Lecture A

Exhaustive and non-exhaustive algorithms without resampling

Samuel Müller and Garth Tarr

THE UNIVERSITY OF SYDNEY

> "Doubt is not a pleasant mental state, but certainty is a ridiculous one."
>
> — *Voltaire (1694-1778)*

- Necessity to select **models** is great despite philosophical issues
- Myriad model/variable selection methods based on loglikelihood, RSS, prediction errors, resampling, etc

Some motivating questions

- How to **practically select model(s)**?
- How to cope with an ever **increasing number of observed variables**?
- How to **visualise** the model building process?
- How to assess the **stability** of a selected model?

Topics

A1. Selecting models

- Stepwise model selection
- Information criteria including AIC and BIC
- Exhaustive and non-exhaustive searches

A2 Regularisation methods

A3 Marginality constraints

A1. Selecting models

Body fat data

- $N = 252$ measurements of men (available in R package `mfp`)
- Training sample size $n = 128$ (available in R package `mplot`)
- Response variable: body fat percentage: `Bodyfat`
- 13 or 14 explanatory variables

```
data("bodyfat", package = "mplot")
dim(bodyfat)
```

```
## [1] 128  15
```

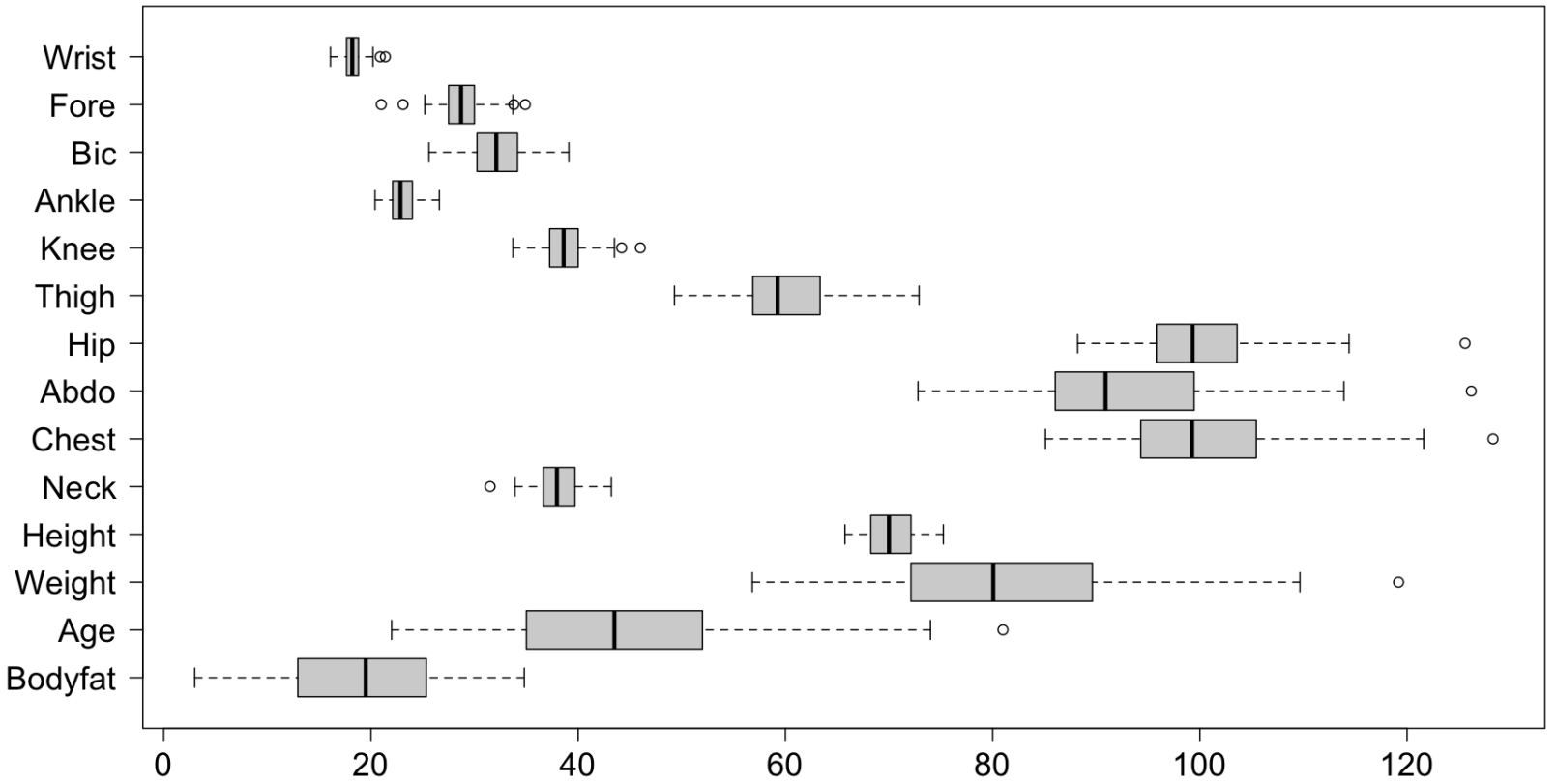
```
names(bodyfat)
```

```
##  [1] "Id"      "Bodyfat" "Age"     "Weight"  "Height"  "Neck"
##  [7] "Chest"   "Abdo"    "Hip"     "Thigh"   "Knee"    "Ankle"
## [13] "Bic"     "Fore"    "Wrist"
```

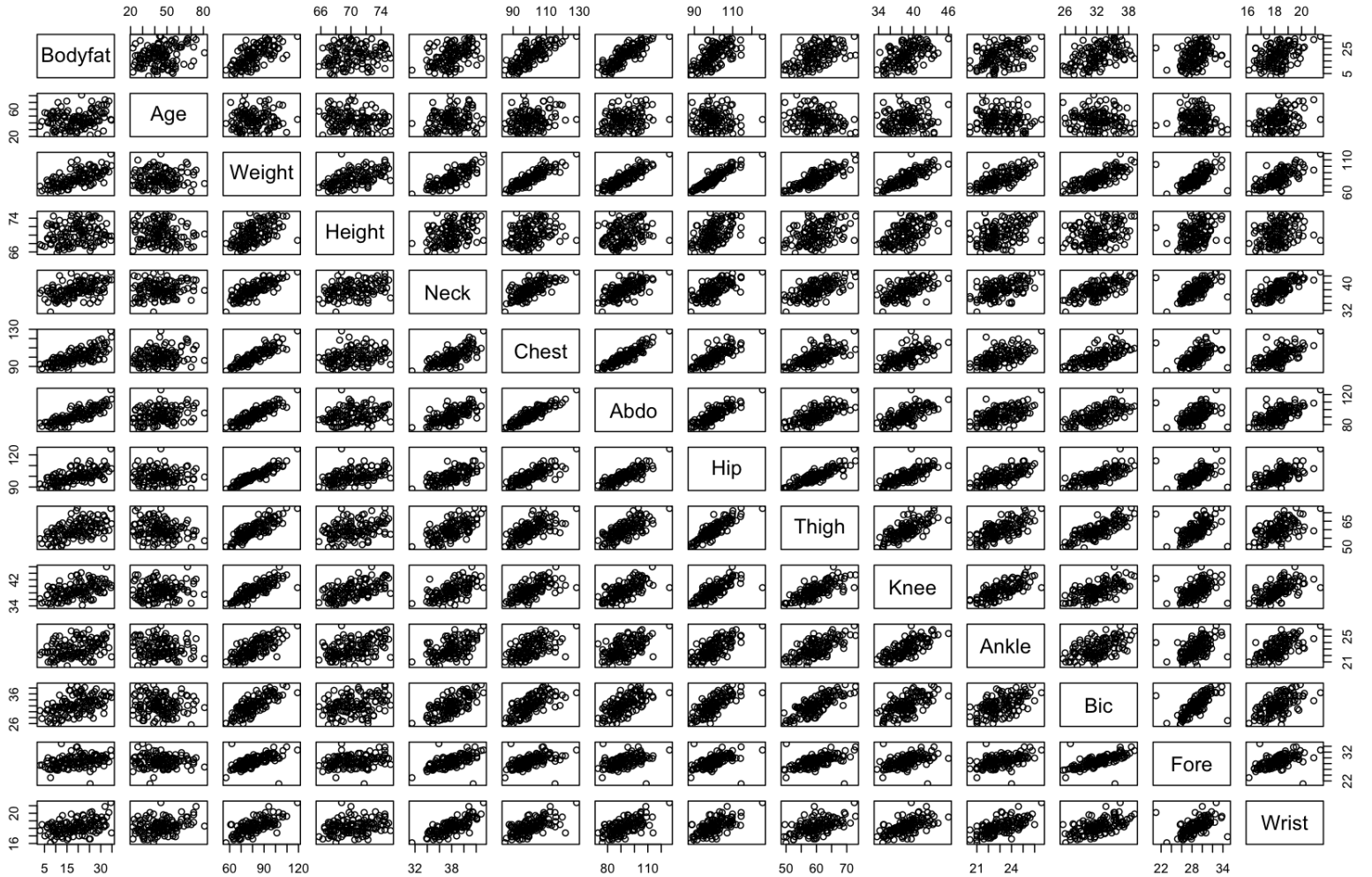
Source: Johnson (1996)

Body fat: Boxplots

```
bfat = bodyfat[,-1] # remove the ID column
boxplot(bfat, horizontal = TRUE, las = 1)
```



```
pairs(bfat)
```


