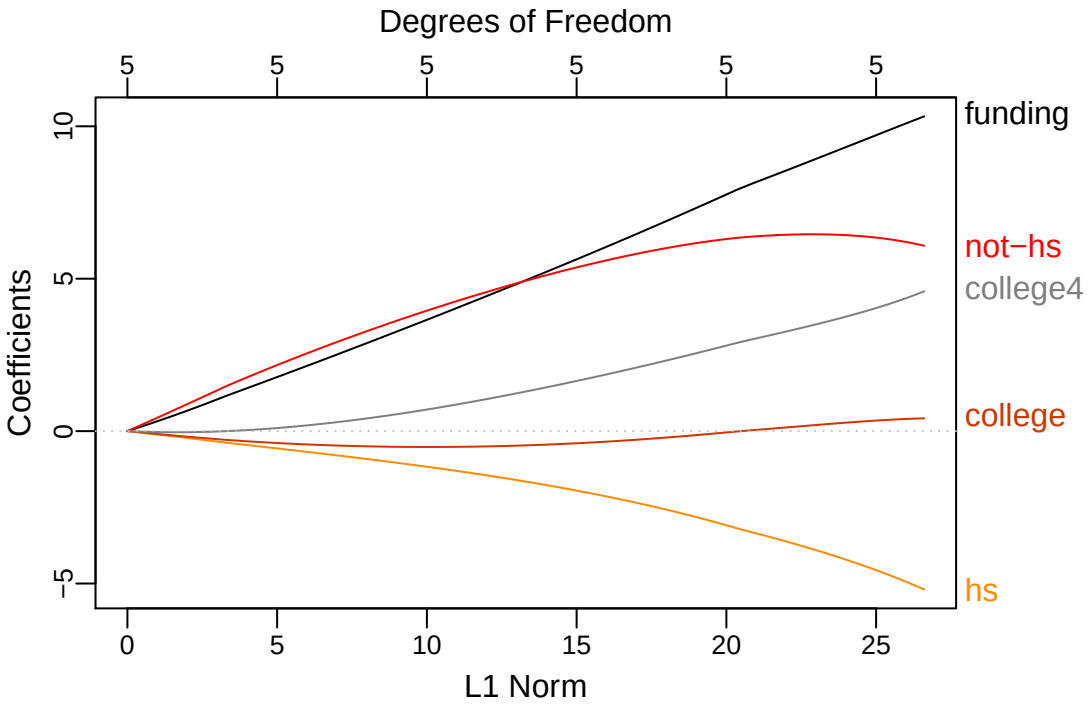

```
library(plotmo) # for plot_glmnet # install.packages("plotmo")
plot_glmnet(crime_lasso, xvar = "norm")
```




```
crime_ridge <- glmnet(`crime rate` ~ ., alpha = 0, data = crime) ## alpha = 0: Ridge Regression
plot(crime_ridge)
```

