A Minimal Book Example

Yihui Xie

2019-05-12

Contents

Chapter 1

Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

Chapter 2

Compare classification algorithms

```
# load packages
library(mlbench)
library(caret)
# load the dataset
data(PimaIndiansDiabetes)
```

```
dplyr::glimpse(PimaIndiansDiabetes)
```

```
#> Observations: 768
#> Variables: 9
#> $ pregnant <dbl> 6, 1, 8, 1, 0, 5, 3, 10, 2, 8, 4, 10, 10, 1, 5, 7, 0,...
#> $ glucose  <dbl> 148, 85, 183, 89, 137, 116, 78, 115, 197, 125, 110, 1...
#> $ pressure <dbl> 72, 66, 64, 66, 40, 74, 50, 0, 70, 96, 92, 74, 80, 60...
#> $ triceps  <dbl> 35, 29, 0, 23, 35, 0, 32, 0, 45, 0, 0, 0, 0, 23, 19, ...
#> $ insulin  <dbl> 0, 0, 0, 94, 168, 0, 88, 0, 543, 0, 0, 0, 0, 846, 175...
#> $ mass     <dbl> 33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31.0, 35.3, 30.5,...
#> $ pedigree <dbl> 0.627, 0.351, 0.672, 0.167, 2.288, 0.201, 0.248, 0.13...
#> $ age      <dbl> 50, 31, 32, 21, 33, 30, 26, 29, 53, 54, 30, 34, 57, 5...
#> $ diabetes <fct> pos, neg, pos, neg, pos, neg, pos, neg, pos, pos, neg...
```

```
tibble::as_tibble(PimaIndiansDiabetes)
```

```
#> # A tibble: 768 x 9
#>    pregnant glucose pressure triceps insulin  mass pedigree   age diabetes
#>       <dbl>   <dbl>    <dbl>   <dbl>   <dbl> <dbl>    <dbl> <dbl> <fct>
#>  1        6     148       72      35       0  33.6    0.627    50 pos
#>  2        1      85       66      29       0  26.6    0.351    31 neg
#>  3        8     183       64       0       0  23.3    0.672    32 pos
#>  4        1      89       66      23      94  28.1    0.167    21 neg
#>  5        0     137       40      35     168  43.1    2.29     33 pos
#>  6        5     116       74       0       0  25.6    0.201    30 neg
#>  7        3      78       50      32      88  31      0.248    26 pos
#>  8       10     115        0       0       0  35.3    0.134    29 neg
#>  9        2     197       70      45     543  30.5    0.158    53 pos
#> 10        8     125       96       0       0   0      0.232    54 pos
#> # ... with 758 more rows
```

2.1 Train the models

```
# prepare training scheme
trainControl <- trainControl(method = "repeatedcv", number=10, repeats=3)
```

```
# CART
set.seed(7)
fit.cart <- train(diabetes~., data=PimaIndiansDiabetes,
                  method = "rpart", trControl=trainControl)
```

```
# LDA: Linear Discriminant Analysis
set.seed(7)
fit.lda <- train(diabetes~., data=PimaIndiansDiabetes,
                 method="lda", trControl=trainControl)
```

```
# SVM
set.seed(7)
fit.svm <- train(diabetes~., data=PimaIndiansDiabetes,
                 method="svmRadial", trControl=trainControl)
```

```
# KNN
set.seed(7)
fit.knn <- train(diabetes~., data=PimaIndiansDiabetes,
                 method="knn", trControl=trainControl)
```

```
# Random Forest
set.seed(7)
fit.rf <- train(diabetes~., data=PimaIndiansDiabetes,
                method="rf", trControl=trainControl)
```

```
# collect resamples
results <- resamples(list(CART=fit.cart,
                          LDA=fit.lda,
                          SVM=fit.svm,
                          KNN=fit.knn,
                          RF=fit.rf))
```

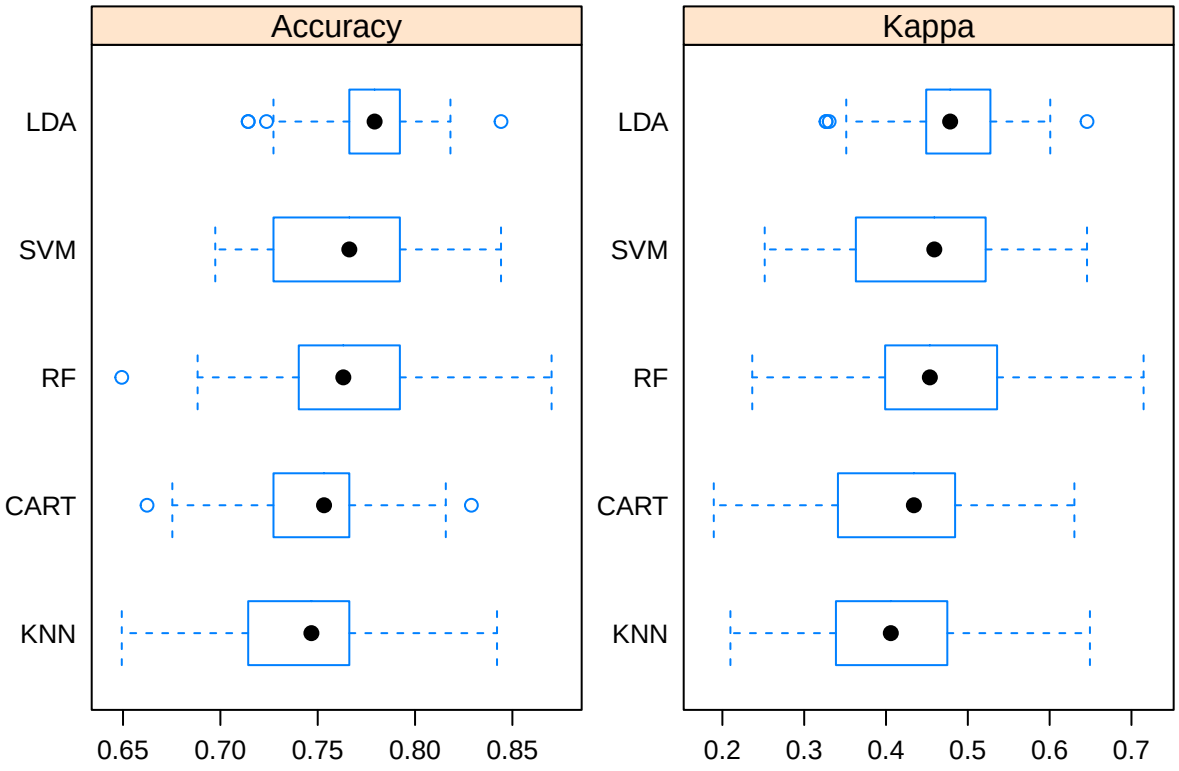
2.2 Compare models

```
# summarize differences between models
summary(results)
```

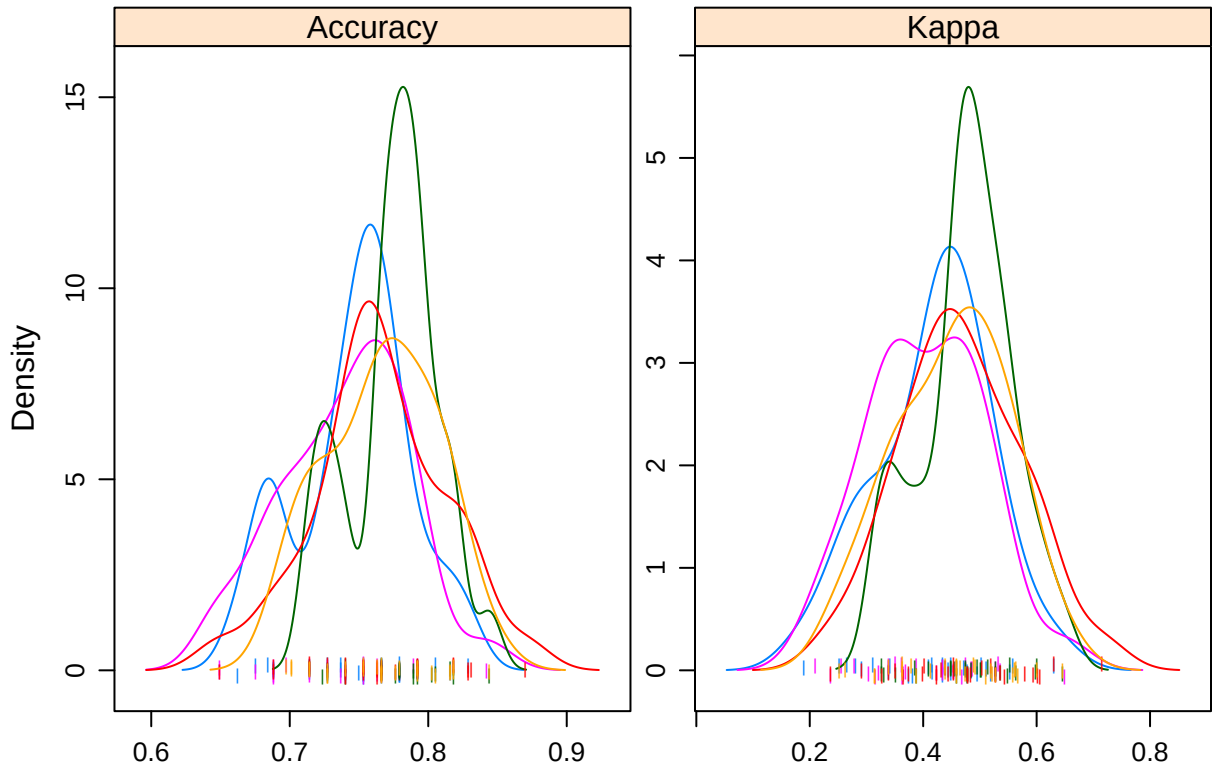
```
#>
#> Call:
#> summary.resamples(object = results)
#>
#> Models: CART, LDA, SVM, KNN, RF
#> Number of resamples: 30
#>
#> Accuracy
#>           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> CART 0.6623377 0.7272727 0.7532468 0.7465596 0.7662338 0.8289474    0
#> LDA  0.7142857 0.7662338 0.7792208 0.7747608 0.7922078 0.8441558    0
```

```
#> SVM  0.6973684 0.7305195 0.7662338 0.7665243 0.7922078 0.8441558    0
#> KNN  0.6493506 0.7142857 0.7467532 0.7400832 0.7662338 0.8421053    0
#> RF   0.6493506 0.7402597 0.7631579 0.7643825 0.7922078 0.8701299    0
#>
#> Kappa
#>           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> CART 0.1894737 0.3495477 0.4340426 0.4184446 0.4827744 0.6302395    0
#> LDA  0.3267091 0.4491256 0.4784008 0.4764965 0.5276074 0.6457055    0
#> SVM  0.2517123 0.3670435 0.4590164 0.4500126 0.5211405 0.6457055    0
#> KNN  0.2098062 0.3388961 0.4058531 0.4047018 0.4733369 0.6492308    0
#> RF   0.2365039 0.3997116 0.4535834 0.4649871 0.5337481 0.7148148    0
```

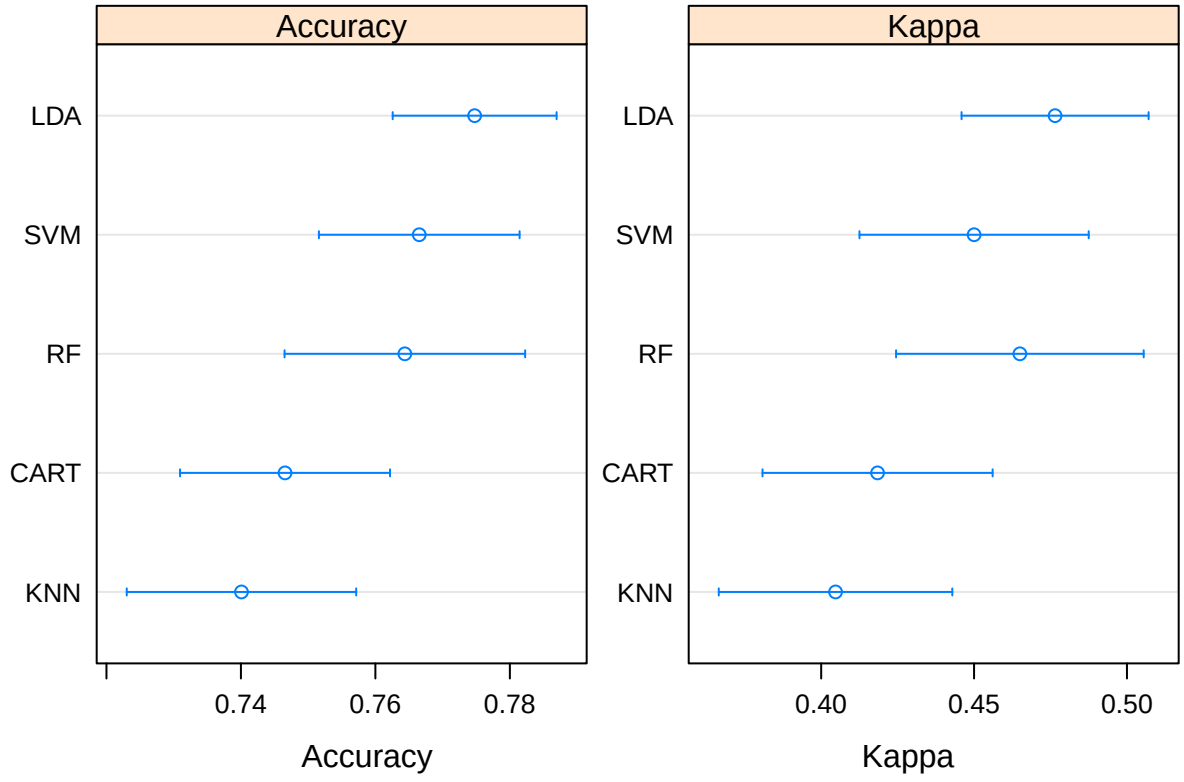
```
# box and whisker plots to compare models
scales <- list(x=list(relation="free"), y=list(relation="free"))
bwplot(results, scales=scales)
```




```
# density plots of accuracy
scales <- list(x=list(relation="free"), y=list(relation="free"))
densityplot(results, scales=scales, pch = "|")
```

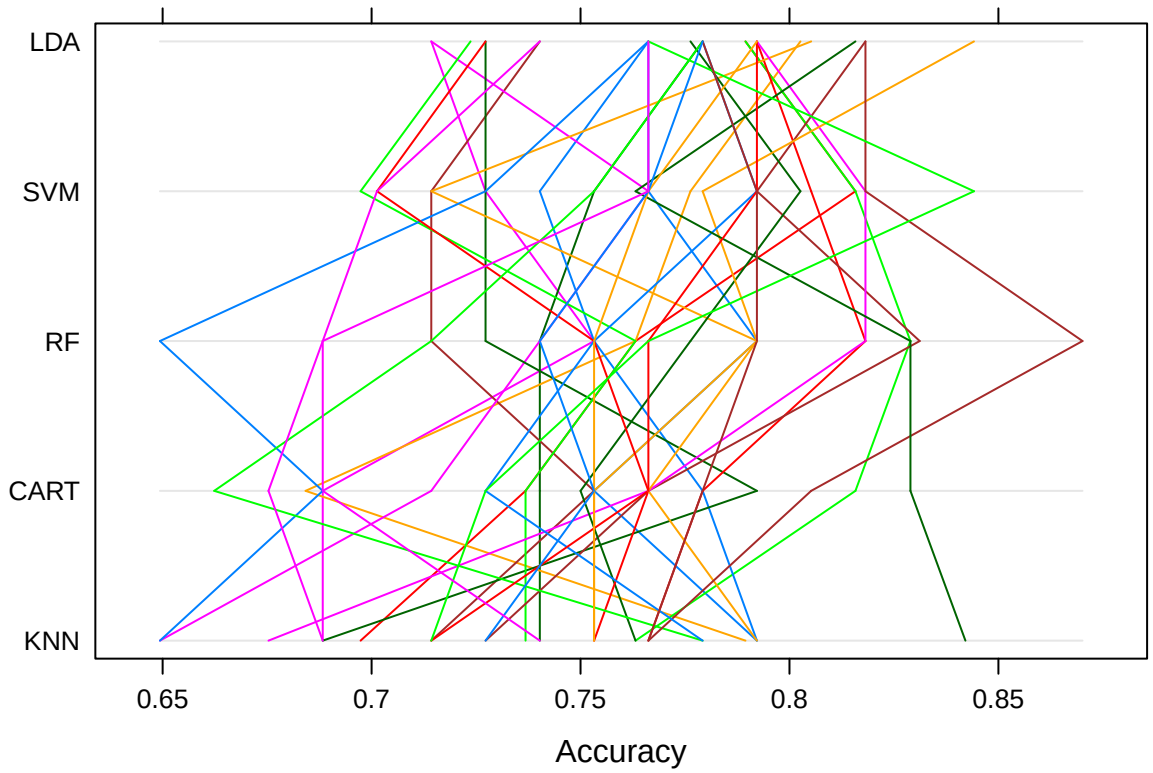



```
# dot plots of accuracy
scales <- list(x=list(relation="free"), y=list(relation="free"))
dotplot(results, scales=scales)
```

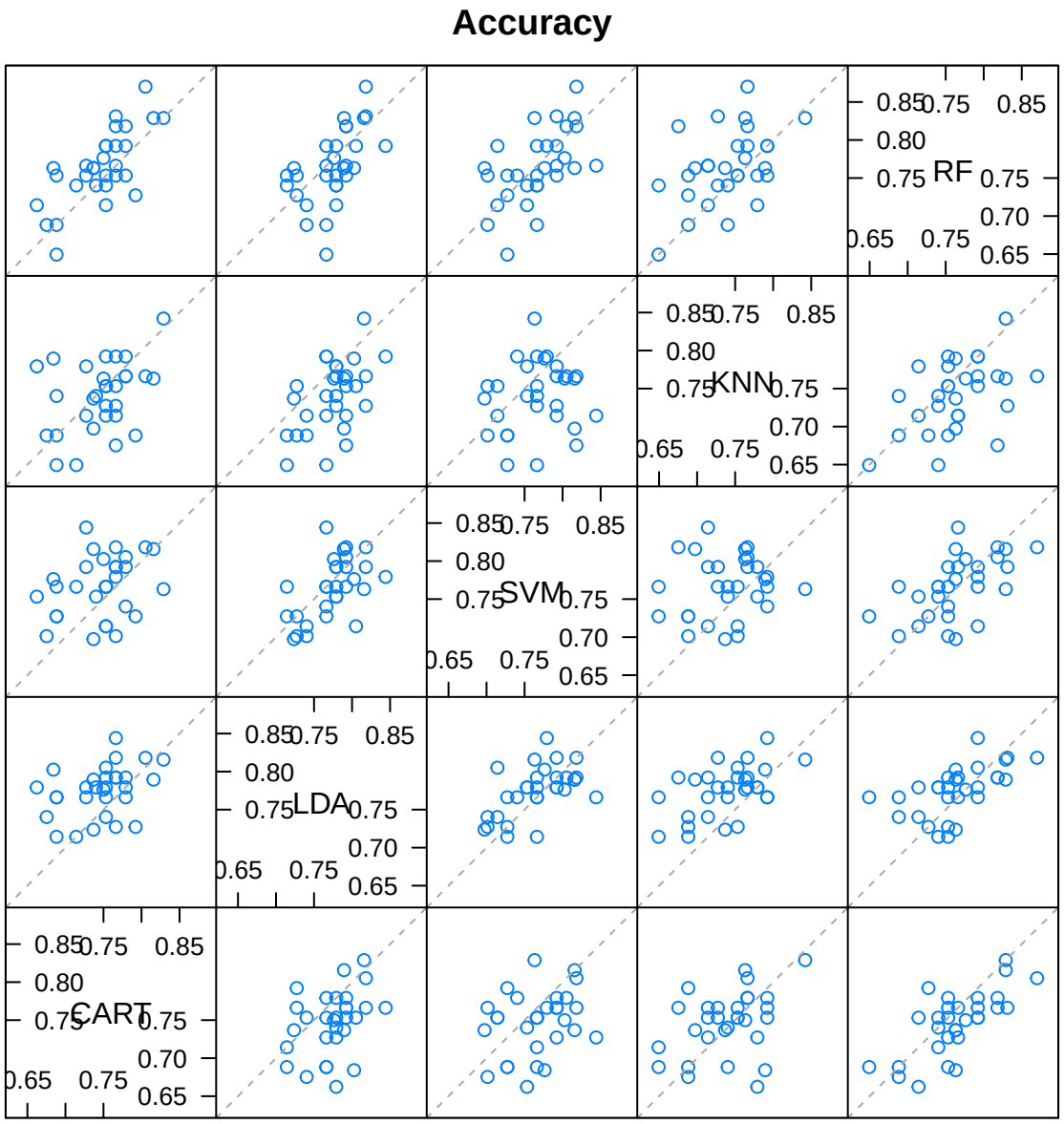


Confidence Level: 0.95

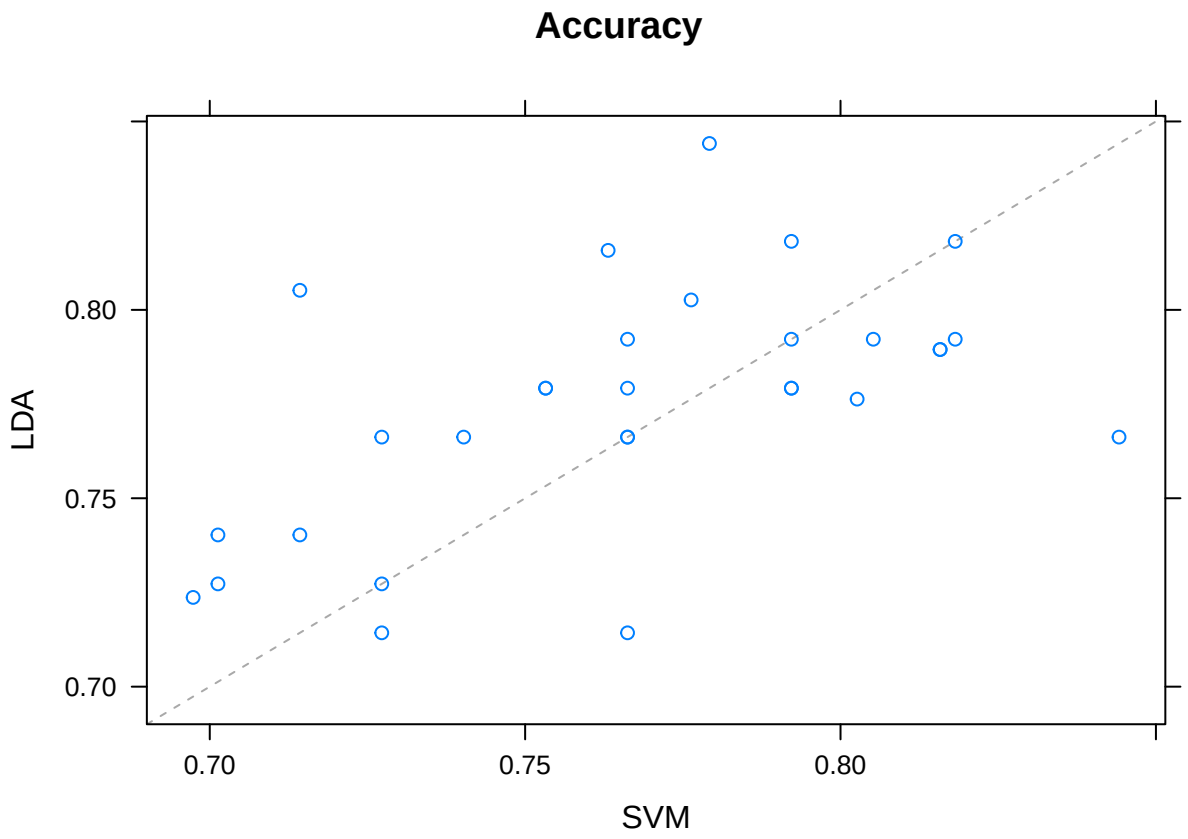
```
# parallel plots to compare models
parallelplot(results)
```




```
# pairwise scatter plots of predictions to compare models
splom(results)
```




```
# xyplot plots to compare models
xyplot(results, models=c("LDA", "SVM"))
```




```
# difference in model predictions
diffs <- diff(results)
# summarize p-values for pairwise comparisons
summary(diffs)
```

```
#>
#> Call:
#> summary.diff.resamples(object = diffs)
#>
#> p-value adjustment: bonferroni
#> Upper diagonal: estimates of the difference
#> Lower diagonal: p-value for H0: difference = 0
#>
#> Accuracy
#>      CART      LDA       SVM       KNN       RF
#> CART           -0.028201 -0.019965  0.006476 -0.017823
#> LDA  0.0109883            0.008237  0.034678  0.010378
#> SVM  0.3270843 1.0000000            0.026441  0.002142
#> KNN  1.0000000 0.0004698 0.1545886           -0.024299
#> RF   0.0770235 1.0000000 1.0000000 0.1030800
#>
#> Kappa
#>      CART     LDA      SVM      KNN      RF
#> CART          -0.05805 -0.03157  0.01374 -0.04654
#> LDA  0.025797           0.02648  0.07179  0.01151
#> SVM  1.000000 0.986395           0.04531 -0.01497
#> KNN  1.000000 0.001399 0.637573          -0.06029
#> RF   0.034983 1.000000 1.000000 0.047566
```

Chapter 3

Temperature modeling using nested dataframes

3.1 Prepare the data

http://ijlyttle.github.io/isugg_purrr/presentation.html#(1)

3.1.1 Packages to run this presentation

```
library("readr")
library("tibble")
library("dplyr")
library("tidyr")
library("stringr")
library("ggplot2")
library("purrr")
library("broom")
```

3.1.2 Motivation

As you know, purrr is a recent package from Hadley Wickham, focused on lists and functional programming, like dplyr is focused on data-frames.

I figure a good way to learn a new package is to try to solve a problem, so we have a dataset:

- you can view or download
- you can download the source of this presentation
- these are three temperatures recorded simultaneously in a piece of electronics
- it will be very valuable to be able to characterize the transient temperature for each sensor
- we want to apply the same set of models across all three sensors
- it will be easier to show using pictures

3.1.3 Let's get the data into shape

Using the readr package

```
temperature_wide <-
  read_csv(file.path(data_raw_dir, "temperature.csv")) %>%
  print()
```

```
# A tibble: 327 x 4
   instant             temperature_a temperature_b temperature_c
   <dttm>                      <dbl>         <dbl>         <dbl>
 1 2015-11-13 06:10:19          116.          91.7          84.2
 2 2015-11-13 06:10:23          116.          91.7          84.2
 3 2015-11-13 06:10:27          116.          91.6          84.2
 4 2015-11-13 06:10:31          116.          91.7          84.2
 5 2015-11-13 06:10:36          116.          91.7          84.2
 6 2015-11-13 06:10:41          116.          91.6          84.2
 7 2015-11-13 06:10:46          116.          91.5          84.2
 8 2015-11-13 06:10:51          116.          91.5          84.2
 9 2015-11-13 06:10:56          116.          91.5          84.2
10 2015-11-13 06:11:01          115.          91.5          84.2
# ... with 317 more rows
```

3.1.4 Is `temperature_wide` "tidy"?

```
# A tibble: 327 x 4
   instant             temperature_a temperature_b temperature_c
   <dttm>                      <dbl>         <dbl>         <dbl>
 1 2015-11-13 06:10:19          116.          91.7          84.2
 2 2015-11-13 06:10:23          116.          91.7          84.2
 3 2015-11-13 06:10:27          116.          91.6          84.2
 4 2015-11-13 06:10:31          116.          91.7          84.2
 5 2015-11-13 06:10:36          116.          91.7          84.2
 6 2015-11-13 06:10:41          116.          91.6          84.2
 7 2015-11-13 06:10:46          116.          91.5          84.2
 8 2015-11-13 06:10:51          116.          91.5          84.2
 9 2015-11-13 06:10:56          116.          91.5          84.2
10 2015-11-13 06:11:01          115.          91.5          84.2
# ... with 317 more rows
```

Why or why not?

3.1.5 Tidy data

1. Each column is a variable
2. Each row is an observation
3. Each cell is a value

(http://www.jstatsoft.org/v59/i10/paper)

My personal observation is that "tidy" can depend on the context, on what you want to do with the data.

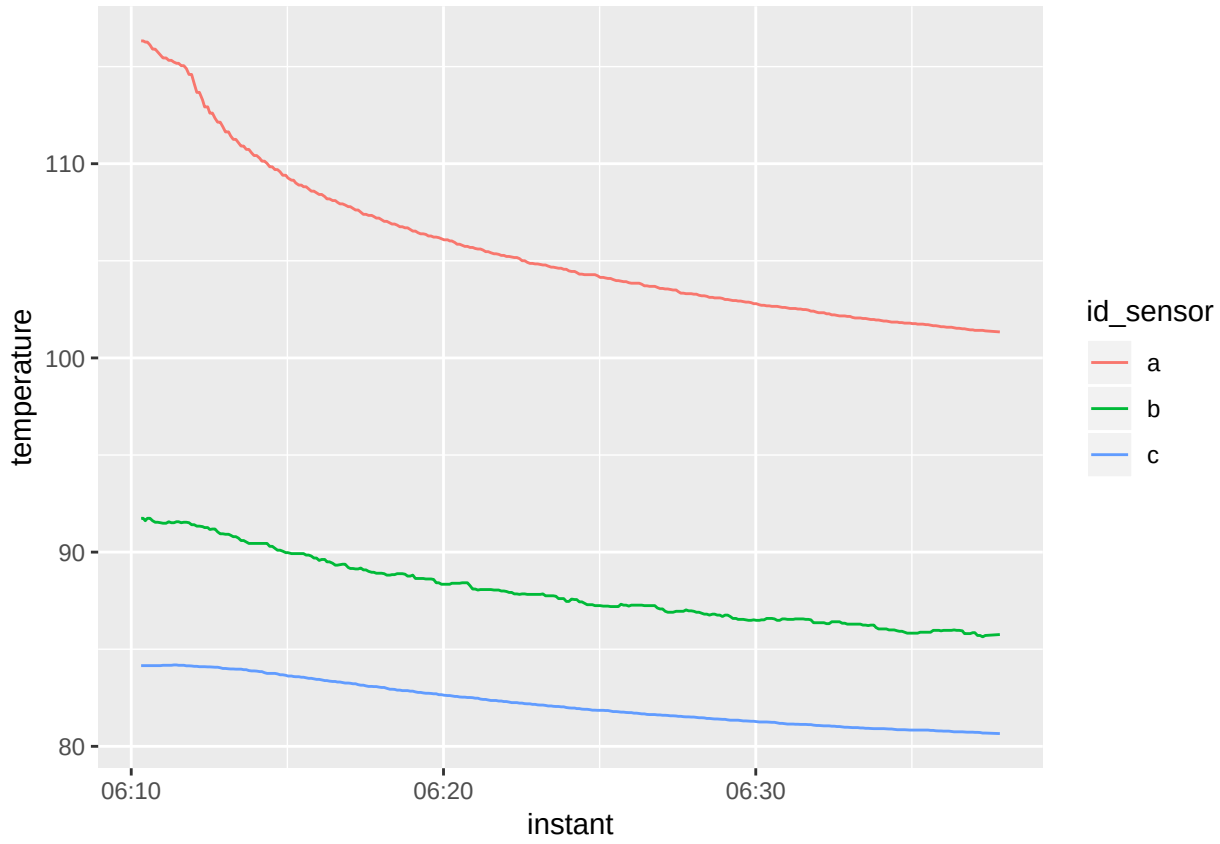
3.1.6 Let's get this into a tidy form

```
temperature_tall <-
  temperature_wide %>%
  gather(key = "id_sensor", value = "temperature", starts_with("temp")) %>%
  mutate(id_sensor = str_replace(id_sensor, "temperature_", "")) %>%
  print()
```

```
# A tibble: 981 x 3
   instant             id_sensor temperature
   <dttm>              <chr>           <dbl>
 1 2015-11-13 06:10:19 a                116.
 2 2015-11-13 06:10:23 a                116.
 3 2015-11-13 06:10:27 a                116.
 4 2015-11-13 06:10:31 a                116.
 5 2015-11-13 06:10:36 a                116.
 6 2015-11-13 06:10:41 a                116.
 7 2015-11-13 06:10:46 a                116.
 8 2015-11-13 06:10:51 a                116.
 9 2015-11-13 06:10:56 a                116.
10 2015-11-13 06:11:01 a                115.
# ... with 971 more rows
```

3.1.7 Now, it's easier to visualize

```
temperature_tall %>%
  ggplot(aes(x = instant, y = temperature, color = id_sensor)) +
  geom_line()
```



3.1.8 Calculate delta time (Δt) and delta temperature (ΔT)

delta_time Δt

change in time since event started, s

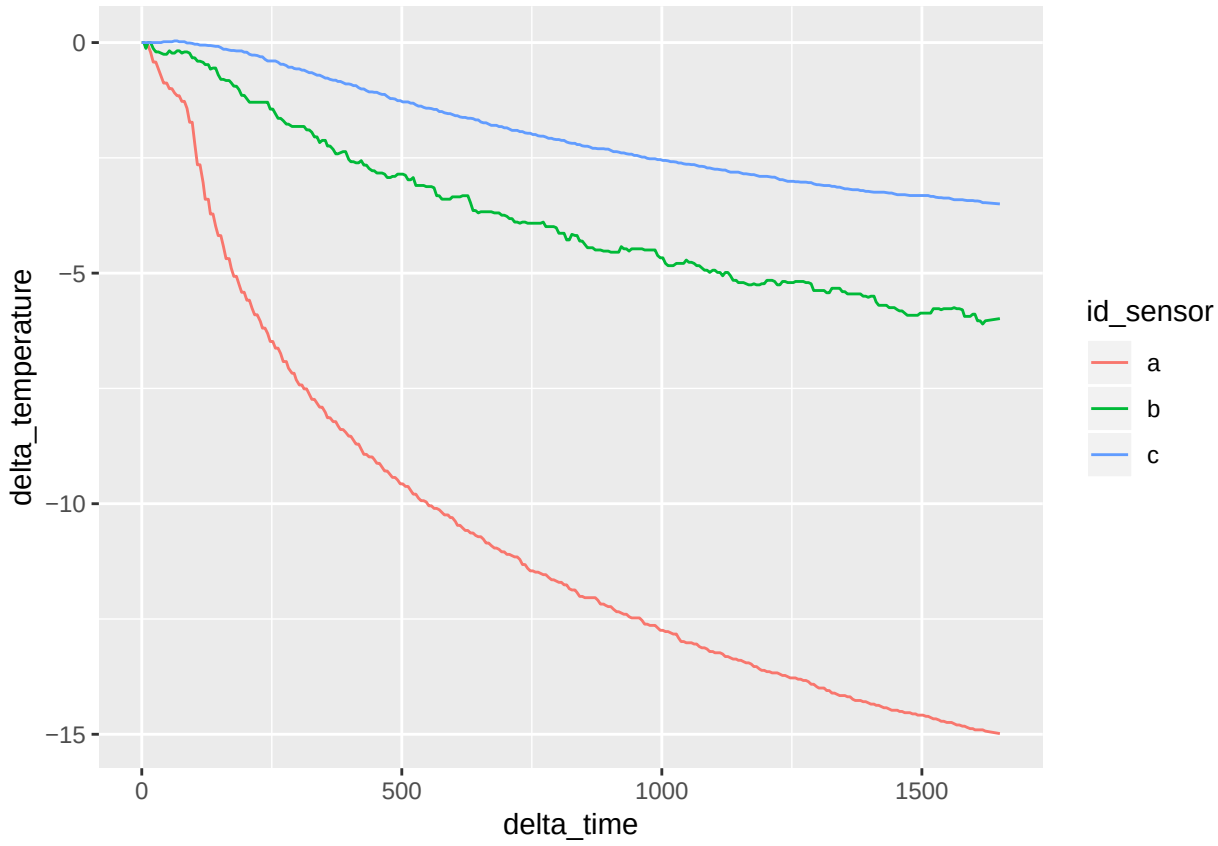
delta_temperature: ΔT

change in temperature since event started, °C

```
delta <-
  temperature_tall %>%
  arrange(id_sensor, instant) %>%
  group_by(id_sensor) %>%
  mutate(
    delta_time = as.numeric(instant) - as.numeric(instant[[1]]),
    delta_temperature = temperature - temperature[[1]]
  ) %>%
  select(id_sensor, delta_time, delta_temperature)
```