Bodyfat
Age
Weight
Height
Neck
Chest
Abdo
Hip
Thigh
Knee
Ankle
Bic
Fore
Wrist

Body fat data: Full model and power set

```
names(bfat)
```

```
##  [1] "Bodyfat" "Age"     "Weight"  "Height"  "Neck"    "Chest"
##  [7] "Abdo"    "Hip"     "Thigh"   "Knee"    "Ankle"   "Bic"
## [13] "Fore"    "Wrist"
```

Full model

$$\begin{aligned}\text{Bodyfat}=&\beta_1+\beta_2\text{Age}+\beta_3\text{Weight}+\beta_4\text{Height}+\beta_5\text{Height}+\\&\beta_6\text{Chest}+\beta_7\text{Abdo}+\beta_8\text{Hip}+\beta_9\text{Thigh}+\beta_{10}\text{Knee}+\\&\beta_{11}\text{Ankle}+\beta_{12}\text{Bic}+\beta_{13}\text{Fore}+\beta_{14}\text{Wrist}+\epsilon\end{aligned}$$

Power set

Here, the intercept is not subject to selection. Therefore, $p-1=13$ and thus the number of possible regression models is:

$$M=\#\mathcal{A}=2^{13}=8192$$

Body fat: Best bivariate fitted model

```
M0 = lm(Bodyfat ~ Abdo , data = bfat)
summary(M0)
```

```
##
## Call:
## lm(formula = Bodyfat ~ Abdo, data = bfat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.3542  -2.9928   0.2191   2.4967  10.0106
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -43.19058    3.63431  -11.88   <2e-16 ***
## Abdo          0.67411    0.03907   17.26   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.249 on 126 degrees of freedom
## Multiple R-squared:  0.7027,    Adjusted R-squared:  0.7003
## F-statistic: 297.8 on 1 and 126 DF,  p-value: < 2.2e-16
```

Body fat: Full fitted model

```
M1 = lm(Bodyfat ~ ., data = bfat)
summary(M1)
```

```
##
## Call:
## lm(formula = Bodyfat ~ ., data = bfat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.3767 -2.5514 -0.1723  2.6391  9.1393
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -52.553646  40.062856  -1.312   0.1922
## Age           0.009288   0.043470   0.214   0.8312
## Weight       -0.271016   0.243569  -1.113   0.2682
## Height        0.258388   0.320810   0.805   0.4223
## Neck         -0.592669   0.322125  -1.840   0.0684 .
## Chest         0.090883   0.164738   0.552   0.5822
## Abdo          0.995184   0.123072   8.086 7.29e-13 ***
## Hip          -0.141981   0.204533  -0.694   0.4890
## Thigh         0.101272   0.200714   0.505   0.6148
## Knee         -0.096682   0.325889  -0.297   0.7673
## Ankle        -0.048017   0.507695  -0.095   0.9248
## Bic           0.075332   0.244105   0.309   0.7582
## Fore          0.412107   0.272144   1.514   0.1327
## Wrist        -0.263067   0.745145  -0.353   0.7247
##
```

The AIC and BIC

- AIC is the most widely known and used model selection method (of course this does not imply it is the best/recommended method to use)
- AIC Akaike (1973):

$$\text{AIC} = -2 \times \text{LogLik} + 2 \times p$$

- BIC Schwarz (1978):

$$\text{BIC} = -2 \times \text{LogLik} + \log(n) \times p$$

- The smaller the AIC/BIC the better the model
- These and many other criteria choose models by minimizing an expression that can be written as

$$\text{Loss} + \text{Penalty}$$

Stepwise variable selection

Stepwise, forward and backward:

1. Start with **some** model, typically null model (with no explanatory variables) or full model (with all variables)
2. For each variable in the current model, **investigate** effect of **removing** it
3. **Remove** the least informative variable, unless this variable is nonetheless supplying significant information about the response
4. For each variable not in the current model, **investigate** effect of **including** it
5. **Include** the most statistically significant variable not currently in model (unless no significant variable exists)
6. Go to step 2. Stop only if no change in steps 2-5

Body fat: Forward search using BIC

```
M0 = lm(Bodyfat ~ 1, data = bfat)  # Null model
M1 = lm(Bodyfat ~ ., data = bfat)  # Full model
step.fwd.bic = step(M0, scope = list(lower = M0, upper = M1),
                    direction = "forward",
                    trace = FALSE,
                    k = log(128)) # BIC: k=log(n)
# summary(step.fwd.bic)
round(summary(step.fwd.bic)$coef, 3)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -47.991      3.691 -13.003        0
## Abdo           0.939      0.080  11.743        0
## Weight        -0.243      0.065  -3.740        0
```

- Best model has two features: Abdo and Weight
- Backward search using BIC gives the same model (confirm yourself)

Body fat: Backward search using AIC

```
M0 = lm(Bodyfat ~ 1, data = bfat)  # Null model
M1 = lm(Bodyfat ~ ., data = bfat)  # Full model
step.bwd.aic = step(M1, scope = list(lower = M0, upper = M1),
                    direction = "backward",
                    trace = FALSE,
                    k = 2)
round(summary(step.bwd.aic)$coef, 3)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -41.883      8.310  -5.040    0.000
## Weight        -0.229      0.079  -2.915    0.004
## Neck          -0.619      0.266  -2.325    0.022
## Abdo           0.980      0.081  12.036    0.000
## Fore           0.434      0.238   1.821    0.071
```

- Best model has four variables
- Using AIC gives larger model because $2 < \log(128) = 4.85$
- `k=2` is the default in `step()`

Body fat: Stepwise using AIC

```
M0 = lm(Bodyfat ~ 1, data = bfat)  # Null model
M1 = lm(Bodyfat ~ ., data = bfat)  # Full model
step.aic = step(M1, scope = list(lower = M0, upper = M1),
                trace = FALSE)
round(summary(step.aic)$coef, 3)
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -41.883      8.310  -5.040    0.000
## Weight        -0.229      0.079  -2.915    0.004
## Neck          -0.619      0.266  -2.325    0.022
## Abdo           0.980      0.081  12.036    0.000
## Fore           0.434      0.238   1.821    0.071
```

- Here, default stepwise gives the same as backward using AIC
- This does not hold in general

Criticism of stepwise procedures

- **Never run (p value based) automatic stepwise procedures on their own!**

> Stepwise regression is probably the most abused computerized statistical technique ever devised. If you think you need stepwise regression to solve a particular problem you have, it is almost certain you do not. Professional statisticians rarely use automated stepwise regression.
>
> —*Wilkinson (1998)*

- See Wiegand (2010) for a more recent simulation study and review of the performance of stepwise procedures

Exhaustive search

Exhaustive search is the only technique **guaranteed** to find the predictor variable subset with the best evaluation criterion.

- Since we look over the whole model space, we can identify the best model(s) at each model size
- Sometimes known as **best subsets** model selection
- **Loss** component is (typically) the residual sum of squares
- Main drawback: exhaustive searching is **computationally intensive**

The `lmSubsets` package

- Main function: `lmSubsets()` performs an **exhaustive search** of the model space to find all-subsets selection for predicting `y` in **linear regression** (Hofmann, Gatu, Kontoghiorghes, Colubi, and Zeileis, 2020)
- Uses an efficient **branch-and-bound** algorithm
- The `lmSelect()` function performs best-subset selection

Historical note: the `leaps` package also performs regression subset selection including **exhaustive searches** (Lumley and Miller, 2009) for *linear models*, the `lmSubsets` package seems to be faster at achieving a similar goal and offers a few more features.

The `lmSubsets` function

- Input: **full model** and various optional parameters (e.g. `include` and `exclude` parameters to force variables in or out of the model space; `nmin` and `nmax` to specify the smallest and largest model space; and `nbest` for the number of best subsets to report in each model size).
- Output: a `lmSubsets` object but you interact with it using the `summary` function which reports model summary statistics (e.g. $\hat{\sigma}$, R^2, AIC, BIC)
- Plot: there's also a plot method which can be applied to the `lmSubsets` object

The `lmSelect` function

The `lmSelect` function is similar to the `lmSubsets` function but it helps the user choose a "best" model.

- Input: **full model** and various optional parameters including the `penalty` parameter
  - `penalty = "BIC"` is the default
  - `penalty = 2` or `penalty = "AIC"` gives AIC
  - or a more general function `penalty = function (size, rss)` where where `size` is the number of regressors, and `rss` the residual sum of squares of the corresponding submodel
- Output: a `lmSelect` object. In particular, the list has elements such as `subset` (identifies the selected variables)

The `refit()` can be used to fit the selected model and return an `lm` object.