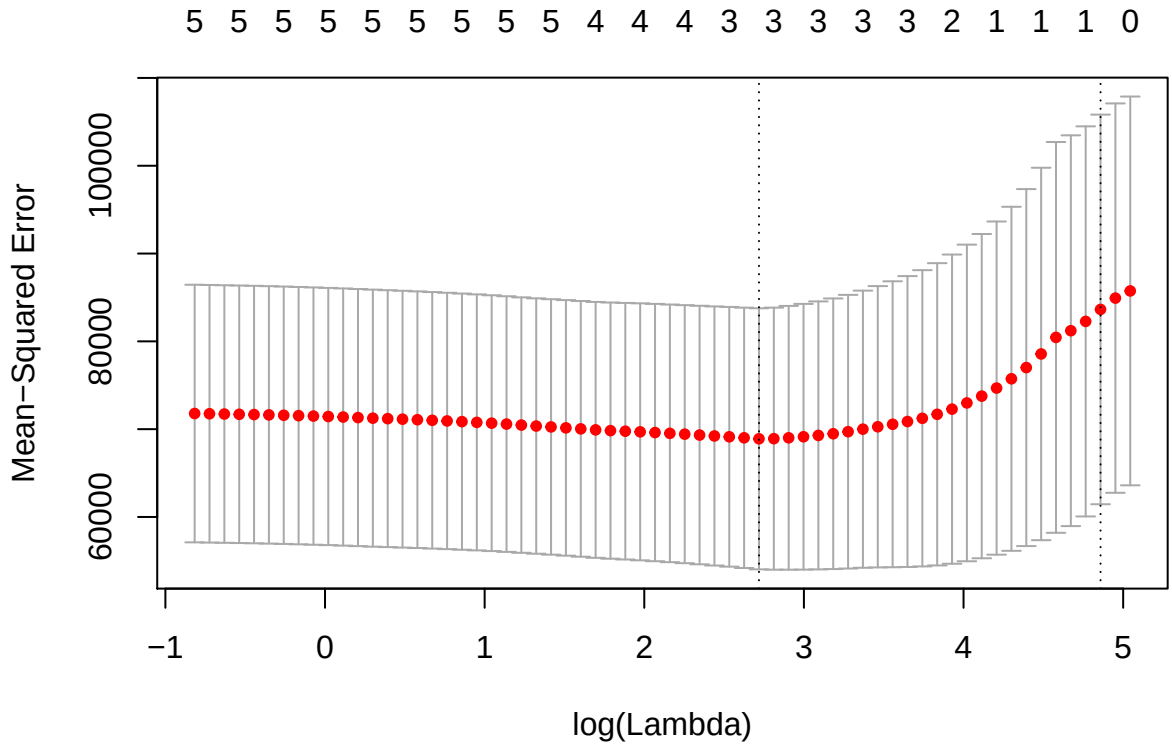



```
plot_glmnet(crime_ridge, xvar = "norm")
```

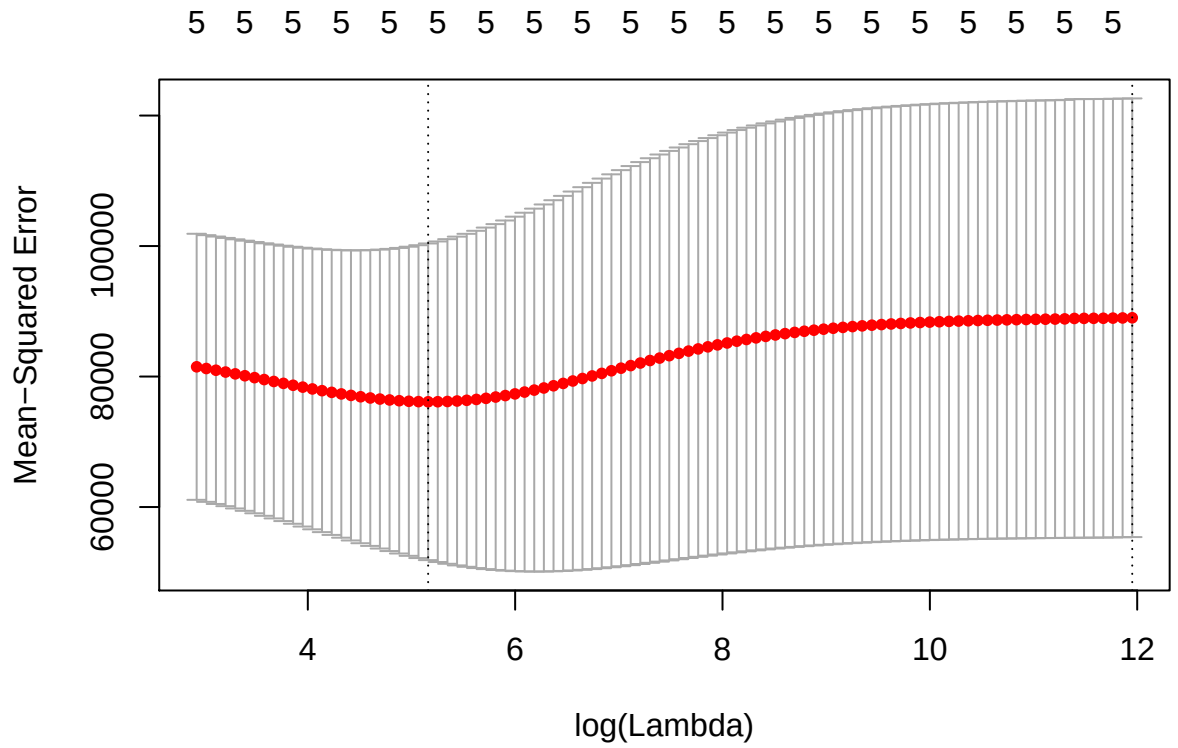


What should λ be?

```
cv_glmnet_lasso <- cv.glmnet(`crime rate` ~ ., alpha = 1, data = crime)
plot(cv_glmnet_lasso)
```