3.1.9 Let's have a look

```
# plot delta time vs delta temperature, by sensor
delta %>%
  ggplot(aes(x = delta_time, y = delta_temperature, color = id_sensor)) +
  geom_line()
```



3.2 Define the models

We want to see how three different curve-fits might perform on these three data-sets:

3.2.0.1 Newtonian cooling

$$\Delta T = \Delta T_0 * (1 - e^{-\frac{\delta t}{\tau_0}})$$

3.2.1 Semi-infinite solid

$$\Delta T = \Delta T_0 * erfc(\sqrt{\frac{\tau_0}{\delta t}}))$$

3.2.2 Semi-infinite solid with convection

$$\Delta T = \Delta T_0 * \left[\mathrm{e}\, rfc(\sqrt{\frac{\tau_0}{\delta t}}) - e^{Bi_0 + (\frac{Bi_0}{2})^2 \frac{\delta t}{\tau_0}} * \mathrm{e}\, rfc(\sqrt{\frac{\tau_0}{\delta t}} + \frac{Bi_0}{2} * \sqrt{\frac{\delta t}{\tau_0}} \right]$$

3.2.3 `erf` and `erfc` functions

```
# reference: http://stackoverflow.com/questions/29067916/r-error-function-erfz
# (see Abramowitz and Stegun 29.2.29)
erf <- function(x) 2 * pnorm(x * sqrt(2)) - 1
erfc <- function(x) 2 * pnorm(x * sqrt(2), lower = FALSE)
```

3.2.4 Newton cooling equation

```
newton_cooling <- function(x) {
  nls(
    delta_temperature ~ delta_temperature_0 * (1 - exp(-delta_time/tau_0)),
    start = list(delta_temperature_0 = -10, tau_0 = 50),
    data = x
  )
}
```

3.2.5 Temperature models: simple and convection

```
semi_infinite_simple <- function(x) {
  nls(
    delta_temperature ~ delta_temperature_0 * erfc(sqrt(tau_0 / delta_time)),
    start = list(delta_temperature_0 = -10, tau_0 = 50),
    data = x
  )
}

semi_infinite_convection <- function(x){
  nls(
    delta_temperature ~
      delta_temperature_0 * (
        erfc(sqrt(tau_0 / delta_time)) -
        exp(Bi_0 + (Bi_0/2)^2 * delta_time / tau_0) *
          erfc(sqrt(tau_0 / delta_time) +
        (Bi_0/2) * sqrt(delta_time / tau_0))
      ),
    start = list(delta_temperature_0 = -5, tau_0 = 50, Bi_0 = 1.e6),
    data = x
  )
}
```

3.3 Test modeling on one dataset

3.3.1 Before going into purrr

Before doing anything, we want to show that we can do something with one dataset and one model-function:

```
# only one sensor; it is a test
tmp_data <- delta %>% filter(id_sensor == "a")

tmp_model <- newton_cooling(tmp_data)
```

```
summary(tmp_model)
```

```
Formula: delta_temperature ~ delta_temperature_0 * (1 - exp(-delta_time/tau_0))

Parameters:
                     Estimate Std. Error t value Pr(>|t|)
delta_temperature_0 -15.06085    0.05262  -286.2   <2e-16 ***
tau_0               500.01382    4.83673   103.4   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3267 on 325 degrees of freedom

Number of iterations to convergence: 7
Achieved convergence tolerance: 4.136e-06
```

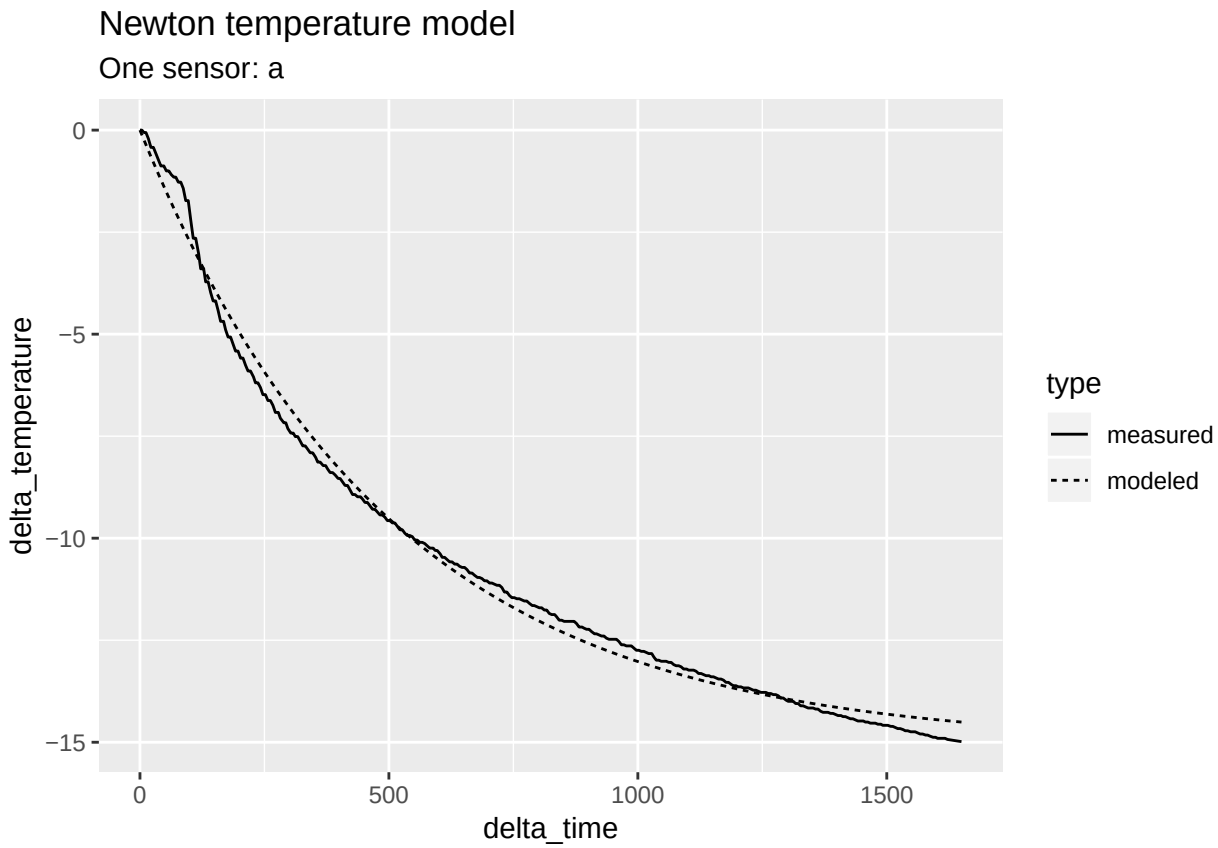
3.3.2 Look at predictions

```
# apply prediction and make it tidy
tmp_pred <-
  tmp_data %>%
  mutate(modeled = predict(tmp_model, data = .)) %>%
  select(id_sensor, delta_time, measured = delta_temperature, modeled) %>%
  gather("type", "delta_temperature", measured:modeled) %>%
  print()
```

```
# A tibble: 654 x 4
# Groups:   id_sensor [1]
   id_sensor delta_time type     delta_temperature
   <chr>          <dbl> <chr>                <dbl>
 1 a                  0 measured             0
 2 a                  4 measured             0
 3 a                  8 measured            -0.06
 4 a                 12 measured            -0.06
 5 a                 17 measured            -0.211
 6 a                 22 measured            -0.423
 7 a                 27 measured            -0.423
 8 a                 32 measured            -0.574
 9 a                 37 measured            -0.726
10 a                 42 measured            -0.878
# ... with 644 more rows
```

3.3.3 Plot Newton model

```
tmp_pred %>%
  ggplot(aes(x = delta_time, y = delta_temperature, linetype = type)) +
  geom_line() +
  labs(title = "Newton temperature model", subtitle = "One sensor: a")
```



3.3.4 "Regular" data-frame (deltas)

```
print(delta)
```

```
# A tibble: 981 x 3
# Groups:   id_sensor [3]
   id_sensor delta_time delta_temperature
   <chr>          <dbl>             <dbl>
 1 a                  0              0
 2 a                  4              0
 3 a                  8             -0.06
 4 a                 12             -0.06
 5 a                 17             -0.211
 6 a                 22             -0.423
 7 a                 27             -0.423
 8 a                 32             -0.574
 9 a                 37             -0.726
10 a                 42             -0.878
# ... with 971 more rows
```

Each column of the dataframe is a vector - in this case, a character vector and two doubles

3.4 Making a nested dataframe

3.4.1 How to make a weird data-frame

Here's where the fun starts - a column of a data-frame can be a list.

- use `tidyr::nest()` to makes a column `data`, which is a list of data-frames
- this seems like a stronger expression of the `dplyr::group_by()` idea

```
# nest delta_time and delta_temperature variables
delta_nested <-
  delta %>%
  nest(-id_sensor) %>%
  print()
```

```
# A tibble: 3 x 2
  id_sensor data
  <chr>     <list>
1 a         <tibble [327 x 2]>
2 b         <tibble [327 x 2]>
3 c         <tibble [327 x 2]>
```

3.4.2 Map dataframes to a modeling function (Newton)

- `map()` is like `lapply()`
- `map()` returns a list-column (it keeps the weirdness)

```
model_nested <-
  delta_nested %>%
  mutate(model = map(data, newton_cooling)) %>%
  print()
```

```
# A tibble: 3 x 3
  id_sensor data               model
  <chr>     <list>             <list>
1 a         <tibble [327 x 2]> <nls>
2 b         <tibble [327 x 2]> <nls>
3 c         <tibble [327 x 2]> <nls>
```

We get an additional list-column `model`.

3.4.3 We can use `map2()` to make the predictions

- `map2()` is like `mapply()`
- designed to map two colunms (`model`, `data`) to a function `predict()`

```
predict_nested <-
  model_nested %>%
  mutate(pred = map2(model, data, predict)) %>%
  print()
```

```
# A tibble: 3 x 4
  id_sensor data               model  pred
  <chr>     <list>             <list> <list>
```

```
1 a         <tibble [327 x 2]> <nls>  <dbl [327]>
2 b         <tibble [327 x 2]> <nls>  <dbl [327]>
3 c         <tibble [327 x 2]> <nls>  <dbl [327]>
```

Another list-column `pred` for the prediction results.

3.4.4 We need to get out of the weirdness

- use `unnest()` to get back to a regular data-frame

```
predict_unnested <-
  predict_nested %>%
  unnest(data, pred) %>%
  print()
```

```
# A tibble: 981 x 4
   id_sensor   pred delta_time delta_temperature
   <chr>      <dbl>      <dbl>             <dbl>
 1 a          0              0             0
 2 a         -0.120          4             0
 3 a         -0.239          8            -0.06
 4 a         -0.357         12            -0.06
 5 a         -0.503         17            -0.211
 6 a         -0.648         22            -0.423
 7 a         -0.792         27            -0.423
 8 a         -0.934         32            -0.574
 9 a         -1.07          37            -0.726
10 a         -1.21          42            -0.878
# ... with 971 more rows
```

3.4.5 We can wrangle the predictions

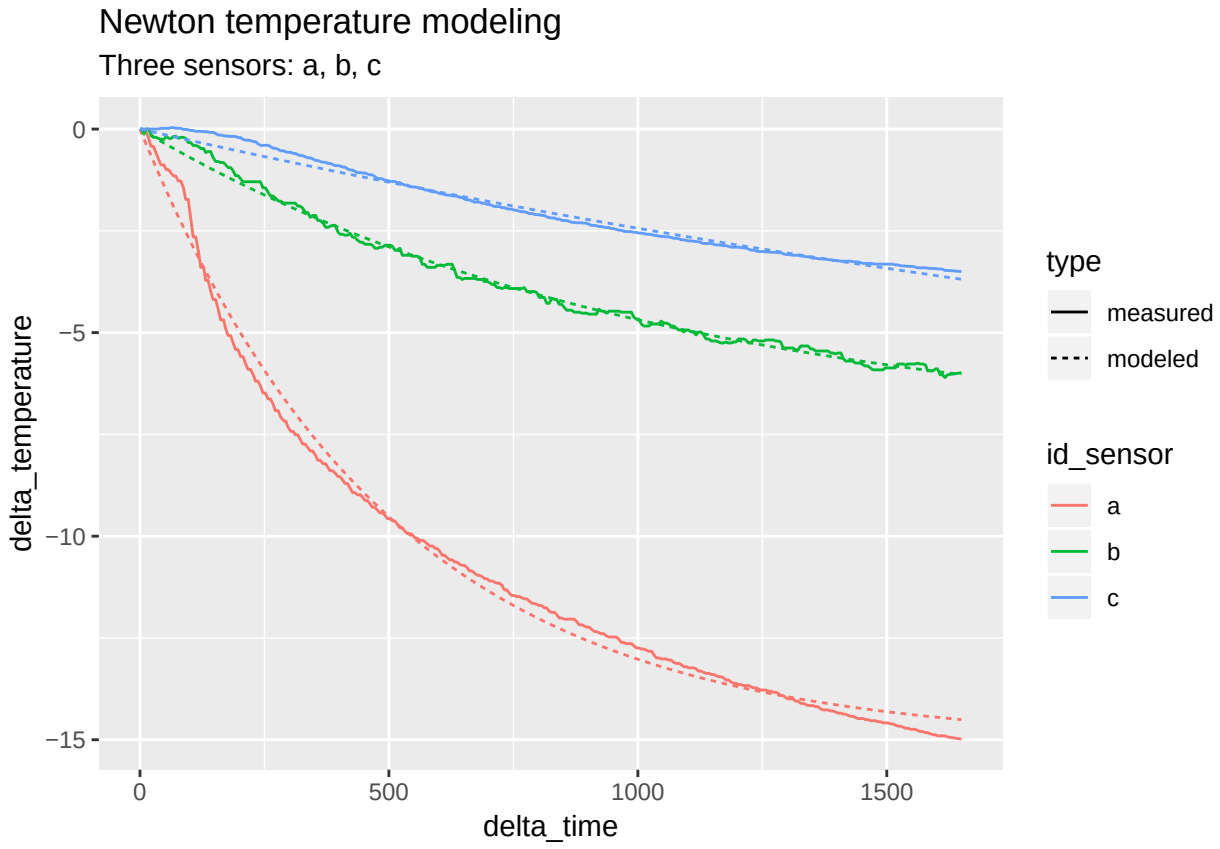
- get into a form that makes it easier to plot

```
predict_tall <-
  predict_unnested %>%
  rename(modeled = pred, measured = delta_temperature) %>%
  gather("type", "delta_temperature", modeled, measured) %>%
  print()
```

```
# A tibble: 1,962 x 4
   id_sensor delta_time type    delta_temperature
   <chr>          <dbl> <chr>               <dbl>
 1 a                  0 modeled             0
 2 a                  4 modeled            -0.120
 3 a                  8 modeled            -0.239
 4 a                 12 modeled            -0.357
 5 a                 17 modeled            -0.503
 6 a                 22 modeled            -0.648
 7 a                 27 modeled            -0.792
 8 a                 32 modeled            -0.934
 9 a                 37 modeled            -1.07
10 a                 42 modeled            -1.21
# ... with 1,952 more rows
```

3.4.6 We can visualize the predictions

```
predict_tall %>%
  ggplot(aes(x = delta_time, y = delta_temperature)) +
  geom_line(aes(color = id_sensor, linetype = type)) +
  labs(title = "Newton temperature modeling",
       subtitle = "Three sensors: a, b, c")
```



3.5 Apply multiple models on a nested structure

3.5.1 Step 1: Selection of models

Make a list of functions to model:

```
list_model <-
  list(
    newton_cooling = newton_cooling,
    semi_infinite_simple = semi_infinite_simple,
    semi_infinite_convection = semi_infinite_convection
  )
```

3.5.2 Step 2: write a function to define the "inner" loop

```
# add additional variable with the model name

fn_model <- function(.model, df) {
  # one parameter for the model in the list, the second for the data
  # safer to avoid non-standard evaluation
  # df %>% mutate(model = map(data, .model))

  df$model <- map(df$data, possibly(.model, NULL))
  df
}
```

- for a given model-function and a given (weird) data-frame, return a modified version of that data-frame with a column `model`, which is the model-function applied to each element of the data-frame's `data` column (which is itself a list of data-frames)
- the purrr functions `safely()` and `possibly()` are **very** interesting. I think they could be useful outside of purrr as a friendlier way to do error-handling.

3.5.3 Step 3: Use `map_df()` to define the "outer" loop

```
# this dataframe will be the second input of fn_model
delta_nested %>%
  print()
```

```
# A tibble: 3 x 2
  id_sensor data
  <chr>     <list>
1 a         <tibble [327 x 2]>
2 b         <tibble [327 x 2]>
3 c         <tibble [327 x 2]>
```

```
# fn_model is receiving two inputs: one from list_model and from delta_nested
model_nested_new <-
  list_model %>%
  map_df(fn_model, delta_nested, .id = "id_model") %>%
  print()
```

```
# A tibble: 9 x 4
  id_model                id_sensor data               model
  <chr>                   <chr>     <list>             <list>
1 newton_cooling          a         <tibble [327 x 2]> <nls>
2 newton_cooling          b         <tibble [327 x 2]> <nls>
3 newton_cooling          c         <tibble [327 x 2]> <nls>
4 semi_infinite_simple    a         <tibble [327 x 2]> <nls>
5 semi_infinite_simple    b         <tibble [327 x 2]> <nls>
6 semi_infinite_simple    c         <tibble [327 x 2]> <nls>
7 semi_infinite_convection a        <tibble [327 x 2]> <NULL>
8 semi_infinite_convection b        <tibble [327 x 2]> <NULL>
9 semi_infinite_convection c        <tibble [327 x 2]> <NULL>
```

- for each element of a list of model-functions, run the inner-loop function, and row-bind the results into a data-frame

- we want to discard the rows where the model failed
- we also want to investigate why they failed, but that's a different talk

3.5.4 Step 4: Use `map()` to identify the null models

```
model_nested_new <-
  list_model %>%
  map_df(fn_model, delta_nested, .id = "id_model") %>%
  mutate(is_null = map(model, is.null)) %>%
  print()
```

```
# A tibble: 9 x 5
  id_model                id_sensor data              model  is_null
  <chr>                   <chr>     <list>            <list> <list>
1 newton_cooling          a         <tibble [327 x 2]> <nls>  <lgl [1]>
2 newton_cooling          b         <tibble [327 x 2]> <nls>  <lgl [1]>
3 newton_cooling          c         <tibble [327 x 2]> <nls>  <lgl [1]>
4 semi_infinite_simple    a         <tibble [327 x 2]> <nls>  <lgl [1]>
5 semi_infinite_simple    b         <tibble [327 x 2]> <nls>  <lgl [1]>
6 semi_infinite_simple    c         <tibble [327 x 2]> <nls>  <lgl [1]>
7 semi_infinite_convection a        <tibble [327 x 2]> <NULL> <lgl [1]>
8 semi_infinite_convection b        <tibble [327 x 2]> <NULL> <lgl [1]>
9 semi_infinite_convection c        <tibble [327 x 2]> <NULL> <lgl [1]>
```

- using `map(model, is.null)` returns a list column
- to use `filter()`, we have to escape the weirdness

3.5.5 Step 5: `map_lgl()` to identify nulls and get out of the weirdness

```
model_nested_new <-
  list_model %>%
  map_df(fn_model, delta_nested, .id = "id_model") %>%
  mutate(is_null = map_lgl(model, is.null)) %>%
  print()
```

```
# A tibble: 9 x 5
  id_model                id_sensor data              model  is_null
  <chr>                   <chr>     <list>            <list> <lgl>
1 newton_cooling          a         <tibble [327 x 2]> <nls>  FALSE
2 newton_cooling          b         <tibble [327 x 2]> <nls>  FALSE
3 newton_cooling          c         <tibble [327 x 2]> <nls>  FALSE
4 semi_infinite_simple    a         <tibble [327 x 2]> <nls>  FALSE
5 semi_infinite_simple    b         <tibble [327 x 2]> <nls>  FALSE
6 semi_infinite_simple    c         <tibble [327 x 2]> <nls>  FALSE
7 semi_infinite_convection a        <tibble [327 x 2]> <NULL> TRUE
8 semi_infinite_convection b        <tibble [327 x 2]> <NULL> TRUE
9 semi_infinite_convection c        <tibble [327 x 2]> <NULL> TRUE
```

- using `map_lgl(model, is.null)` returns a vector column

3.5.6 Step 6: `filter()` nulls and `select()` variables to clean up

```
model_nested_new <-
  list_model %>%
  map_df(fn_model, delta_nested, .id = "id_model") %>%
  mutate(is_null = map_lgl(model, is.null)) %>%
  filter(!is_null) %>%
  select(-is_null) %>%
  print()
```

```
# A tibble: 6 x 4
  id_model             id_sensor data              model
  <chr>                <chr>     <list>            <list>
1 newton_cooling       a         <tibble [327 x 2]> <nls>
2 newton_cooling       b         <tibble [327 x 2]> <nls>
3 newton_cooling       c         <tibble [327 x 2]> <nls>
4 semi_infinite_simple a         <tibble [327 x 2]> <nls>
5 semi_infinite_simple b         <tibble [327 x 2]> <nls>
6 semi_infinite_simple c         <tibble [327 x 2]> <nls>
```

3.5.7 Step 7: Calculate predictions on nested dataframe

```
predict_nested <-
  model_nested_new %>%
  mutate(pred = map2(model, data, predict)) %>%
  print()
```

```
# A tibble: 6 x 5
  id_model             id_sensor data              model  pred
  <chr>                <chr>     <list>            <list> <list>
1 newton_cooling       a         <tibble [327 x 2]> <nls>  <dbl [327]>
2 newton_cooling       b         <tibble [327 x 2]> <nls>  <dbl [327]>
3 newton_cooling       c         <tibble [327 x 2]> <nls>  <dbl [327]>
4 semi_infinite_simple a         <tibble [327 x 2]> <nls>  <dbl [327]>
5 semi_infinite_simple b         <tibble [327 x 2]> <nls>  <dbl [327]>
6 semi_infinite_simple c         <tibble [327 x 2]> <nls>  <dbl [327]>
```

3.5.8 `unnest()`, make it tall and tidy

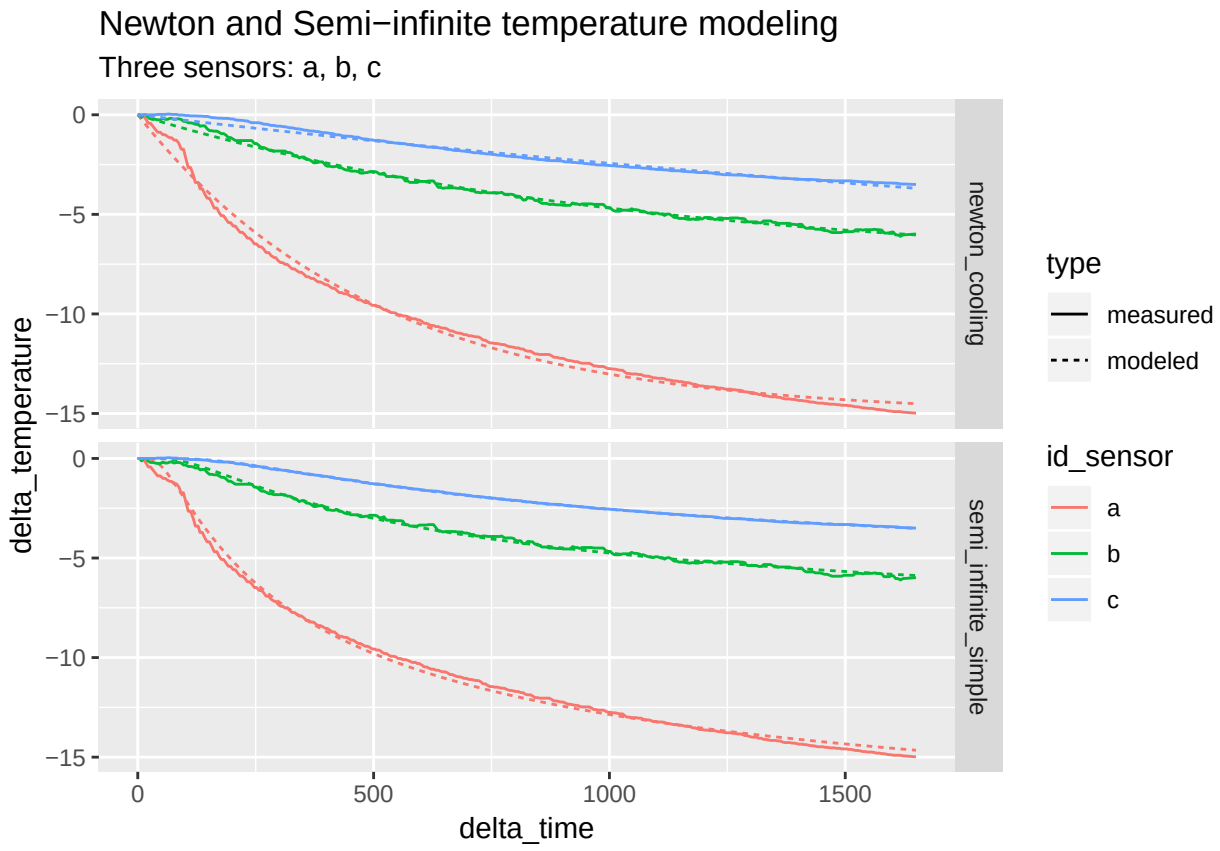
```
predict_tall <-
  predict_nested %>%
  unnest(data, pred) %>%
  rename(modeled = pred, measured = delta_temperature) %>%
  gather("type", "delta_temperature", modeled, measured) %>%
  print()
```

```
# A tibble: 3,924 x 5
   id_model       id_sensor delta_time type    delta_temperature
   <chr>          <chr>          <dbl> <chr>               <dbl>
 1 newton_cooling a                  0 modeled             0
 2 newton_cooling a                  4 modeled            -0.120
 3 newton_cooling a                  8 modeled            -0.239
```

```
 4 newton_cooling a                12 modeled            -0.357
 5 newton_cooling a                17 modeled            -0.503
 6 newton_cooling a                22 modeled            -0.648
 7 newton_cooling a                27 modeled            -0.792
 8 newton_cooling a                32 modeled            -0.934
 9 newton_cooling a                37 modeled            -1.07
10 newton_cooling a                42 modeled            -1.21
# ... with 3,914 more rows
```

3.5.9 Visualize the predictions

```
predict_tall %>%
  ggplot(aes(x = delta_time, y = delta_temperature)) +
  geom_line(aes(color = id_sensor, linetype = type)) +
  facet_grid(id_model ~ .) +
  labs(title = "Newton and Semi-infinite temperature modeling",
       subtitle = "Three sensors: a, b, c")
```



3.5.10 Let's get the residuals

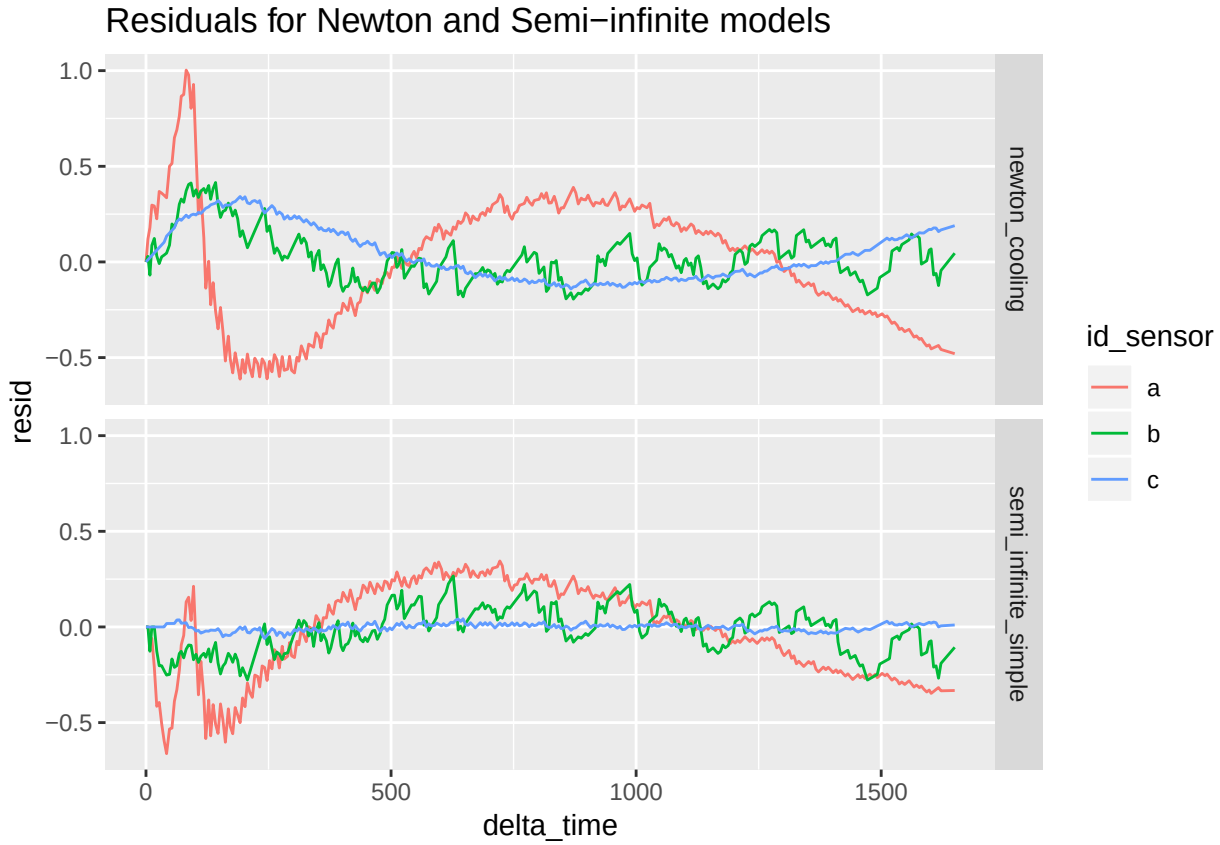
```
resid <-
  model_nested_new %>%
  mutate(resid = map(model, resid)) %>%
```

```
  unnest(data, resid) %>%
  print()
```

```
# A tibble: 1,962 x 5
   id_model       id_sensor resid delta_time delta_temperature
   <chr>          <chr>     <dbl>      <dbl>             <dbl>
 1 newton_cooling a         0              0              0
 2 newton_cooling a         0.120          4              0
 3 newton_cooling a         0.179          8             -0.06
 4 newton_cooling a         0.297         12             -0.06
 5 newton_cooling a         0.292         17             -0.211
 6 newton_cooling a         0.225         22             -0.423
 7 newton_cooling a         0.369         27             -0.423
 8 newton_cooling a         0.360         32             -0.574
 9 newton_cooling a         0.348         37             -0.726
10 newton_cooling a         0.335         42             -0.878
# ... with 1,952 more rows
```

3.5.11 And visualize them

```
resid %>%
  ggplot(aes(x = delta_time, y = resid)) +
  geom_line(aes(color = id_sensor)) +
  facet_grid(id_model ~ .) +
  labs(title = "Residuals for Newton and Semi-infinite models")
```



3.6 Using broom package to look at model-statistics

We will use a previous defined dataframe with the model and data:

```
model_nested_new %>%
  print()
```

```
# A tibble: 6 x 4
  id_model             id_sensor data               model
  <chr>                <chr>     <list>             <list>
1 newton_cooling       a         <tibble [327 x 2]> <nls>
2 newton_cooling       b         <tibble [327 x 2]> <nls>
3 newton_cooling       c         <tibble [327 x 2]> <nls>
4 semi_infinite_simple a         <tibble [327 x 2]> <nls>
5 semi_infinite_simple b         <tibble [327 x 2]> <nls>
6 semi_infinite_simple c         <tibble [327 x 2]> <nls>
```

The `tidy()` function extracts statistics from a model.

```
# apply over model_nested_new but only three variables
model_parameters <-
  model_nested_new %>%
  select(id_model, id_sensor, model) %>%
  mutate(tidy = map(model, tidy)) %>%
  select(-model) %>%
  unnest() %>%
  print()
```

```
# A tibble: 12 x 7
   id_model      id_sensor term      estimate std.error statistic   p.value
   <chr>         <chr>     <chr>        <dbl>     <dbl>     <dbl>     <dbl>
 1 newton_cooli~ a         delta_te~   -15.1     0.0526    -286.  0.
 2 newton_cooli~ a         tau_0       500.      4.84       103.  1.07e-250
 3 newton_cooli~ b         delta_te~    -7.59    0.0676    -112.  6.38e-262
 4 newton_cooli~ b         tau_0      1041.     16.2         64.2 9.05e-187
 5 newton_cooli~ c         delta_te~    -9.87    0.704      -14.0 3.16e- 35
 6 newton_cooli~ c         tau_0      3525.    299.          11.8 5.61e- 27
 7 semi_infinit~ a         delta_te~   -21.5     0.0649    -332.  0.
 8 semi_infinit~ a         tau_0       139.      1.15       121.  2.14e-272
 9 semi_infinit~ b         delta_te~   -10.6     0.0515    -206.  0.
10 semi_infinit~ b         tau_0       287.      2.58       111.  1.46e-260
11 semi_infinit~ c         delta_te~    -8.04    0.0129    -626.  0.
12 semi_infinit~ c         tau_0       500.      1.07       468.  0.
```

3.6.1 Get a sense of the coefficients

```
model_summary <-
  model_parameters %>%
  select(id_model, id_sensor, term, estimate) %>%
  spread(key = "term", value = "estimate") %>%
  print()
```

```
# A tibble: 6 x 4
  id_model             id_sensor delta_temperature_0 tau_0
  <chr>                <chr>                   <dbl> <dbl>
```

```
1 newton_cooling       a                    -15.1   500.
2 newton_cooling       b                     -7.59 1041.
3 newton_cooling       c                     -9.87 3525.
4 semi_infinite_simple a                    -21.5   139.
5 semi_infinite_simple b                    -10.6   287.
6 semi_infinite_simple c                     -8.04  500.
```

3.6.2 Summary

- this is just a smalll part of purrr
- there seem to be parallels between `tidyr::nest()/purrr::map()` and `dplyr::group_by()/dplyr::do()`
  - to my mind, the purrr framework is more understandable
  - update tweet from Hadley

References from Hadley:

- purrr 0.1.0 announcement
- purrr 0.2.0 announcement
- chapter from Garrett Grolemund and Hadley's forthcoming book

Chapter 4

Standalone Model

Classification problem

- `mtry`: Number of variables randomly sampled as candidates at each split.
- `ntree`: Number of trees to grow.