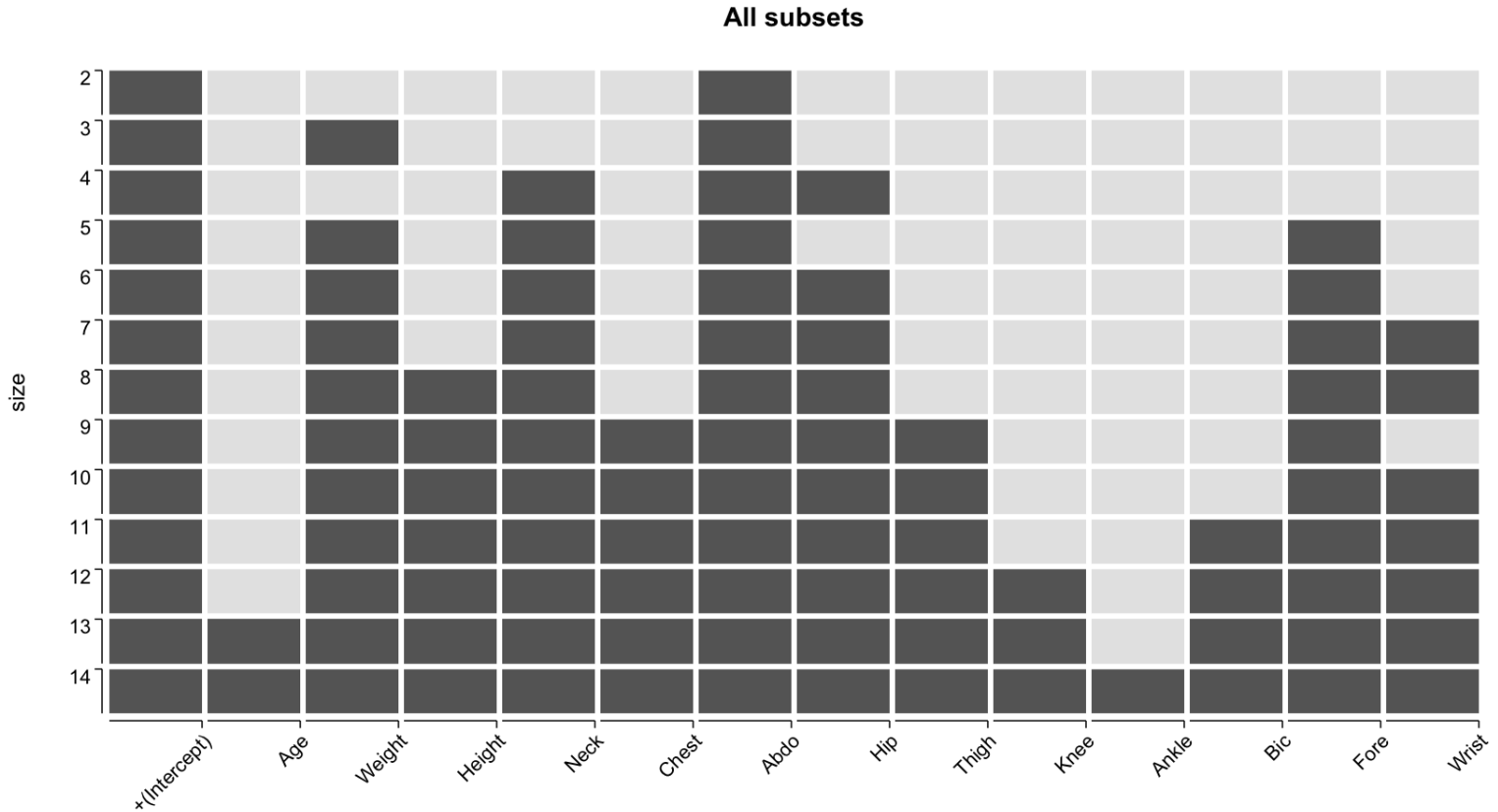
Body fat: Exhaustive search

```
library(lmSubsets)
rgsbst.out = lmSubsets(Bodyfat ~ ., data = bfat,
                       nbest = 1, # 1 best model for each dimension
                       nmax = NULL, # NULL for no limit on number of variables
                       include = NULL, exclude = NULL)
rgsbst.out
```

```
## Call:
##   lmSubsets(formula = Bodyfat ~ ., data = bfat, nbest = 1, nmax = NULL,
##     include = NULL, exclude = NULL)
##
## Deviance:
##   [best, size (tolerance)] = RSS
##       2 (0)    3 (0)    4 (0)    5 (0)    6 (0)   7 (0)    8 (0)
##   1st 2274.919 2045.958 1975.423 1939.099 1922.31 1914.939 1912.117
##       9 (0)    10 (0)   11 (0)   12 (0)  13 (0)   14 (0)
##   1st 1905.726 1902.934 1900.909 1899.58 1898.741 1898.592
##
## Subset:
##   [variable, best] = size
##                 1st
##   +(Intercept) 2-14
##   Age          13-14
##   Weight       3,5-14
##   Height       8-14
##   Neck         4-14
##   Chest        9-14
##   Abdo         2-14
##   Hip          4,6-14
```

Body fat: Exhaustive search

```
image(rgsbst.out)
```



Body fat: Exhaustive search

```
summary.out = summary(rgsbst.out)
names(summary.out)
```

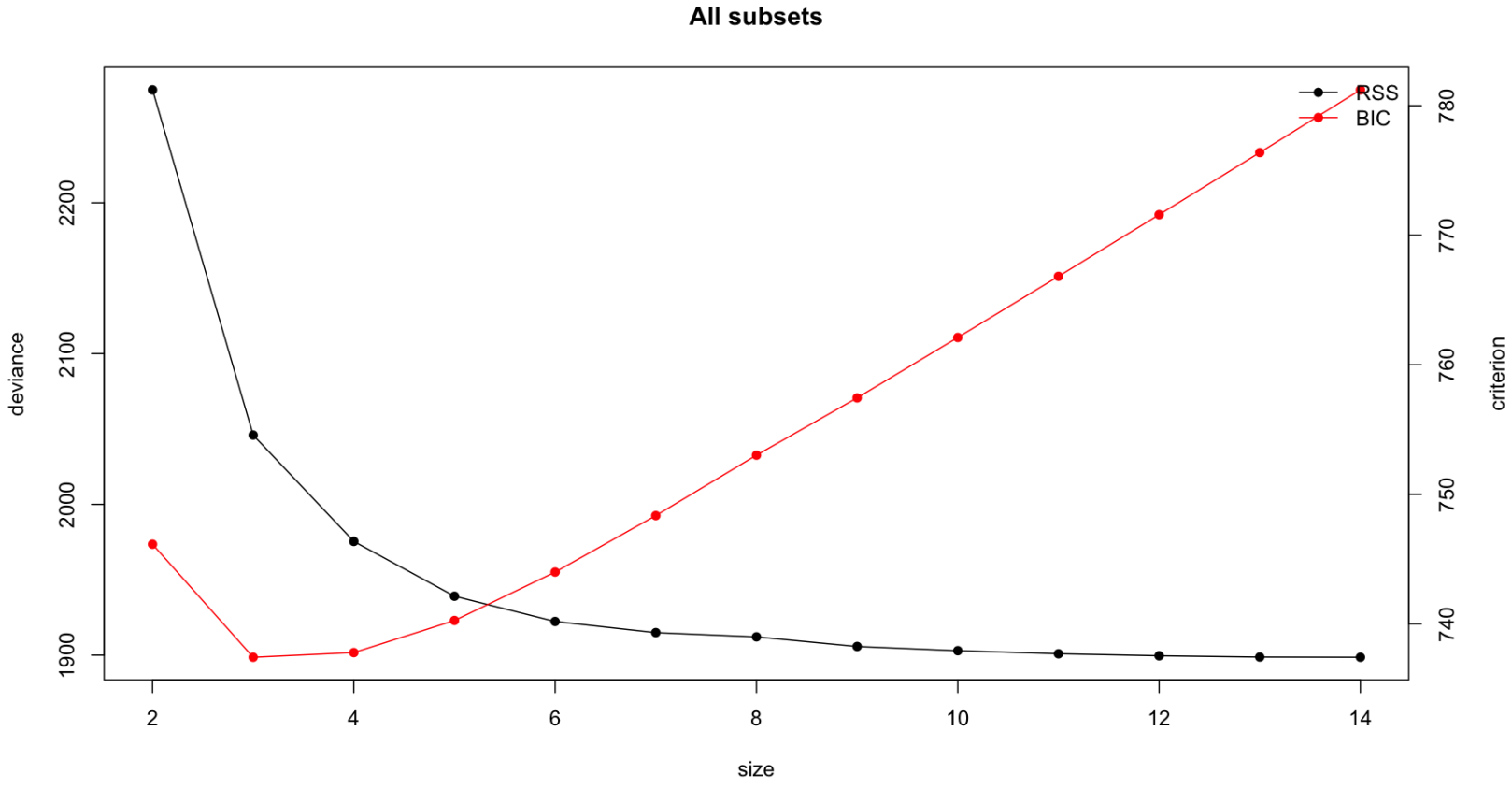
```
## [1] "call"  "terms" "nvar"  "nbest" "size"  "stats"
```

```
round(summary.out$stats, 2)
```

```
##    SIZE BEST sigma   R2 R2adj pval    Cp    AIC    BIC
## 2     2    1  4.25 0.70  0.70    0 12.60 737.59 746.15
## 3     3    1  4.05 0.73  0.73    0  0.85 726.01 737.42
## 4     4    1  3.99 0.74  0.74    0 -1.39 723.52 737.78
## 5     5    1  3.97 0.75  0.74    0 -1.57 723.15 740.26
## 6     6    1  3.97 0.75  0.74    0 -0.58 724.03 744.00
## 7     7    1  3.98 0.75  0.74    0  0.98 725.54 748.36
## 8     8    1  3.99 0.75  0.74    0  2.81 727.35 753.02
## 9     9    1  4.00 0.75  0.73    0  4.43 728.92 757.44
## 10   10    1  4.02 0.75  0.73    0  6.26 730.74 762.11
## 11   11    1  4.03 0.75  0.73    0  8.14 732.60 766.82
## 12   12    1  4.05 0.75  0.73    0 10.06 734.51 771.59
## 13   13    1  4.06 0.75  0.73    0 12.01 736.45 776.38
## 14   14    1  4.08 0.75  0.72    0 14.00 738.44 781.22
```