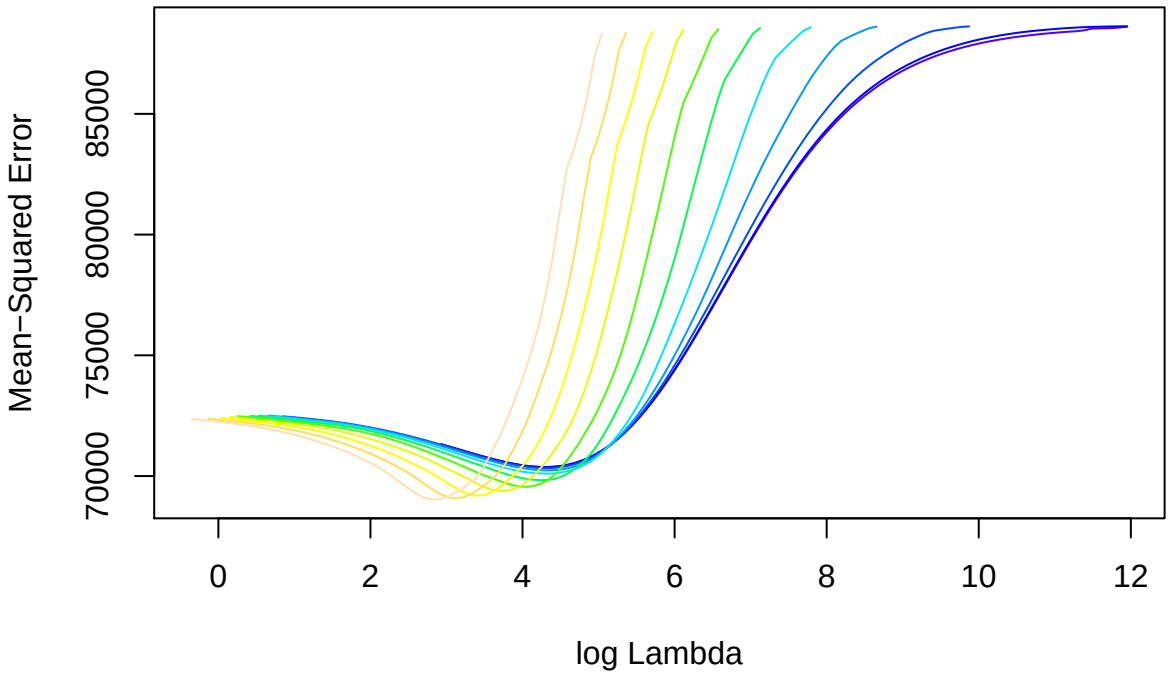
```
cv_glmnet_ridge <- cv.glmnet(`crime rate` ~ ., alpha = 0, data = crime) ## alpha = 0: Ridge
plot(cv_glmnet_ridge)
```