4.1 Load libraries

```
# load packages
library(caret)
library(mlbench)
library(randomForest)

# load dataset
data(Sonar)
set.seed(7)
```

4.2 Explore data

```
dplyr::glimpse(Sonar)
```

```
#> Observations: 208
#> Variables: 61
#> $ V1    <dbl> 0.0200, 0.0453, 0.0262, 0.0100, 0.0762, 0.0286, 0.0317, ...
#> $ V2    <dbl> 0.0371, 0.0523, 0.0582, 0.0171, 0.0666, 0.0453, 0.0956, ...
#> $ V3    <dbl> 0.0428, 0.0843, 0.1099, 0.0623, 0.0481, 0.0277, 0.1321, ...
#> $ V4    <dbl> 0.0207, 0.0689, 0.1083, 0.0205, 0.0394, 0.0174, 0.1408, ...
#> $ V5    <dbl> 0.0954, 0.1183, 0.0974, 0.0205, 0.0590, 0.0384, 0.1674, ...
#> $ V6    <dbl> 0.0986, 0.2583, 0.2280, 0.0368, 0.0649, 0.0990, 0.1710, ...
#> $ V7    <dbl> 0.1539, 0.2156, 0.2431, 0.1098, 0.1209, 0.1201, 0.0731, ...
#> $ V8    <dbl> 0.1601, 0.3481, 0.3771, 0.1276, 0.2467, 0.1833, 0.1401, ...
#> $ V9    <dbl> 0.3109, 0.3337, 0.5598, 0.0598, 0.3564, 0.2105, 0.2083, ...
#> $ V10   <dbl> 0.2111, 0.2872, 0.6194, 0.1264, 0.4459, 0.3039, 0.3513, ...
#> $ V11   <dbl> 0.1609, 0.4918, 0.6333, 0.0881, 0.4152, 0.2988, 0.1786, ...
#> $ V12   <dbl> 0.1582, 0.6552, 0.7060, 0.1992, 0.3952, 0.4250, 0.0658, ...
```

```
#> $ V13   <dbl> 0.2238, 0.6919, 0.5544, 0.0184, 0.4256, 0.6343, 0.0513, ...
#> $ V14   <dbl> 0.0645, 0.7797, 0.5320, 0.2261, 0.4135, 0.8198, 0.3752, ...
#> $ V15   <dbl> 0.0660, 0.7464, 0.6479, 0.1729, 0.4528, 1.0000, 0.5419, ...
#> $ V16   <dbl> 0.2273, 0.9444, 0.6931, 0.2131, 0.5326, 0.9988, 0.5440, ...
#> $ V17   <dbl> 0.3100, 1.0000, 0.6759, 0.0693, 0.7306, 0.9508, 0.5150, ...
#> $ V18   <dbl> 0.2999, 0.8874, 0.7551, 0.2281, 0.6193, 0.9025, 0.4262, ...
#> $ V19   <dbl> 0.5078, 0.8024, 0.8929, 0.4060, 0.2032, 0.7234, 0.2024, ...
#> $ V20   <dbl> 0.4797, 0.7818, 0.8619, 0.3973, 0.4636, 0.5122, 0.4233, ...
#> $ V21   <dbl> 0.5783, 0.5212, 0.7974, 0.2741, 0.4148, 0.2074, 0.7723, ...
#> $ V22   <dbl> 0.5071, 0.4052, 0.6737, 0.3690, 0.4292, 0.3985, 0.9735, ...
#> $ V23   <dbl> 0.4328, 0.3957, 0.4293, 0.5556, 0.5730, 0.5890, 0.9390, ...
#> $ V24   <dbl> 0.5550, 0.3914, 0.3648, 0.4846, 0.5399, 0.2872, 0.5559, ...
#> $ V25   <dbl> 0.6711, 0.3250, 0.5331, 0.3140, 0.3161, 0.2043, 0.5268, ...
#> $ V26   <dbl> 0.6415, 0.3200, 0.2413, 0.5334, 0.2285, 0.5782, 0.6826, ...
#> $ V27   <dbl> 0.7104, 0.3271, 0.5070, 0.5256, 0.6995, 0.5389, 0.5713, ...
#> $ V28   <dbl> 0.8080, 0.2767, 0.8533, 0.2520, 1.0000, 0.3750, 0.5429, ...
#> $ V29   <dbl> 0.6791, 0.4423, 0.6036, 0.2090, 0.7262, 0.3411, 0.2177, ...
#> $ V30   <dbl> 0.3857, 0.2028, 0.8514, 0.3559, 0.4724, 0.5067, 0.2149, ...
#> $ V31   <dbl> 0.1307, 0.3788, 0.8512, 0.6260, 0.5103, 0.5580, 0.5811, ...
#> $ V32   <dbl> 0.2604, 0.2947, 0.5045, 0.7340, 0.5459, 0.4778, 0.6323, ...
#> $ V33   <dbl> 0.5121, 0.1984, 0.1862, 0.6120, 0.2881, 0.3299, 0.2965, ...
#> $ V34   <dbl> 0.7547, 0.2341, 0.2709, 0.3497, 0.0981, 0.2198, 0.1873, ...
#> $ V35   <dbl> 0.8537, 0.1306, 0.4232, 0.3953, 0.1951, 0.1407, 0.2969, ...
#> $ V36   <dbl> 0.8507, 0.4182, 0.3043, 0.3012, 0.4181, 0.2856, 0.5163, ...
#> $ V37   <dbl> 0.6692, 0.3835, 0.6116, 0.5408, 0.4604, 0.3807, 0.6153, ...
#> $ V38   <dbl> 0.6097, 0.1057, 0.6756, 0.8814, 0.3217, 0.4158, 0.4283, ...
#> $ V39   <dbl> 0.4943, 0.1840, 0.5375, 0.9857, 0.2828, 0.4054, 0.5479, ...
#> $ V40   <dbl> 0.2744, 0.1970, 0.4719, 0.9167, 0.2430, 0.3296, 0.6133, ...
#> $ V41   <dbl> 0.0510, 0.1674, 0.4647, 0.6121, 0.1979, 0.2707, 0.5017, ...
#> $ V42   <dbl> 0.2834, 0.0583, 0.2587, 0.5006, 0.2444, 0.2650, 0.2377, ...
#> $ V43   <dbl> 0.2825, 0.1401, 0.2129, 0.3210, 0.1847, 0.0723, 0.1957, ...
#> $ V44   <dbl> 0.4256, 0.1628, 0.2222, 0.3202, 0.0841, 0.1238, 0.1749, ...
#> $ V45   <dbl> 0.2641, 0.0621, 0.2111, 0.4295, 0.0692, 0.1192, 0.1304, ...
#> $ V46   <dbl> 0.1386, 0.0203, 0.0176, 0.3654, 0.0528, 0.1089, 0.0597, ...
#> $ V47   <dbl> 0.1051, 0.0530, 0.1348, 0.2655, 0.0357, 0.0623, 0.1124, ...
#> $ V48   <dbl> 0.1343, 0.0742, 0.0744, 0.1576, 0.0085, 0.0494, 0.1047, ...
#> $ V49   <dbl> 0.0383, 0.0409, 0.0130, 0.0681, 0.0230, 0.0264, 0.0507, ...
#> $ V50   <dbl> 0.0324, 0.0061, 0.0106, 0.0294, 0.0046, 0.0081, 0.0159, ...
#> $ V51   <dbl> 0.0232, 0.0125, 0.0033, 0.0241, 0.0156, 0.0104, 0.0195, ...
#> $ V52   <dbl> 0.0027, 0.0084, 0.0232, 0.0121, 0.0031, 0.0045, 0.0201, ...
#> $ V53   <dbl> 0.0065, 0.0089, 0.0166, 0.0036, 0.0054, 0.0014, 0.0248, ...
#> $ V54   <dbl> 0.0159, 0.0048, 0.0095, 0.0150, 0.0105, 0.0038, 0.0131, ...
#> $ V55   <dbl> 0.0072, 0.0094, 0.0180, 0.0085, 0.0110, 0.0013, 0.0070, ...
#> $ V56   <dbl> 0.0167, 0.0191, 0.0244, 0.0073, 0.0015, 0.0089, 0.0138, ...
#> $ V57   <dbl> 0.0180, 0.0140, 0.0316, 0.0050, 0.0072, 0.0057, 0.0092, ...
#> $ V58   <dbl> 0.0084, 0.0049, 0.0164, 0.0044, 0.0048, 0.0027, 0.0143, ...
#> $ V59   <dbl> 0.0090, 0.0052, 0.0095, 0.0040, 0.0107, 0.0051, 0.0036, ...
#> $ V60   <dbl> 0.0032, 0.0044, 0.0078, 0.0117, 0.0094, 0.0062, 0.0103, ...
#> $ Class <fct> R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R,...
```

```
tibble::as_tibble(Sonar)
```

```
#> # A tibble: 208 x 61
#>        V1     V2     V3     V4     V5     V6     V7     V8     V9    V10
#>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
```

```
#>  1 0.02   0.0371 0.0428 0.0207 0.0954 0.0986 0.154  0.160  0.311  0.211
#>  2 0.0453 0.0523 0.0843 0.0689 0.118  0.258  0.216  0.348  0.334  0.287
#>  3 0.0262 0.0582 0.110  0.108  0.0974 0.228  0.243  0.377  0.560  0.619
#>  4 0.01   0.0171 0.0623 0.0205 0.0205 0.0368 0.110  0.128  0.0598 0.126
#>  5 0.0762 0.0666 0.0481 0.0394 0.059  0.0649 0.121  0.247  0.356  0.446
#>  6 0.0286 0.0453 0.0277 0.0174 0.0384 0.099  0.120  0.183  0.210  0.304
#>  7 0.0317 0.0956 0.132  0.141  0.167  0.171  0.0731 0.140  0.208  0.351
#>  8 0.0519 0.0548 0.0842 0.0319 0.116  0.0922 0.103  0.0613 0.146  0.284
#>  9 0.0223 0.0375 0.0484 0.0475 0.0647 0.0591 0.0753 0.0098 0.0684 0.149
#> 10 0.0164 0.0173 0.0347 0.007  0.0187 0.0671 0.106  0.0697 0.0962 0.0251
#> # ... with 198 more rows, and 51 more variables: V11 <dbl>, V12 <dbl>,
#> #   V13 <dbl>, V14 <dbl>, V15 <dbl>, V16 <dbl>, V17 <dbl>, V18 <dbl>,
#> #   V19 <dbl>, V20 <dbl>, V21 <dbl>, V22 <dbl>, V23 <dbl>, V24 <dbl>,
#> #   V25 <dbl>, V26 <dbl>, V27 <dbl>, V28 <dbl>, V29 <dbl>, V30 <dbl>,
#> #   V31 <dbl>, V32 <dbl>, V33 <dbl>, V34 <dbl>, V35 <dbl>, V36 <dbl>,
#> #   V37 <dbl>, V38 <dbl>, V39 <dbl>, V40 <dbl>, V41 <dbl>, V42 <dbl>,
#> #   V43 <dbl>, V44 <dbl>, V45 <dbl>, V46 <dbl>, V47 <dbl>, V48 <dbl>,
#> #   V49 <dbl>, V50 <dbl>, V51 <dbl>, V52 <dbl>, V53 <dbl>, V54 <dbl>,
#> #   V55 <dbl>, V56 <dbl>, V57 <dbl>, V58 <dbl>, V59 <dbl>, V60 <dbl>,
#> #   Class <fct>
```

```
# create 80%/20% for training and validation datasets
validationIndex <- createDataPartition(Sonar$Class, p=0.80, list=FALSE)
validation <- Sonar[-validationIndex,]
training   <- Sonar[validationIndex,]
```

```
# train a model and summarize model
set.seed(7)
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
fit.rf <- train(Class~., data=training,
                method = "rf",
                metric = "Accuracy",
                trControl = trainControl,
                ntree = 2000)

print(fit.rf)
```

```
#> Random Forest
#>
#> 167 samples
#>  60 predictor
#>   2 classes: 'M', 'R'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 3 times)
#> Summary of sample sizes: 150, 151, 150, 150, 150, 151, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>    2    0.8525572  0.7008578
#>   31    0.8184722  0.6341306
#>   60    0.7944526  0.5856805
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 2.
```

```
print(fit.rf$finalModel)
```

```
#>
#> Call:
#>  randomForest(x = x, y = y, ntree = 2000, mtry = param$mtry)
#>                Type of random forest: classification
#>                      Number of trees: 2000
#> No. of variables tried at each split: 2
#>
#>         OOB estimate of  error rate: 12.57%
#> Confusion matrix:
#>    M  R class.error
#> M 84  5  0.05617978
#> R 16 62  0.20512821
```

Accuracy: 85.26% at mtry=2

4.3 Apply tuning parameters for final model

```
# create standalone model using all training data
set.seed(7)
finalModel <- randomForest(Class~., training, mtry=2, ntree=2000)

# make a predictions on "new data" using the final model
finalPredictions <- predict(finalModel, validation[,1:60])
confusionMatrix(finalPredictions, validation$Class)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  M  R
#>          M 20  5
#>          R  2 14
#>
#>                Accuracy : 0.8293
#>                  95% CI : (0.6794, 0.9285)
#>     No Information Rate : 0.5366
#>     P-Value [Acc > NIR] : 8.511e-05
#>
#>                   Kappa : 0.653
#>
#>  Mcnemar's Test P-Value : 0.4497
#>
#>             Sensitivity : 0.9091
#>             Specificity : 0.7368
#>          Pos Pred Value : 0.8000
#>          Neg Pred Value : 0.8750
#>              Prevalence : 0.5366
#>          Detection Rate : 0.4878
#>    Detection Prevalence : 0.6098
#>       Balanced Accuracy : 0.8230
#>
```

```
#>        'Positive' Class : M
#>
```

Accuracy: 82.93%

4.4 Save model

```
# save the model to disk
saveRDS(finalModel, file.path(model_out_dir, "sonar-finalModel.rds"))
```

4.5 Use the saved model

```
# load the model
superModel <- readRDS(file.path(model_out_dir, "sonar-finalModel.rds"))
print(superModel)
```

```
#>
#> Call:
#>  randomForest(formula = Class ~ ., data = training, mtry = 2,      ntree = 2000)
#>                Type of random forest: classification
#>                      Number of trees: 2000
#> No. of variables tried at each split: 2
#>
#>         OOB estimate of  error rate: 14.97%
#> Confusion matrix:
#>    M  R class.error
#> M 81  8  0.08988764
#> R 17 61  0.21794872
```

4.6 Make prediction with new data

```
# make a predictions on "new data" using the final model
finalPredictions <- predict(superModel, validation[,1:60])
confusionMatrix(finalPredictions, validation$Class)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  M  R
#>          M 20  5
#>          R  2 14
#>
#>                Accuracy : 0.8293
#>                  95% CI : (0.6794, 0.9285)
#>     No Information Rate : 0.5366
#>     P-Value [Acc > NIR] : 8.511e-05
#>
#>                   Kappa : 0.653
#>
```

```
#>  Mcnemar's Test P-Value : 0.4497
#>
#>             Sensitivity : 0.9091
#>             Specificity : 0.7368
#>          Pos Pred Value : 0.8000
#>          Neg Pred Value : 0.8750
#>              Prevalence : 0.5366
#>          Detection Rate : 0.4878
#>    Detection Prevalence : 0.6098
#>       Balanced Accuracy : 0.8230
#>
#>        'Positive' Class : M
#>
```

Chapter 5

Glass classification

https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf

Glass Classification In this example, we use the glass data from the UCI Repository of Machine Learning Databases for classification. The task is to predict the type of a glass on basis of its chemical analysis. We start by splitting the data into a train and test set:

```
library(caret)
library(e1071)
library(rpart)

data(Glass, package="mlbench")
str(Glass)
```

```
#> 'data.frame':    214 obs. of  10 variables:
#>  $ RI  : num  1.52 1.52 1.52 1.52 1.52 ...
#>  $ Na  : num  13.6 13.9 13.5 13.2 13.3 ...
#>  $ Mg  : num  4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
#>  $ Al  : num  1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
#>  $ Si  : num  71.8 72.7 73 72.6 73.1 ...
#>  $ K   : num  0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
#>  $ Ca  : num  8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
#>  $ Ba  : num  0 0 0 0 0 0 0 0 0 0 ...
#>  $ Fe  : num  0 0 0 0 0 0.26 0 0 0 0.11 ...
#>  $ Type: Factor w/ 6 levels "1","2","3","5",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
## split data into a train and test set
index <- 1:nrow(Glass)
testindex <- sample(index, trunc(length(index)/3))
testset  <- Glass[testindex,]
trainset <- Glass[-testindex,]
```

Both for the SVM and the partitioning tree (via `rpart()`), we fit the model and try to predict the test set values:

```
## svm
svm.model <- svm(Type ~ ., data = trainset, cost = 100, gamma = 1)
svm.pred  <- predict(svm.model, testset[,-10])
```

(The dependent variable, Type, has column number 10. cost is a general penalizing parameter for C-classification and `gamma` is the radial basis function-specific `kernel` parameter.)

```
## rpart
rpart.model <- rpart(Type ~ ., data = trainset)
rpart.pred <- predict(rpart.model, testset[,-10], type = "class")
```

A cross-tabulation of the true versus the predicted values yields:

```
## compute svm confusion matrix
table(pred = svm.pred, true = testset[,10])
```

```
#>     true
#> pred  1  2  3  5  6  7
#>    1 17  5  2  0  0  0
#>    2  3 22  1  5  0  3
#>    3  1  1  0  0  0  0
#>    5  0  0  0  0  0  0
#>    6  0  0  0  0  1  0
#>    7  0  0  0  0  0 10
```

```
## compute rpart confusion matrix
table(pred = rpart.pred, true = testset[,10])
```

```
#>     true
#> pred  1  2  3  5  6  7
#>    1 14  4  1  0  0  1
#>    2  7 19  0  1  0  0
#>    3  0  2  2  0  0  0
#>    5  0  3  0  4  1  1
#>    6  0  0  0  0  0  0
#>    7  0  0  0  0  0 11
```

5.0.1 Comparison test sets

```
confusionMatrix(svm.pred, testset$Type)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  1  2  3  5  6  7
#>          1 17  5  2  0  0  0
#>          2  3 22  1  5  0  3
#>          3  1  1  0  0  0  0
#>          5  0  0  0  0  0  0
#>          6  0  0  0  0  1  0
#>          7  0  0  0  0  0 10
#>
#> Overall Statistics
#>
#>                Accuracy : 0.7042
#>                  95% CI : (0.5841, 0.8067)
#>     No Information Rate : 0.3944
#>     P-Value [Acc > NIR] : 1.235e-07
#>
#>                   Kappa : 0.5676
#>
```

```
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
#> Sensitivity            0.8095   0.7857  0.00000  0.00000  1.00000   0.7692
#> Specificity            0.8600   0.7209  0.97059  1.00000  1.00000   1.0000
#> Pos Pred Value         0.7083   0.6471  0.00000      NaN  1.00000   1.0000
#> Neg Pred Value         0.9149   0.8378  0.95652  0.92958  1.00000   0.9508
#> Prevalence             0.2958   0.3944  0.04225  0.07042  0.01408   0.1831
#> Detection Rate         0.2394   0.3099  0.00000  0.00000  0.01408   0.1408
#> Detection Prevalence   0.3380   0.4789  0.02817  0.00000  0.01408   0.1408
#> Balanced Accuracy      0.8348   0.7533  0.48529  0.50000  1.00000   0.8846
```

```
confusionMatrix(rpart.pred, testset$Type)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  1  2  3  5  6  7
#>          1 14  4  1  0  0  1
#>          2  7 19  0  1  0  0
#>          3  0  2  2  0  0  0
#>          5  0  3  0  4  1  1
#>          6  0  0  0  0  0  0
#>          7  0  0  0  0  0 11
#>
#> Overall Statistics
#>
#>                Accuracy : 0.7042
#>                  95% CI : (0.5841, 0.8067)
#>     No Information Rate : 0.3944
#>     P-Value [Acc > NIR] : 1.235e-07
#>
#>                   Kappa : 0.5932
#>
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
#> Sensitivity            0.6667   0.6786  0.66667  0.80000  0.00000   0.8462
#> Specificity            0.8800   0.8140  0.97059  0.92424  1.00000   1.0000
#> Pos Pred Value         0.7000   0.7037  0.50000  0.44444      NaN   1.0000
#> Neg Pred Value         0.8627   0.7955  0.98507  0.98387  0.98592   0.9667
#> Prevalence             0.2958   0.3944  0.04225  0.07042  0.01408   0.1831
#> Detection Rate         0.1972   0.2676  0.02817  0.05634  0.00000   0.1549
#> Detection Prevalence   0.2817   0.3803  0.05634  0.12676  0.00000   0.1549
#> Balanced Accuracy      0.7733   0.7463  0.81863  0.86212  0.50000   0.9231
```

5.0.2 Comparison with resamples

Finally, we compare the performance of the two methods by computing the respective accuracy rates and the kappa indices (as computed by `classAgreement()` also contained in package `e1071`). In Table 1, we

summarize the results of 10 replications—Support Vector Machines show better results.

```
set.seed(1234567)

# SVM
fit.svm <- train(Type ~., data = trainset,
                 method = "svmRadial")

# Random Forest
fit.rpart <- train(Type ~., data = trainset,
                method="rpart")

# collect resamples
results <- resamples(list(svm = fit.svm,
                          rpart  = fit.rpart))

summary(results)
```

```
#>
#> Call:
#> summary.resamples(object = results)
#>
#> Models: svm, rpart
#> Number of resamples: 25
#>
#> Accuracy
#>            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> svm   0.5652174 0.6181818 0.6545455 0.6602196 0.6875000 0.7636364    0
#> rpart 0.4218750 0.5686275 0.6078431 0.6085792 0.6666667 0.7291667    0
#>
#> Kappa
#>            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> svm   0.3750000 0.4611973 0.5127538 0.5141015 0.5516812 0.6705069    0
#> rpart 0.1983751 0.3745819 0.4350282 0.4425535 0.5098926 0.6183486    0
```

Chapter 6

Ozone SVM

https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf

```
library(e1071)
library(rpart)

data(Ozone, package="mlbench")
## split data into a train and test set
index <- 1:nrow(Ozone)
testindex <- sample(index, trunc(length(index)/3))
testset <- na.omit(Ozone[testindex,-3])
trainset <- na.omit(Ozone[-testindex,-3])
```

```
## svm
svm.model <- svm(V4 ~ ., data = trainset, cost = 1000, gamma = 0.0001)
svm.pred  <- predict(svm.model, testset[,-3])
crossprod(svm.pred - testset[,3]) / length(testindex)
```

```
##          [,1]
## [1,] 9.389549
```

```
## rpart
rpart.model <- rpart(V4 ~ ., data = trainset)
rpart.pred  <- predict(rpart.model, testset[,-3])
crossprod(rpart.pred - testset[,3]) / length(testindex)
```

```
##          [,1]
## [1,] 20.39648
```

Chapter 7

A gentle introduction to support vector machines using R

https://eight2late.wordpress.com/2017/02/07/a-gentle-introduction-to-support-vector-machines-using-r/

7.1 Support vector machines in R

In this demo we'll use the svm interface that is implemented in the `e1071` R package. This interface provides R programmers access to the comprehensive `libsvm` library written by Chang and Lin. I'll use two toy datasets: the famous iris dataset available with the base R package and the sonar dataset from the mlbench package. I won't describe details of the datasets as they are discussed at length in the documentation that I have linked to. However, it is worth mentioning the reasons why I chose these datasets:

As mentioned earlier, no real life dataset is linearly separable, but the iris dataset is almost so. Consequently, it is a good illustration of using linear SVMs. Although one almost never uses these in practice, I have illustrated their use primarily for pedagogical reasons. The sonar dataset is a good illustration of the benefits of using RBF kernels in cases where the dataset is hard to visualise (60 variables in this case!). In general, one would almost always use RBF (or other nonlinear) kernels in practice.

With that said, let's get right to it. I assume you have R and RStudio installed. For instructions on how to do this, have a look at the first article in this series. The processing preliminaries – loading libraries, data and creating training and test datasets are much the same as in my previous articles so I won't dwell on these here. For completeness, however, I'll list all the code so you can run it directly in R or R studio (a complete listing of the code can be found here):

7.2 SVM on `iris` dataset

7.2.1 Training and test datasets

```
#load required library
library(e1071)

#load built-in iris dataset
data(iris)
```

```
#set seed to ensure reproducible results
set.seed(42)

#split into training and test sets
iris[, "train"] <- ifelse(runif(nrow(iris)) < 0.8, 1, 0)

#separate training and test sets
trainset <- iris[iris$train == 1,]
testset <- iris[iris$train == 0,]

#get column index of train flag
trainColNum <- grep("train", names(trainset))

#remove train flag column from train and test sets
trainset <- trainset[,-trainColNum]
testset <- testset[,-trainColNum]

dim(trainset)
```

```
## [1] 115   5
```

```
dim(testset)
```

```
## [1] 35  5
```

7.2.2 Build the SVM model

```
#get column index of predicted variable in dataset
typeColNum <- grep("Species", names(iris))

#build model - linear kernel and C-classification (soft margin) with default cost (C=1)
svm_model <- svm(Species~ ., data = trainset,
                 method = "C-classification",
                 kernel = "linear")
svm_model
```

```
##
## Call:
## svm(formula = Species ~ ., data = trainset, method = "C-classification",
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.25
##
## Number of Support Vectors:  24
```

The output from the SVM model show that there are 24 support vectors. If desired, these can be examined using the SV variable in the model – i.e via svm_model$SV.

7.2.3 Support Vectors

```
# support vectors
svm_model$SV
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 19   -0.25639203  1.76683447   -1.3228618  -1.3054201
## 42   -1.70055936 -1.70445193   -1.5591789  -1.3054201
## 45   -0.97847569  1.76683447   -1.2047033  -1.1709010
## 53    1.18777530  0.14690082    0.5676747   0.3088091
## 55    0.70638619 -0.54735646    0.3904369   0.3088091
## 57    0.46569164  0.60973900    0.4495161   0.4433282
## 58   -1.21917025 -1.47303284   -0.3775936  -0.3637864
## 69    0.34534436 -1.93587102    0.3313576   0.3088091
## 71   -0.01569747  0.37831991    0.5085954   0.7123664
## 73    0.46569164 -1.24161374    0.5676747   0.3088091
## 78    0.94708075 -0.08451828    0.6267539   0.5778473
## 84    0.10464981 -0.77877556    0.6858332   0.4433282
## 85   -0.61743386 -0.08451828    0.3313576   0.3088091
## 86    0.10464981  0.84115809    0.3313576   0.4433282
## 99   -0.97847569 -1.24161374   -0.5548314  -0.2292673
## 107  -1.21917025 -1.24161374    0.3313576   0.5778473
## 111   0.70638619  0.37831991    0.6858332   0.9814046
## 117   0.70638619 -0.08451828    0.9221503   0.7123664
## 124   0.46569164 -0.77877556    0.5676747   0.7123664
## 130   1.54881714 -0.08451828    1.0993881   0.4433282
## 138   0.58603892  0.14690082    0.9221503   0.7123664
## 139   0.10464981 -0.08451828    0.5085954   0.7123664
## 147   0.46569164 -1.24161374    0.6267539   0.8468855
## 150  -0.01569747 -0.08451828    0.6858332   0.7123664
```

The test prediction accuracy indicates that the linear performs quite well on this dataset, confirming that it is indeed near linearly separable. To check performance by class, one can create a confusion matrix as described in my post on random forests. I'll leave this as an exercise for you. Another point is that we have used a soft-margin classification scheme with a cost C=1. You can experiment with this by explicitly changing the value of C. Again, I'll leave this for you an exercise.

7.2.4 Predictions on training model

```
# training set predictions
pred_train <- predict(svm_model, trainset)
mean(pred_train == trainset$Species)
```

```
## [1] 0.9826087
```

```
# [1] 0.9826087
```

7.2.5 Predictions on test model

```
# test set predictions
pred_test <-predict(svm_model, testset)
mean(pred_test == testset$Species)
```

```
## [1] 0.9142857
```

```
# [1] 0.9142857
```

7.2.6 Confusion matrix and Accuracy

```
# confusion matrix
cm <- table(pred_test, testset$Species)
cm
```

```
##
## pred_test    setosa versicolor virginica
##   setosa         18          0         0
##   versicolor      0          5         3
##   virginica       0          0         9
```

```
# accuracy
sum(diag(cm)) / sum(cm)
```

```
## [1] 0.9142857
```

7.3 SVM with Radial Basis Function kernel. Linear

7.3.1 Training and test sets

```
#load required library (assuming e1071 is already loaded)
library(mlbench)

#load Sonar dataset
data(Sonar)
#set seed to ensure reproducible results
set.seed(42)
#split into training and test sets
Sonar[, "train"] <- ifelse(runif(nrow(Sonar))<0.8,1,0)

#separate training and test sets
trainset <- Sonar[Sonar$train==1,]
testset <- Sonar[Sonar$train==0,]

#get column index of train flag
trainColNum <- grep("train",names(trainset))
#remove train flag column from train and test sets
trainset <- trainset[,-trainColNum]
testset <- testset[,-trainColNum]

#get column index of predicted variable in dataset
typeColNum <- grep("Class",names(Sonar))
```

7.3.2 Predictions on Training model

```
#build model - linear kernel and C-classification with default cost (C=1)
svm_model <- svm(Class~ ., data=trainset,
                 method="C-classification",
                 kernel="linear")

#training set predictions
pred_train <-predict(svm_model,trainset)
mean(pred_train==trainset$Class)
```

```
## [1] 0.969697
```

7.3.3 Predictions on test model

```
#test set predictions
pred_test <-predict(svm_model,testset)
mean(pred_test==testset$Class)
```

```
## [1] 0.6046512
```

I'll leave you to examine the contents of the model. The important point to note here is that the performance of the model with the test set is quite dismal compared to the previous case. This simply indicates that the linear kernel is not appropriate here. Let's take a look at what happens if we use the RBF kernel with default values for the parameters:

7.4 SVM with Radial Basis Function kernel. Non-linear

7.4.1 Predictions on training model

```
#build model: radial kernel, default params
svm_model <- svm(Class~ ., data=trainset,
                 method="C-classification",
                 kernel="radial")
# print params
svm_model$cost
```

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

```
svm_model$gamma
```

```
## [1] 0.01666667
```

```
#training set predictions
pred_train <-predict(svm_model,trainset)
mean(pred_train==trainset$Class)
```

```
## [1] 0.9878788
```

7.4.2 Predictions on test model

```
#test set predictions
pred_test <-predict(svm_model,testset)
mean(pred_test==testset$Class)
```

```
## [1] 0.7674419
```

That's a pretty decent improvement from the linear kernel. Let's see if we can do better by doing some parameter tuning. To do this we first invoke tune.svm and use the parameters it gives us in the call to svm:

7.4.3 Tuning of parameters

```
# find optimal parameters in a specified range
tune_out <- tune.svm(x = trainset[,-typeColNum],
                     y = trainset[, typeColNum],
                     gamma = 10^(-3:3),
                     cost = c(0.01, 0.1, 1, 10, 100, 1000),
                     kernel = "radial")

#print best values of cost and gamma
tune_out$best.parameters$cost
```

```
## [1] 100
```

```
tune_out$best.parameters$gamma
```

```
## [1] 0.01
```

```
#build model
svm_model <- svm(Class~ ., data = trainset,
                 method = "C-classification",
                 kernel = "radial",
                 cost = tune_out$best.parameters$cost,
                 gamma = tune_out$best.parameters$gamma)
```

7.4.4 Prediction on training model with new parameters

```
# training set predictions
pred_train <-predict(svm_model,trainset)
mean(pred_train==trainset$Class)
```

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

7.4.5 Prediction on test model with new parameters

```
# test set predictions
pred_test <-predict(svm_model,testset)
mean(pred_test==testset$Class)
```

```
## [1] 0.8139535
```

Which is fairly decent improvement on the un-optimised case.

7.5 Wrapping up

This bring us to the end of this introductory exploration of SVMs in R. To recap, the distinguishing feature of SVMs in contrast to most other techniques is that they attempt to construct optimal separation boundaries between different categories.

SVMs are quite versatile and have been applied to a wide variety of domains ranging from chemistry to pattern recognition. They are best used in binary classification scenarios. This brings up a question as to where SVMs are to be preferred to other binary classification techniques such as logistic regression. The honest response is, "it depends" – but here are some points to keep in mind when choosing between the two. A general point to keep in mind is that SVM algorithms tend to be expensive both in terms of memory and computation, issues that can start to hurt as the size of the dataset increases.

Given all the above caveats and considerations, the best way to figure out whether an SVM approach will work for your problem may be to do what most machine learning practitioners do: try it out!