Body fat: Exhaustive search

```
plot(rgsbst.out)
```



Body fat: Exhaustive search with selection

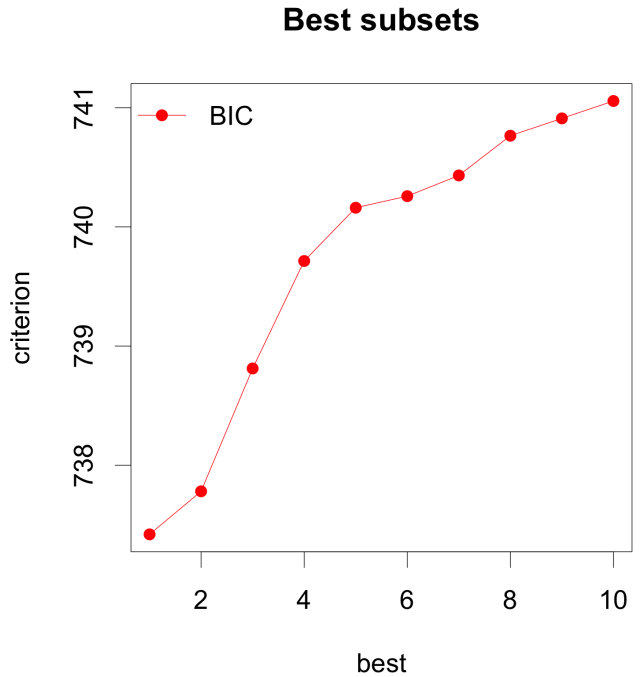
```
best_bic = lmSelect(Bodyfat ~ ., data = bfat)
best_aic = lmSelect(rgsbst.out, penalty = "AIC")
rbind(best_bic$subset,
      best_aic$subset)*1
```

```
##   (Intercept) Age Weight Height Neck Chest Abdo Hip Thigh Knee Ankle Bic Fore Wrist
## 1           1   0      1      0    0     0    1   0     0    0     0   0    0     0
## 2           1   0      1      0    1     0    1   0     0    0     0   0    1     0
```

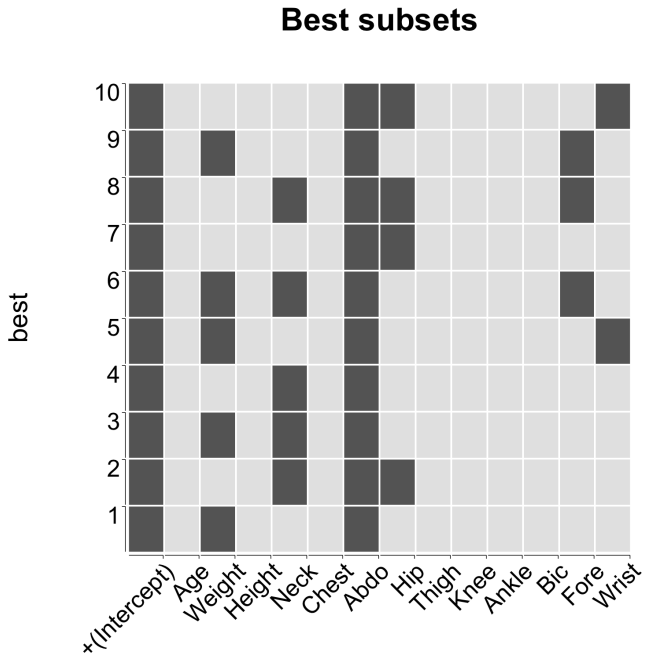
Body fat: Best subsets with BIC

```
best_10_bic = lmSelect(Bodyfat ~ ., data = bfat, nbest = 10)
```

```
plot(best_10_bic)
```




```
image(best_10_bic)
```



A2. Regularisation methods

Regularisation methods with `glmnet`

> Regularisation methods **shrink** estimated regression coefficients by imposing a penalty on their sizes

- There are different choices for the **penalty**
- The penalty choice drives the properties of the method

In particular, `glmnet` implements

- **Lasso regression** using a L_1 penalty
- **Ridge regression** using a L_2 penalty
- **Elastic-net** using both, an L_1 and an L_2 penalty
- **Adaptive lasso**, it takes advantage of feature weights

Lasso is least squares with L_1 penalty

- Simultaneously estimate and select coefficients through

$$\hat{\boldsymbol{\beta}}_{\text{lasso}} = \hat{\boldsymbol{\beta}}_{\text{lasso}}(\lambda) = \text{argmin}_{\boldsymbol{\beta} \in \mathbb{R}^p} \{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda \, ||\boldsymbol{\beta}||_1\}$$

- positive tuning parameter λ controls the shrinkage
- Optimisation problem above is equivalent to solving

$$\text{minimise } \text{RSS}(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leq t.$$

Example: Diabetes

- We consider the `diabetes` data and we only analyse main effects
- The data from the `lars` package is the same as the `mplot` package but is structured and scaled differently

```
data("diabetes", package = "lars")
names(diabetes) # list, slot x has 10 main effects
```

```
## [1] "x"  "y"  "x2"
```

```
dim(diabetes)
```

```
## [1] 442   3
```

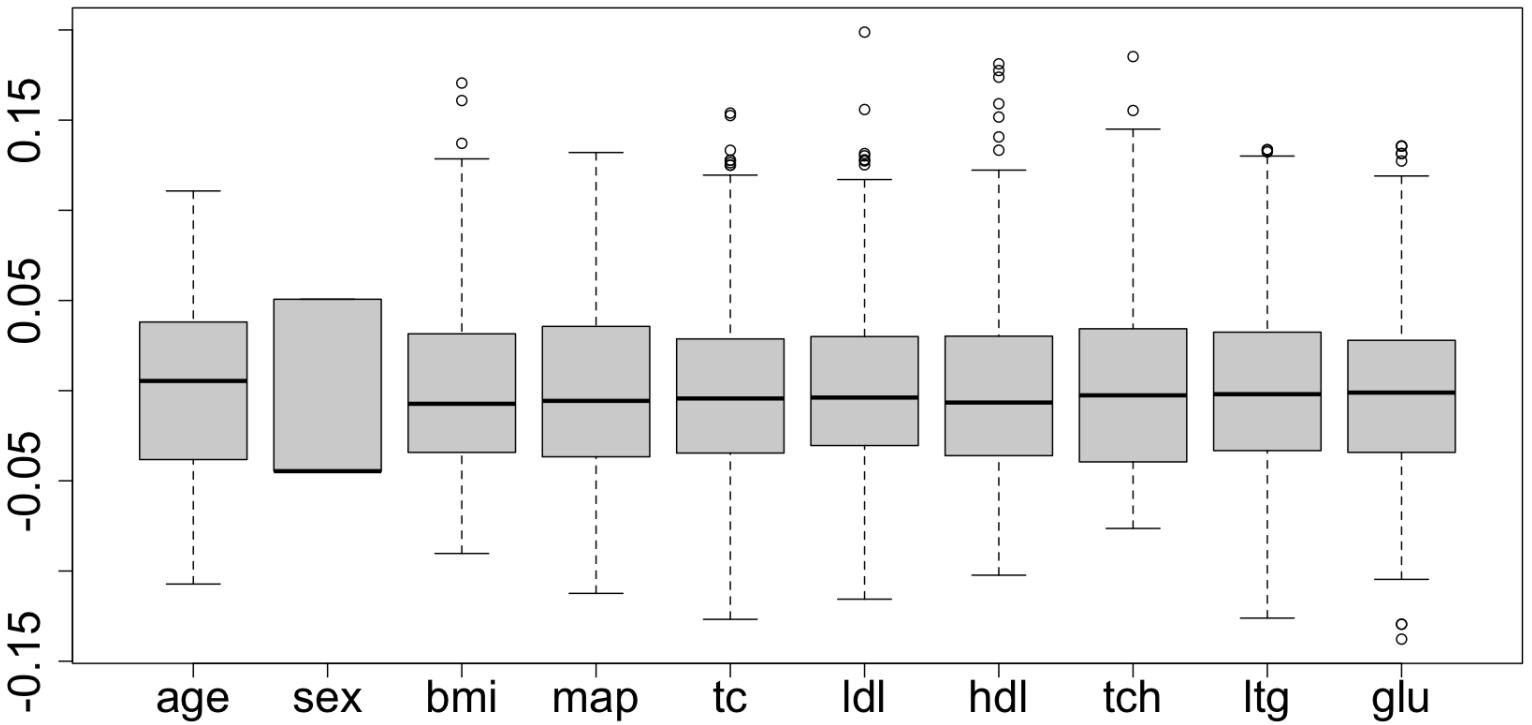
```
x = diabetes$x
dim(x)
```

```
## [1] 442  10
```

```
y = diabetes$y
```

Example: Diabetes

```
class(x) = NULL # remove the AsIs class; otherwise single boxplot drawn
boxplot(x, cex.axis = 2)
```