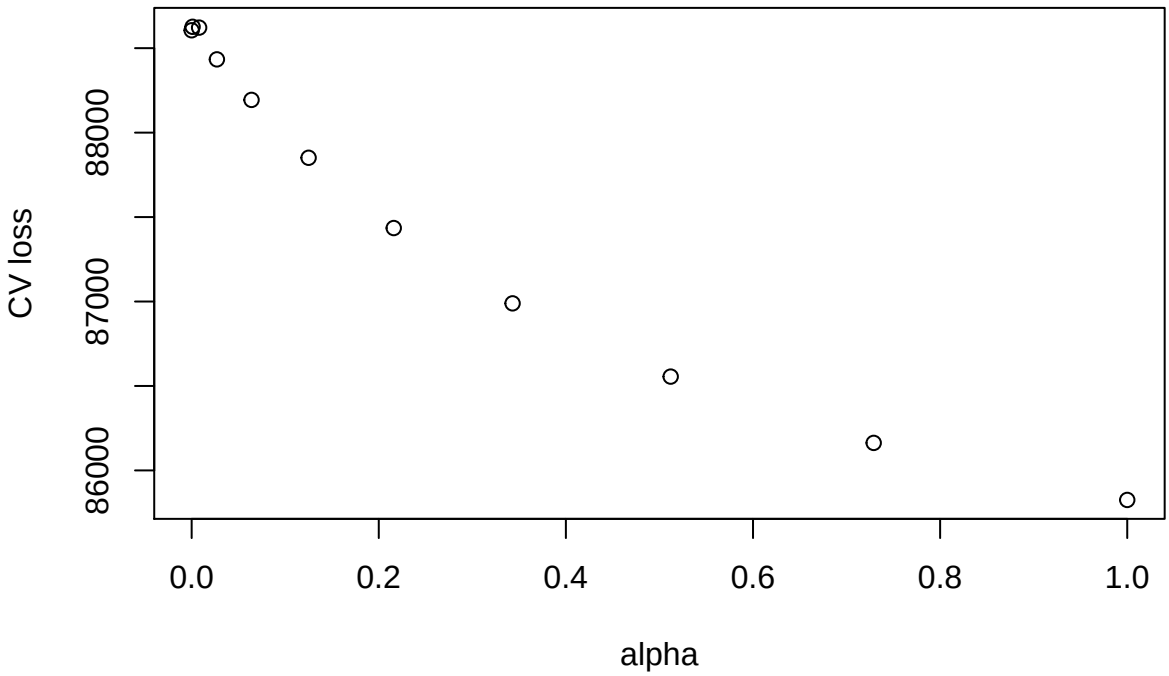
Elastic net approach: What should α be?

```
cv_glmnet_enet <- cva.glmnet(`crime rate` ~ ., data = crime)
```

```
plot(cv_glmnet_enet)
```




```
minlossplot(cv_glmnet_enet)
```



Other types of regression

There are many different types of regressions one would be interested in:

- Logistic regression (binary outcomes): `family = "binomial"`
- Count regression (Poisson distribution): `family = "poisson"`
- Multiclass (multinomial data): `family = "multinomial"`

- Survival analysis (Cox proportional hazard model): `family = "cox"`
- Multivariate normal (multivariate Gaussian): `family = "mgaussian"`

Classification And Regression Trees

In R using `rpart`

August 21, 2018

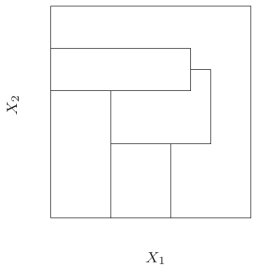
Torben Tvedebrink
tvede@math.aau.dk

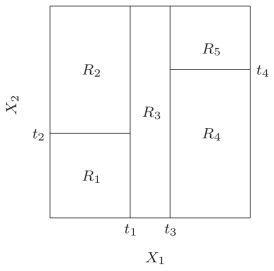
Data Science using R

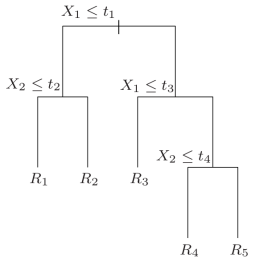
AALBORG UNIVERSITY
DENMARK

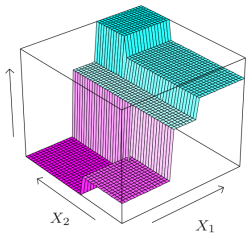
CART: Classification And Regression Trees

Link: Introduction to `rpart`









CART: Regression

For regression the CART methodoloy fits a piecewise 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 construcs a piecewise constant function