Chapter 8

Multiclass classification. iris

```
# load the caret package
library(caret)
# attach the iris dataset to the environment
data(iris)
# rename the dataset
dataset <- iris
```

We will split the loaded dataset into two, 80% of which we will use to train our models and 20% that we will hold back as a validation dataset.

```
# create a list of 80% of the rows in the original dataset we can use for training
validationIndex <- createDataPartition(dataset$Species, p=0.80, list=FALSE)
# select 20% of the data for validation
validation <- dataset[-validationIndex,]

# use the remaining 80% of data to training and testing the models
dataset <- dataset[validationIndex,]

# dimensions of dataset
dim(dataset)
```

```
#> [1] 120   5
```

```
# list types for each attribute
sapply(dataset, class)
```

```
#> Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
#>    "numeric"    "numeric"    "numeric"    "numeric"     "factor"
```

8.1 Peek at the dataset

```
# take a peek at the first 5 rows of the data
head(dataset)
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 2          4.9         3.0          1.4         0.2  setosa
#> 3          4.7         3.2          1.3         0.2  setosa
#> 4          4.6         3.1          1.5         0.2  setosa
```

```
#> 5          5.0         3.6          1.4         0.2  setosa
#> 6          5.4         3.9          1.7         0.4  setosa
#> 7          4.6         3.4          1.4         0.3  setosa
```

```
library(dplyr)

glimpse(dataset)
```

```
#> Observations: 120
#> Variables: 5
#> $ Sepal.Length <dbl> 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 4.4, 4.9, 5.4, 4.8,...
#> $ Sepal.Width  <dbl> 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 2.9, 3.1, 3.7, 3.4,...
#> $ Petal.Length <dbl> 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.4, 1.5, 1.5, 1.6,...
#> $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.1, 0.2, 0.2,...
#> $ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, s...
```

```
library(skimr)

skim(dataset)
```

```
#> Skim summary statistics
#>  n obs: 120
#>  n variables: 5
#>
#> -- Variable type:factor -------------------------------------------------------
#>  variable missing complete   n n_unique                       top_counts
#>   Species       0      120 120        3 set: 40, ver: 40, vir: 40, NA: 0
#>  ordered
#>    FALSE
#>
#> -- Variable type:numeric ------------------------------------------------------
#>      variable missing complete   n mean   sd  p0 p25 p50 p75 p100     hist
#>  Petal.Length       0      120 120 3.74 1.76 1   1.6 4.3 5.1  6.7
#>   Petal.Width       0      120 120 1.2  0.76 0.1 0.3 1.3 1.8  2.5
#>  Sepal.Length       0      120 120 5.83 0.84 4.3 5.1 5.7 6.5  7.7
#>   Sepal.Width       0      120 120 3.06 0.42 2   2.8 3   3.3  4.4
```

8.2 Levels of the class

```
# list the levels for the class
levels(dataset$Species)
```

```
#> [1] "setosa"     "versicolor" "virginica"
```

8.3 class distribution

```
# summarize the class distribution
percentage <- prop.table(table(dataset$Species)) * 100
cbind(freq=table(dataset$Species), percentage=percentage)
```

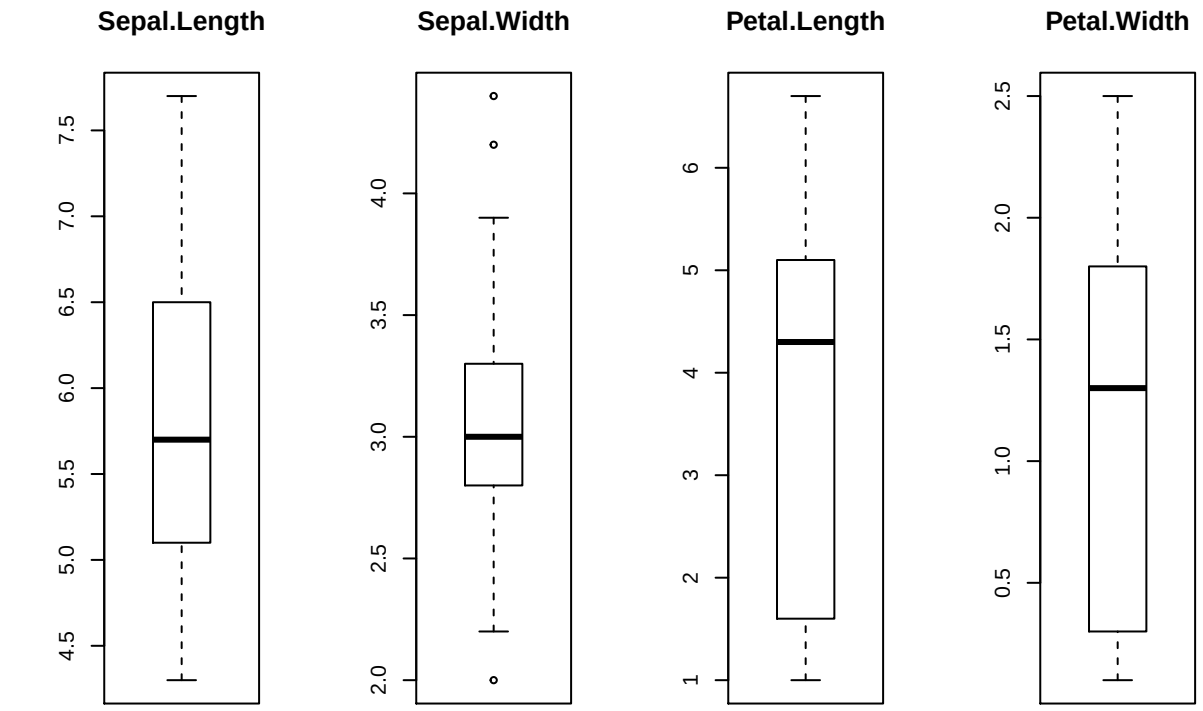
```
#>            freq percentage
#> setosa       40   33.33333
```

```
#> versicolor    40    33.33333
#> virginica     40    33.33333
```

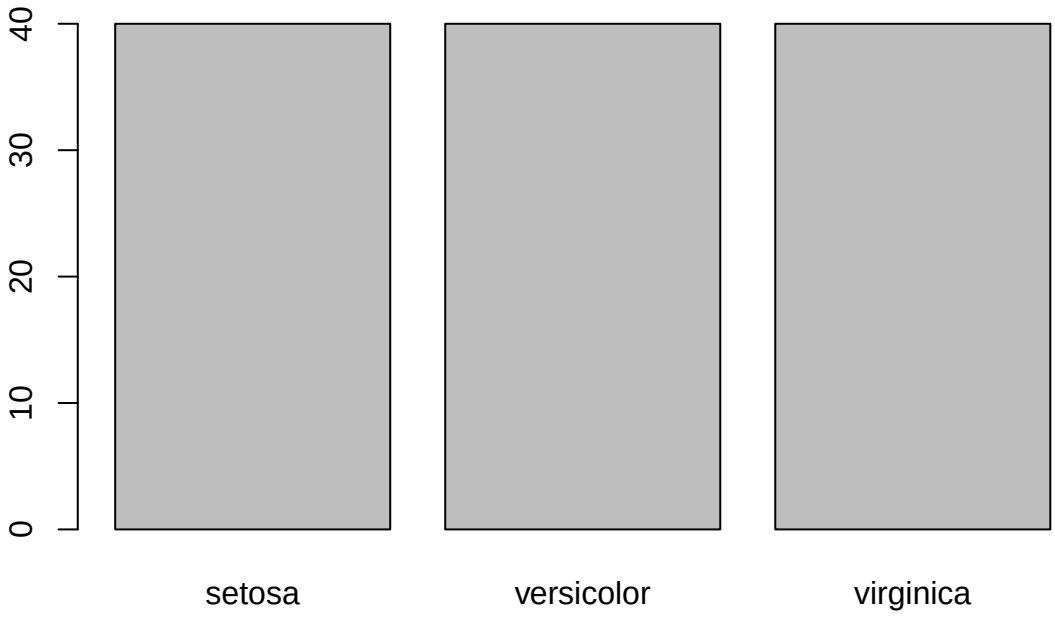
8.4 Visualize the dataset

```
# split input and output
x <- dataset[,1:4]
y <- dataset[,5]
```

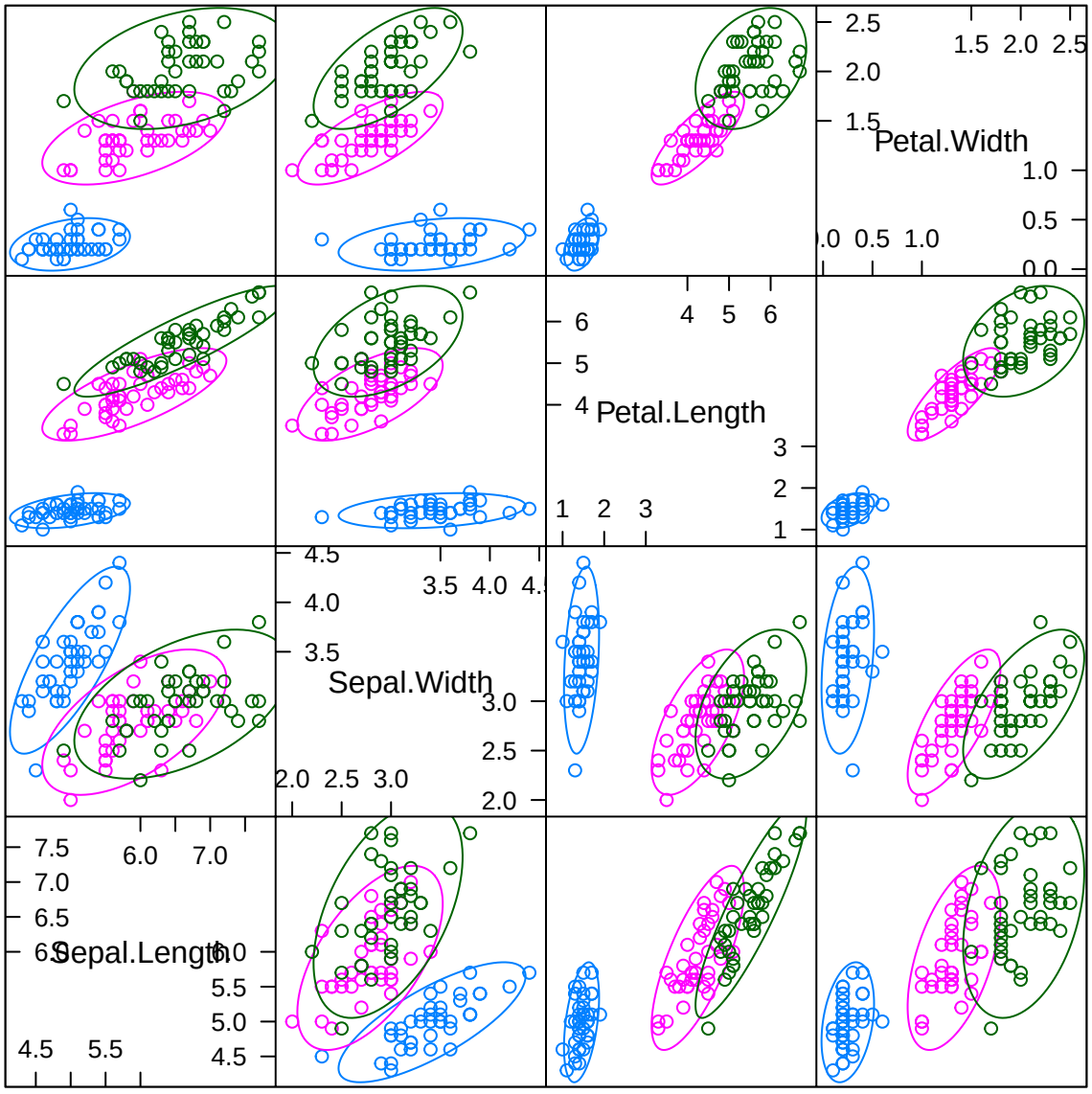
```
# boxplot for each attribute on one image
par(mfrow=c(1,4))
for(i in 1:4) {
    boxplot(x[,i], main=names(dataset)[i])
}
```




```
# barplot for class breakdown
plot(y)
```

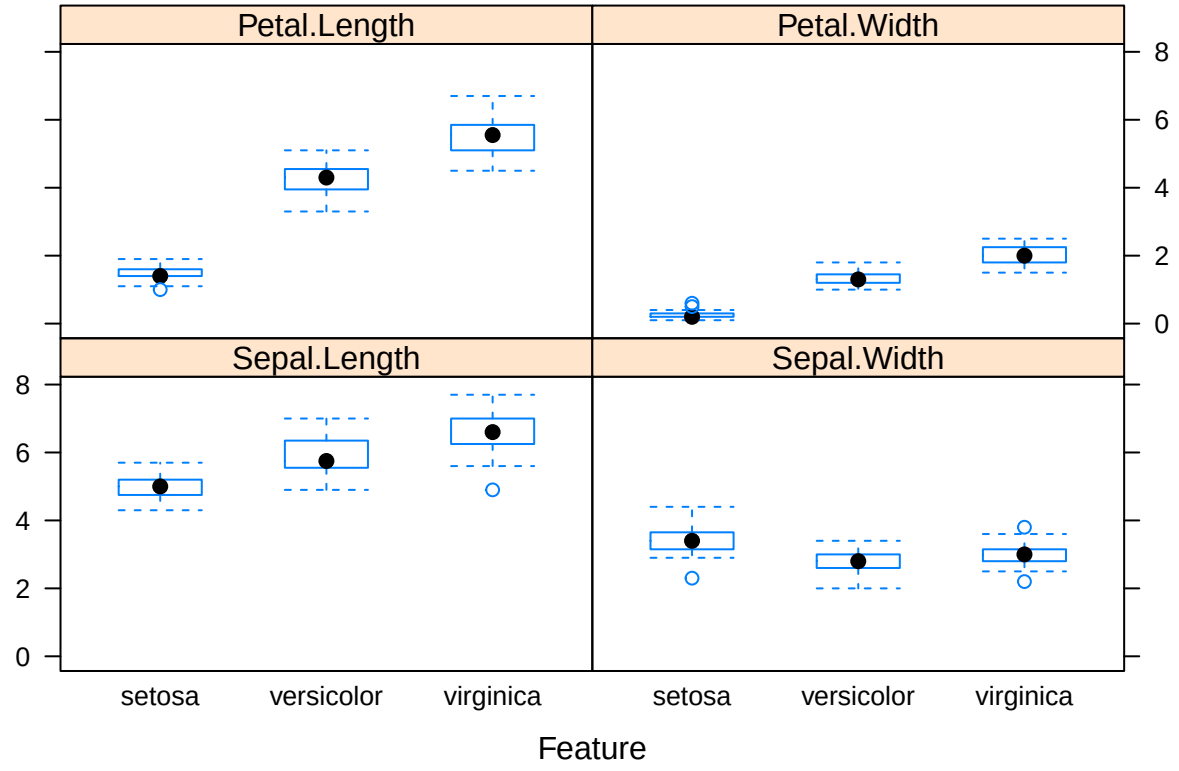



```
# scatter plot matrix
featurePlot(x=x, y=y, plot="ellipse")
```

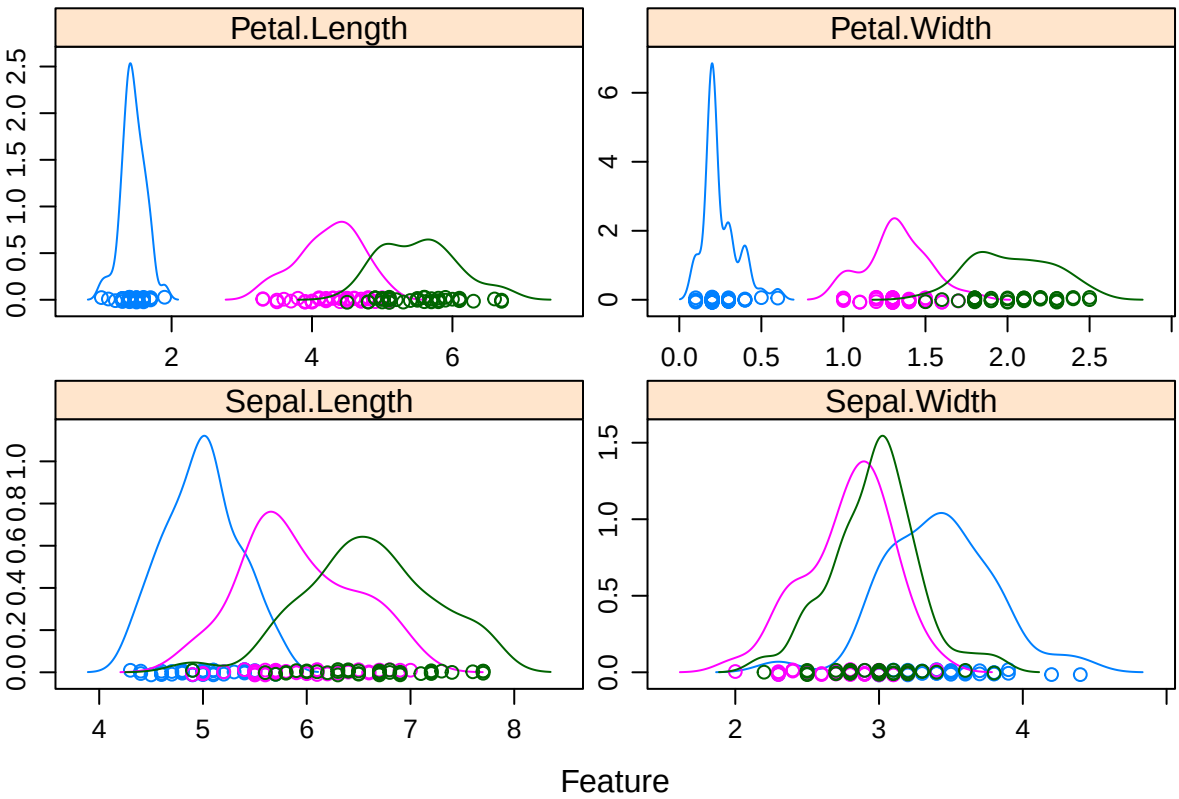


Scatter Plot Matrix

```
# box and whisker plots for each attribute
featurePlot(x=x, y=y, plot="box")
```




```
# density plots for each attribute by class value
scales <- list(x=list(relation="free"), y=list(relation="free"))
featurePlot(x=x, y=y, plot="density", scales=scales)
```



8.5 Evaluate algorithms

8.5.1 split and metrics

```
# Run algorithms using 10-fold cross-validation
trainControl <- trainControl(method="cv", number=10)
metric <- "Accuracy"
```

8.5.2 build models

```
# LDA
set.seed(7)
fit.lda <- train(Species~., data=dataset, method = "lda",
                 metric=metric, trControl=trainControl)
# CART
set.seed(7)
fit.cart <- train(Species~., data=dataset, method = "rpart",
                  metric=metric, trControl=trainControl)
# KNN
set.seed(7)
fit.knn <- train(Species~., data=dataset, method = "knn",
                 metric=metric, trControl=trainControl)
# SVM
set.seed(7)
fit.svm <- train(Species~., data=dataset, method = "svmRadial",
                 metric=metric, trControl=trainControl)
# Random Forest
set.seed(7)
fit.rf <- train(Species~., data=dataset, method = "rf",
                metric=metric, trControl=trainControl)
```

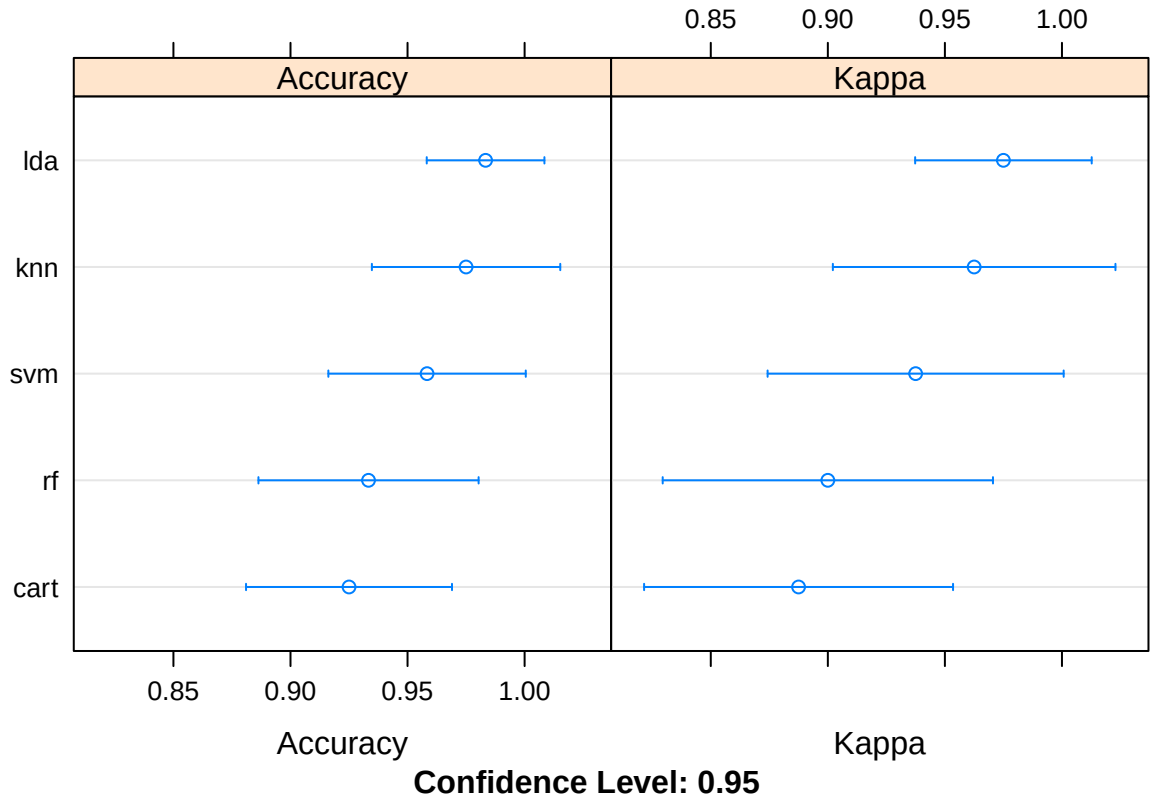
8.5.3 compare

```
#summarize accuracy of models
results <- resamples(list(lda  = fit.lda,
                          cart = fit.cart,
                          knn  = fit.knn,
                          svm  = fit.svm,
                          rf   = fit.rf))
summary(results)
```

```
#>
#> Call:
#> summary.resamples(object = results)
#>
#> Models: lda, cart, knn, svm, rf
#> Number of resamples: 10
#>
#> Accuracy
#>           Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
```

```
#> lda  0.9166667 1.0000000 1.0000000 0.9833333 1.0000000    1    0
#> cart 0.8333333 0.9166667 0.9166667 0.9250000 0.9791667    1    0
#> knn  0.8333333 1.0000000 1.0000000 0.9750000 1.0000000    1    0
#> svm  0.8333333 0.9166667 1.0000000 0.9583333 1.0000000    1    0
#> rf   0.8333333 0.9166667 0.9166667 0.9333333 1.0000000    1    0
#>
#> Kappa
#>       Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
#> lda  0.875   1.000  1.000 0.9750 1.00000    1    0
#> cart 0.750   0.875  0.875 0.8875 0.96875    1    0
#> knn  0.750   1.000  1.000 0.9625 1.00000    1    0
#> svm  0.750   0.875  1.000 0.9375 1.00000    1    0
#> rf   0.750   0.875  0.875 0.9000 1.00000    1    0
```

```
# compare accuracy of models
dotplot(results)
```




```
# summarize Best Model
print(fit.lda)
```

```
#> Linear Discriminant Analysis
#>
#> 120 samples
#>   4 predictor
#>   3 classes: 'setosa', 'versicolor', 'virginica'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 108, 108, 108, 108, 108, 108, ...
```

```
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.9833333  0.975
```

8.6 Make predictions

```
# estimate skill of LDA on the validation dataset
predictions <- predict(fit.lda, validation)
confusionMatrix(predictions, validation$Species)
```

```
#> Confusion Matrix and Statistics
#>
#>             Reference
#> Prediction   setosa versicolor virginica
#>   setosa         10          0         0
#>   versicolor      0         10         1
#>   virginica       0          0         9
#>
#> Overall Statistics
#>
#>                Accuracy : 0.9667
#>                  95% CI : (0.8278, 0.9992)
#>     No Information Rate : 0.3333
#>     P-Value [Acc > NIR] : 2.963e-13
#>
#>                   Kappa : 0.95
#>
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: setosa Class: versicolor Class: virginica
#> Sensitivity                 1.0000            1.0000           0.9000
#> Specificity                 1.0000            0.9500           1.0000
#> Pos Pred Value              1.0000            0.9091           1.0000
#> Neg Pred Value              1.0000            1.0000           0.9524
#> Prevalence                  0.3333            0.3333           0.3333
#> Detection Rate              0.3333            0.3333           0.3000
#> Detection Prevalence        0.3333            0.3667           0.3000
#> Balanced Accuracy           1.0000            0.9750           0.9500
```

Chapter 9

Regression. Boston

Comparison of various algorithms.

```
# load packages
library(mlbench)
library(caret)
library(corrplot)

# attach the BostonHousing dataset
data(BostonHousing)
```

```
dplyr::glimpse(BostonHousing)
```

```
#> Observations: 506
#> Variables: 14
#> $ crim    <dbl> 0.00632, 0.02731, 0.02729, 0.03237, 0.06905, 0.02985, ...
#> $ zn      <dbl> 18.0, 0.0, 0.0, 0.0, 0.0, 0.0, 12.5, 12.5, 12.5, 12.5,...
#> $ indus   <dbl> 2.31, 7.07, 7.07, 2.18, 2.18, 2.18, 7.87, 7.87, 7.87, ...
#> $ chas    <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ nox     <dbl> 0.538, 0.469, 0.469, 0.458, 0.458, 0.458, 0.524, 0.524...
#> $ rm      <dbl> 6.575, 6.421, 7.185, 6.998, 7.147, 6.430, 6.012, 6.172...
#> $ age     <dbl> 65.2, 78.9, 61.1, 45.8, 54.2, 58.7, 66.6, 96.1, 100.0,...
#> $ dis     <dbl> 4.0900, 4.9671, 4.9671, 6.0622, 6.0622, 6.0622, 5.5605...
#> $ rad     <dbl> 1, 2, 2, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, ...
#> $ tax     <dbl> 296, 242, 242, 222, 222, 222, 311, 311, 311, 311, 311,...
#> $ ptratio <dbl> 15.3, 17.8, 17.8, 18.7, 18.7, 18.7, 15.2, 15.2, 15.2, ...
#> $ b       <dbl> 396.90, 396.90, 392.83, 394.63, 396.90, 394.12, 395.60...
#> $ lstat   <dbl> 4.98, 9.14, 4.03, 2.94, 5.33, 5.21, 12.43, 19.15, 29.9...
#> $ medv    <dbl> 24.0, 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, ...
```

```
tibble::as_tibble(BostonHousing)
```

```
#> # A tibble: 506 x 14
#>       crim    zn indus chas    nox    rm   age   dis   rad   tax ptratio
#>      <dbl> <dbl> <dbl> <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
#>  1 0.00632  18    2.31 0     0.538  6.58  65.2  4.09     1   296    15.3
#>  2 0.0273    0    7.07 0     0.469  6.42  78.9  4.97     2   242    17.8
#>  3 0.0273    0    7.07 0     0.469  7.18  61.1  4.97     2   242    17.8
#>  4 0.0324    0    2.18 0     0.458  7.00  45.8  6.06     3   222    18.7
#>  5 0.0690    0    2.18 0     0.458  7.15  54.2  6.06     3   222    18.7
```

```
#>  6 0.0298    0    2.18 0     0.458  6.43  58.7  6.06     3   222    18.7
#>  7 0.0883   12.5  7.87 0     0.524  6.01  66.6  5.56     5   311    15.2
#>  8 0.145    12.5  7.87 0     0.524  6.17  96.1  5.95     5   311    15.2
#>  9 0.211    12.5  7.87 0     0.524  5.63 100    6.08     5   311    15.2
#> 10 0.170    12.5  7.87 0     0.524  6.00  85.9  6.59     5   311    15.2
#> # ... with 496 more rows, and 3 more variables: b <dbl>, lstat <dbl>,
#> #   medv <dbl>
```

```
# Split out validation dataset
# create a list of 80% of the rows in the original dataset we can use for training
set.seed(7)
validationIndex <- createDataPartition(BostonHousing$medv,
                                       p=0.80, list=FALSE)

# select 20% of the data for validation
validation <- BostonHousing[-validationIndex,]

# use the remaining 80% of data to training and testing the models
dataset <- BostonHousing[validationIndex,]
```

```
# dimensions of dataset
dim(validation)
```

```
#> [1] 99 14
```

```
dim(dataset)
```

```
#> [1] 407  14
```

```
# list types for each attribute
sapply(dataset, class)
```

```
#>      crim        zn     indus      chas       nox        rm       age
#> "numeric" "numeric" "numeric"  "factor" "numeric" "numeric" "numeric"
#>       dis       rad       tax   ptratio         b     lstat      medv
#> "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
```

```
# take a peek at the first 20 rows of the data
head(dataset, n=20)
```

```
#>       crim   zn indus chas   nox    rm   age    dis rad tax ptratio      b
#> 2  0.02731  0.0  7.07    0 0.469 6.421  78.9 4.9671   2 242    17.8 396.90
#> 3  0.02729  0.0  7.07    0 0.469 7.185  61.1 4.9671   2 242    17.8 392.83
#> 4  0.03237  0.0  2.18    0 0.458 6.998  45.8 6.0622   3 222    18.7 394.63
#> 5  0.06905  0.0  2.18    0 0.458 7.147  54.2 6.0622   3 222    18.7 396.90
#> 6  0.02985  0.0  2.18    0 0.458 6.430  58.7 6.0622   3 222    18.7 394.12
#> 7  0.08829 12.5  7.87    0 0.524 6.012  66.6 5.5605   5 311    15.2 395.60
#> 8  0.14455 12.5  7.87    0 0.524 6.172  96.1 5.9505   5 311    15.2 396.90
#> 9  0.21124 12.5  7.87    0 0.524 5.631 100.0 6.0821   5 311    15.2 386.63
#> 13 0.09378 12.5  7.87    0 0.524 5.889  39.0 5.4509   5 311    15.2 390.50
#> 14 0.62976  0.0  8.14    0 0.538 5.949  61.8 4.7075   4 307    21.0 396.90
#> 15 0.63796  0.0  8.14    0 0.538 6.096  84.5 4.4619   4 307    21.0 380.02
#> 16 0.62739  0.0  8.14    0 0.538 5.834  56.5 4.4986   4 307    21.0 395.62
#> 17 1.05393  0.0  8.14    0 0.538 5.935  29.3 4.4986   4 307    21.0 386.85
#> 18 0.78420  0.0  8.14    0 0.538 5.990  81.7 4.2579   4 307    21.0 386.75
#> 19 0.80271  0.0  8.14    0 0.538 5.456  36.6 3.7965   4 307    21.0 288.99
#> 20 0.72580  0.0  8.14    0 0.538 5.727  69.5 3.7965   4 307    21.0 390.95
#> 23 1.23247  0.0  8.14    0 0.538 6.142  91.7 3.9769   4 307    21.0 396.90
```

```
#> 25 0.75026  0.0  8.14    0 0.538 5.924  94.1 4.3996   4 307    21.0 394.33
#> 26 0.84054  0.0  8.14    0 0.538 5.599  85.7 4.4546   4 307    21.0 303.42
#> 27 0.67191  0.0  8.14    0 0.538 5.813  90.3 4.6820   4 307    21.0 376.88
#>    lstat medv
#> 2   9.14 21.6
#> 3   4.03 34.7
#> 4   2.94 33.4
#> 5   5.33 36.2
#> 6   5.21 28.7
#> 7  12.43 22.9
#> 8  19.15 27.1
#> 9  29.93 16.5
#> 13 15.71 21.7
#> 14  8.26 20.4
#> 15 10.26 18.2
#> 16  8.47 19.9
#> 17  6.58 23.1
#> 18 14.67 17.5
#> 19 11.69 20.2
#> 20 11.28 18.2
#> 23 18.72 15.2
#> 25 16.30 15.6
#> 26 16.51 13.9
#> 27 14.81 16.6
```

```
library(skimr)
skim_with(numeric = list(hist = NULL))
skim(dataset)
```

```
#> Skim summary statistics
#>  n obs: 407
#>  n variables: 14
#>
#> -- Variable type:factor -------------------------------------------------------
#>  variable missing complete   n n_unique           top_counts ordered
#>      chas       0      407 407        2 0: 376, 1: 31, NA: 0   FALSE
#>
#> -- Variable type:numeric ------------------------------------------------------
#>  variable missing complete   n   mean     sd       p0     p25    p50
#>       age       0      407 407  68.83  28.38   2.9     45.05   77.7
#>         b       0      407 407 357.88  87.66   0.32   374.5   391.13
#>      crim       0      407 407   3.58   8.32   0.0091   0.086   0.29
#>       dis       0      407 407   3.73   2.07   1.13     2.03    3.22
#>     indus       0      407 407  11.36   6.82   0.46     5.19    9.9
#>     lstat       0      407 407  12.83   7.29   1.73     6.89   11.5
#>      medv       0      407 407  22.61   9.38   5       17.05   21.2
#>       nox       0      407 407   0.56   0.12   0.39     0.45    0.54
#>   ptratio       0      407 407  18.49   2.12  12.6     17.4    19
#>       rad       0      407 407   9.46   8.64   1        4       5
#>        rm       0      407 407   6.28   0.72   3.86     5.87    6.18
#>       tax       0      407 407 405.62 166.3  188      279     330
#>        zn       0      407 407  10.57  22.54   0        0       0
#>     p75   p100
#>   94.55 100
#>  396.27 396.9
```

```
#>    3.5   88.98
#>    5.1   10.71
#>   18.1   27.74
#>   17.18  37.97
#>   25     50
#>    0.63   0.87
#>   20.2   22
#>   24     24
#>    6.61   8.78
#>  666    711
#>    0     95
```

```
dataset[,4] <- as.numeric(as.character(dataset[,4]))
```

```
skim(dataset)
```

```
#> Skim summary statistics
#>  n obs: 407
#>  n variables: 14
#>
#> -- Variable type:numeric -------------------------------------------------------
#>  variable missing complete   n    mean     sd       p0     p25    p50
#>       age       0      407 407  68.83   28.38   2.9     45.05   77.7
#>         b       0      407 407 357.88   87.66   0.32   374.5   391.13
#>      chas       0      407 407   0.076   0.27   0        0       0
#>      crim       0      407 407   3.58    8.32   0.0091   0.086   0.29
#>       dis       0      407 407   3.73    2.07   1.13     2.03    3.22
#>     indus       0      407 407  11.36    6.82   0.46     5.19    9.9
#>     lstat       0      407 407  12.83    7.29   1.73     6.89   11.5
#>      medv       0      407 407  22.61    9.38   5       17.05   21.2
#>       nox       0      407 407   0.56    0.12   0.39     0.45    0.54
#>   ptratio       0      407 407  18.49    2.12  12.6     17.4    19
#>       rad       0      407 407   9.46    8.64   1        4       5
#>        rm       0      407 407   6.28    0.72   3.86     5.87    6.18
#>       tax       0      407 407 405.62  166.3  188      279     330
#>        zn       0      407 407  10.57   22.54   0        0       0
#>     p75   p100
#>   94.55 100
#>  396.27 396.9
#>    0      1
#>    3.5   88.98
#>    5.1   10.71
#>   18.1   27.74
#>   17.18  37.97
#>   25     50
#>    0.63   0.87
#>   20.2   22
#>   24     24
#>    6.61   8.78
#>  666    711
#>    0     95
```

no more factors or character variables

```
# find correlation between variables
cor(dataset[,1:13])
```

```
#>                crim          zn       indus        chas         nox
#> crim     1.00000000 -0.19790631  0.40597009 -0.05713065  0.4232413
#> zn      -0.19790631  1.00000000 -0.51895069 -0.04843477 -0.5058512
#> indus    0.40597009 -0.51895069  1.00000000  0.08003629  0.7665481
#> chas    -0.05713065 -0.04843477  0.08003629  1.00000000  0.1027366
#> nox      0.42324132 -0.50585121  0.76654811  0.10273656  1.0000000
#> rm      -0.21513269  0.28942883 -0.37673408  0.08252441 -0.2988506
#> age      0.35438190 -0.57070265  0.65858310  0.10938121  0.7238371
#> dis     -0.39050970  0.65618742 -0.72305885 -0.11142420 -0.7708680
#> rad      0.64240501 -0.29952976  0.56774365 -0.00901245  0.5851676
#> tax      0.60622608 -0.28791668  0.68070916 -0.02779018  0.6521787
#> ptratio  0.28929828 -0.35341215  0.32920610 -0.13554380  0.1416616
#> b       -0.30211854  0.16927489 -0.33597951  0.04724420 -0.3620791
#> lstat    0.47537617 -0.39712686  0.59212718 -0.04569239  0.5819645
#>                  rm        age        dis         rad         tax
#> crim    -0.21513269  0.3543819 -0.3905097  0.64240501  0.60622608
#> zn       0.28942883 -0.5707027  0.6561874 -0.29952976 -0.28791668
#> indus   -0.37673408  0.6585831 -0.7230588  0.56774365  0.68070916
#> chas     0.08252441  0.1093812 -0.1114242 -0.00901245 -0.02779018
#> nox     -0.29885055  0.7238371 -0.7708680  0.58516760  0.65217875
#> rm       1.00000000 -0.2325359  0.1952159 -0.19149122 -0.26794733
#> age     -0.23253586  1.0000000 -0.7503321  0.45235421  0.50164657
#> dis      0.19521590 -0.7503321  1.0000000 -0.49382744 -0.52649325
#> rad     -0.19149122  0.4523542 -0.4938274  1.00000000  0.92137876
#> tax     -0.26794733  0.5016466 -0.5264932  0.92137876  1.00000000
#> ptratio -0.32000372  0.2564318 -0.2021897  0.45312318  0.44192428
#> b        0.15539923 -0.2512574  0.2826819 -0.41033069 -0.41848779
#> lstat   -0.62038075  0.5932128 -0.4957302  0.47306604  0.52339243
#>             ptratio          b       lstat
#> crim     0.2892983 -0.3021185  0.47537617
#> zn      -0.3534121  0.1692749 -0.39712686
#> indus    0.3292061 -0.3359795  0.59212718
#> chas    -0.1355438  0.0472442 -0.04569239
#> nox      0.1416616 -0.3620791  0.58196447
#> rm      -0.3200037  0.1553992 -0.62038075
#> age      0.2564318 -0.2512574  0.59321281
#> dis     -0.2021897  0.2826819 -0.49573024
#> rad      0.4531232 -0.4103307  0.47306604
#> tax      0.4419243 -0.4184878  0.52339243
#> ptratio  1.0000000 -0.1495283  0.35375936
#> b       -0.1495283  1.0000000 -0.37661571
#> lstat    0.3537594 -0.3766157  1.00000000
```

```
library(dplyr)

m <- cor(dataset[,1:13])
diag(m) <- 0
```

```
# select variables with correlation 0.7 and above
threshold <- 0.7
ok <- apply(abs(m) >= threshold, 1, any)
m[ok, ok]
```

```
#>            indus        nox        age        dis        rad        tax
#> indus  0.0000000  0.7665481  0.6585831 -0.7230588  0.5677436  0.6807092
```

```
#> nox    0.7665481  0.0000000  0.7238371 -0.7708680  0.5851676  0.6521787
#> age    0.6585831  0.7238371  0.0000000 -0.7503321  0.4523542  0.5016466
#> dis   -0.7230588 -0.7708680 -0.7503321  0.0000000 -0.4938274 -0.5264932
#> rad    0.5677436  0.5851676  0.4523542 -0.4938274  0.0000000  0.9213788
#> tax    0.6807092  0.6521787  0.5016466 -0.5264932  0.9213788  0.0000000
```

```
# values of correlation >= 0.7
ind <- sapply(1:13, function(x) abs(m[, x]) > 0.7)
m[ind]
```

```
#>  [1]  0.7665481 -0.7230588  0.7665481  0.7238371 -0.7708680  0.7238371
#>  [7] -0.7503321 -0.7230588 -0.7708680 -0.7503321  0.9213788  0.9213788
```

```
# defining a index for selecting if the condition is met
cind <- apply(m, 2, function(x) any(abs(x) > 0.7))
cm <- m[, cind] # since col6 only has values less than 0.5 it is not taken
cm
```