Example: Diabetes

```
lm.fit = lm(y~x) #least squares fit for the full model first
summary(lm.fit)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -155.829  -38.534   -0.227   37.806  151.355
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  152.133      2.576  59.061  < 2e-16 ***
## xage         -10.012     59.749  -0.168 0.867000
## xsex        -239.819     61.222  -3.917 0.000104 ***
## xbmi         519.840     66.534   7.813 4.30e-14 ***
## xmap         324.390     65.422   4.958 1.02e-06 ***
## xtc         -792.184    416.684  -1.901 0.057947 .
## xldl         476.746    339.035   1.406 0.160389
## xhdl         101.045    212.533   0.475 0.634721
## xtch         177.064    161.476   1.097 0.273456
## xltg         751.279    171.902   4.370 1.56e-05 ***
## xglu          67.625     65.984   1.025 0.305998
```

Fitting a lasso regression

Lasso regression with the `glmnet()` function in `library(glmnet)` has as a first argument the design matrix and as a second argument the response vector, i.e.

```
library(glmnet)
lasso.fit = glmnet(x, y)
```

Comments

- `glmnet.fit` is an object of class glmnet that contains all the relevant information of the fitted model for further use
- Various methods are provided for the object such as `coef`, `plot`, `predict` and `print` that extract and present useful object information
- The additional material shows that cross-validation can be used to choose a good value for λ through using the function `cv.glmnet`

```
set.seed(1) # to get a reproducible lambda.min value
lasso.cv = cv.glmnet(x, y)
lasso.cv$lambda.min
```

```
## [1] 0.09729434
```

Lasso regression coefficients

Clearly the regression coefficients are smaller for the lasso regression than for least squares

```
lasso = as.matrix(coef(lasso.fit, s = c(5, 1, 0.5, 0)))
LS = coef(lm.fit)
data.frame(lasso = lasso, LS)
```

Variable	lasso.1	lasso.2	lasso.3	lasso.4	LS
(Intercept)	152.1	152.1	152.1	152.1	152.1
age	0.0	0.0	0.0	-9.2	-10.0
sex	-45.3	-195.9	-216.4	-239.1	-239.8
bmi	509.1	522.1	525.1	520.5	519.8
map	217.2	296.2	308.3	323.6	324.4
tc	0.0	-101.9	-161.0	-716.5	-792.2
ldl	0.0	0.0	0.0	418.4	476.7
hdl	-147.7	-223.2	-180.5	65.4	101.0
tch	0.0	0.0	66.1	164.6	177.1
ltg	446.3	513.6	524.2	723.7	751.3
glu	0.0	53.9	61.2	67.5	67.6
Sum of abs(coef)	**1365.7**	**1906.7**	**2042.6**	**3248.5**	**3460.0**

Lasso regression vs least squares

```
beta_lasso_5 = coef(lasso.fit, s = 5)
sum(abs(beta_lasso_5[-1]))
```

```
## [1] 1365.672
```

```
beta_lasso_1 = coef(lasso.fit, s = 1)
sum(abs(beta_lasso_1[-1]))
```

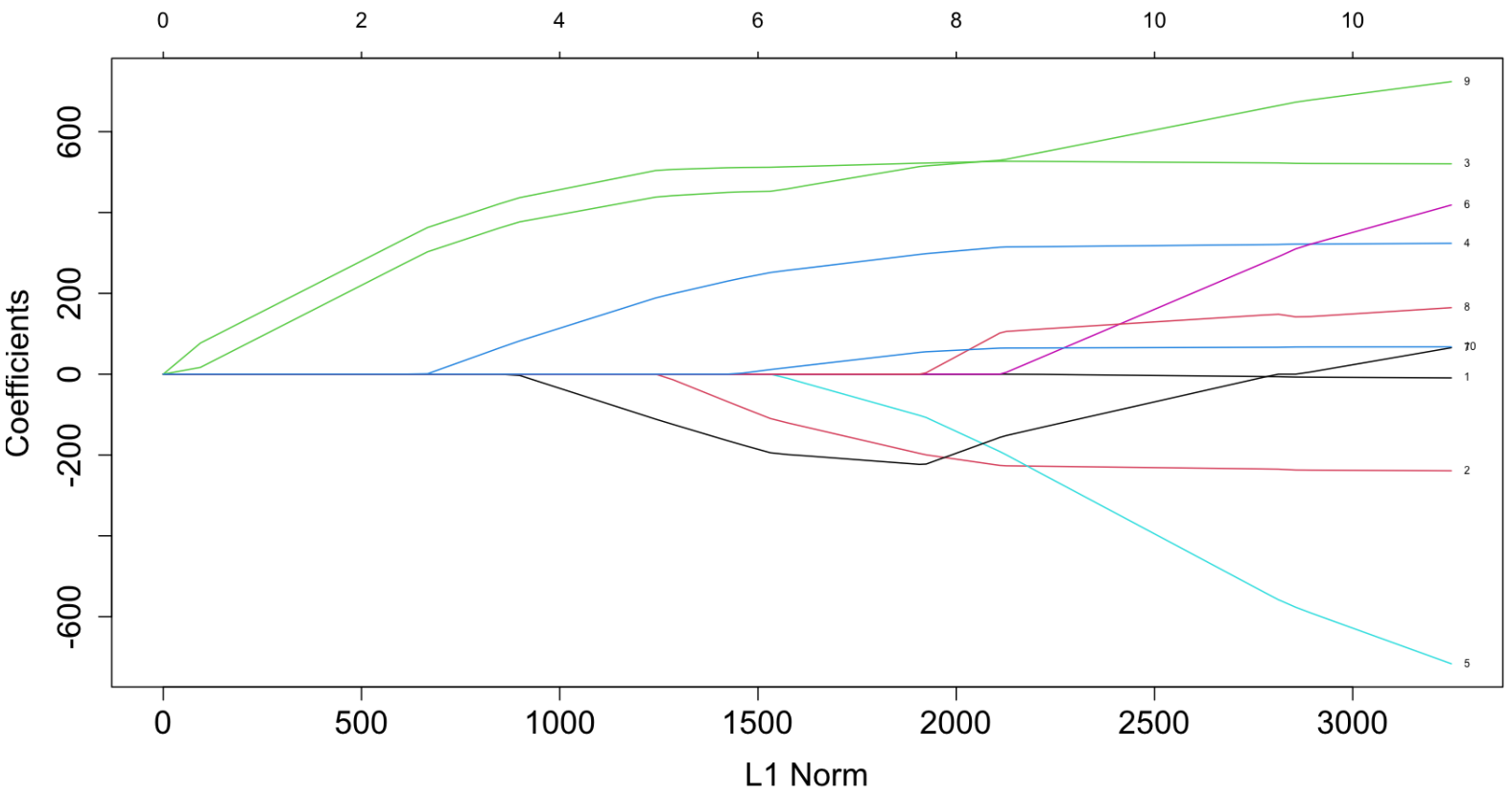
```
## [1] 1906.658
```

```
beta_ls = coef(lm.fit)
sum(abs(beta_ls[-1]))
```

```
## [1] 3460.005
```

Lasso coefficient plot

```
plot(lasso.fit, label=TRUE, cex.axis = 1.5, cex.lab = 1.5)
```