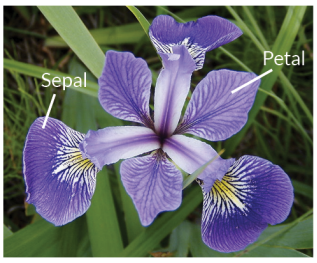
$$\hat{Y}_{\text{CART}}(\boldsymbol{x}) = \sum_{j=1}^{R} \beta_j \mathbb{I}(\boldsymbol{x} \in R_j),$$

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

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

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

Eksempel
Iris data – three species



Iris Versicolor

Iris Setosa

Iris Virginica

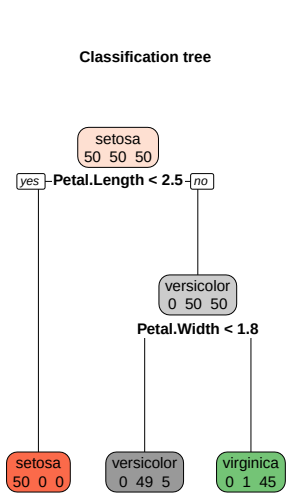
```
> iris[c(1:2,51:52,101:102),]
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
51           7.0         3.2          4.7         1.4 versicolor
52           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
```

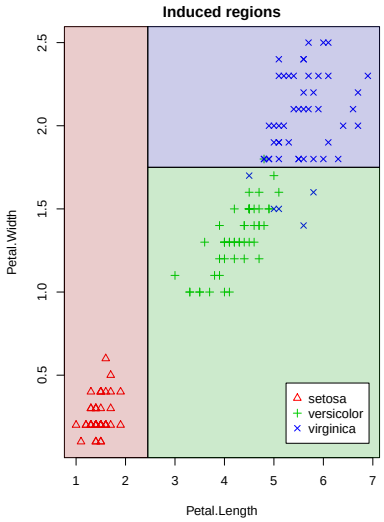

Eksempel
Iris data

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 exhautive 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 $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})$ 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 $R_1, \ldots, R_r$, split each R_j into R_{j_1} and R_{j_2} where

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

   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 $R_{1_1}, R_{1_2}, \ldots, R_{r_1}, R_{r_2}$ 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 untill each observations has its own leaf (terminal node). This gives an error rate of null, but not very enlightning!.

Hence, stop before that, but when?

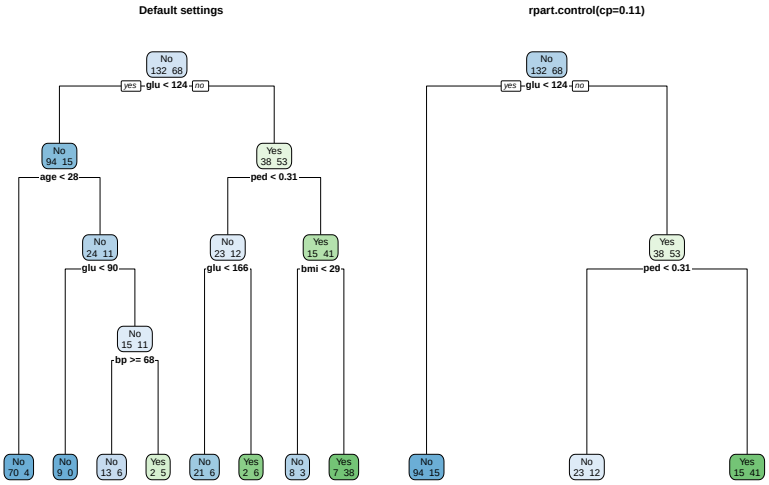
Example
Pima indians

Female descendents from the Pima indians above 21 years of age and living near Phonix, Arizona, was included in a study. Each female was tested for diabetes according to WHO's criteria.

The variables in the data includes apart from diabetest status (`type`), information on

- number of pregnancies (`npreg`),
- plasma glucose concentration (`glu`)
- blood pressure (`bp`),
- triceps skin fold thickness (mm) (`skin`),
- BMI (`bmi`),
- diabetes pedigree function (`ped`) and
- age (`age`).

Two different trees
Pima indians – Cont'd



Bias vs. variance

Why did I choose `rpart.control(cp=0.11)` in the analysis of the Pima indians? This *tuning parameter* 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 (overfitting!), 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 unknwon, but may be approximated using cross-validation.