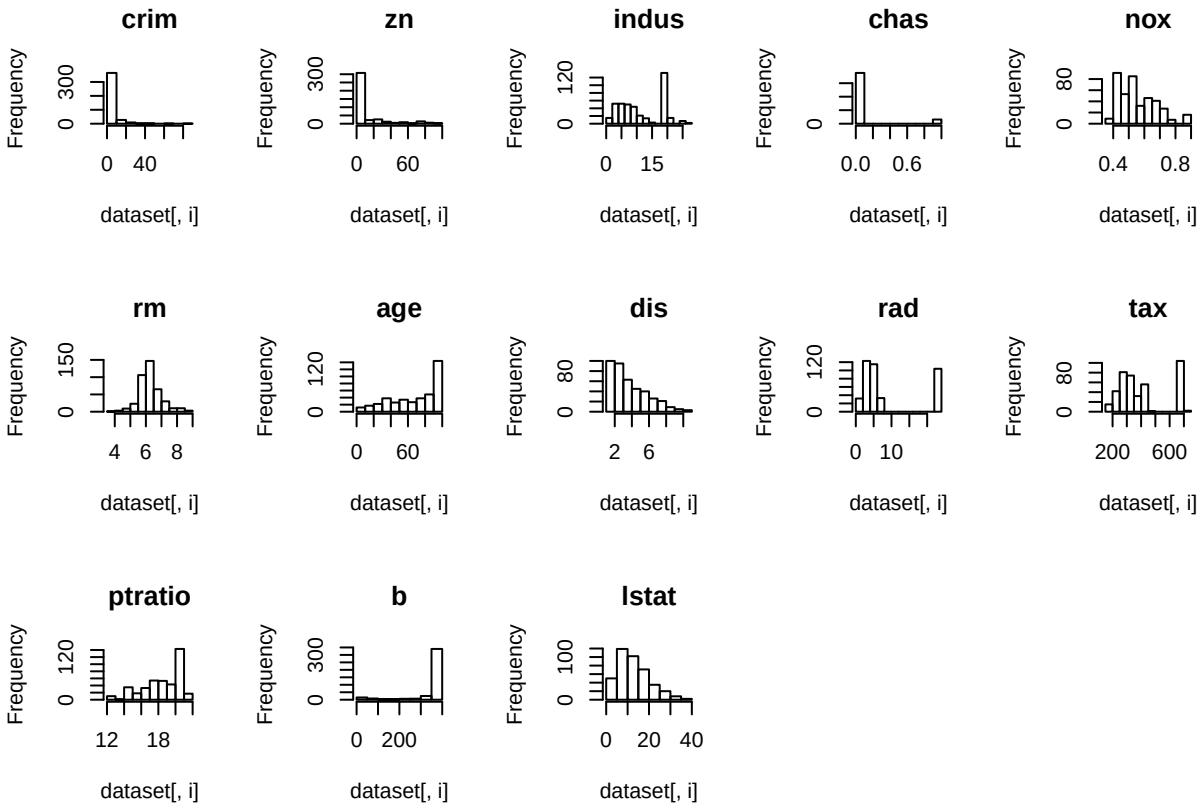
```
#>              indus        nox        age        dis         rad
#> crim     0.40597009  0.4232413  0.3543819 -0.3905097  0.64240501
#> zn      -0.51895069 -0.5058512 -0.5707027  0.6561874 -0.29952976
#> indus    0.00000000  0.7665481  0.6585831 -0.7230588  0.56774365
#> chas     0.08003629  0.1027366  0.1093812 -0.1114242 -0.00901245
#> nox      0.76654811  0.0000000  0.7238371 -0.7708680  0.58516760
#> rm      -0.37673408 -0.2988506 -0.2325359  0.1952159 -0.19149122
#> age      0.65858310  0.7238371  0.0000000 -0.7503321  0.45235421
#> dis     -0.72305885 -0.7708680 -0.7503321  0.0000000 -0.49382744
#> rad      0.56774365  0.5851676  0.4523542 -0.4938274  0.00000000
#> tax      0.68070916  0.6521787  0.5016466 -0.5264932  0.92137876
#> ptratio  0.32920610  0.1416616  0.2564318 -0.2021897  0.45312318
#> b       -0.33597951 -0.3620791 -0.2512574  0.2826819 -0.41033069
#> lstat    0.59212718  0.5819645  0.5932128 -0.4957302  0.47306604
#>                tax
#> crim     0.60622608
#> zn      -0.28791668
#> indus    0.68070916
#> chas    -0.02779018
#> nox      0.65217875
#> rm      -0.26794733
#> age      0.50164657
#> dis     -0.52649325
#> rad      0.92137876
#> tax      0.00000000
#> ptratio  0.44192428
#> b       -0.41848779
#> lstat    0.52339243
```

```
rind <- apply(cm, 1, function(x) any(abs(x) > 0.7))
rm <- cm[rind, ]
rm
```

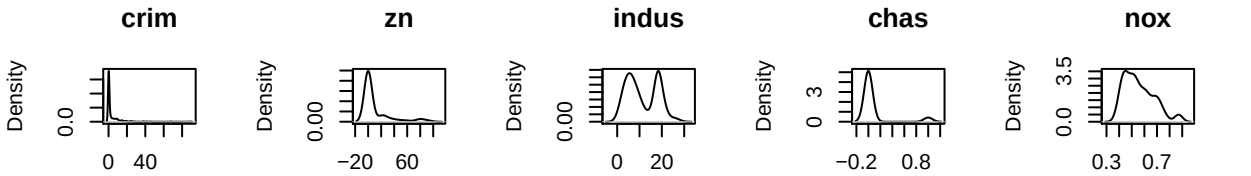
```
#>            indus        nox        age        dis        rad        tax
#> indus  0.0000000  0.7665481  0.6585831 -0.7230588  0.5677436  0.6807092
#> nox    0.7665481  0.0000000  0.7238371 -0.7708680  0.5851676  0.6521787
#> age    0.6585831  0.7238371  0.0000000 -0.7503321  0.4523542  0.5016466
#> dis   -0.7230588 -0.7708680 -0.7503321  0.0000000 -0.4938274 -0.5264932
#> rad    0.5677436  0.5851676  0.4523542 -0.4938274  0.0000000  0.9213788
```

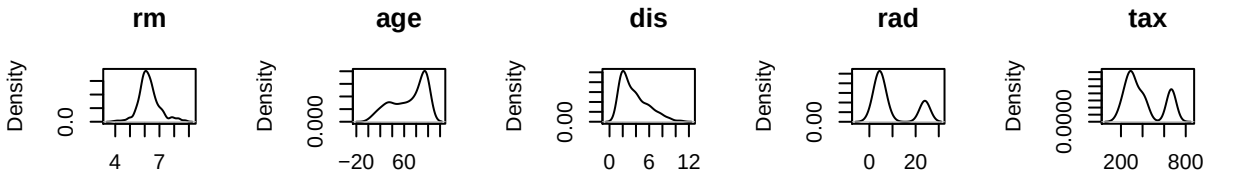
```
#> tax    0.6807092  0.6521787  0.5016466 -0.5264932  0.9213788  0.0000000
```

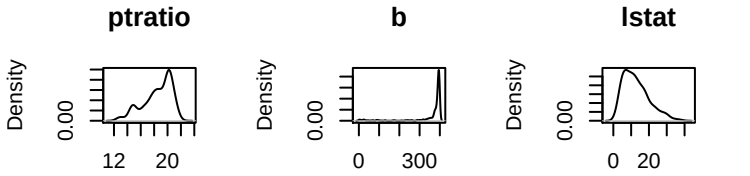
```
# histograms for each attribute
par(mfrow=c(3,5))
for(i in 1:13) {
    hist(dataset[,i], main=names(dataset)[i])
}
```



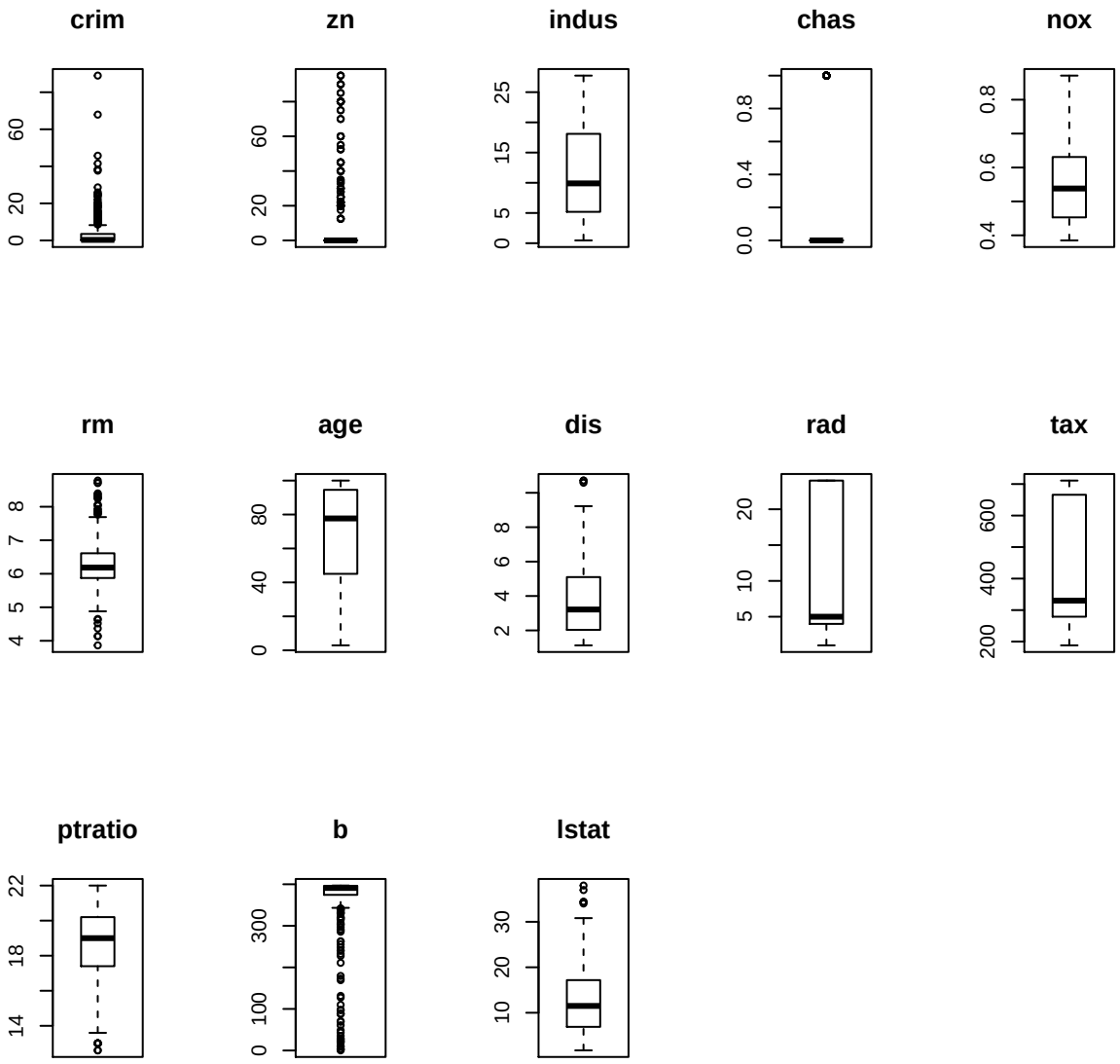

```
# density plot for each attribute
par(mfrow=c(3,5))
for(i in 1:13) {
plot(density(dataset[,i]), main=names(dataset)[i])
}
```



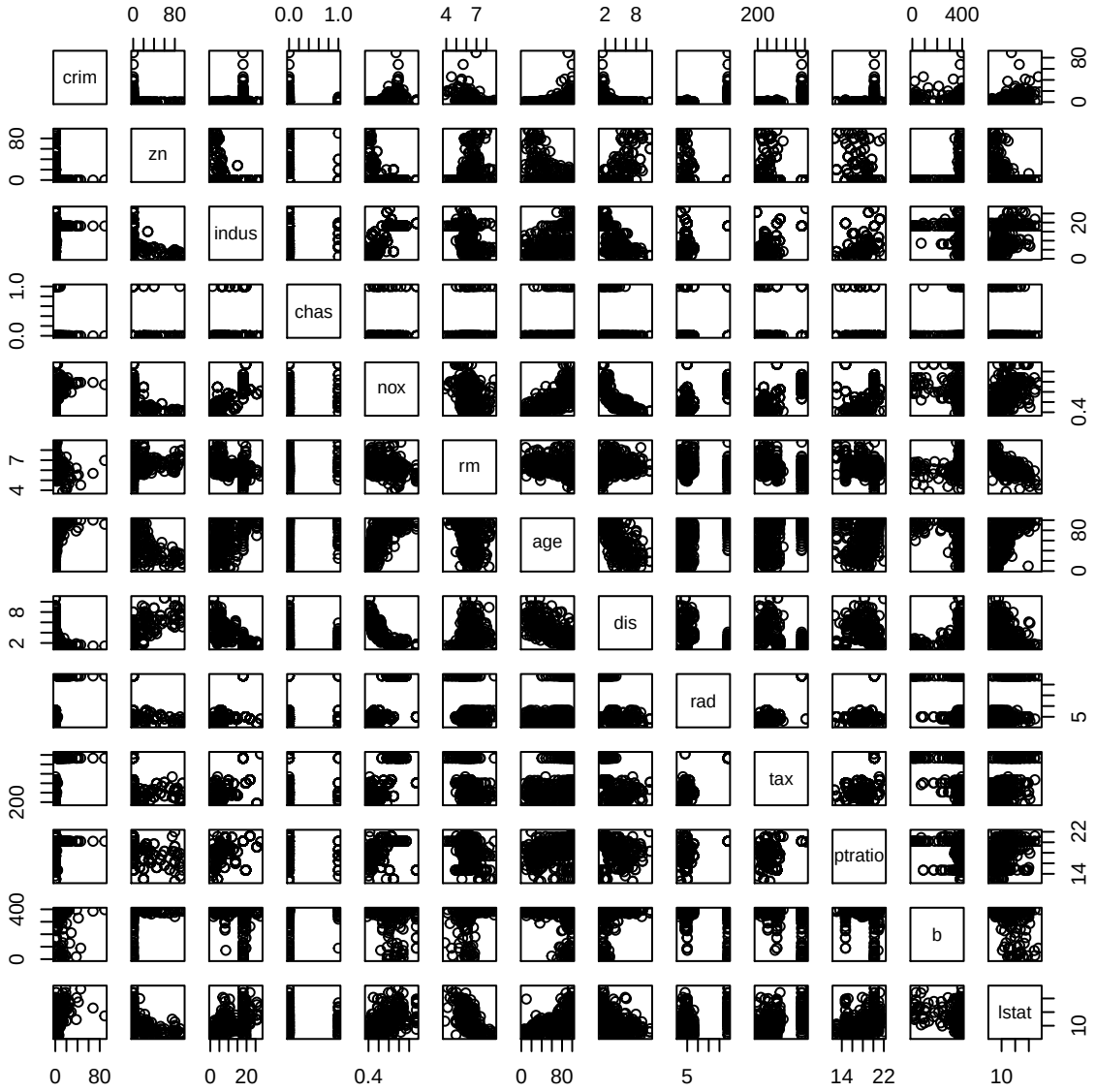




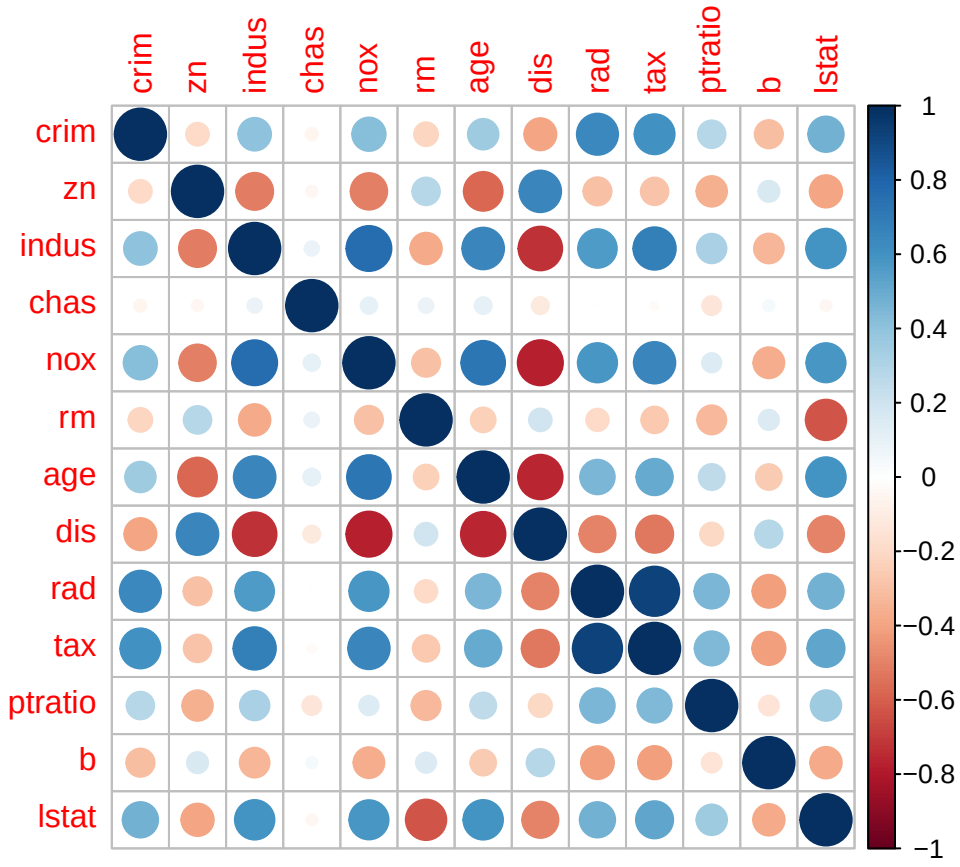

```
# boxplots for each attribute
par(mfrow=c(3,5))
for(i in 1:13) {
boxplot(dataset[,i], main=names(dataset)[i])
}
```




```
# scatter plot matrix
pairs(dataset[,1:13])
```




```
# correlation plot
correlations <- cor(dataset[,1:13])
corrplot(correlations, method="circle")
```



9.1 Evaluation

```
# Run algorithms using 10-fold cross-validation
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
```

```
# LM
set.seed(7)
fit.lm <- train(medv~., data=dataset, method="lm",
                metric=metric, preProc=c("center", "scale"),
                trControl=trainControl)
# GLM
set.seed(7)
fit.glm <- train(medv~., data=dataset, method="glm",
                 metric=metric, preProc=c("center", "scale"),
                 trControl=trainControl)
# GLMNET
set.seed(7)
fit.glmnet <- train(medv~., data=dataset, method="glmnet",
                    metric=metric,
                    preProc=c("center", "scale"),
                    trControl=trainControl)
# SVM
set.seed(7)
```

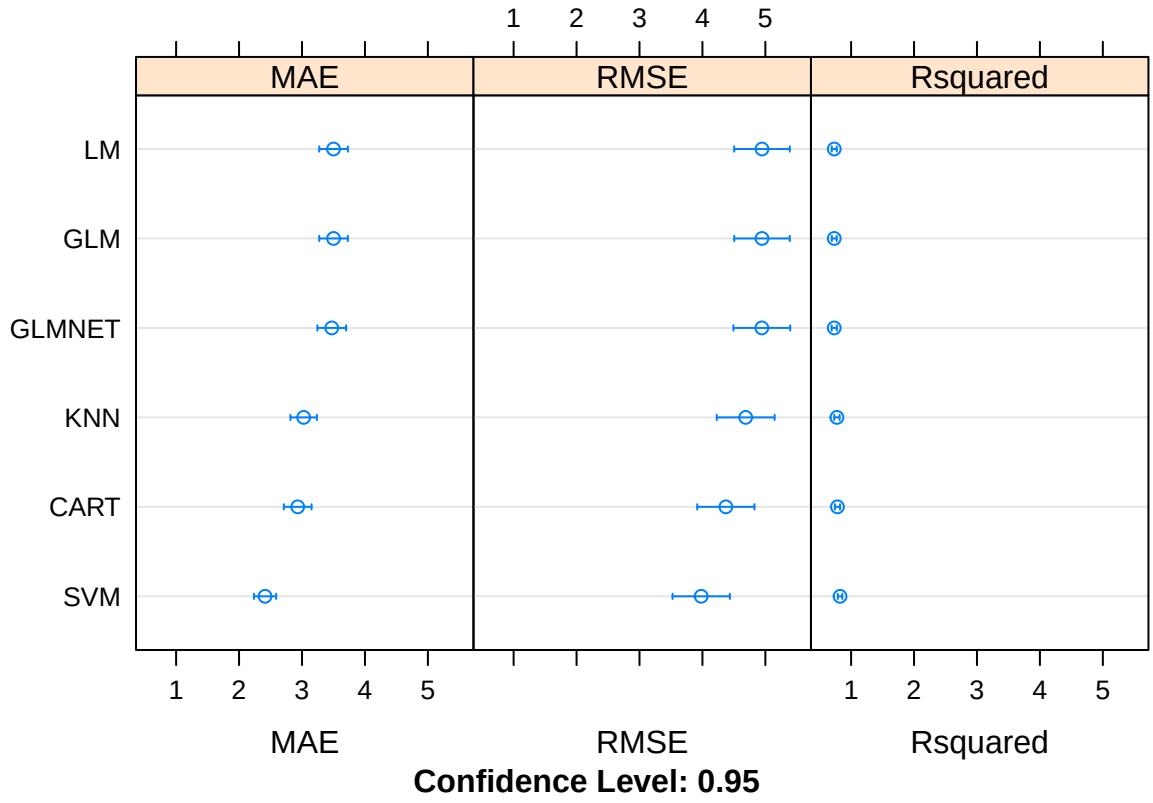
```
fit.svm <- train(medv~., data=dataset, method="svmRadial",
                 metric=metric,
                 preProc=c("center", "scale"),
                 trControl=trainControl)
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
fit.cart <- train(medv~., data=dataset, method="rpart",
                  metric=metric, tuneGrid=grid,
                  preProc=c("center", "scale"),
                  trControl=trainControl)
# KNN
set.seed(7)
fit.knn <- train(medv~., data=dataset, method="knn",
                 metric=metric, preProc=c("center", "scale"),
                 trControl=trainControl)
```

```
# Compare algorithms
results <- resamples(list(LM     = fit.lm,
                          GLM    = fit.glm,
                          GLMNET = fit.glmnet,
                          SVM    = fit.svm,
                          CART   = fit.cart,
                          KNN    = fit.knn))
summary(results)
```

```
#>
#> Call:
#> summary.resamples(object = results)
#>
#> Models: LM, GLM, GLMNET, SVM, CART, KNN
#> Number of resamples: 30
#>
#> MAE
#>            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
#> LM     2.275798 3.105566 3.476184 3.501075 3.988069 4.806931    0
#> GLM    2.275798 3.105566 3.476184 3.501075 3.988069 4.806931    0
#> GLMNET 2.249747 3.054908 3.436463 3.472943 4.002405 4.819706    0
#> SVM    1.636837 2.083861 2.395264 2.412779 2.615126 3.597281    0
#> CART   2.043357 2.509889 2.875049 2.931782 3.294371 4.707060    0
#> KNN    2.036111 2.675617 3.023173 3.026927 3.346350 4.663144    0
#>
#> RMSE
#>            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
#> LM     2.945061 4.120871 4.590366 4.946754 5.622836 7.966565    0
#> GLM    2.945061 4.120871 4.590366 4.946754 5.622836 7.966565    0
#> GLMNET 2.900783 4.135902 4.580667 4.944112 5.641409 8.044256    0
#> SVM    2.164717 2.979377 3.736229 3.980410 4.723911 7.251589    0
#> CART   2.519930 3.428198 4.294325 4.372673 5.169167 7.856569    0
#> KNN    3.100655 3.690470 4.579753 4.687796 5.442480 8.369137    0
#>
#> Rsquared
#>             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> LM     0.4723270 0.6698297 0.7479362 0.7323877 0.8184113 0.8830690    0
```

```
#> GLM    0.4723270 0.6698297 0.7479362 0.7323877 0.8184113 0.8830690    0
#> GLMNET 0.4602065 0.6704187 0.7505348 0.7325322 0.8182185 0.8904240    0
#> SVM    0.5715369 0.7795105 0.8572078 0.8249574 0.8968088 0.9474025    0
#> CART   0.4803188 0.7100007 0.7991753 0.7837238 0.8504914 0.9358565    0
#> KNN    0.4295926 0.7514504 0.7924037 0.7746890 0.8521054 0.9166735    0
```

```
dotplot(results)
```



9.2 Feature selection

```
# remove correlated attributes
# find attributes that are highly correlated
set.seed(7)
cutoff <- 0.70
correlations <- cor(dataset[,1:13])
highlyCorrelated <- findCorrelation(correlations, cutoff=cutoff)

for (value in highlyCorrelated) {
    print(names(dataset)[value])
}
```

```
#> [1] "indus"
#> [1] "nox"
#> [1] "tax"
#> [1] "dis"
```

```
# create a new dataset without highly correlated features
datasetFeatures <- dataset[,-highlyCorrelated]
dim(datasetFeatures)
```

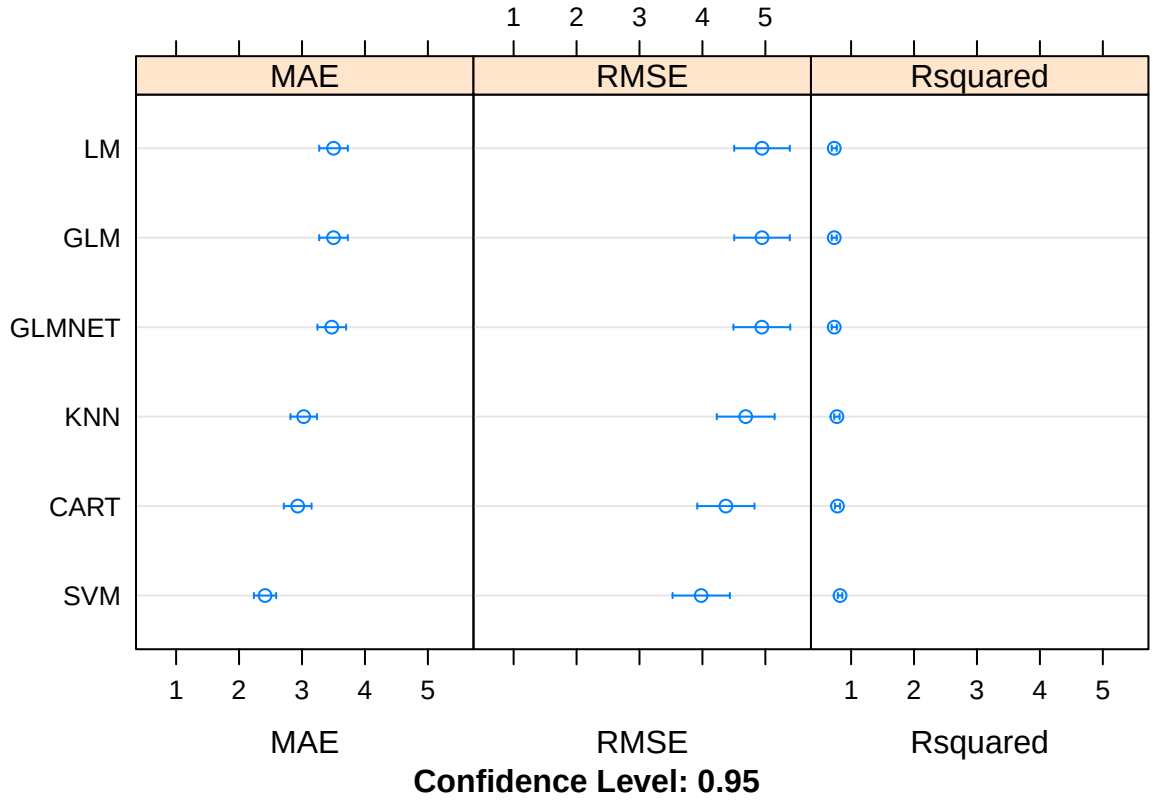
```
#> [1] 407  10
```

```
# Run algorithms using 10-fold cross-validation
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"

# LM
set.seed(7)
fit.lm <- train(medv~., data=dataset, method="lm",
                metric=metric, preProc=c("center", "scale"),
                trControl=trainControl)
# GLM
set.seed(7)
fit.glm <- train(medv~., data=dataset, method="glm",
                 metric=metric, preProc=c("center", "scale"),
                 trControl=trainControl)
# GLMNET
set.seed(7)
fit.glmnet <- train(medv~., data=dataset, method="glmnet",
                    metric=metric,
                    preProc=c("center", "scale"),
                    trControl=trainControl)
# SVM
set.seed(7)
fit.svm <- train(medv~., data=dataset, method="svmRadial",
                 metric=metric,
                 preProc=c("center", "scale"),
                 trControl=trainControl)
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
fit.cart <- train(medv~., data=dataset, method="rpart",
                  metric=metric, tuneGrid=grid,
                  preProc=c("center", "scale"),
                  trControl=trainControl)
# KNN
set.seed(7)
fit.knn <- train(medv~., data=dataset, method="knn",
                 metric=metric, preProc=c("center", "scale"),
                 trControl=trainControl)

# Compare algorithms
feature_results <- resamples(list(LM     = fit.lm,
                                  GLM    = fit.glm,
                                  GLMNET = fit.glmnet,
                                  SVM    = fit.svm,
                                  CART   = fit.cart,
                                  KNN    = fit.knn))
summary(feature_results)
```

```
#>
```

```
#> Call:
#> summary.resamples(object = feature_results)
#>
#> Models: LM, GLM, GLMNET, SVM, CART, KNN
#> Number of resamples: 30
#>
#> MAE
#>            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
#> LM     2.275798 3.105566 3.476184 3.501075 3.988069 4.806931    0
#> GLM    2.275798 3.105566 3.476184 3.501075 3.988069 4.806931    0
#> GLMNET 2.249747 3.054908 3.436463 3.472943 4.002405 4.819706    0
#> SVM    1.636837 2.083861 2.395264 2.412779 2.615126 3.597281    0
#> CART   2.043357 2.509889 2.875049 2.931782 3.294371 4.707060    0
#> KNN    2.036111 2.675617 3.023173 3.026927 3.346350 4.663144    0
#>
#> RMSE
#>            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
#> LM     2.945061 4.120871 4.590366 4.946754 5.622836 7.966565    0
#> GLM    2.945061 4.120871 4.590366 4.946754 5.622836 7.966565    0
#> GLMNET 2.900783 4.135902 4.580667 4.944112 5.641409 8.044256    0
#> SVM    2.164717 2.979377 3.736229 3.980410 4.723911 7.251589    0
#> CART   2.519930 3.428198 4.294325 4.372673 5.169167 7.856569    0
#> KNN    3.100655 3.690470 4.579753 4.687796 5.442480 8.369137    0
#>
#> Rsquared
#>             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> LM     0.4723270 0.6698297 0.7479362 0.7323877 0.8184113 0.8830690    0
#> GLM    0.4723270 0.6698297 0.7479362 0.7323877 0.8184113 0.8830690    0
#> GLMNET 0.4602065 0.6704187 0.7505348 0.7325322 0.8182185 0.8904240    0
#> SVM    0.5715369 0.7795105 0.8572078 0.8249574 0.8968088 0.9474025    0
#> CART   0.4803188 0.7100007 0.7991753 0.7837238 0.8504914 0.9358565    0
#> KNN    0.4295926 0.7514504 0.7924037 0.7746890 0.8521054 0.9166735    0
```

```
dotplot(feature_results)
```



Comparing the results, we can see that this has made the RMSE worse for the linear and the nonlinear algorithms. The correlated attributes we removed are contributing to the accuracy of the models.

9.3 Evaluate Algorithms: Box-Cox Transform

We know that some of the attributes have a skew and others perhaps have an exponential distribution. One option would be to explore squaring and log transforms respectively (you could try this!). Another approach would be to use a power transform and let it figure out the amount to correct each attribute. One example is the `Box-Cox` power transform. Let's try using this transform to rescale the original data and evaluate the effect on the same 6 algorithms. We will also leave in the centering and scaling for the benefit of the instance-based methods.