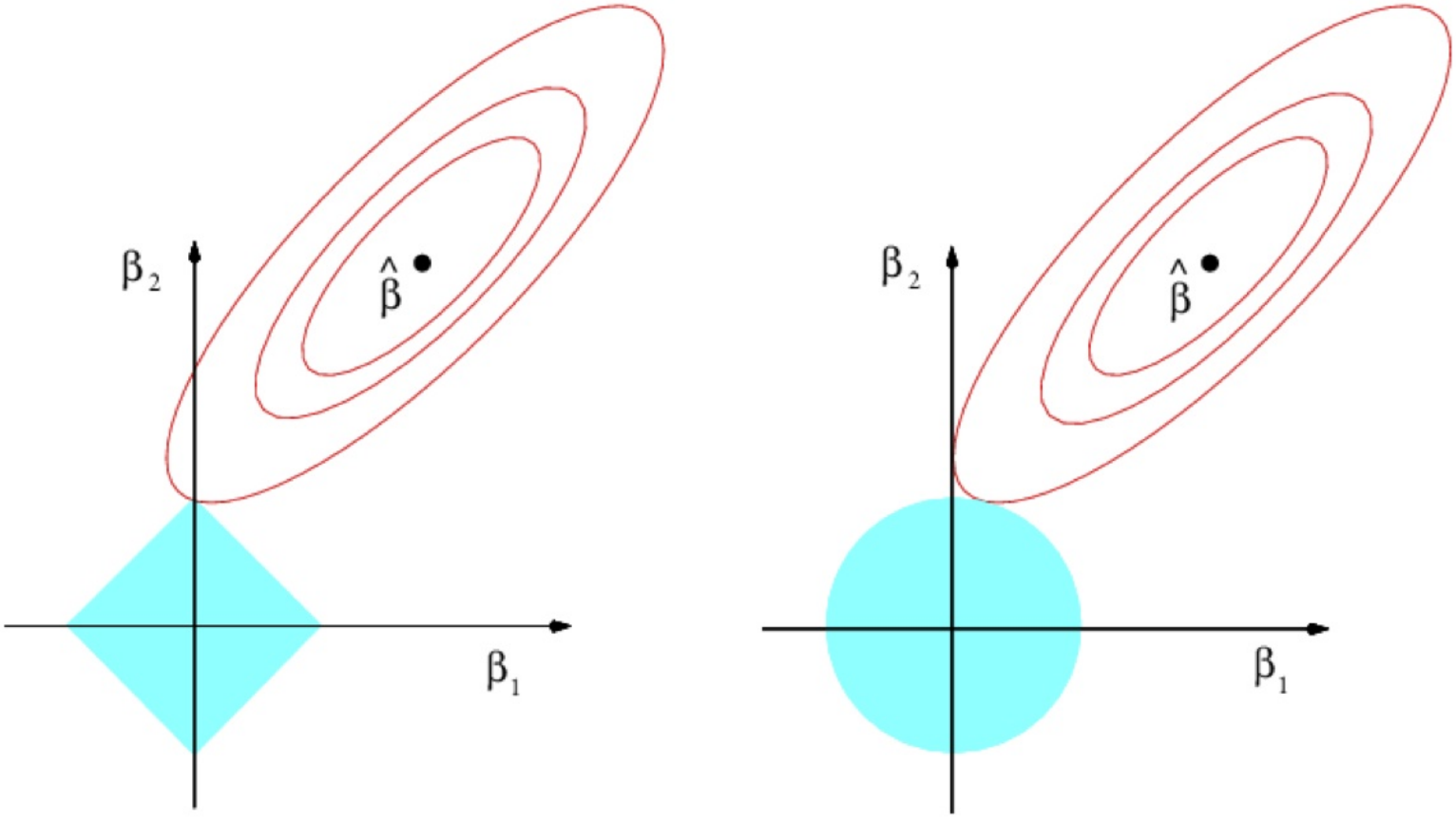
We observe that the lasso

- Can shrink coefficients exactly to zero
- Simultaneously estimates and selects coefficients
- Paths are not necessarily monotone (e.g. coefficients for `hdl` in above table for different values of `s`)

Further remarks on lasso

- **lasso** is an acronym for **l**east **a**bsolute **s**election and **s**hrinkage **o**perator
- Theoretically, when the tuning parameter $\lambda = 0$, there is no bias and $\hat{\boldsymbol{\beta}}_{\text{lasso}} = \hat{\boldsymbol{\beta}}_{\text{LS}}$ ($p \leq n$) however, `glmnet` *approximates* the solution
- When $\lambda = \infty$, we get $\hat{\boldsymbol{\beta}}_{\text{lasso}} = 0 \Rightarrow$ an estimator with zero variance
- For λ in between, we are balancing bias and variance
- As λ increases, more coefficients are set to zero (less variables are selected), and among the nonzero coefficients, more shrinkage is employed

Why does the lasso give zero coefficients?



This figure is taken from James, Witten, Hastie, and Tibshirani (2013) "An Introduction to Statistical Learning, with applications in R" with permission from the authors.

Ridge is least squares with L_2 penalty

- As the lasso, ridge regression shrinks the estimated regression coefficients by imposing a penalty on their sizes

$$\text{minimise } \text{RSS}(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \text{ subject to } \sum_{j=1}^{p} \beta_j^2 \leq t,$$

  where the positive tuning parameter λ controls the shrinkage

- However, unlike the lasso, ridge regression does not shrink the parameters all the way to zero
- Ridge coefficients have closed form solution (even when $n < p$)

$$\hat{\boldsymbol{\beta}}_{\text{ridge}}(\lambda) = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Fitting a ridge regression

Ridge regression with the `glmnet()` function by changing one of its default options to `alpha=0`

```
ridge.fit = glmnet(x, y, alpha = 0)
ridge.cv = cv.glmnet(x, y, alpha = 0)
ridge.cv$lambda.min
```

```
## [1] 4.516003
```

Comments

- In general, the optimal λ for ridge is very different than the optimal λ for the lasso
- The scale of $\sum \beta_j^2$ and $\sum |\beta_j|$ is very different
- α is known as the elastic-net mixing parameter

Elastic-net: best of both worlds?

- Minimising the (penalised) log-loglikelihood in linear regression with iid errors is equivalent to minimising the (penalised) RSS
- More generally, the elastic-net minimises the following objective function over β_0 and $\boldsymbol{\beta}$ over a grid of values of λ covering the entire range

$$\frac{1}{n}\sum_{i=1}^{N} w_i l(y_i, \beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}_i) + \lambda \left[(1-\alpha) \sum_{j=1}^{p} \beta_j^2 + \alpha \sum_{j=1}^{p} |\beta_j| \right]$$

- This is general enough to include logistic regression, generalised linear models and Cox regression, where $l(y_i)$ is the log-likelihood of observation i and w_i an optional observation weight

A3. Marginality principle

Generalised linear models

New concepts

- Logistic regression models
- Marginality principle for interactions
- **Selected R functions**:
  - `glm()` and `glmnet()`
  - `AIC(...,k=log(n))`
  - `bestglm()`
  - `hierNet()` incl `hierNet.logistic()`

Rock-wallaby data

- Rock-wallaby colony in the Warrambungles National Park (NSW)
- Sample size $n = 200$ sites
- Zero/one responses y_i's: presence or absence of scats
- Five main effects and three special interaction terms:
  - **Main effects**: x_2, x_3, x_4, x_5, x_6
  - **Interactions**: $x_7 = x_2 \times x_4, x_8 = x_2 \times x_5, x_9 = x_4 \times x_5$

```
data("wallabies", package = "mplot")
names(wallabies)
```

```
## [1] "rw"       "edible"   "inedible" "canopy"   "distance"
## [6] "shelter"  "lat"      "long"
```

```
wdat = data.frame(subset(wallabies, select = -c(lat, long)),
                  EaD = wallabies$edible * wallabies$distance,
                  EaS = wallabies$edible * wallabies$shelter,
                  DaS = wallabies$distance * wallabies$shelter)
```

Source: Tuft, Crowther, Connell, Müller, and McArthur (2011).

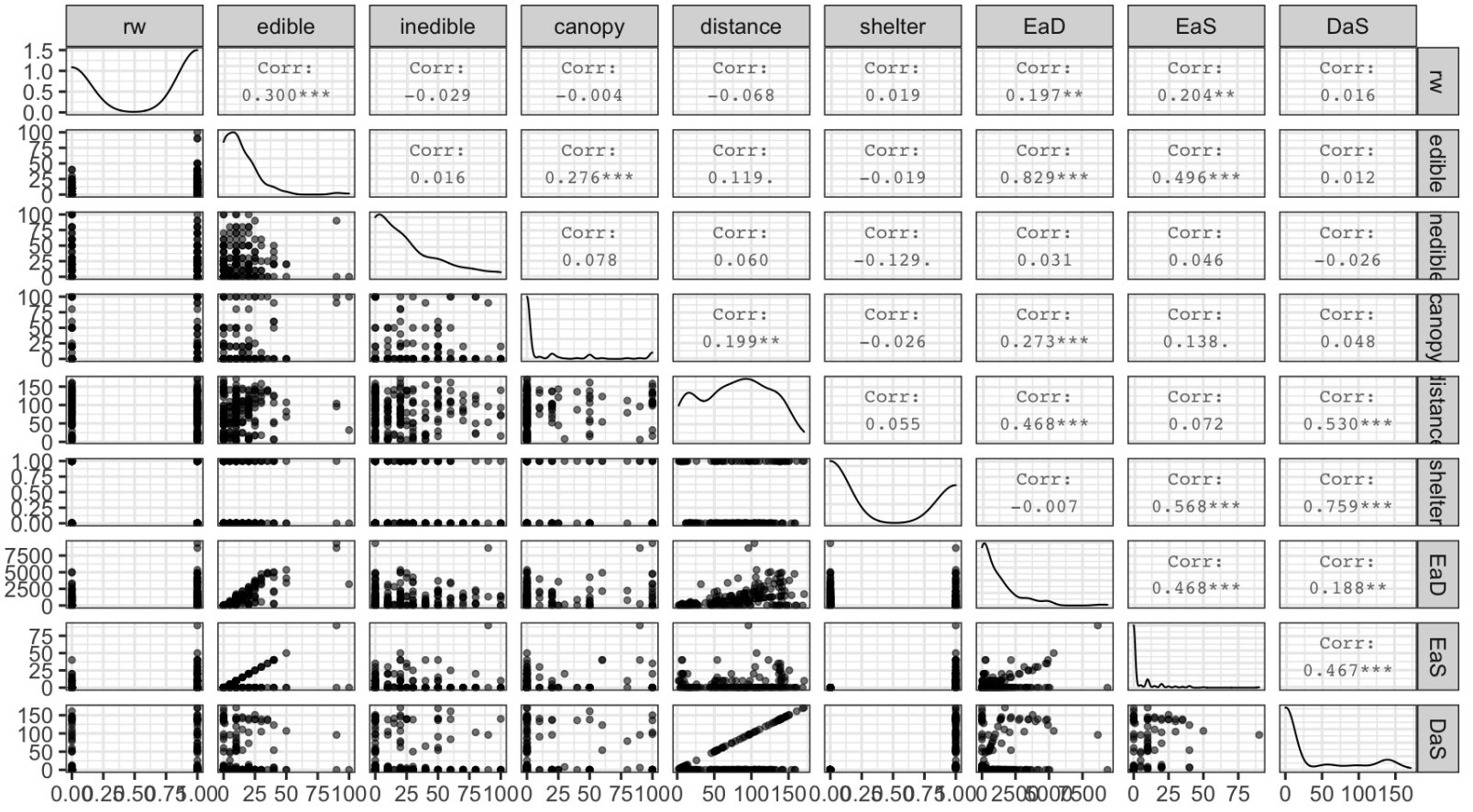
Rock-wallaby: Correlation matrix

```
round(cor(wdat), 1)
```

```
##              rw edible inedible canopy distance shelter EaD EaS DaS
## rw          1.0    0.3      0.0    0.0     -0.1     0.0 0.2 0.2 0.0
## edible      0.3    1.0      0.0    0.3      0.1     0.0 0.8 0.5 0.0
## inedible    0.0    0.0      1.0    0.1      0.1    -0.1 0.0 0.0 0.0
## canopy      0.0    0.3      0.1    1.0      0.2     0.0 0.3 0.1 0.0
## distance   -0.1    0.1      0.1    0.2      1.0     0.1 0.5 0.1 0.5
## shelter     0.0    0.0     -0.1    0.0      0.1     1.0 0.0 0.6 0.8
## EaD         0.2    0.8      0.0    0.3      0.5     0.0 1.0 0.5 0.2
## EaS         0.2    0.5      0.0    0.1      0.1     0.6 0.5 1.0 0.5
## DaS         0.0    0.0      0.0    0.0      0.5     0.8 0.2 0.5 1.0
```