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

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

Torben Tvedebrink
tvede@math.aau.dk

Choosing the *optimal* tree
Tuning parameter α

CART

We wants to search for the *optimal* tree T^*, that minimises the *true* test error, $\text{Error}_{\text{Test}}$. This quantity is unknwon, 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.

Regression
Classification
Example
Estimation
Partitioning
14 Model complexity
Pruning
Surrogates

22
Torben Tvedebrink
tvede@math.aau.dk

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

CART

Regression
Classification
Example
Estimation
Partitioning
Model complexity
15 Pruning
Surrogates

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

Selecting α

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

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

Selecting α

CART

Regression
Classification
Example
Estimation
Partitioning
Model complexity
16 Pruning
Surrogates

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 $I_1 = [0, \alpha_1]$, $I_2 = (\alpha_1, \alpha_2]$, $\ldots$, $I_m = (\alpha_{m-1}, \infty]$. 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:

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

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'.

Choise 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 `xerorr`.

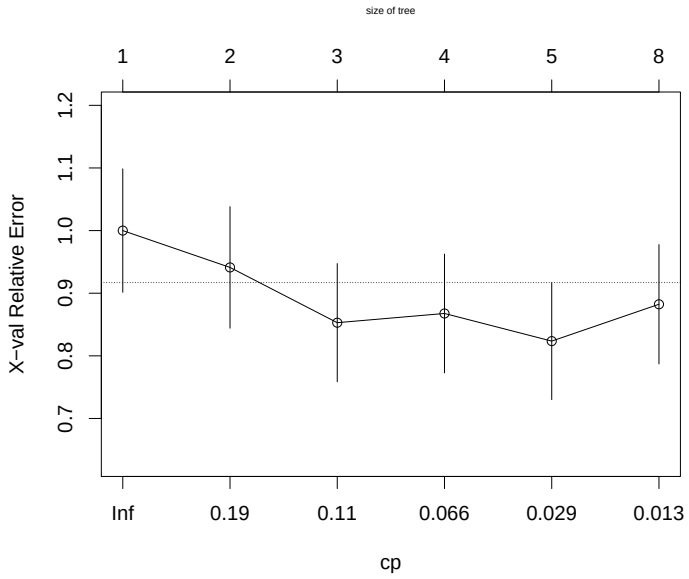
In the `plotcp`-plot the dotted line shows `xerror`+`xstd` relative to the cp-value with smallest `xerror`.

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).

Eksempel

Pima indians – Cont'd



Eksempel
Pima indianere – Cont'd


```
set.seed(13454)
pima.cp <- rpart(type~.,data=Pima.tr,cp=0.012)
printcp(pima.cp)

Classification tree:
rpart(formula = type ~ ., data = Pima.tr, cp = 0.012)

Variables actually used in tree construction:
[1] age bmi bp  glu ped

Root node error: 68/200 = 0.34

n= 200

        CP nsplit rel error  xerror     xstd
1 0.220588      0   1.00000 1.00000 0.098518
2 0.161765      1   0.77941 0.97059 0.097791
3 0.073529      2   0.61765 0.79412 0.092331
4 0.058824      3   0.54412 0.77941 0.091785
5 0.014706      4   0.48529 0.69118 0.088180
6 0.012000      7   0.44118 0.77941 0.091785
```


Surrogates

A nice feature of the CART methodoloty are the so called *surrogates*. These are variables in the data that are not choosen 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.

rpart - pima indians

Torben

August 21, 2018

```
library(rpart)
library(rpart.plot)
set.seed(123)
```

The data on the Pima indians can be found in the `MASS` package

```
data(Pima.tr, package = "MASS")
head(Pima.tr)
```