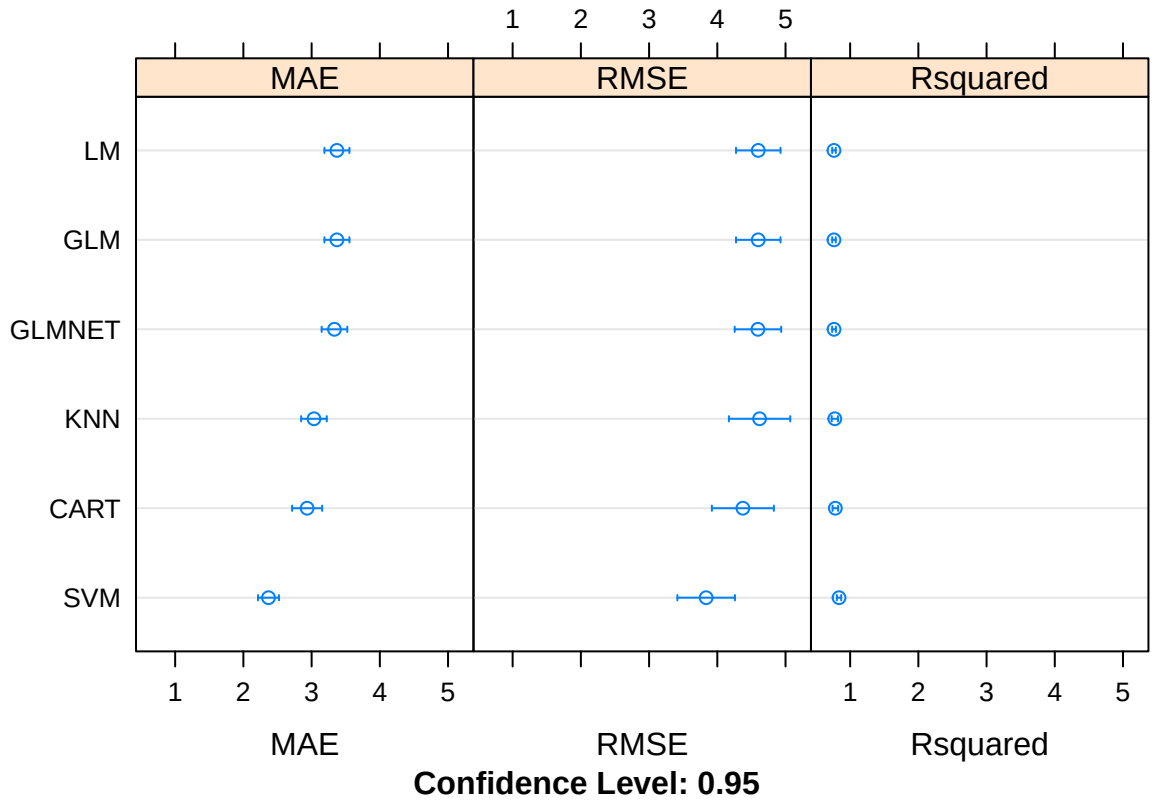
```
# Run algorithms using 10-fold cross-validation
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"

# lm
set.seed(7)
fit.lm <- train(medv~., data=dataset, method="lm", metric=metric,
                preProc=c("center", "scale", "BoxCox"),
                trControl=trainControl)
# GLM
set.seed(7)
fit.glm <- train(medv~., data=dataset, method="glm", metric=metric,
                 preProc=c("center", "scale", "BoxCox"),
                 trControl=trainControl)
# GLMNET
```

```
set.seed(7)
fit.glmnet <- train(medv~., data=dataset, method="glmnet", metric=metric,
                    preProc=c("center", "scale", "BoxCox"),
                    trControl=trainControl)
# SVM
set.seed(7)
fit.svm <- train(medv~., data=dataset, method="svmRadial", metric=metric,
                 preProc=c("center", "scale", "BoxCox"),
                 trControl=trainControl)
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
fit.cart <- train(medv~., data=dataset, method="rpart", metric=metric,
                  tuneGrid=grid,
                  preProc=c("center", "scale", "BoxCox"),
                  trControl=trainControl)
# KNN
set.seed(7)
fit.knn <- train(medv~., data=dataset, method="knn", metric=metric,
                 preProc=c("center", "scale", "BoxCox"),
                 trControl=trainControl)

# Compare algorithms
transformResults <- resamples(list(LM     = fit.lm,
                                   GLM    = fit.glm,
                                   GLMNET = fit.glmnet,
                                   SVM    = fit.svm,
                                   CART   = fit.cart,
                                   KNN    = fit.knn))
summary(transformResults)
```

```
#>
#> Call:
#> summary.resamples(object = transformResults)
#>
#> Models: LM, GLM, GLMNET, SVM, CART, KNN
#> Number of resamples: 30
#>
#> MAE
#>            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
#> LM     2.520337 2.963410 3.385355 3.371279 3.611259 4.311583    0
#> GLM    2.520337 2.963410 3.385355 3.371279 3.611259 4.311583    0
#> GLMNET 2.505158 2.922326 3.328423 3.334650 3.596593 4.377119    0
#> SVM    1.616495 2.143104 2.342498 2.368048 2.642978 3.443715    0
#> CART   2.043357 2.509889 2.875049 2.934449 3.294371 4.707060    0
#> KNN    2.193333 2.641599 3.140515 3.034677 3.300678 4.373442    0
#>
#> RMSE
#>            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
#> LM     3.327180 3.936074 4.441338 4.600223 5.183245 6.733523    0
#> GLM    3.327180 3.936074 4.441338 4.600223 5.183245 6.733523    0
#> GLMNET 3.373948 3.925110 4.431670 4.595578 5.186260 6.877090    0
#> SVM    2.301160 2.881686 3.601308 3.836561 4.498855 7.014149    0
#> CART   2.563901 3.428198 4.294325 4.374859 5.169167 7.856569    0
```

```
#> KNN    3.172856 3.757953 4.464893 4.619555 5.116853 8.451237    0
#>
#> Rsquared
#>              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> LM     0.5972973 0.7270836 0.7727291 0.7633673 0.8114439 0.8901553    0
#> GLM    0.5972973 0.7270836 0.7727291 0.7633673 0.8114439 0.8901553    0
#> GLMNET 0.5928990 0.7276325 0.7765796 0.7641157 0.8171126 0.8912449    0
#> SVM    0.6000736 0.7997356 0.8603609 0.8374185 0.8892423 0.9533119    0
#> CART   0.4803188 0.7100007 0.7991753 0.7834822 0.8504914 0.9333419    0
#> KNN    0.4128347 0.7518341 0.8064478 0.7754822 0.8439104 0.9195463    0
```

```
dotplot(transformResults)
```



9.4 Tune SVM

```
print(fit.svm)
```

```
#> Support Vector Machines with Radial Basis Function Kernel
#>
#> 407 samples
#>  13 predictor
#>
#> Pre-processing: centered (13), scaled (13), Box-Cox transformation (11)
#> Resampling: Cross-Validated (10 fold, repeated 3 times)
#> Summary of sample sizes: 366, 367, 366, 367, 365, 367, ...
#> Resampling results across tuning parameters:
#>
```

```
#>   C     RMSE      Rsquared   MAE
#>   0.25  4.692364  0.7795827  2.776964
#>   0.50  4.251248  0.8083441  2.524725
#>   1.00  3.836561  0.8374185  2.368048
#>
#> Tuning parameter 'sigma' was held constant at a value of 0.08878349
#> RMSE was used to select the optimal model using the smallest value.
#> The final values used for the model were sigma = 0.08878349 and C = 1.
```

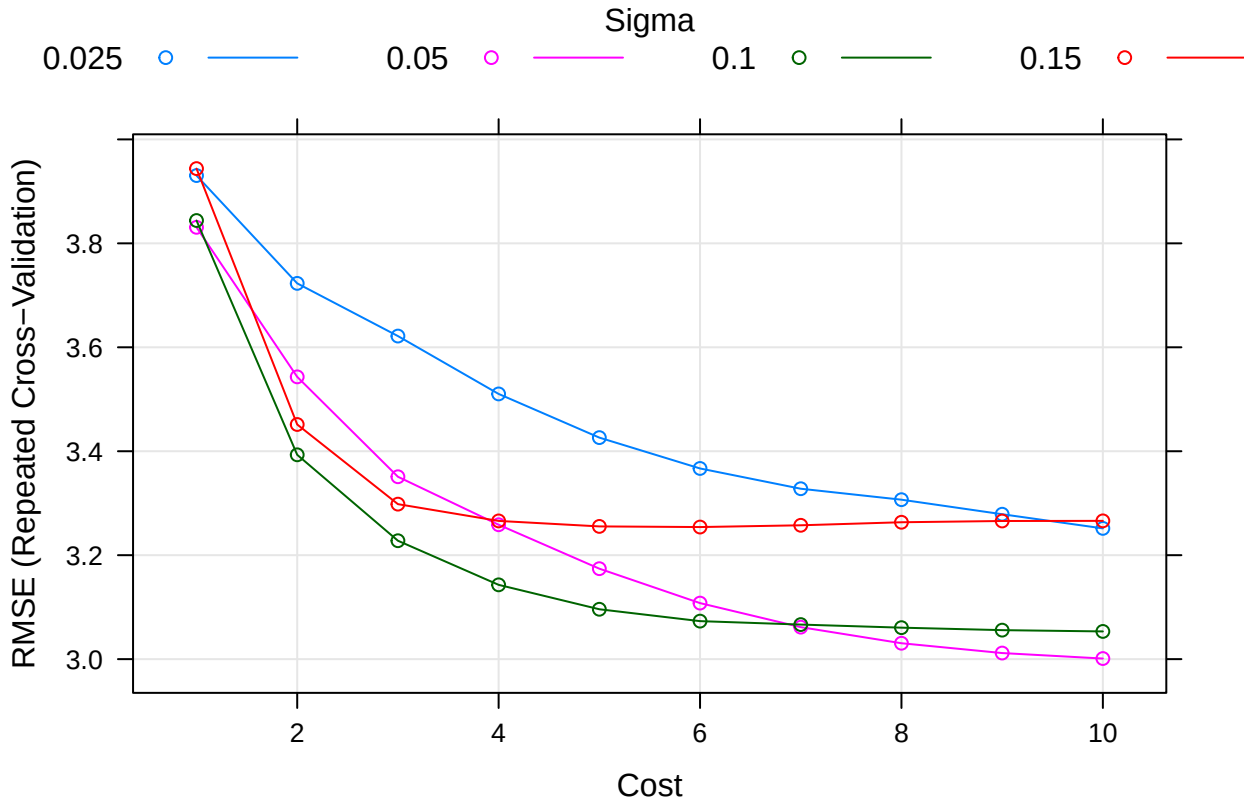
Let's design a grid search around a C value of 1. We might see a small trend of decreasing RMSE with increasing C, so let's try all integer C values between 1 and 10. Another parameter that caret let us tune is the sigma parameter. This is a smoothing parameter. Good sigma values often start around 0.1, so we will try numbers before and after.

```
# tune SVM sigma and C parametres
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
set.seed(7)

grid <- expand.grid(.sigma = c(0.025, 0.05, 0.1, 0.15),
                    .C = seq(1, 10, by=1))

fit.svm <- train(medv~., data=dataset, method="svmRadial", metric=metric,
                 tuneGrid=grid,
                 preProc=c("BoxCox"), trControl=trainControl)
print(fit.svm)
```

```
#> Support Vector Machines with Radial Basis Function Kernel
#>
#> 407 samples
#>  13 predictor
#>
#> Pre-processing: Box-Cox transformation (11)
#> Resampling: Cross-Validated (10 fold, repeated 3 times)
#> Summary of sample sizes: 366, 367, 366, 367, 365, 367, ...
#> Resampling results across tuning parameters:
#>
#>   sigma  C   RMSE      Rsquared   MAE
#>   0.025   1  3.930458  0.8276275  2.439609
#>   0.025   2  3.723009  0.8409733  2.328927
#>   0.025   3  3.621614  0.8482064  2.280724
#>   0.025   4  3.510280  0.8565961  2.231254
#>   0.025   5  3.426241  0.8629571  2.203999
#>   0.025   6  3.366889  0.8675244  2.184939
#>   0.025   7  3.327847  0.8706149  2.173270
#>   0.025   8  3.306757  0.8721949  2.165520
#>   0.025   9  3.278829  0.8742127  2.155252
#>   0.025  10  3.251596  0.8762207  2.144621
#>   0.050   1  3.830832  0.8363320  2.356105
#>   0.050   2  3.543118  0.8554896  2.254228
#>   0.050   3  3.350911  0.8689186  2.173351
#>   0.050   4  3.258575  0.8755807  2.134096
#>   0.050   5  3.174041  0.8815445  2.092097
#>   0.050   6  3.107810  0.8861029  2.063514
#>   0.050   7  3.061503  0.8892309  2.049936
```

```
#>   0.050   8  3.030597  0.8913244  2.042967
#>   0.050   9  3.011698  0.8926385  2.039596
#>   0.050  10  3.001111  0.8933696  2.039105
#>   0.100   1  3.843944  0.8374210  2.369040
#>   0.100   2  3.393117  0.8671391  2.176772
#>   0.100   3  3.227843  0.8775505  2.104811
#>   0.100   4  3.142971  0.8836300  2.081783
#>   0.100   5  3.095937  0.8869367  2.066293
#>   0.100   6  3.073112  0.8885969  2.056102
#>   0.100   7  3.066507  0.8891503  2.054790
#>   0.100   8  3.060573  0.8896855  2.054430
#>   0.100   9  3.055764  0.8900990  2.056769
#>   0.100  10  3.053300  0.8903036  2.061258
#>   0.150   1  3.943847  0.8317027  2.387425
#>   0.150   2  3.451454  0.8626185  2.204039
#>   0.150   3  3.298200  0.8729186  2.143969
#>   0.150   4  3.265943  0.8759410  2.141796
#>   0.150   5  3.255373  0.8766262  2.142137
#>   0.150   6  3.254155  0.8767197  2.151371
#>   0.150   7  3.257504  0.8764573  2.164027
#>   0.150   8  3.263358  0.8759398  2.176578
#>   0.150   9  3.265779  0.8756556  2.188381
#>   0.150  10  3.266036  0.8754983  2.197180
#>
#> RMSE was used to select the optimal model using the smallest value.
#> The final values used for the model were sigma = 0.05 and C = 10.
```

```
plot(fit.svm)
```



9.5 Ensembling

We can try some ensemble methods on the problem and see if we can get a further decrease in our RMSE.

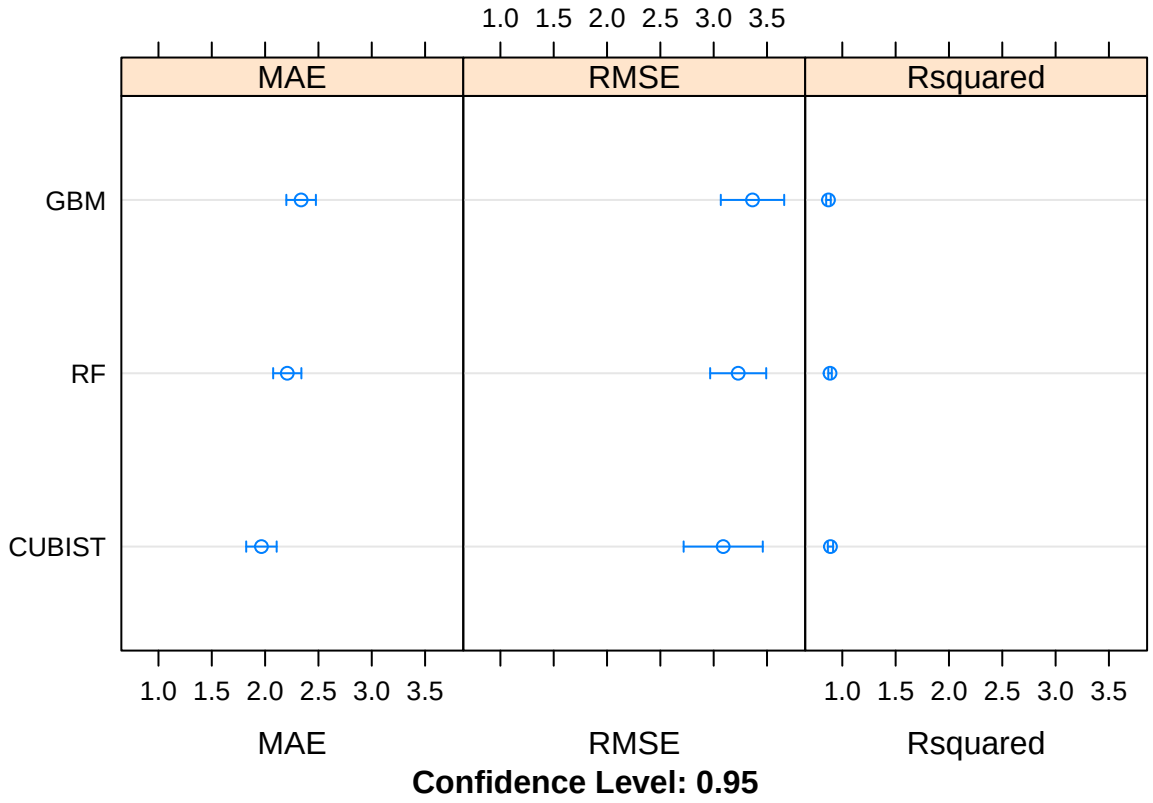
- Random Forest, bagging (RF).
- Gradient Boosting Machines (GBM).
- Cubist, boosting (CUBIST).

```
# try ensembles
seed <- 7
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"

# Random Forest
set.seed(seed)
fit.rf <- train(medv~., data=dataset, method="rf", metric=metric,
                preProc=c("BoxCox"),
                trControl=trainControl)

# Stochastic Gradient Boosting
set.seed(seed)
fit.gbm <- train(medv~., data=dataset, method="gbm", metric=metric,
                 preProc=c("BoxCox"),
                 trControl=trainControl, verbose=FALSE)
# Cubist
set.seed(seed)
fit.cubist <- train(medv~., data=dataset, method="cubist", metric=metric,
                    preProc=c("BoxCox"), trControl=trainControl)
# Compare algorithms
ensembleResults <- resamples(list(RF  = fit.rf,
                                  GBM = fit.gbm,
                                  CUBIST = fit.cubist))
summary(ensembleResults)
```

```
#>
#> Call:
#> summary.resamples(object = ensembleResults)
#>
#> Models: RF, GBM, CUBIST
#> Number of resamples: 30
#>
#> MAE
#>            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
#> RF     1.550247 1.995197 2.198848 2.207868 2.308026 3.102807    0
#> GBM    1.684321 2.055839 2.296176 2.337663 2.488155 3.073921    0
#> CUBIST 1.225568 1.631722 1.998192 1.965454 2.248008 2.652877    0
#>
#> RMSE
#>            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
#> RF     2.258527 2.722467 3.059156 3.229802 3.502506 5.075363    0
#> GBM    2.296971 2.684094 3.146213 3.363994 4.213876 4.949682    0
#> CUBIST 1.624740 2.323381 2.765466 3.089250 3.847823 5.259804    0
#>
#> Rsquared
#>             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
```

```
#> RF     0.7885328 0.8536977 0.8927022 0.8849676 0.9186262 0.9490067    0
#> GBM    0.7395492 0.8346060 0.8798001 0.8702191 0.9127787 0.9603358    0
#> CUBIST 0.6764145 0.8541497 0.9083048 0.8888146 0.9361016 0.9692201    0
```

```
dotplot(ensembleResults)
```



Let's dive deeper into Cubist and see if we can tune it further and get more skill out of it. Cubist has two parameters that are tunable with caret: committees which is the number of boosting operations and neighbors which is used during prediction and is the number of instances used to correct the rule-based prediction (although the documentation is perhaps a little ambiguous on this).

```
# look at parameters used for Cubist
print(fit.cubist)
```

```
#> Cubist
#>
#> 407 samples
#>  13 predictor
#>
#> Pre-processing: Box-Cox transformation (11)
#> Resampling: Cross-Validated (10 fold, repeated 3 times)
#> Summary of sample sizes: 366, 367, 366, 367, 365, 367, ...
#> Resampling results across tuning parameters:
#>
#>   committees  neighbors  RMSE      Rsquared   MAE
#>    1          0          3.753432  0.8372372  2.468735
#>    1          5          3.396743  0.8654549  2.207527
#>    1          9          3.470867  0.8618118  2.237505
#>   10          0          3.372730  0.8700258  2.233735
#>   10          5          3.104315  0.8879016  1.978900
```

```
#>   10          9          3.164513  0.8841651  2.012710
#>   20          0          3.373670  0.8694293  2.228746
#>   20          5          3.089250  0.8888146  1.965454
#>   20          9          3.148450  0.8854011  1.992379
#>
#> RMSE was used to select the optimal model using the smallest value.
#> The final values used for the model were committees = 20 and neighbors = 5.
```

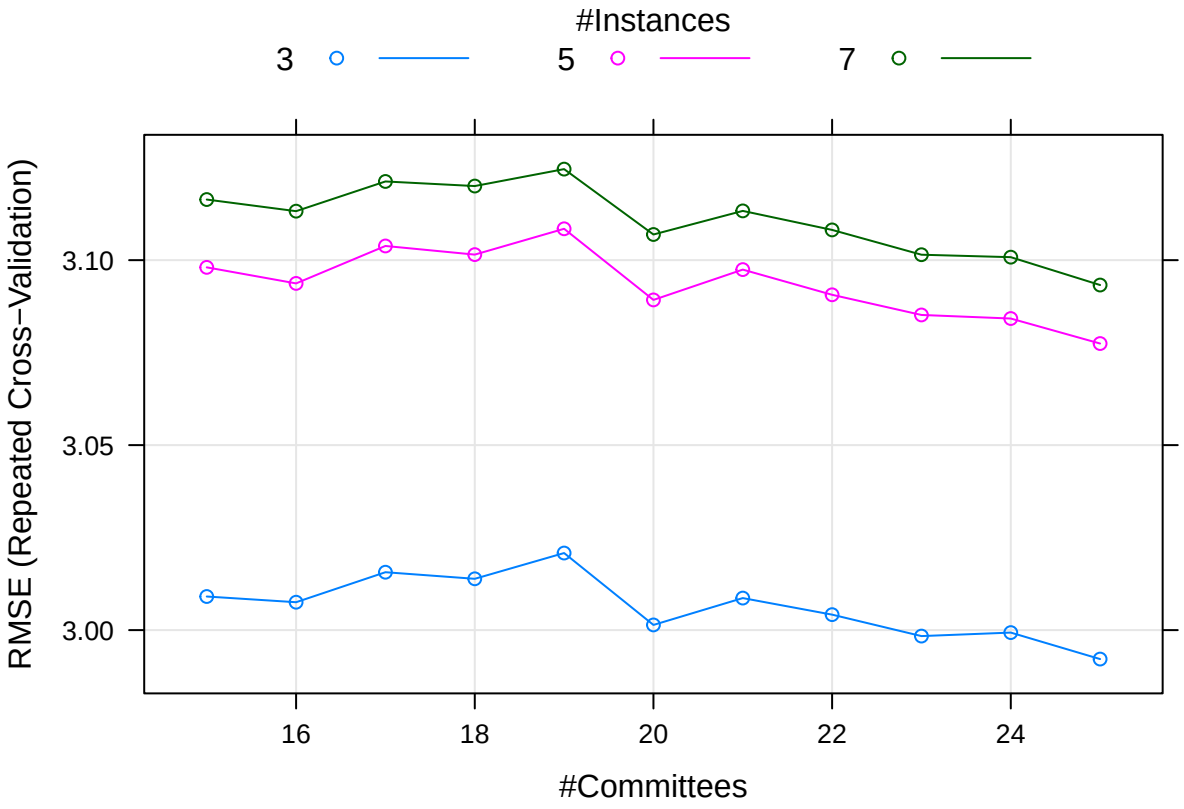
Let's use a grid search to tune around those values. We'll try all committees between 15 and 25 and spot-check a neighbors value above and below 5.

```
library(Cubist)
# Tune the Cubist algorithm
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
set.seed(7)
grid <- expand.grid(.committees = seq(15, 25, by=1),
                    .neighbors = c(3, 5, 7))

tune.cubist <- train(medv~., data=dataset, method = "cubist", metric=metric,
                     preProc=c("BoxCox"),
                     tuneGrid=grid, trControl=trainControl)
print(tune.cubist)
```

```
#> Cubist
#>
#> 407 samples
#>  13 predictor
#>
#> Pre-processing: Box-Cox transformation (11)
#> Resampling: Cross-Validated (10 fold, repeated 3 times)
#> Summary of sample sizes: 366, 367, 366, 367, 365, 367, ...
#> Resampling results across tuning parameters:
#>
#>   committees  neighbors  RMSE      Rsquared   MAE
#>   15          3          3.009082  0.8941074  1.919529
#>   15          5          3.098053  0.8884253  1.976015
#>   15          7          3.116386  0.8874419  1.979539
#>   16          3          3.007537  0.8944762  1.919600
#>   16          5          3.093709  0.8889310  1.975111
#>   16          7          3.113242  0.8878123  1.978636
#>   17          3          3.015645  0.8937711  1.923880
#>   17          5          3.103834  0.8880055  1.977733
#>   17          7          3.121269  0.8871594  1.982526
#>   18          3          3.013869  0.8937863  1.921819
#>   18          5          3.101501  0.8880896  1.973930
#>   18          7          3.120030  0.8871029  1.978973
#>   19          3          3.020826  0.8932599  1.921864
#>   19          5          3.108476  0.8875403  1.974250
#>   19          7          3.124609  0.8868355  1.978953
#>   20          3          3.001396  0.8944892  1.911889
#>   20          5          3.089250  0.8888146  1.965454
#>   20          7          3.106933  0.8879802  1.969943
#>   21          3          3.008654  0.8939703  1.913532
#>   21          5          3.097445  0.8881274  1.968988
```

```
#>    21          7          3.113314  0.8874920  1.973820
#>    22          3          3.004181  0.8945197  1.913736
#>    22          5          3.090638  0.8887730  1.966745
#>    22          7          3.108176  0.8879460  1.971023
#>    23          3          2.998382  0.8948426  1.910446
#>    23          5          3.085185  0.8891198  1.964406
#>    23          7          3.101469  0.8883363  1.969268
#>    24          3          2.999325  0.8948105  1.912269
#>    24          5          3.084216  0.8891824  1.964024
#>    24          7          3.100808  0.8883295  1.968827
#>    25          3          2.992166  0.8951772  1.907429
#>    25          5          3.077461  0.8896248  1.959972
#>    25          7          3.093274  0.8888028  1.963996
#>
#> RMSE was used to select the optimal model using the smallest value.
#> The final values used for the model were committees = 25 and neighbors = 3.
```

```
plot(tune.cubist)
```



We can see that we have achieved a more accurate model again with an RMSE of 2.822 using committees = 18 and neighbors = 3.

It looks like the results for the Cubist algorithm are the most accurate. Let's finalize it by creating a new standalone Cubist model with the parameters above trained using the whole dataset. We must also use the Box-Cox power transform.