Rock-wallaby: Visualised

```
GGally::ggpairs(wdat, mapping = aes(alpha = 0.2)) + theme_bw(base_size =
```



Logistic regression models

- **Logistic regression model** is of the form

$$\mathrm{P}(Y_i = 1|\mathbf{X}_\alpha) = h(\mathbf{x}_{\alpha i}^\top \beta_\alpha),\ i = 1, \ldots, n,$$

  where $h(\cdot)$ is the logistic function, $h(\mu) = \dfrac{1}{1 + e^{-\mu}}$

- In R, using ML, the full model can be fitted through
  - `glm(y ~ ., family = binomial, data = dat)`
- Calculate **BIC** value through either,
  - `-2 * logLik(glm(...)) + log(n) * p`
  - `AIC(glm(...), k = log(n))`
- Calculate **AIC** value through either,
  - `-2 * logLik(glm(...)) + 2 * p`
  - `AIC(glm(...))`

Rock-wallaby: Possible models

- Here, $p_{\alpha_f} = 9$, intercept part of all models, thus,
- all possible submodels, i.e. $M = \#\mathcal{A} = 2^{p_{\alpha_f}-1} = 2^8 = 256$

Marginality principle

> In general, **never remove the main effect** if a term involving its interaction is included in the model.

- I.e. include main effects if **interaction terms** are modeled
- This reduces the number of submodels to $M = 72$

Rock-wallaby: Full model

```
M1 = glm(rw ~ ., family = binomial, data = wdat)
summary(M1)
```

```
##
## Call:
## glm(formula = rw ~ ., family = binomial, data = wdat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2762  -1.0810   0.4916   1.0107   1.6596
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.1785976  0.5547714   0.322  0.74751
## edible       0.1244071  0.0435224   2.858  0.00426 **
## inedible    -0.0035853  0.0060614  -0.591  0.55419
## canopy      -0.0017489  0.0056838  -0.308  0.75831
## distance    -0.0073732  0.0068719  -1.073  0.28329
## shelter     -1.1439199  0.7052026  -1.622  0.10478
## EaD         -0.0006349  0.0004034  -1.574  0.11546
## EaS          0.0313602  0.0371986   0.843  0.39920
## DaS          0.0118275  0.0076204   1.552  0.12064
## ---
```

The `bestglm` package

- The `bestglm` packages performs complete enumeration for generalised linear models and uses the `leaps` package for linear models.
- The best fit may be found using the information criterion IC: **AIC**, **BIC**.
- It is also able to perform model selection through **cross-validation** with `IC="CV"` (or the special case `IC="LOOCV"`).

See the vignette for further information.

Annoyingly, the vignette doesn't seem to be accessible in the usual way. You can try to access it locally using:

```
path = system.file("bestglm.pdf", package = "bestglm")
system(paste0('open "', path, '"'))
```

Rock-wallaby: Best AIC model

- If the marginality principle is not observed then we search over all $2^8 = 256$ models.

```
library(bestglm)
```

```
## Loading required package: leaps
```

```
X = subset(wdat, select = -rw)
y = wdat$rw
Xy = as.data.frame(cbind(X, y))
bestglm(Xy, IC = "AIC", family = binomial)
```

```
## Morgan-Tatar search since family is non-gaussian.

## AIC
## BICq equivalent for q in (0.419147940650539, 0.842477608830578)
## Best Model:
##                  Estimate   Std. Error   z value     Pr(>|z|)
## (Intercept) -0.5187681581 0.2294865608 -2.260560 0.0237865121
## edible       0.1309190259 0.0355298982  3.684757 0.0002289213
## EaD         -0.0006527505 0.0002864895 -2.278444 0.0227001134
```

Rock-wallaby: Best BIC model

```
bestglm(Xy, IC = "BIC", family = binomial)
```

```
## Morgan-Tatar search since family is non-gaussian.

## BIC
## BICq equivalent for q in (0.419147940650539, 0.842477608830578)
## Best Model:
##                  Estimate   Std. Error   z value     Pr(>|z|)
## (Intercept) -0.5187681581 0.2294865608 -2.260560 0.0237865121
## edible       0.1309190259 0.0355298982  3.684757 0.0002289213
## EaD         -0.0006527505 0.0002864895 -2.278444 0.0227001134
```

- Both the **AIC** and **BIC** suggest the optimal model is the one with main effect `edible` and the interaction term `edible*distance`

How do we enforce marginality?

- Hardcode
- **Hierarchical regularisation**, e.g. with `hierNet()`
  - Let's reframe the problem: consider all main effects and any of the $\binom{5}{2}$ interaction terms (i.e. no quadratic effects `diagonal=FALSE`)
  - Either weak hierarchy (`strong=FALSE`; $x_1 + x_1x_2$, $x_2 + x_1x_2$ and $x_1 + x_2 + x_1x_2$ allowed, i.e. at least one of the main effects is present) or strong hierarchy (`strong=TRUE`; only $x_1 + x_2 + x_1x_2$ allowed)

```
library(hierNet)
Xm = X[, 1:5]  # main effects only
Xm=scale(Xm, center = TRUE, scale = TRUE)
#no quadratic terms and strong marginality
fit = hierNet.logistic.path(Xm, y, diagonal=FALSE,
                            strong=TRUE)
set.seed(9)
fitcv=hierNet.cv(fit,Xm,y,trace=0)
```

Bien, Taylor, and Tibshirani (2013)

Rock-wallaby: Strong hierarchical Lasso

```
fitcv$lamhat.1se
```

```
## [1] 23.15706
```

```
fit.1se = hierNet.logistic(Xm, y, lam=fitcv$lamhat.1se,
                           diagonal = FALSE, strong = TRUE)
```

```
print(fit.1se)
```

```
## Call:
## hierNet.logistic(x = Xm, y = y, lam = fitcv$lamhat.1se, diagonal = FALSE,
##     strong = TRUE)
##
## Non-zero coefficients:
##   (No interactions in this model)
##
##   Main effect
## 1       0.137
```

Rock-wallaby: Weak hierarchical Lasso model

```
fit.5 = hierNet.logistic(Xm, y, lam = 5,
                         diagonal = FALSE, strong = FALSE)
```

```
print(fit.5)
```

```
## Call:
## hierNet.logistic(x = Xm, y = y, lam = 5, diagonal = FALSE, strong = FALSE)
##
## Non-zero coefficients:
##   (Rows are predictors with nonzero main effects)
##   (1st column is main effect)
##   (Next columns are nonzero interactions of row predictor)
##   (Last column indicates whether hierarchy constraint is tight.)
##
##   Main effect 2       3       4 5      Tight?
## 1 0.7821      0       0       0 0
## 2 -0.0404     0       -0.0511 0 0      *
## 3 -0.0618     -0.0511 0       0 0
## 4 -0.1605     0       0       0 0.0802
```

References

Akaike, H. (1973). "Information theory and and extension of the maximum likelihood principle". In: *Second International Symposium on Information Theory*. Ed. by B. Petrov and F. Csaki. Budapest: Academiai Kiado, pp. 267-281.

Bien, J, J. Taylor, and R. Tibshirani (2013). "A LASSO for hierarchical interactions". In: *Ann. Statist.* 41.3, pp. 1111-1141. ISSN: 0090-5364. DOI: 10.1214/13-AOS1096. URL: https://doi.org/10.1214/13-AOS1096.

Chen, J. and Z. Chen (2008). "Extended Bayesian information criteria for model selection with large model spaces". In: *Biometrika* 95.3, pp. 759-771. DOI: 10.1093/biomet/asn034.

Hofmann, M, C. Gatu, E. Kontoghiorghes, et al. (2020). "lmSubsets: Exact Variable-Subset Selection in Linear Regression for R". In: *Journal of Statistical Software* 93.3, pp. 1-21. ISSN: 1548-7660. DOI: 10.18637/jss.v093.i03.

James, G, D. Witten, T. Hastie, et al. (2013). *An introduction to statistical learning with applications in R*. New York, NY: Springer. ISBN: 9781461471370. URL: http://www-bcf.usc.edu/~gareth/ISL/.

Johnson, R. W. (1996). "Fitting Percentage of Body Fat to Simple Body Measurements". In: *Journal of Statistics Education* 4.1. URL: http://www.amstat.org/publications/jse/v4n1/datasets.johnson

Lumley, T. and A. Miller (2009). *leaps: regression subset selection*. R package version 2.9. URL: http://CRAN.R-project.org/package=leaps.