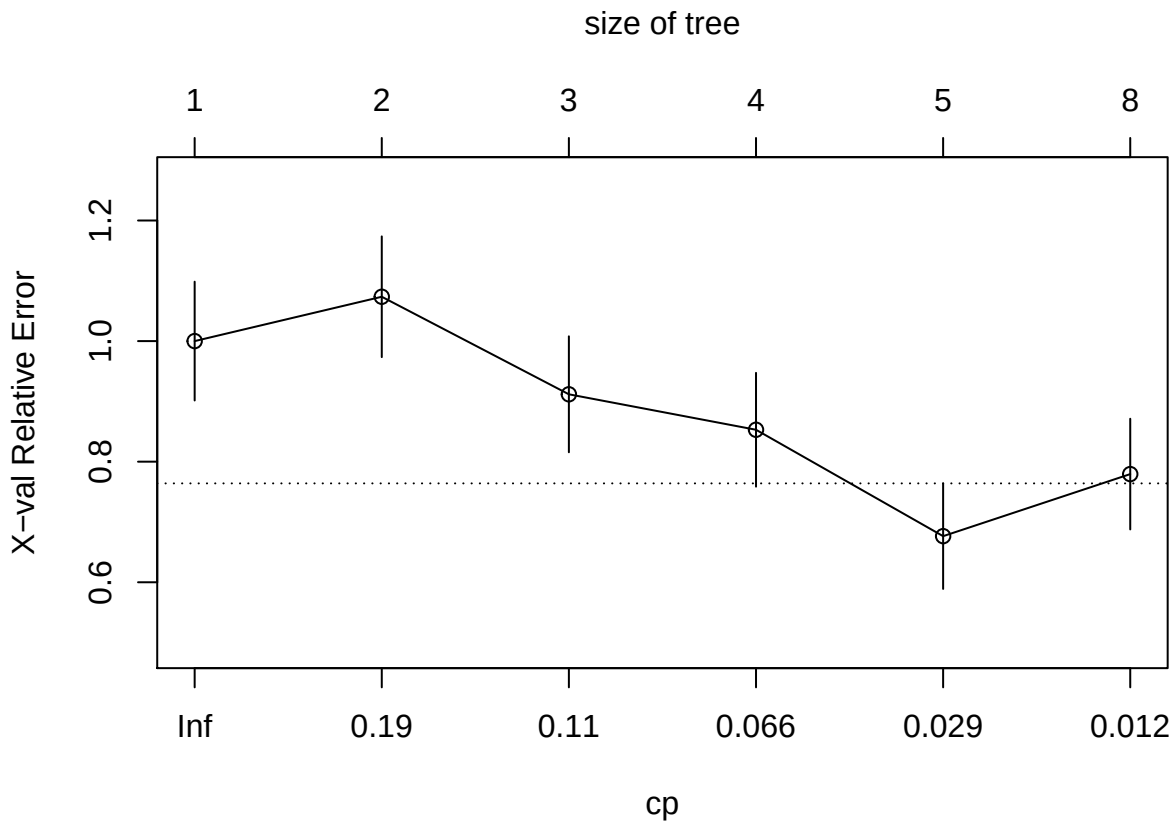
```
##   npreg glu bp skin  bmi   ped age type
## 1     5  86 68   28 30.2 0.364  24   No
## 2     7 195 70   33 25.1 0.163  55  Yes
## 3     5  77 82   41 35.8 0.156  35   No
## 4     0 165 76   43 47.9 0.259  26   No
## 5     0 107 60   25 26.4 0.133  23   No
## 6     5  97 76   27 35.6 0.378  52  Yes
```

Fit the a `rpart` model by default settings

```
pima_rp1 <- rpart(type ~ ., data = Pima.tr)
```