9.6 Finalize the model

```
# prepare the data transform using training data
set.seed(7)
x <- dataset[,1:13]
y <- dataset[,14]

# transform
preprocessParams <- preProcess(x, method=c("BoxCox"))
transX <- predict(preprocessParams, x)

# train the final model
finalModel <- cubist(x = transX, y=y, committees=18)
summary(finalModel)
```

```
#>
#> Call:
#> cubist.default(x = transX, y = y, committees = 18)
#>
#>
#> Cubist [Release 2.07 GPL Edition]  Sun May 12 23:16:35 2019
#> -------------------------------
#>
#>     Target attribute `outcome'
#>
#> Read 407 cases (14 attributes) from undefined.data
#>
#> Model 1:
#>
#>   Rule 1/1: [84 cases, mean 13.75, range 5 to 25, est err 1.85]
#>
#>     if
#>  nox > -0.497006
#>     then
#>  outcome = 28.8 - 3.48 lstat + 8.2 nox - 1.41 crim + 5.3 dis + 3e-05 b
#>
#>   Rule 1/2: [166 cases, mean 19.48, range 7 to 31, est err 2.05]
#>
#>     if
#>  nox <= -0.497006
#>  lstat > 2.858393
#>     then
#>  outcome = 158.68 - 2.35 lstat + 1.8 rad - 75 tax - 2.6 dis
#>            - 0.037 ptratio + 10 rm - 0.0075 age + 2.3e-05 b + 1.6 chas
#>
#>   Rule 1/3: [107 cases, mean 25.59, range 18.6 to 37.2, est err 1.86]
#>
#>     if
#>  rm <= 1.954587
#>  dis > 0.5868974
#>  lstat <= 2.858393
#>     then
#>  outcome = 17.94 + 49.3 rm - 4.3 dis - 1.71 lstat - 0.014 age
```

```
#>            - 0.46 indus - 0.023 ptratio - 36 tax - 0.5 nox + 0.1 rad
#>
#>   Rule 1/4: [4 cases, mean 31.15, range 23.3 to 50, est err 1.69]
#>
#>     if
#>  dis <= 0.5868974
#>  b <= 66469.73
#>  lstat <= 2.858393
#>     then
#>  outcome = 71.04 - 60.5 dis - 4.99 lstat
#>
#>   Rule 1/5: [9 cases, mean 38.21, range 17.8 to 50, est err 2.17]
#>
#>     if
#>  rm > 1.954587
#>  dis > 0.5868974
#>  tax > 1.894297
#>     then
#>  outcome = 2234.56 - 1152 tax - 11.9 dis - 0.9 lstat + 7.6 rm
#>            - 0.012 ptratio
#>
#>   Rule 1/6: [38 cases, mean 40.13, range 31 to 50, est err 2.55]
#>
#>     if
#>  rm > 1.954587
#>  tax <= 1.894297
#>     then
#>  outcome = 391.64 - 0.000497 b + 80.8 rm - 246 tax - 0.0294 age
#>            - 0.047 ptratio - 1.52 lstat - 2.4 dis
#>
#>   Rule 1/7: [4 cases, mean 50.00, range 50 to 50, est err 0.00]
#>
#>     if
#>  dis <= 0.5868974
#>  b > 66469.73
#>  lstat <= 2.858393
#>     then
#>  outcome = 50
#>
#> Model 2:
#>
#>   Rule 2/1: [10 cases, mean 7.92, range 5 to 12.3, est err 3.49]
#>
#>     if
#>  nox > -0.4727541
#>  dis <= 0.462979
#>  ptratio > 149.145
#>     then
#>  outcome = 244.69 + 544.4 nox - 0.3 dis - 0.16 lstat
#>
#>   Rule 2/2: [9 cases, mean 12.56, range 10.2 to 15, est err 3.06]
#>
#>     if
#>  nox <= -0.4727541
```

```
#>  dis <= 0.462979
#>  b > 67032.41
#>  lstat > 1.931448
#>     then
#>  outcome = 31.3 - 0.48 lstat - 0.3 dis - 8 tax + 1.4 rm - 0.004 ptratio
#>            - 0.06 indus
#>
#>   Rule 2/3: [145 cases, mean 18.41, range 7 to 35.2, est err 2.38]
#>
#>     if
#>  dis > 0.462979
#>  ptratio > 149.145
#>  b <= 77263.3
#>  lstat > 1.931448
#>     then
#>  outcome = 42.95 - 4.01 lstat - 0.042 ptratio + 5.7e-05 b + 5 rm
#>            - 0.0055 age - 0.4 dis - 7 tax - 0.06 indus
#>
#>   Rule 2/4: [116 cases, mean 22.59, range 8.3 to 43.8, est err 2.25]
#>
#>     if
#>  dis > 0.462979
#>  tax > 1.857866
#>  ptratio > 149.145
#>  b > 77263.3
#>     then
#>  outcome = 120.83 - 0.001783 b + 36.4 rm - 0.026 age - 0.52 lstat
#>            - 0.5 dis - 0.11 indus - 10 tax
#>
#>   Rule 2/5: [74 cases, mean 23.58, range 11.8 to 50, est err 2.37]
#>
#>     if
#>  ptratio <= 149.145
#>  lstat > 1.931448
#>     then
#>  outcome = -65.34 + 53.6 rm - 0.112 ptratio + 7.7e-05 b - 0.32 lstat
#>            - 0.3 dis
#>
#>   Rule 2/6: [11 cases, mean 23.62, range 6.3 to 50, est err 7.87]
#>
#>     if
#>  dis <= 0.462979
#>  ptratio > 149.145
#>  b <= 67032.41
#>     then
#>  outcome = 72.68 - 117.8 dis - 37.4 nox - 4.66 lstat - 0.000222 b
#>
#>   Rule 2/7: [11 cases, mean 24.03, range 15.7 to 39.8, est err 5.34]
#>
#>     if
#>  tax <= 1.857866
#>  lstat > 1.931448
#>     then
#>  outcome = 130.83 - 0.477 ptratio - 4.17 lstat - 0.0344 age - 0.2 dis
```

```
#>
#>   Rule 2/8: [9 cases, mean 28.52, range 22.5 to 50, est err 7.28]
#>
#>     if
#>  rm <= 1.895669
#>  lstat <= 1.931448
#>     then
#>  outcome = 77.55 - 33.16 lstat + 3.4 rm - 0.7 dis
#>
#>   Rule 2/9: [61 cases, mean 36.05, range 21 to 50, est err 3.01]
#>
#>     if
#>  rm > 1.895669
#>  tax <= 1.894297
#>     then
#>  outcome = 398.49 + 98.7 rm - 288 tax - 0.0433 age - 7.5 dis - 0.28 lstat
#>
#>   Rule 2/10: [13 cases, mean 42.53, range 30.5 to 50, est err 2.91]
#>
#>     if
#>  tax > 1.894297
#>  lstat <= 1.931448
#>     then
#>  outcome = -2.05 + 0.032 age + 21.8 rm - 3.2 dis
#>
#> Model 3:
#>
#>   Rule 3/1: [68 cases, mean 13.12, range 5 to 25, est err 2.21]
#>
#>     if
#>  nox > -0.497006
#>  ptratio > 109.02
#>     then
#>  outcome = 51.26 + 41.9 nox - 2.06 crim - 4.11 lstat
#>
#>   Rule 3/2: [171 cases, mean 19.94, range 7 to 50, est err 2.60]
#>
#>     if
#>  nox <= -0.497006
#>  rm <= 1.831781
#>  lstat > 1.739722
#>     then
#>  outcome = 76.77 + 25.7 rm - 0.063 ptratio - 4.2 dis + 0.56 crim
#>            - 1.15 lstat - 0.01 age - 42 tax - 0.37 indus + 3e-05 b
#>            + 0.012 zn
#>
#>   Rule 3/3: [35 cases, mean 24.17, range 11.8 to 50, est err 3.79]
#>
#>     if
#>  ptratio <= 109.02
#>  lstat > 1.739722
#>     then
#>  outcome = 113.14 - 0.352 ptratio - 5.85 lstat - 1.3 dis + 5.3 rm
#>            - 1.5 nox - 24 tax + 0.0024 age
```

```
#>
#>   Rule 3/4: [118 cases, mean 27.40, range 16.1 to 50, est err 2.09]
#>
#>     if
#>  nox <= -0.497006
#>  rm > 1.831781
#>  lstat > 1.739722
#>     then
#>  outcome = 67.96 + 56.8 rm - 0.05 ptratio - 1.89 lstat - 71 tax - 1.3 dis
#>            + 0.24 crim - 0.16 indus - 0.0032 age + 1.3e-05 b + 0.005 zn
#>
#>   Rule 3/5: [25 cases, mean 38.41, range 24.8 to 50, est err 3.51]
#>
#>     if
#>  dis > 1.162901
#>  lstat <= 1.739722
#>     then
#>  outcome = 283.78 + 96.9 rm - 0.0574 age - 219 tax - 7.6 dis
#>            - 0.067 ptratio
#>
#>   Rule 3/6: [9 cases, mean 49.42, range 44.8 to 50, est err 3.35]
#>
#>     if
#>  dis <= 1.162901
#>  lstat <= 1.739722
#>     then
#>  outcome = 56.35 - 9 dis
#>
#> Model 4:
#>
#>   Rule 4/1: [90 cases, mean 13.70, range 5 to 27.9, est err 3.46]
#>
#>     if
#>  dis <= 0.7244036
#>  lstat > 2.858393
#>     then
#>  outcome = 203.21 - 19.6 dis - 7.53 lstat + 0.0679 age - 6.3 nox - 80 tax
#>            - 5.1e-05 b - 8 rm
#>
#>   Rule 4/2: [247 cases, mean 17.48, range 5 to 31, est err 2.91]
#>
#>     if
#>  lstat > 2.858393
#>     then
#>  outcome = -2.29 + 21.8 rm - 0.0239 age - 2.18 lstat - 0.034 ptratio
#>            + 5.2e-05 b + 2.5 nox - 0.1 crim - 0.3 dis
#>
#>   Rule 4/3: [41 cases, mean 20.49, range 14.4 to 29.6, est err 2.58]
#>
#>     if
#>  nox <= -1.04499
#>  lstat > 2.858393
#>     then
#>  outcome = 62.87 - 0.000266 b + 10.1 nox - 5.9 dis
```

```
#>
#>   Rule 4/4: [103 cases, mean 25.88, range 18.6 to 50, est err 2.66]
#>
#>     if
#>  rm <= 1.937158
#>  lstat <= 2.858393
#>     then
#>  outcome = -38.51 + 52.8 rm - 4.7 dis - 1.39 lstat - 0.012 age - 0.8 nox
#>            + 0.12 crim - 12 tax - 0.006 ptratio - 0.09 indus + 5e-06 b
#>
#>   Rule 4/5: [47 cases, mean 38.35, range 23.6 to 50, est err 3.20]
#>
#>     if
#>  rm > 1.937158
#>  tax <= 1.896025
#>     then
#>  outcome = 697.02 - 0.000827 b + 102.5 rm - 418 tax - 0.0401 age
#>            - 4.8 dis - 0.038 ptratio
#>
#>   Rule 4/6: [11 cases, mean 38.45, range 21.9 to 50, est err 6.32]
#>
#>     if
#>  rm > 1.937158
#>  dis > 0.5868974
#>  tax > 1.896025
#>  lstat <= 2.858393
#>     then
#>  outcome = -78.2 + 65 rm - 0.087 ptratio
#>
#>   Rule 4/7: [8 cases, mean 40.58, range 23.3 to 50, est err 22.46]
#>
#>     if
#>  dis <= 0.5868974
#>  lstat <= 2.858393
#>     then
#>  outcome = 58.89 - 104.3 dis + 11.72 lstat
#>
#> Model 5:
#>
#>   Rule 5/1: [84 cases, mean 13.75, range 5 to 25, est err 3.17]
#>
#>     if
#>  nox > -0.497006
#>     then
#>  outcome = 40.91 + 12.2 dis - 2.6 crim + 8.3 nox - 2.85 lstat - 10.2 rm
#>            + 3e-05 b
#>
#>   Rule 5/2: [12 cases, mean 15.95, range 13.2 to 23.7, est err 3.47]
#>
#>     if
#>  nox <= -0.497006
#>  lstat > 4.439301
#>     then
#>  outcome = 17.92
```

```
#>
#>   Rule 5/3: [156 cases, mean 19.81, range 10.2 to 29.6, est err 1.91]
#>
#>     if
#>  nox <= -0.497006
#>  rm <= 1.831301
#>  dis > 0.5626968
#>     then
#>  outcome = 139.74 + 23.4 rm - 79 tax + 1.8 rad - 0.041 ptratio - 2.8 dis
#>            - 1.24 lstat - 0.0069 age + 2.3e-05 b + 1.7 chas + 0.016 zn
#>
#>   Rule 5/4: [124 cases, mean 23.08, range 7.2 to 50, est err 3.41]
#>
#>     if
#>  rm > 1.831301
#>  lstat > 1.987397
#>     then
#>  outcome = 102.95 + 43.6 rm - 0.053 ptratio - 79 tax - 1.02 lstat
#>            - 1.1 dis + 0.5 rad - 0.9 nox + 0.0019 age + 0.008 zn
#>
#>   Rule 5/5: [10 cases, mean 26.73, range 7 to 50, est err 9.26]
#>
#>     if
#>  nox <= -0.497006
#>  rm <= 1.831301
#>  dis <= 0.5626968
#>  lstat <= 4.439301
#>     then
#>  outcome = 116.88 - 24.41 lstat - 3.1 dis
#>
#>   Rule 5/6: [62 cases, mean 36.86, range 21.9 to 50, est err 4.83]
#>
#>     if
#>  lstat <= 1.987397
#>     then
#>  outcome = -111.16 + 78.2 rm - 6.3 dis - 1.52 crim - 0.43 lstat
#>
#>   Rule 5/7: [13 cases, mean 43.86, range 21.9 to 50, est err 9.72]
#>
#>     if
#>  age > 229.474
#>  lstat <= 1.987397
#>     then
#>  outcome = -60.3 + 0.4787 age - 1.09 lstat - 1 dis - 1.2 nox - 14 tax
#>            - 0.006 ptratio + 1.6 rm + 0.2 rad
#>
#> Model 6:
#>
#>   Rule 6/1: [29 cases, mean 13.33, range 7 to 27.5, est err 4.31]
#>
#>     if
#>  b <= 16084.5
#>     then
#>  outcome = 19.31 + 0.000779 b - 0.38 lstat - 0.4 dis + 1 rm
```

```
#>            - 0.003 ptratio - 5 tax
#>
#>   Rule 6/2: [247 cases, mean 17.48, range 5 to 31, est err 2.54]
#>
#>     if
#>  lstat > 2.858393
#>     then
#>  outcome = 67.94 - 3.35 lstat - 0.68 crim - 0.0142 age + 3.3 nox
#>            - 0.015 ptratio + 2.6e-05 b + 0.4 rad - 17 tax
#>
#>   Rule 6/3: [9 cases, mean 21.33, range 15.7 to 29.6, est err 3.41]
#>
#>     if
#>  tax <= 1.857866
#>  lstat > 2.858393
#>     then
#>  outcome = 4.61 + 32.7 dis - 0.36 lstat - 0.003 ptratio + 0.9 rm - 5 tax
#>
#>   Rule 6/4: [114 cases, mean 28.38, range 18.6 to 50, est err 2.17]
#>
#>     if
#>  dis > 1.230868
#>  lstat <= 2.858393
#>     then
#>  outcome = -84.69 + 73.9 rm - 5.8 dis - 0.0285 age - 0.84 indus
#>            - 0.64 lstat + 0.13 crim - 0.7 nox - 0.004 ptratio - 6 tax
#>
#>   Rule 6/5: [29 cases, mean 33.81, range 20.6 to 50, est err 3.83]
#>
#>     if
#>  dis <= 1.230868
#>  b > 73935.01
#>  lstat <= 2.858393
#>     then
#>  outcome = -82.01 - 14.9 dis + 69.6 rm + 0.92 crim - 1.6 lstat - 3.2 nox
#>            - 0.48 indus - 0.0095 age
#>
#>   Rule 6/6: [12 cases, mean 39.08, range 21.9 to 50, est err 8.85]
#>
#>     if
#>  dis <= 0.6604174
#>  lstat <= 2.858393
#>     then
#>  outcome = -5.71 - 62.2 dis + 0.001034 b - 0.21 lstat
#>
#>   Rule 6/7: [8 cases, mean 41.85, range 25 to 50, est err 11.11]
#>
#>     if
#>  dis > 0.6604174
#>  dis <= 1.230868
#>  b <= 73935.01
#>  lstat <= 2.858393
#>     then
#>  outcome = -98.17 + 74.9 rm - 11.2 dis + 0.000142 b + 0.68 crim
```

```
#>            - 1.17 lstat - 2.3 nox - 0.35 indus - 0.0069 age
#>
#> Model 7:
#>
#>   Rule 7/1: [84 cases, mean 13.75, range 5 to 25, est err 2.49]
#>
#>     if
#>  nox > -0.497006
#>     then
#>  outcome = 25.08 + 11 dis - 2.31 crim - 3.33 lstat + 6.9 nox + 2.8e-05 b
#>
#>   Rule 7/2: [59 cases, mean 14.73, range 5 to 50, est err 7.01]
#>
#>     if
#>  dis <= 0.5626968
#>  lstat > 1.931448
#>     then
#>  outcome = 16.83 + 0.06 crim + 1.1 rm - 0.003 ptratio - 0.2 dis
#>
#>   Rule 7/3: [236 cases, mean 21.99, range 10.2 to 50, est err 2.22]
#>
#>     if
#>  nox <= -0.497006
#>  dis > 0.5626968
#>  tax > 1.865769
#>  lstat > 1.931448
#>     then
#>  outcome = -18.15 + 35.4 rm - 0.0248 age - 0.045 ptratio + 0.58 crim
#>            - 2.2 dis + 5.8e-05 b + 1.2 lstat - 1 nox - 9 tax
#>
#>   Rule 7/4: [10 cases, mean 25.41, range 7 to 50, est err 12.49]
#>
#>     if
#>  nox <= -0.497006
#>  dis <= 0.5626968
#>  b <= 67032.41
#>     then
#>  outcome = 81.16 - 115.7 dis - 0.000217 b - 0.49 lstat
#>
#>   Rule 7/5: [58 cases, mean 25.55, range 16.5 to 50, est err 2.33]
#>
#>     if
#>  nox <= -0.497006
#>  ptratio <= 149.145
#>  lstat > 1.931448
#>     then
#>  outcome = -22.5 + 47.6 rm - 0.089 ptratio + 3.1 nox - 0.66 lstat
#>            - 0.6 dis - 12 tax + 0.0019 age + 0.08 crim
#>
#>   Rule 7/6: [20 cases, mean 29.26, range 15.7 to 50, est err 4.96]
#>
#>     if
#>  tax <= 1.865769
#>  ptratio > 149.145
```

```
#>     then
#>  outcome = 53.7 - 0.394 ptratio + 12 dis + 17.6 nox + 2.64 crim + 33.6 rm
#>            - 2.74 lstat
#>
#>   Rule 7/7: [6 cases, mean 29.48, range 22.5 to 50, est err 6.03]
#>
#>     if
#>  rm <= 1.882057
#>  lstat <= 1.931448
#>     then
#>  outcome = 220.69 - 106 rm
#>
#>   Rule 7/8: [37 cases, mean 38.17, range 22.8 to 50, est err 3.89]
#>
#>     if
#>  rm > 1.882057
#>  tax <= 1.894297
#>  lstat <= 1.931448
#>     then
#>  outcome = 767.63 + 111.8 rm - 506 tax - 0.0255 age - 0.33 lstat
#>            - 0.5 nox - 0.3 dis - 0.003 ptratio
#>
#>   Rule 7/9: [12 cases, mean 41.91, range 30.5 to 50, est err 2.56]
#>
#>     if
#>  rm > 1.882057
#>  tax > 1.894297
#>  lstat <= 1.931448
#>     then
#>  outcome = 46.61 + 0.0476 age + 10.8 rm - 1.11 lstat - 1.6 nox - 1 dis
#>            - 0.009 ptratio - 16 tax + 0.1 crim
#>
#> Model 8:
#>
#>   Rule 8/1: [39 cases, mean 12.55, range 5 to 27.9, est err 3.96]
#>
#>     if
#>  crim > 2.382708
#>     then
#>  outcome = 54.82 - 5.35 crim - 6.25 lstat
#>
#>   Rule 8/2: [141 cases, mean 17.25, range 6.3 to 31, est err 2.50]
#>
#>     if
#>  crim <= 2.382708
#>  nox > -0.8796992
#>  lstat > 2.858393
#>     then
#>  outcome = 60.02 - 5.63 lstat + 7.7 nox - 0.97 crim + 1.4 dis
#>            - 0.0078 age + 3.2 rm + 1.7e-05 b - 0.008 ptratio - 12 tax
#>
#>   Rule 8/3: [67 cases, mean 20.84, range 14.4 to 29.6, est err 2.09]
#>
#>     if
```

```
#>  nox <= -0.8796992
#>  lstat > 2.858393
#>     then
#>  outcome = 48.44 + 8.1 nox - 1.47 lstat + 2.6 rm - 0.5 dis
#>            - 0.007 ptratio - 0.11 crim - 9 tax
#>
#>   Rule 8/4: [160 cases, mean 30.52, range 18.6 to 50, est err 3.26]
#>
#>     if
#>  lstat <= 2.858393
#>     then
#>  outcome = -31.85 + 60.1 rm - 6.3 dis - 2.81 lstat - 0.0153 age
#>            + 0.27 crim - 18 tax + 0.015 zn - 0.009 ptratio
#>
#>   Rule 8/5: [8 cases, mean 40.58, range 23.3 to 50, est err 9.14]
#>
#>     if
#>  dis <= 0.5868974
#>  lstat <= 2.858393
#>     then
#>  outcome = 75.2 - 87.2 dis
#>
#> Model 9:
#>
#>   Rule 9/1: [83 cases, mean 13.65, range 5 to 25, est err 2.25]
#>
#>     if
#>  nox > -0.497006
#>  lstat > 2.150069
#>     then
#>  outcome = 22.54 + 11.7 dis + 10.8 nox - 1.57 crim + 6.3e-05 b
#>            - 0.15 lstat - 6 tax
#>
#>   Rule 9/2: [29 cases, mean 19.16, range 7 to 50, est err 7.45]
#>
#>     if
#>  nox <= -0.497006
#>  rm <= 1.85661
#>  dis <= 0.7285143
#>  lstat > 2.150069
#>     then
#>  outcome = 387.41 - 44.9 dis + 2.15 crim - 4.7 lstat - 199 tax + 32 rm
#>
#>   Rule 9/3: [167 cases, mean 20.49, range 12.7 to 29.6, est err 2.00]
#>
#>     if
#>  nox <= -0.497006
#>  rm <= 1.85661
#>  dis > 0.7285143
#>     then
#>  outcome = 132.11 + 25.2 rm - 0.057 ptratio - 0.0213 age - 3.3 dis
#>            - 75 tax + 1.4 rad + 0.49 crim + 0.21 indus + 0.44 lstat
#>
#>   Rule 9/4: [60 cases, mean 26.86, range 16.1 to 50, est err 2.12]
```

```
#>
#>     if
#>  nox <= -0.497006
#>  rm > 1.85661
#>  lstat > 2.150069
#>     then
#>  outcome = 117.46 + 69.1 rm - 113 tax - 0.053 ptratio - 0.7 dis
#>            - 0.0034 age + 0.14 crim + 0.3 rad
#>
#>   Rule 9/5: [76 cases, mean 35.06, range 20.6 to 50, est err 5.99]
#>
#>     if
#>  lstat <= 2.150069
#>     then
#>  outcome = 147.59 + 35.6 rm - 93 tax - 0.046 ptratio - 1.41 lstat
#>            + 2.2 nox - 1.4 dis + 0.033 zn + 0.5 rad + 0.0031 age
#>            + 0.6 chas
#>
#>   Rule 9/6: [26 cases, mean 38.57, range 23.6 to 50, est err 4.54]
#>
#>     if
#>  nox <= -0.6286645
#>  rm > 1.914272
#>  tax > 1.877141
#>  lstat <= 2.150069
#>     then
#>  outcome = -1249.52 + 111.9 rm + 580 tax + 20.9 nox - 0.0495 age
#>            - 1.29 lstat - 0.014 ptratio + 0.005 zn
#>
#>   Rule 9/7: [19 cases, mean 38.87, range 28.5 to 50, est err 4.77]
#>
#>     if
#>  nox <= -0.6286645
#>  rm > 1.914272
#>  tax <= 1.877141
#>  lstat <= 2.150069
#>     then
#>  outcome = -191.22 + 159.9 rm + 3.92 lstat + 2.7 nox - 45 tax
#>            - 0.022 ptratio + 0.019 zn
#>
#>   Rule 9/8: [6 cases, mean 45.12, range 21.9 to 50, est err 25.03]
#>
#>     if
#>  nox > -0.6286645
#>  lstat <= 2.150069
#>     then
#>  outcome = -119.53 - 313.9 nox - 9.3 chas
#>
#> Model 10:
#>
#>   Rule 10/1: [221 cases, mean 18.34, range 5 to 50, est err 3.01]
#>
#>     if
#>  rm <= 1.831301
```

```
#>  lstat > 1.931448
#>     then
#>  outcome = 98.79 - 4.59 lstat - 0.76 crim + 10 rm - 1.9 dis - 41 tax
#>            - 0.36 indus + 0.8 rad - 0.017 ptratio
#>
#>   Rule 10/2: [185 cases, mean 27.55, range 7.2 to 50, est err 3.26]
#>
#>     if
#>  rm > 1.831301
#>     then
#>  outcome = 143.21 - 6.01 lstat + 34.3 rm - 3.5 dis - 84 tax - 0.33 indus
#>            - 0.016 ptratio - 0.11 crim + 0.2 rad
#>
#>   Rule 10/3: [9 cases, mean 28.52, range 22.5 to 50, est err 5.73]
#>
#>     if
#>  rm <= 1.895669
#>  lstat <= 1.931448
#>     then
#>  outcome = 201.43 - 91.7 rm - 2.4 dis
#>
#>   Rule 10/4: [40 cases, mean 35.86, range 22.5 to 50, est err 3.20]
#>
#>     if
#>  crim <= -1.051538
#>  lstat <= 1.931448
#>     then
#>  outcome = -136.13 + 96.1 rm - 7.9 dis - 0.0206 age
#>
#>   Rule 10/5: [12 cases, mean 46.13, range 31.5 to 50, est err 8.21]
#>
#>     if
#>  crim > -1.051538
#>  rm > 1.895669
#>  lstat <= 1.931448
#>     then
#>  outcome = 4.13 + 3.95 crim - 7.6 dis + 28.7 rm - 0.0197 age
#>
#> Model 11:
#>
#>   Rule 11/1: [84 cases, mean 13.75, range 5 to 25, est err 2.87]
#>
#>     if
#>  nox > -0.497006
#>     then
#>  outcome = 41.07 + 15.3 dis + 13.5 nox - 2.3 crim - 1.64 lstat - 13.3 rm
#>            + 6.1e-05 b
#>
#>   Rule 11/2: [169 cases, mean 19.61, range 7 to 33.8, est err 3.09]
#>
#>     if
#>  nox <= -0.497006
#>  lstat > 2.848535
#>     then
```

```
#>  outcome = 36.73 - 0.069 ptratio + 0.84 crim - 3.2 dis - 0.0122 age
#>            + 4.8e-05 b - 0.33 lstat - 0.4 nox + 1 rm
#>
#>   Rule 11/3: [157 cases, mean 30.59, range 18.6 to 50, est err 4.09]
#>
#>     if
#>  lstat <= 2.848535
#>     then
#>  outcome = 54.82 + 87.9 rm - 0.026 age - 0.062 ptratio + 1.01 crim
#>            - 91 tax + 4.8 nox - 1.1 dis
#>
#>   Rule 11/4: [11 cases, mean 39.32, range 21.9 to 50, est err 28.68]
#>
#>     if
#>  dis <= 0.6492998
#>  lstat <= 2.848535
#>     then
#>  outcome = 58.77 - 179.7 dis - 16.74 crim + 9.48 lstat + 31.1 rm
#>
#> Model 12:
#>
#>   Rule 12/1: [300 cases, mean 19.06, range 5 to 50, est err 2.96]
#>
#>     if
#>  rm <= 1.887978
#>  lstat > 1.805082
#>     then
#>  outcome = 118.53 - 5.23 lstat + 13.1 rm - 0.59 indus - 2.2 dis - 53 tax
#>
#>   Rule 12/2: [67 cases, mean 28.08, range 7.5 to 50, est err 3.89]
#>
#>     if
#>  rm > 1.887978
#>  lstat > 1.805082
#>     then
#>  outcome = 243.66 + 32.5 rm - 143 tax - 4.5 dis - 2.36 lstat - 5 nox
#>            - 0.53 indus
#>
#>   Rule 12/3: [31 cases, mean 38.23, range 22.8 to 50, est err 3.37]
#>
#>     if
#>  tax <= 1.899749
#>  lstat <= 1.805082
#>     then
#>  outcome = -126.48 + 82.5 rm - 3.09 crim - 8.4e-05 b - 0.81 lstat
#>
#>   Rule 12/4: [9 cases, mean 46.42, range 32.9 to 50, est err 8.91]
#>
#>     if
#>  tax > 1.899749
#>  lstat <= 1.805082
#>     then
#>  outcome = 53.55
#>
```

```
#> Model 13:
#>
#>   Rule 13/1: [84 cases, mean 13.75, range 5 to 25, est err 3.29]
#>
#>     if
#>  nox > -0.497006
#>     then
#>  outcome = 36.89 + 16.5 dis - 3.04 crim + 15.4 nox - 14.3 rm + 7.1e-05 b
#>
#>   Rule 13/2: [169 cases, mean 19.61, range 7 to 33.8, est err 3.36]
#>
#>     if
#>  nox <= -0.497006
#>  lstat > 2.848535
#>     then
#>  outcome = 215.56 + 3.5 rad - 0.074 ptratio - 98 tax - 3.4 dis
#>            - 0.0119 age + 5.5e-05 b + 0.33 indus
#>
#>   Rule 13/3: [298 cases, mean 24.85, range 7 to 50, est err 3.84]
#>
#>     if
#>  crim <= 0.9690653
#>     then
#>  outcome = -128.58 + 90.9 rm - 5.5 dis - 0.0176 age + 0.77 crim
#>            - 0.028 ptratio
#>
#>   Rule 13/4: [135 cases, mean 29.21, range 18.6 to 50, est err 2.85]
#>
#>     if
#>  dis > 0.9708925
#>  lstat <= 2.848535
#>     then
#>  outcome = 159.83 + 79.8 rm + 1.65 crim - 0.0294 age - 138 tax - 4.7 dis
#>            - 0.058 ptratio
#>
#>   Rule 13/5: [9 cases, mean 38.40, range 21.9 to 50, est err 11.82]
#>
#>     if
#>  crim > 0.9690653
#>  lstat <= 2.848535
#>     then
#>  outcome = -9.01 - 41.5 dis + 40.7 rm
#>
#> Model 14:
#>
#>   Rule 14/1: [218 cases, mean 18.28, range 5 to 50, est err 3.15]
#>
#>     if
#>  rm <= 1.831301
#>  lstat > 2.150069
#>     then
#>  outcome = 92.35 - 4.33 lstat - 0.7 crim - 35 tax + 5 rm - 0.23 indus
#>
#>   Rule 14/2: [112 cases, mean 22.42, range 7.2 to 50, est err 2.59]
```

```
#>
#>     if
#>  rm > 1.831301
#>  tax > 1.857866
#>  lstat > 2.150069
#>     then
#>  outcome = 34.48 - 5.61 lstat + 40.2 rm - 36 tax - 0.33 indus - 0.9 dis
#>
#>   Rule 14/3: [10 cases, mean 23.18, range 15.7 to 39.8, est err 5.98]
#>
#>     if
#>  tax <= 1.857866
#>  lstat > 2.150069
#>     then
#>  outcome = 32.05 + 30.2 dis - 4.58 lstat + 1.8 rm - 10 tax - 0.09 indus
#>
#>   Rule 14/4: [8 cases, mean 26.65, range 20.6 to 50, est err 6.28]
#>
#>     if
#>  rm <= 1.849714
#>  lstat <= 2.150069
#>     then
#>  outcome = 260.66 - 127.5 rm - 0.56 lstat - 0.3 dis - 0.07 crim
#>
#>   Rule 14/5: [68 cases, mean 36.05, range 21.9 to 50, est err 5.20]
#>
#>     if
#>  rm > 1.849714
#>  lstat <= 2.150069
#>     then
#>  outcome = -143.78 + 95.9 rm - 2.43 crim - 6.6 dis - 0.94 lstat
#>            - 0.008 ptratio
#>
#>   Rule 14/6: [20 cases, mean 40.28, range 21.9 to 50, est err 9.15]
#>
#>     if
#>  rm > 1.849714
#>  age > 195.4278
#>  lstat <= 2.150069
#>     then
#>  outcome = 41.37 + 0.1192 age - 10.24 lstat - 0.162 ptratio + 3.1 rm
#>
#> Model 15:
#>
#>   Rule 15/1: [84 cases, mean 13.75, range 5 to 25, est err 2.84]
#>
#>     if
#>  nox > -0.497006
#>     then
#>  outcome = 41.69 + 13.1 dis + 15.4 nox - 2.35 crim - 2 lstat - 11.9 rm
#>            + 6e-05 b
#>
#>   Rule 15/2: [166 cases, mean 19.48, range 7 to 31, est err 2.83]
#>
```

```
#>     if
#>  nox <= -0.497006
#>  lstat > 2.858393
#>     then
#>  outcome = 47.22 - 0.081 ptratio - 4.8 dis + 0.99 crim + 12.7 rm
#>            - 0.0128 age + 5.1e-05 b - 0.67 lstat - 1 nox + 0.4 rad
#>            - 15 tax
#>
#>   Rule 15/3: [160 cases, mean 30.52, range 18.6 to 50, est err 4.39]
#>
#>     if
#>  lstat <= 2.858393
#>     then
#>  outcome = 85.06 + 81 rm + 1.46 crim - 4.8 dis - 0.0258 age
#>            - 0.065 ptratio - 100 tax - 0.5 nox - 0.15 lstat + 0.1 rad
#>
#>   Rule 15/4: [11 cases, mean 39.32, range 21.9 to 50, est err 18.74]
#>
#>     if
#>  dis <= 0.6492998
#>  lstat <= 2.858393
#>     then
#>  outcome = 132.31 - 123.9 dis - 5.64 indus
#>
#> Model 16:
#>
#>   Rule 16/1: [62 cases, mean 16.43, range 5 to 50, est err 4.36]
#>
#>     if
#>  dis <= 0.5626968
#>     then
#>  outcome = 71.14 - 35.8 dis - 10.27 lstat + 1 rm
#>
#>   Rule 16/2: [345 cases, mean 23.72, range 7.2 to 50, est err 3.33]
#>
#>     if
#>  dis > 0.5626968
#>     then
#>  outcome = 67.61 + 33.1 rm - 2.21 lstat - 0.65 crim + 3.4 nox - 0.5 indus
#>            - 0.0108 age - 51 tax + 4.9e-05 b
#>
#>   Rule 16/3: [15 cases, mean 29.29, range 15.7 to 50, est err 3.14]
#>
#>     if
#>  tax <= 1.857866
#>     then
#>  outcome = 144.35 - 5.98 lstat + 18.4 rm - 0.78 indus - 2.8 dis - 67 tax
#>
#>   Rule 16/4: [6 cases, mean 30.28, range 22.8 to 50, est err 15.17]
#>
#>     if
#>  rm <= 1.895669
#>  lstat <= 1.805082
#>     then
```

```
#>  outcome = 294.96 - 134.8 rm - 0.000128 b - 0.96 lstat
#>
#>   Rule 16/5: [33 cases, mean 37.96, range 22.8 to 50, est err 4.98]
#>
#>     if
#>  tax <= 1.900249
#>  lstat <= 1.805082
#>     then
#>  outcome = -137.4 + 91.7 rm - 3.85 crim - 0.000181 b - 0.77 lstat
#>
#>   Rule 16/6: [6 cases, mean 50.00, range 50 to 50, est err 12.43]
#>
#>     if
#>  rm > 1.895669
#>  tax > 1.900249
#>  lstat <= 1.805082
#>     then
#>  outcome = 1473.45 - 649 tax - 87.1 rm - 7.3e-05 b - 0.31 lstat
#>
#> Model 17:
#>
#>   Rule 17/1: [84 cases, mean 13.75, range 5 to 25, est err 2.83]
#>
#>     if
#>  nox > -0.497006
#>     then
#>  outcome = 52.92 + 14.8 dis - 3.7 crim - 2.25 lstat - 17.7 rm
#>
#>   Rule 17/2: [9 cases, mean 13.79, range 7.5 to 21.9, est err 9.65]
#>
#>     if
#>  nox > -0.5267175
#>  rm > 1.898219
#>     then
#>  outcome = 19.26 - 5.03 crim
#>
#>   Rule 17/3: [243 cases, mean 21.17, range 7 to 50, est err 2.94]
#>
#>     if
#>  nox <= -0.497006
#>  rm <= 1.898219
#>     then
#>  outcome = 62.1 - 0.074 ptratio - 2.55 lstat - 4.2 dis + 1.8 rad
#>            - 0.9 nox + 2.2 rm - 12 tax + 0.0017 age
#>
#>   Rule 17/4: [47 cases, mean 34.92, range 22 to 50, est err 5.09]
#>
#>     if
#>  nox <= -0.5267175
#>  rm > 1.898219
#>  tax > 1.877141
#>     then
#>  outcome = -63.35 - 0.1167 age + 94.1 rm - 19.2 dis + 23.1 nox
#>            - 0.111 ptratio - 0.28 lstat
```

```
#>
#>   Rule 17/5: [33 cases, mean 38.21, range 26.6 to 50, est err 4.06]
#>
#>     if
#>  rm > 1.898219
#>  tax <= 1.877141
#>     then
#>  outcome = -195.9 + 126.5 rm - 4.9 dis - 1.69 lstat - 0.0141 age
#>            - 0.028 ptratio
#>
#> Model 18:
#>
#>   Rule 18/1: [250 cases, mean 17.59, range 5 to 33.8, est err 2.85]
#>
#>     if
#>  lstat > 2.848535
#>     then
#>  outcome = 176.49 - 2.81 lstat - 92 tax + 15.1 rm - 0.0132 age
#>            + 5.8e-05 b + 1 nox - 0.3 dis + 0.05 crim - 0.04 indus
#>
#>   Rule 18/2: [157 cases, mean 30.59, range 18.6 to 50, est err 4.38]
#>
#>     if
#>  lstat <= 2.848535
#>     then
#>  outcome = 120.3 + 45 rm - 3.67 lstat - 91 tax - 3.2 nox - 0.6 dis
#>            - 0.08 indus + 6e-06 b + 0.05 crim
#>
#>   Rule 18/3: [8 cases, mean 40.58, range 23.3 to 50, est err 18.43]
#>
#>     if
#>  dis <= 0.5868974
#>  lstat <= 2.848535
#>     then
#>  outcome = 90.38 - 114.5 dis
#>
#>
#> Evaluation on training data (407 cases):
#>
#>     Average  |error|               1.72
#>     Relative |error|               0.26
#>     Correlation coefficient        0.97
#>
#>
#>  Attribute usage:
#>    Conds  Model
#>
#>     69%    85%    lstat
#>     37%    52%    nox
#>     35%    88%    rm
#>     24%    87%    dis
#>     12%    73%    tax
#>      6%    66%    crim
#>      6%    63%    ptratio
```

```
#>      4%    50%    b
#>            65%    age
#>            38%    indus
#>            27%    rad
#>            10%    zn
#>             5%    chas
#>
#>
#> Time: 0.2 secs
```

We can now use this model to evaluate our held-out validation dataset. Again, we must prepare the input data using the same Box-Cox transform.

```
# transform the validation dataset
set.seed(7)
valX <- validation[,1:13]
trans_valX <- predict(preprocessParams, valX)
valY <- validation[,14]

# use final model to make predictions on the validation dataset
predictions <- predict(finalModel, newdata = trans_valX, neighbors=3)

# calculate RMSE
rmse <- RMSE(predictions, valY)
r2 <- R2(predictions, valY)
print(rmse)
```

```
#> [1] 2.666336
```

We can see that the estimated RMSE on this unseen data is about 2.666, lower but not too dissimilar from our expected RMSE of 2.822.