Schwarz, G. (1978). "Estimating the Dimension of a Model". In: *The Annals of Statistics* 6.2, pp. 461-464. DOI: 10.1214/aos/1176344136.

Shao, J. (1993). "Linear Model Selection by Cross-Validation". In: *Journal of the American Statistical Association* 88.422, pp. 486-494. DOI: 10.2307/2290328.

Tuft, K. D, M. S. Crowther, K. Connell, et al. (2011). "Predation risk and competitive interactions affect foraging of an endangered refuge-dependent herbivore". In: *Animal Conservation* 14.4, pp. 447-457. DOI: 10.1111/j.1469-1795.2011.00446.x.

Wiegand, R. E. (2010). "Performance of using multiple stepwise algorithms for variable selection". In: *Statistics in Medicine* 29.15, pp. 1647-1659. DOI: 10.1002/sim.3943.

Wilkinson, L. (1998). *SYSTAT 8 statistics manual*. SPSS Inc.

```
devtools::session_info(include_base = FALSE)
```

```
## ─ Session info ──────────────────────────────────────────────────────────────
##  setting  value
##  version  R version 4.0.0 (2020-04-24)
##  os       macOS Catalina 10.15.5
##  system   x86_64, darwin17.0
##  ui       X11
##  language (EN)
##  collate  en_AU.UTF-8
##  ctype    en_AU.UTF-8
##  tz       Australia/Sydney
##  date     2020-07-10
##
## ─ Packages ──────────────────────────────────────────────────────────────────
##  package        * version    date       lib source
##  assertthat       0.2.1      2019-03-21 [1] CRAN (R 4.0.0)
##  backports        1.1.8      2020-06-17 [1] CRAN (R 4.0.0)
##  bestglm        * 0.37.3     2020-03-13 [1] CRAN (R 4.0.0)
##  bibtex           0.4.2.2    2020-01-02 [1] CRAN (R 4.0.0)
##  blob             1.2.1      2020-01-20 [1] CRAN (R 4.0.0)
##  broom            0.5.6      2020-04-20 [1] CRAN (R 4.0.0)
##  callr            3.4.3      2020-03-28 [1] CRAN (R 4.0.0)
##  cellranger       1.1.0      2016-07-27 [1] CRAN (R 4.0.0)
##  cli              2.0.2      2020-02-28 [1] CRAN (R 4.0.0)
##  codetools        0.2-16     2018-12-24 [1] CRAN (R 4.0.0)
##  colorspace       1.4-1      2019-03-18 [1] CRAN (R 4.0.0)
##  crayon           1.3.4      2017-09-16 [1] CRAN (R 4.0.0)
##  DBI              1.1.0      2019-12-15 [1] CRAN (R 4.0.0)
##  dbplyr           1.4.4      2020-05-27 [1] CRAN (R 4.0.0)
##  desc             1.2.0      2018-05-01 [1] CRAN (R 4.0.0)
##  devtools         2.3.0      2020-04-10 [1] CRAN (R 4.0.0)
##  digest           0.6.25     2020-02-23 [1] CRAN (R 4.0.0)
##  doRNG            1.8.2      2020-01-27 [1] CRAN (R 4.0.0)
##  dplyr          * 1.0.0      2020-06-01 [1] Github (tidyverse/dplyr@5e3f3ec)
##  ellipsis         0.3.1      2020-05-15 [1] CRAN (R 4.0.0)
##  evaluate         0.14       2019-05-28 [1] CRAN (R 4.0.0)
##  fansi            0.4.1      2020-01-08 [1] CRAN (R 4.0.0)
##  fastmap          1.0.1      2019-10-08 [1] CRAN (R 4.0.0)
##  forcats        * 0.5.0      2020-03-01 [1] CRAN (R 4.0.0)
##  foreach          1.5.0      2020-03-30 [1] CRAN (R 4.0.0)
##  fs               1.4.1      2020-04-04 [1] CRAN (R 4.0.0)
##  generics         0.0.2      2018-11-29 [1] CRAN (R 4.0.0)
##  GGally         * 2.0.0      2020-06-06 [1] CRAN (R 4.0.0)
##  ggplot2        * 3.3.2      2020-06-19 [1] CRAN (R 4.0.0)
##  glmnet         * 4.0-2      2020-06-16 [1] CRAN (R 4.0.0)
##  glue             1.4.1      2020-05-13 [1] CRAN (R 4.0.0)
##  grpreg           3.3.0      2020-06-10 [1] CRAN (R 4.0.0)
##  gtable           0.3.0      2019-03-25 [1] CRAN (R 4.0.0)
##  haven            2.3.1      2020-06-01 [1] CRAN (R 4.0.0)
##  hierNet        * 1.9        2020-02-05 [1] CRAN (R 4.0.0)
##  highr            0.8        2019-03-20 [1] CRAN (R 4.0.0)
##  hms              0.5.3      2020-01-08 [1] CRAN (R 4.0.0)
##  htmltools        0.5.0      2020-06-16 [1] CRAN (R 4.0.0)
##  httpuv           1.5.4      2020-06-06 [1] CRAN (R 4.0.0)
##  httr             1.4.1      2019-08-05 [1] CRAN (R 4.0.0)
##  iterators        1.0.12     2019-07-26 [1] CRAN (R 4.0.0)
##  jsonlite         1.7.0      2020-06-25 [1] CRAN (R 4.0.0)
##  kableExtra     * 1.1.0.9000 2020-06-01 [1] Github (haozhu233/kableExtra@9
##  knitcitations  * 1.0.10     2019-09-15 [1] CRAN (R 4.0.0)
##  knitr          * 1.29       2020-06-23 [1] CRAN (R 4.0.0)
##  later            1.1.0.1    2020-06-05 [1] CRAN (R 4.0.0)
##  lattice          0.20-41    2020-04-02 [1] CRAN (R 4.0.0)
##  leaps          * 3.1        2020-01-16 [1] CRAN (R 4.0.0)
##  lifecycle        0.2.0      2020-03-06 [1] CRAN (R 4.0.0)
##  lmSubsets      * 0.5-1      2020-05-23 [1] CRAN (R 4.0.0)
##  lubridate        1.7.9      2020-06-08 [1] CRAN (R 4.0.0)
##  magrittr         1.5        2014-11-22 [1] CRAN (R 4.0.0)
##  Matrix         * 1.2-18     2019-11-27 [1] CRAN (R 4.0.0)
##  memoise          1.1.0      2017-04-21 [1] CRAN (R 4.0.0)
##  mime             0.9        2020-02-04 [1] CRAN (R 4.0.0)
##  modelr           0.1.8      2020-05-19 [1] CRAN (R 4.0.0)
##  mplot          * 1.0.4      2020-02-15 [1] CRAN (R 4.0.0)
##  munsell          0.5.0      2018-06-12 [1] CRAN (R 4.0.0)
##  nlme             3.1-148    2020-05-24 [1] CRAN (R 4.0.0)
##  pillar           1.4.4      2020-05-05 [1] CRAN (R 4.0.0)
##  pkgbuild         1.0.8      2020-05-07 [1] CRAN (R 4.0.0)
##  pkgconfig        2.0.3      2019-09-22 [1] CRAN (R 4.0.0)
##  pkgload          1.1.0      2020-05-29 [1] CRAN (R 4.0.0)
##  pls              2.7-2      2019-10-01 [1] CRAN (R 4.0.0)
##  plyr             1.8.6      2020-03-03 [1] CRAN (R 4.0.0)
##  prettyunits      1.1.1      2020-01-24 [1] CRAN (R 4.0.0)
##  processx         3.4.2      2020-02-09 [1] CRAN (R 4.0.0)
##  promises         1.1.1      2020-06-09 [1] CRAN (R 4.0.0)
##  ps               1.3.3      2020-05-08 [1] CRAN (R 4.0.0)
##  purrr          * 0.3.4      2020-04-17 [1] CRAN (R 4.0.0)
##  R6               2.4.1      2019-11-12 [1] CRAN (R 4.0.0)
##  RColorBrewer     1.1-2      2014-12-07 [1] CRAN (R 4.0.0)
##  Rcpp             1.0.4.6    2020-04-09 [1] CRAN (R 4.0.0)
##  readr          * 1.3.1      2018-12-21 [1] CRAN (R 4.0.0)
##  readxl           1.3.1      2019-03-13 [1] CRAN (R 4.0.0)
##  RefManageR       1.2.12     2019-04-03 [1] CRAN (R 4.0.0)
##  remotes          2.1.1      2020-02-15 [1] CRAN (R 4.0.0)
##  reprex           0.3.0      2019-05-16 [1] CRAN (R 4.0.0)
##  reshape          0.8.8      2018-10-23 [1] CRAN (R 4.0.0)
##  rlang            0.4.6      2020-05-02 [1] CRAN (R 4.0.0)
##  rmarkdown        2.3        2020-06-18 [1] CRAN (R 4.0.0)
##  rngtools         1.5        2020-01-23 [1] CRAN (R 4.0.0)
##  rprojroot        1.3-2      2018-01-03 [1] CRAN (R 4.0.0)
##  rstudioapi       0.11       2020-02-07 [1] CRAN (R 4.0.0)
##  rvest            0.3.5      2019-11-08 [1] CRAN (R 4.0.0)
##  scales           1.1.1      2020-05-11 [1] CRAN (R 4.0.0)
##  sessioninfo      1.1.1      2018-11-05 [1] CRAN (R 4.0.0)
##  shape            1.4.4      2018-02-07 [1] CRAN (R 4.0.0)
```