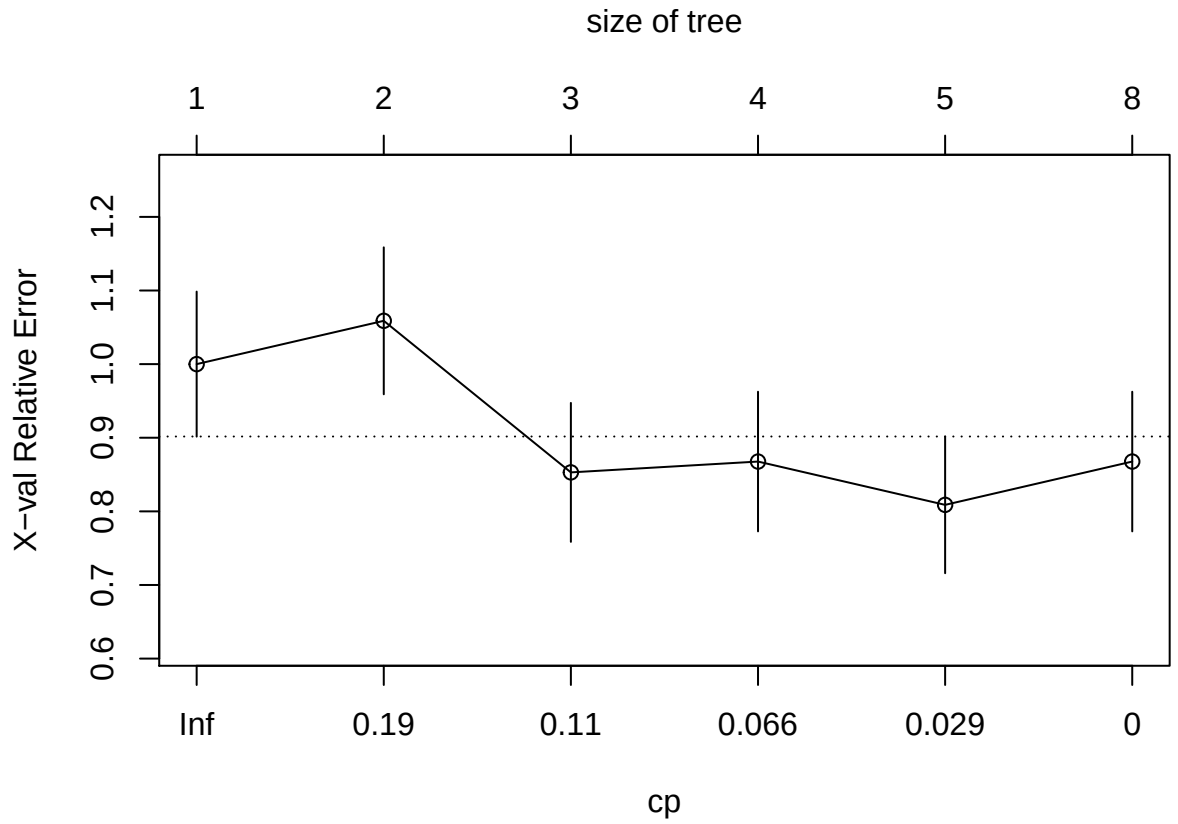
Look at the `cp` complexity parameter

```
plotcp(pima_rp1)
```



Looking for plateau effect

```
set.seed(2018)
pima_rp2 <- rpart(type~.,data=Pima.tr,cp=0)
plotcp(pima_rp2)
```

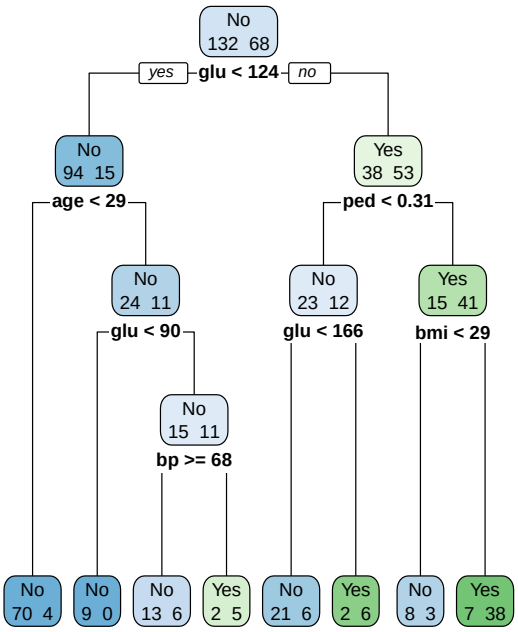



```
pima_rp2_pruned <- prune(pima_rp2, cp = 0.11)
```

Plot the trees

```
par(mfrow=c(1,2))
rpart.plot(pima_rp1, main="Default settings",
           xcompact=FALSE, ycompact=FALSE, type=2, extra=1)

rpart.plot(pima_rp2_pruned, main="rpart.control(cp=0.11)",
           xcompact=FALSE, ycompact=FALSE, type=2, extra=1)
```

Default settings



rpart.control(cp=0.11)