Chapter 10

Breast Cancer

```
# load packages
library(mlbench)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
# Load data
data(BreastCancer)
```

```
dplyr::glimpse(BreastCancer)
```

```
## Observations: 699
## Variables: 11
## $ Id              <chr> "1000025", "1002945", "1015425", "1016277", "1...
## $ Cl.thickness    <ord> 5, 5, 3, 6, 4, 8, 1, 2, 2, 4, 1, 2, 5, 1, 8, 7...
## $ Cell.size       <ord> 1, 4, 1, 8, 1, 10, 1, 1, 1, 2, 1, 1, 3, 1, 7, ...
## $ Cell.shape      <ord> 1, 4, 1, 8, 1, 10, 1, 2, 1, 1, 1, 1, 3, 1, 5, ...
## $ Marg.adhesion   <ord> 1, 5, 1, 1, 3, 8, 1, 1, 1, 1, 1, 1, 3, 1, 10, ...
## $ Epith.c.size    <ord> 2, 7, 2, 3, 2, 7, 2, 2, 2, 2, 1, 2, 2, 2, 7, 6...
## $ Bare.nuclei     <fct> 1, 10, 2, 4, 1, 10, 10, 1, 1, 1, 1, 1, 3, 3, 9...
## $ Bl.cromatin     <fct> 3, 3, 3, 3, 3, 9, 3, 3, 1, 2, 3, 2, 4, 3, 5, 4...
## $ Normal.nucleoli <fct> 1, 2, 1, 7, 1, 7, 1, 1, 1, 1, 1, 1, 4, 1, 5, 3...
## $ Mitoses         <fct> 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 4, 1...
## $ Class           <fct> benign, benign, benign, benign, benign, malign...
```

```
tibble::as_tibble(BreastCancer)
```

```
## # A tibble: 699 x 11
##    Id    Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
##    <chr> <ord>        <ord>     <ord>      <ord>         <ord>
##  1 1000~ 5            1         1          1             2
##  2 1002~ 5            4         4          5             7
##  3 1015~ 3            1         1          1             2
##  4 1016~ 6            8         8          1             3
##  5 1017~ 4            1         1          3             2
##  6 1017~ 8            10        10         8             7
##  7 1018~ 1            1         1          1             2
##  8 1018~ 2            1         2          1             2
```

```
##  9 1033~ 2            1         1          1             2
## 10 1033~ 4            2         1          1             2
## # ... with 689 more rows, and 5 more variables: Bare.nuclei <fct>,
## #   Bl.cromatin <fct>, Normal.nucleoli <fct>, Mitoses <fct>, Class <fct>
```

```
# Split out validation dataset
# create a list of 80% of the rows in the original dataset we can use for training
set.seed(7)
validationIndex <- createDataPartition(BreastCancer$Class,
                                       p=0.80,
                                       list=FALSE)

# select 20% of the data for validation
validation <- BreastCancer[-validationIndex,]
# use the remaining 80% of data to training and testing the models
dataset <- BreastCancer[validationIndex,]
```

```
# dimensions of dataset
dim(validation)
```

```
## [1] 139  11
```

```
dim(dataset)
```

```
## [1] 560  11
```

```
# peek
head(dataset, n=20)
```

```
##         Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
## 2  1002945            5         4          4             5            7
## 4  1016277            6         8          8             1            3
## 5  1017023            4         1          1             3            2
## 6  1017122            8        10         10             8            7
## 7  1018099            1         1          1             1            2
## 9  1033078            2         1          1             1            2
## 10 1033078            4         2          1             1            2
## 11 1035283            1         1          1             1            1
## 12 1036172            2         1          1             1            2
## 14 1043999            1         1          1             1            2
## 15 1044572            8         7          5            10            7
## 16 1047630            7         4          6             4            6
## 17 1048672            4         1          1             1            2
## 18 1049815            4         1          1             1            2
## 19 1050670           10         7          7             6            4
## 20 1050718            6         1          1             1            2
## 21 1054590            7         3          2            10            5
## 22 1054593           10         5          5             3            6
## 23 1056784            3         1          1             1            2
## 24 1057013            8         4          5             1            2
##    Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses     Class
## 2           10           3               2       1    benign
## 4            4           3               7       1    benign
## 5            1           3               1       1    benign
## 6           10           9               7       1 malignant
## 7           10           3               1       1    benign
## 9            1           1               1       5    benign
```

```
## 10           1           2               1       1    benign
## 11           1           3               1       1    benign
## 12           1           2               1       1    benign
## 14           3           3               1       1    benign
## 15           9           5               5       4 malignant
## 16           1           4               3       1 malignant
## 17           1           2               1       1    benign
## 18           1           3               1       1    benign
## 19          10           4               1       2 malignant
## 20           1           3               1       1    benign
## 21          10           5               4       4 malignant
## 22           7           7              10       1 malignant
## 23           1           2               1       1    benign
## 24        <NA>           7               3       1 malignant
```

```
library(skimr)
```

```
##
## Attaching package: 'skimr'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
skim(dataset)
```

```
## Skim summary statistics
##  n obs: 560
##  n variables: 11
##
## -- Variable type:character -------------------------------------------------
##  variable missing complete   n min max empty n_unique
##        Id       0      560 560   5   7     0      526
##
## -- Variable type:factor ----------------------------------------------------
##         variable missing complete   n n_unique
##      Bare.nuclei      13      547 560       10
##      Bl.cromatin       0      560 560       10
##       Cell.shape       0      560 560       10
##        Cell.size       0      560 560       10
##     Cl.thickness       0      560 560       10
##            Class       0      560 560        2
##     Epith.c.size       0      560 560       10
##    Marg.adhesion       0      560 560       10
##          Mitoses       0      560 560        9
##  Normal.nucleoli       0      560 560       10
##                     top_counts ordered
##   1: 327, 10: 99, 5: 28, 2: 23   FALSE
##  2: 133, 3: 133, 1: 124, 7: 57   FALSE
##   1: 278, 2: 51, 3: 46, 10: 44    TRUE
##   1: 307, 10: 51, 3: 48, 2: 33    TRUE
##   1: 120, 5: 104, 3: 85, 4: 61    TRUE
##      ben: 367, mal: 193, NA: 0   FALSE
##    2: 310, 3: 59, 4: 39, 1: 35    TRUE
##   1: 324, 10: 47, 2: 45, 3: 45    TRUE
##    1: 460, 3: 28, 2: 27, 4: 11   FALSE
```

```
##   1: 351, 10: 47, 3: 36, 2: 31   FALSE
```

```
# types
sapply(dataset, class)
```

```
## $Id
## [1] "character"
##
## $Cl.thickness
## [1] "ordered" "factor"
##
## $Cell.size
## [1] "ordered" "factor"
##
## $Cell.shape
## [1] "ordered" "factor"
##
## $Marg.adhesion
## [1] "ordered" "factor"
##
## $Epith.c.size
## [1] "ordered" "factor"
##
## $Bare.nuclei
## [1] "factor"
##
## $Bl.cromatin
## [1] "factor"
##
## $Normal.nucleoli
## [1] "factor"
##
## $Mitoses
## [1] "factor"
##
## $Class
## [1] "factor"
```

We can see that besides the Id, the attributes are factors. This makes sense. I think for modeling it may be more useful to work with the data as numbers than factors. Factors might make things easier for decision tree algorithms (or not). Given that there is an ordinal relationship between the levels we can expose that structure to other algorithms better if we work directly with the integer numbers.

10.1 clean up

```
# Remove redundant variable Id
dataset <- dataset[,-1]
```

```
# convert input values to numeric
for(i in 1:9) {
    dataset[,i] <- as.numeric(as.character(dataset[,i]))
}
```

```r
# summary
summary(dataset)
```

```
##   Cl.thickness      Cell.size        Cell.shape     Marg.adhesion
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Median : 4.000   Median : 1.000   Median : 2.000   Median : 1.000
##  Mean   : 4.384   Mean   : 3.116   Mean   : 3.198   Mean   : 2.875
##  3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000   3rd Qu.: 4.000
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000
##
##   Epith.c.size     Bare.nuclei      Bl.cromatin     Normal.nucleoli
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 2.000   1st Qu.: 1.000
##  Median : 2.000   Median : 1.000   Median : 3.000   Median : 1.000
##  Mean   : 3.232   Mean   : 3.468   Mean   : 3.405   Mean   : 2.877
##  3rd Qu.: 4.000   3rd Qu.: 5.000   3rd Qu.: 4.250   3rd Qu.: 4.000
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000
##                   NA's   :13
##     Mitoses            Class
##  Min.   : 1.000   benign   :367
##  1st Qu.: 1.000   malignant:193
##  Median : 1.000
##  Mean   : 1.611
##  3rd Qu.: 1.000
##  Max.   :10.000
##
```

```r
skim(dataset)
```

```
## Skim summary statistics
##  n obs: 560
##  n variables: 10
##
## -- Variable type:factor -------------------------------------------------------
##  variable missing complete   n n_unique                top_counts ordered
##     Class       0      560 560        2 ben: 367, mal: 193, NA: 0   FALSE
##
## -- Variable type:numeric ------------------------------------------------------
##         variable missing complete   n mean   sd p0 p25 p50  p75 p100
##      Bare.nuclei      13      547 560 3.47 3.59  1   1   1 5      10
##      Bl.cromatin       0      560 560 3.41 2.42  1   2   3 4.25   10
##       Cell.shape       0      560 560 3.2  2.94  1   1   2 5      10
##        Cell.size       0      560 560 3.12 3.02  1   1   1 5      10
##     Cl.thickness       0      560 560 4.38 2.8   1   2   4 6      10
##     Epith.c.size       0      560 560 3.23 2.21  1   2   2 4      10
##    Marg.adhesion       0      560 560 2.88 2.92  1   1   1 4      10
##          Mitoses       0      560 560 1.61 1.71  1   1   1 1      10
##  Normal.nucleoli       0      560 560 2.88 3.04  1   1   1 4      10
##      hist
##
##
##
##
##
```

```
##
##
##
##
```

we can see we have 13 NA values for the Bare.nuclei attribute. This suggests we may need to remove the records (or impute values) with NA values for some analysis and modeling techniques.

10.2 Analyze the class variable

```
# class distribution
cbind(freq = table(dataset$Class),
      percentage = prop.table(table(dataset$Class))*100)
```

```
##           freq percentage
## benign     367   65.53571
## malignant  193   34.46429
```

There is indeed a 65% to 35% split for benign-malignant in the class values which is imbalanced, but not so much that we need to be thinking about rebalancing the dataset, at least not yet.

10.2.1 remove NAs

```
# summarize correlations between input variables
complete_cases <- complete.cases(dataset)
cor(dataset[complete_cases,1:9])
```

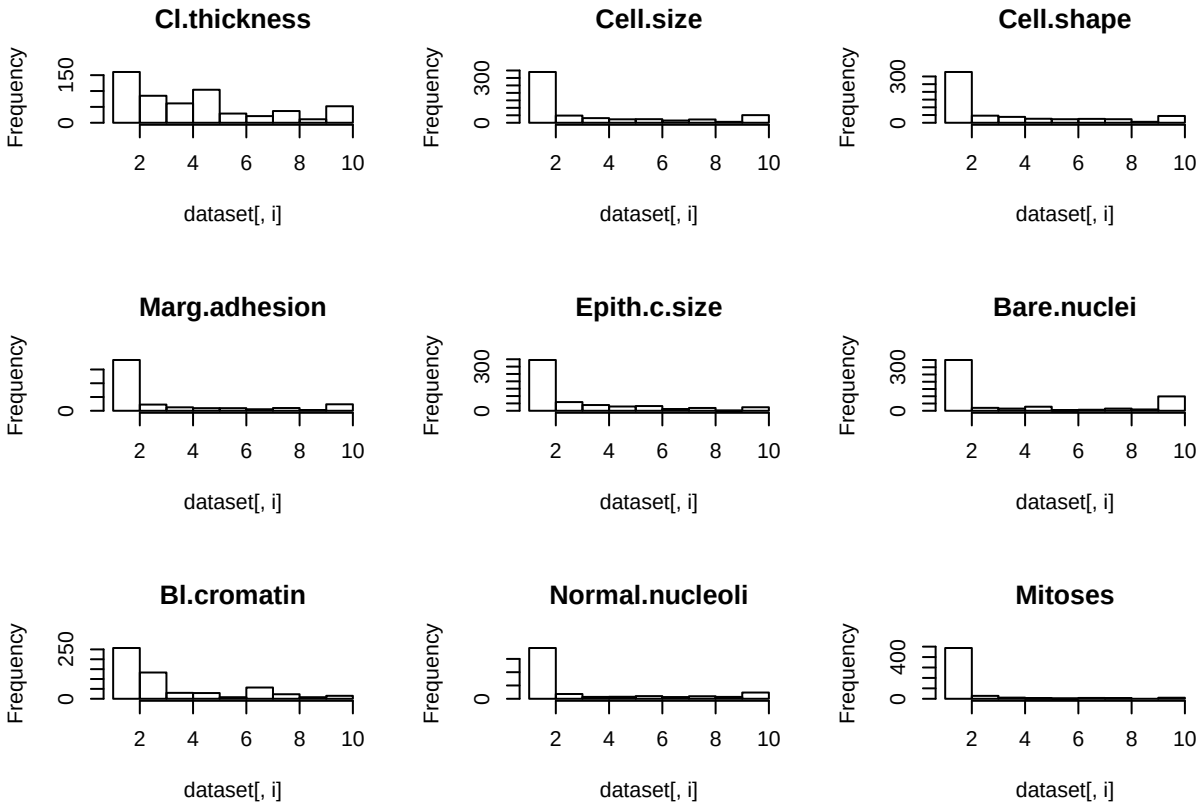
```
##                 Cl.thickness Cell.size Cell.shape Marg.adhesion
## Cl.thickness       1.0000000 0.6200884  0.6302917     0.4741733
## Cell.size          0.6200884 1.0000000  0.9011340     0.7141150
## Cell.shape         0.6302917 0.9011340  1.0000000     0.6846206
## Marg.adhesion      0.4741733 0.7141150  0.6846206     1.0000000
## Epith.c.size       0.5089557 0.7404824  0.7043423     0.5860660
## Bare.nuclei        0.5600770 0.6687226  0.6896724     0.6660165
## Bl.cromatin        0.5290733 0.7502700  0.7276114     0.6660533
## Normal.nucleoli    0.5143933 0.7072182  0.7127155     0.6031036
## Mitoses            0.3426018 0.4506532  0.4345125     0.4314910
##                 Epith.c.size Bare.nuclei Bl.cromatin Normal.nucleoli
## Cl.thickness       0.5089557   0.5600770   0.5290733       0.5143933
## Cell.size          0.7404824   0.6687226   0.7502700       0.7072182
## Cell.shape         0.7043423   0.6896724   0.7276114       0.7127155
## Marg.adhesion      0.5860660   0.6660165   0.6660533       0.6031036
## Epith.c.size       1.0000000   0.5568406   0.6102032       0.6433364
## Bare.nuclei        0.5568406   1.0000000   0.6668483       0.5795794
## Bl.cromatin        0.6102032   0.6668483   1.0000000       0.6838547
## Normal.nucleoli    0.6433364   0.5795794   0.6838547       1.0000000
## Mitoses            0.4775271   0.3539473   0.3545122       0.4084127
##                   Mitoses
## Cl.thickness    0.3426018
## Cell.size       0.4506532
## Cell.shape      0.4345125
## Marg.adhesion   0.4314910
```

```
## Epith.c.size    0.4775271
## Bare.nuclei     0.3539473
## Bl.cromatin     0.3545122
## Normal.nucleoli 0.4084127
## Mitoses         1.0000000
```

We can see some modest to high correlation between some of the attributes. For example between cell shape and cell size at 0.90 correlation.

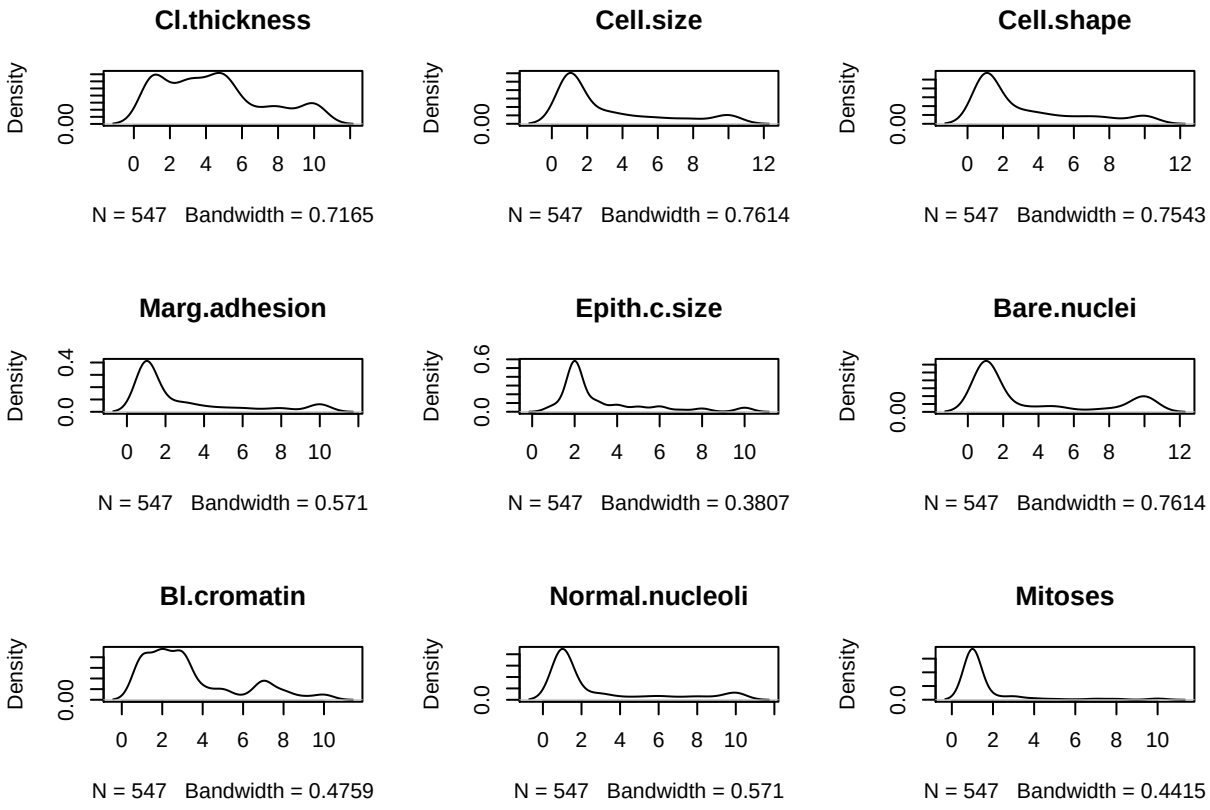
10.3 Unimodal visualization

```
# histograms each attribute
par(mfrow=c(3,3))
for(i in 1:9) {
    hist(dataset[,i], main=names(dataset)[i])
}
```



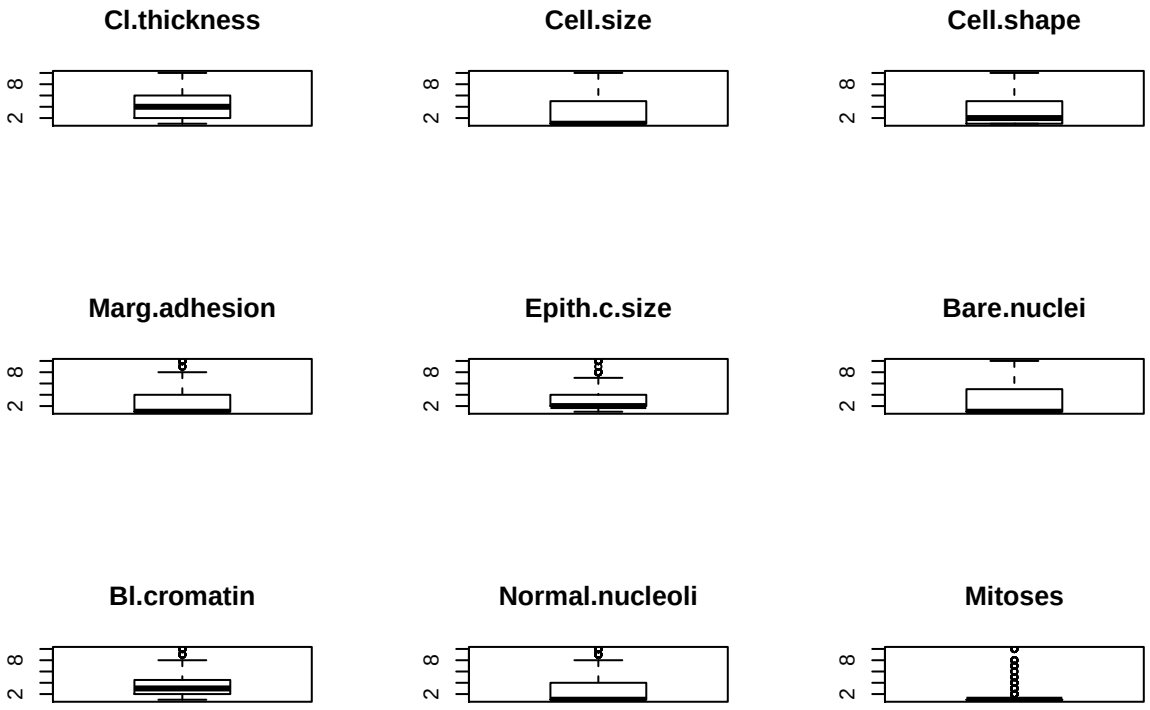
We can see that almost all of the distributions have an exponential or bimodal shape to them.

```
# density plot for each attribute
par(mfrow=c(3,3))
complete_cases <- complete.cases(dataset)
for(i in 1:9) {
    plot(density(dataset[complete_cases,i]), main=names(dataset)[i])
}
```



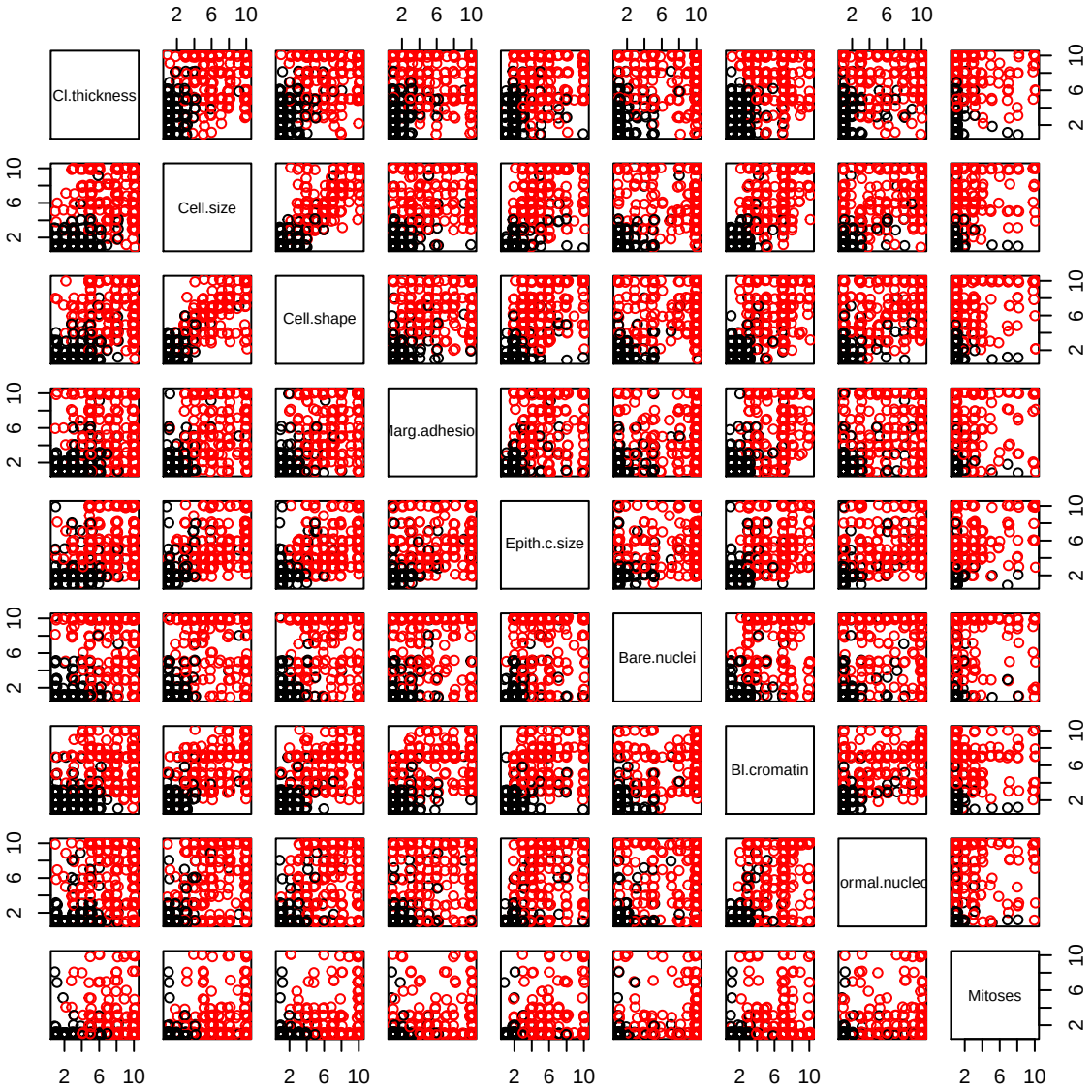
These plots add more support to our initial ideas. We can see bimodal distributions (two bumps) and exponential-looking distributions.

```
# boxplots for each attribute
par(mfrow=c(3,3))
for(i in 1:9) {
    boxplot(dataset[,i], main=names(dataset)[i])
}
```



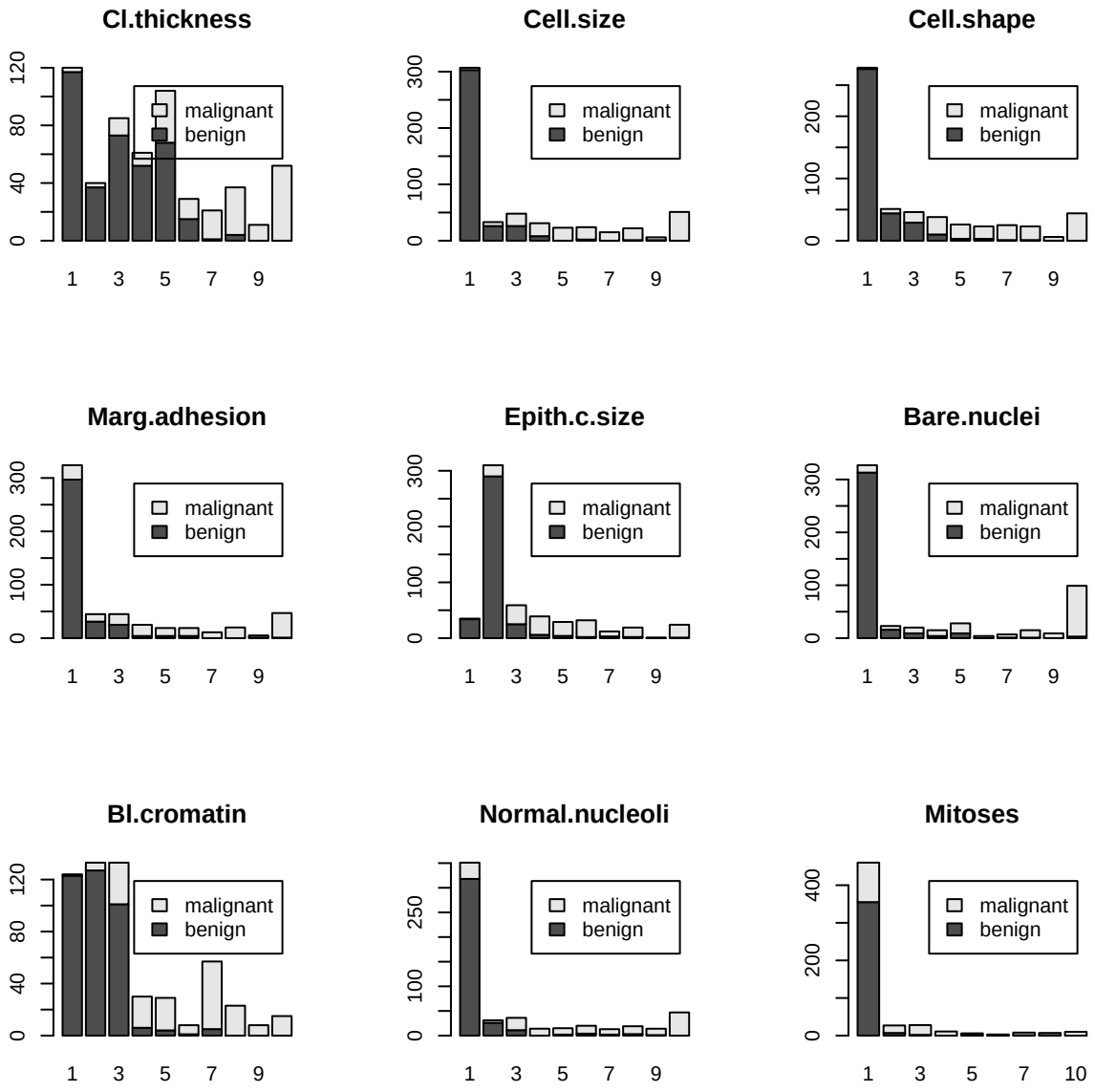
10.4 Multimodal visualization

```
# scatter plot matrix
jittered_x <- sapply(dataset[,1:9], jitter)
pairs(jittered_x, names(dataset[,1:9]), col=dataset$Class)
```



We can see that the black (benign) a part to be clustered around the bottom-right corner (smaller values) and red (malignant) are all over the place.

```
# bar plots of each variable by class
par(mfrow=c(3,3))
for(i in 1:9) {
    barplot(table(dataset$Class,dataset[,i]), main=names(dataset)[i],
            legend.text=unique(dataset$Class))
}
```



10.5 Algorithms Evaluation

- Linear Algorithms: Logistic Regression (LG), Linear Discriminate Analysis (LDA) and Regularized Logistic Regression (GLMNET).
- Nonlinear Algorithms: k-Nearest Neighbors (KNN), Classification and Regression Trees (CART), Naive Bayes (NB) and Support Vector Machines with Radial Basis Functions (SVM).

For simplicity, we will use Accuracy and Kappa metrics. Given that it is a medical test, we could have gone with the Area Under ROC Curve (AUC) and looked at the sensitivity and specificity to select the best algorithms.

```
# 10-fold cross-validation with 3 repeats
trainControl <- trainControl(method = "repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
```

```
# LG
set.seed(7)
```

```
fit.glm <- train(Class~., data=dataset, method="glm", metric=metric,
                 trControl=trainControl, na.action=na.omit)
# LDA
set.seed(7)
fit.lda <- train(Class~., data=dataset, method="lda", metric=metric,
                 trControl=trainControl, na.action=na.omit)
# GLMNET
set.seed(7)
fit.glmnet <- train(Class~., data=dataset, method="glmnet", metric=metric,
                    trControl=trainControl, na.action=na.omit)
# KNN
set.seed(7)
fit.knn <- train(Class~., data=dataset, method="knn", metric=metric,
                 trControl=trainControl, na.action=na.omit)
# CART
set.seed(7)
fit.cart <- train(Class~., data=dataset, method="rpart", metric=metric,
                  trControl=trainControl, na.action=na.omit)
# Naive Bayes
set.seed(7)
fit.nb <- train(Class~., data=dataset, method="nb", metric=metric,
                trControl=trainControl, na.action=na.omit)
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
```

```
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 33

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 33

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 32
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 47

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 32

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 47

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 33

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 33

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 46

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 46

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 32

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 32

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 46

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 46

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 46

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 46
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 32

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 33

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 32

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 33

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 47

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 47

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 49

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 33

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 33

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 46

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 47

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 46

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 47

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 32

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 32

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 39

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 45

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 50

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 46

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 3

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 29

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 30

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 46

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 33

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 11

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 14

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 17

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 19

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 26

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 33

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 35

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 37

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 38

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 40
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 42

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 47

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 13

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 22

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 27

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 31

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 44

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 48

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 2

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 4

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 5

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 10

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 16

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 18

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 20

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 21

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 23

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 24

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 25

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 28

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 43

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 51
```

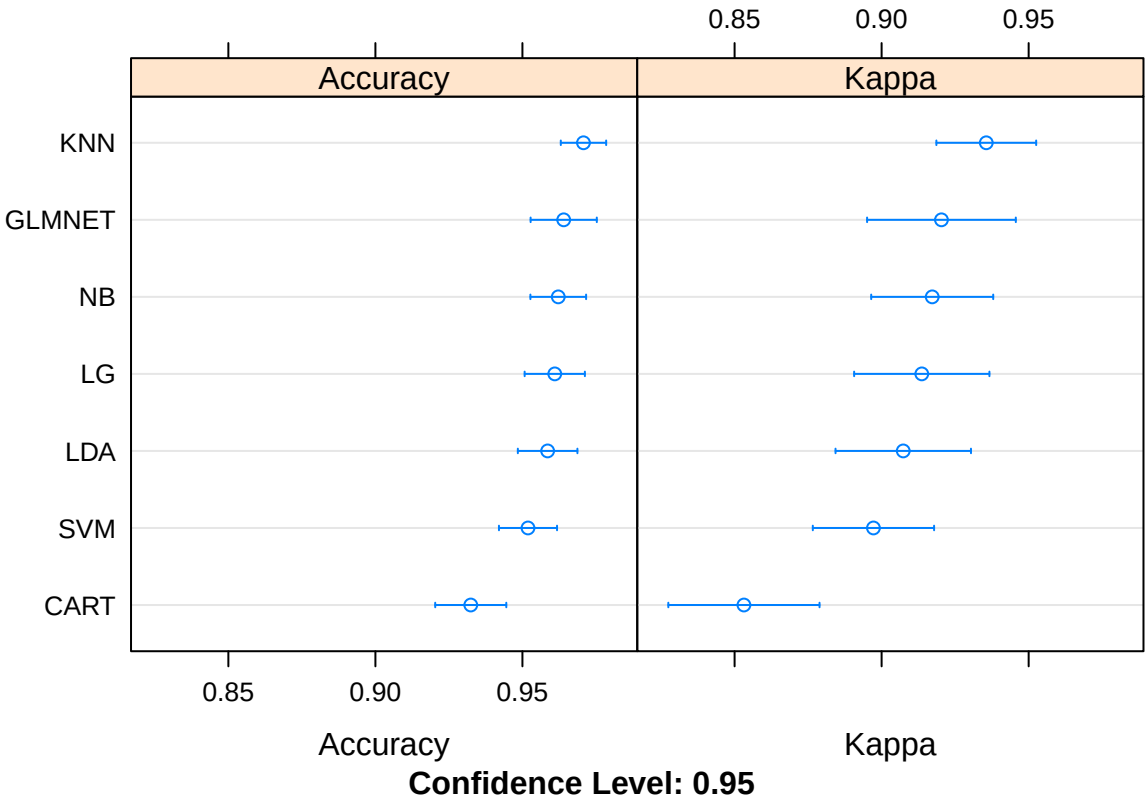
```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 52
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54
```

```
# SVM
set.seed(7)
fit.svm <- train(Class~., data=dataset, method="svmRadial", metric=metric,
                 trControl=trainControl, na.action=na.omit)

# Compare algorithms
results <- resamples(list(LG     = fit.glm,
                          LDA    = fit.lda,
                          GLMNET = fit.glmnet,
                          KNN    = fit.knn,
                          CART   = fit.cart,
                          NB     = fit.nb,
                          SVM    = fit.svm))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: LG, LDA, GLMNET, KNN, CART, NB, SVM
## Number of resamples: 30
##
## Accuracy
##             Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## LG     0.8909091 0.9498316 0.9636364 0.9609977 0.9814815    1    0
## LDA    0.8888889 0.9454545 0.9629630 0.9585734 0.9771825    1    0
## GLMNET 0.8909091 0.9498316 0.9636364 0.9640392 0.9818182    1    0
## KNN    0.9090909 0.9629630 0.9728836 0.9707620 0.9818182    1    0
## CART   0.8571429 0.9090909 0.9272727 0.9324311 0.9461851    1    0
## NB     0.9090909 0.9446970 0.9636364 0.9621878 0.9817340    1    0
## SVM    0.8909091 0.9444444 0.9629630 0.9518951 0.9641234    1    0
##
## Kappa
##             Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## LG     0.7526237 0.8903491 0.9200813 0.9136612 0.9591931    1    0
## LDA    0.7440758 0.8778682 0.9187970 0.9073541 0.9493543    1    0
## GLMNET 0.7526237 0.8903491 0.9207048 0.9203519 0.9601869    1    0
## KNN    0.8014440 0.9195906 0.9402224 0.9355909 0.9598811    1    0
## CART   0.7083333 0.8011046 0.8464753 0.8531748 0.8859322    1    0
## NB     0.7964471 0.8796434 0.9205797 0.9172069 0.9597332    1    0
## SVM    0.7755102 0.8799491 0.9207048 0.8972223 0.9220518    1    0
```

```
dotplot(results)
```



We can see good accuracy across the board. All algorithms have a mean accuracy above 90%, well above the baseline of 65% if we just predicted benign. The problem is learnable. We can see that KNN (97.08%) and logistic regression (NB was 96.2% and GLMNET was 96.4%) had the highest accuracy on the problem.

10.6 Data transform

We know we have some skewed distributions. There are transform methods that we can use to adjust and normalize these distributions. A favorite for positive input attributes (which we have in this case) is the Box-Cox transform.

```
# 10-fold cross-validation with 3 repeats
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"

# LG
set.seed(7)
fit.glm <- train(Class~., data=dataset, method="glm", metric=metric,
                 preProc=c("BoxCox"), trControl=trainControl, na.action=na.omit)
# LDA
set.seed(7)
fit.lda <- train(Class~., data=dataset, method="lda", metric=metric,
                 preProc=c("BoxCox"), trControl=trainControl, na.action=na.omit)
# GLMNET
set.seed(7)
fit.glmnet <- train(Class~., data=dataset, method="glmnet", metric=metric,
                    preProc=c("BoxCox"), trControl=trainControl,
```

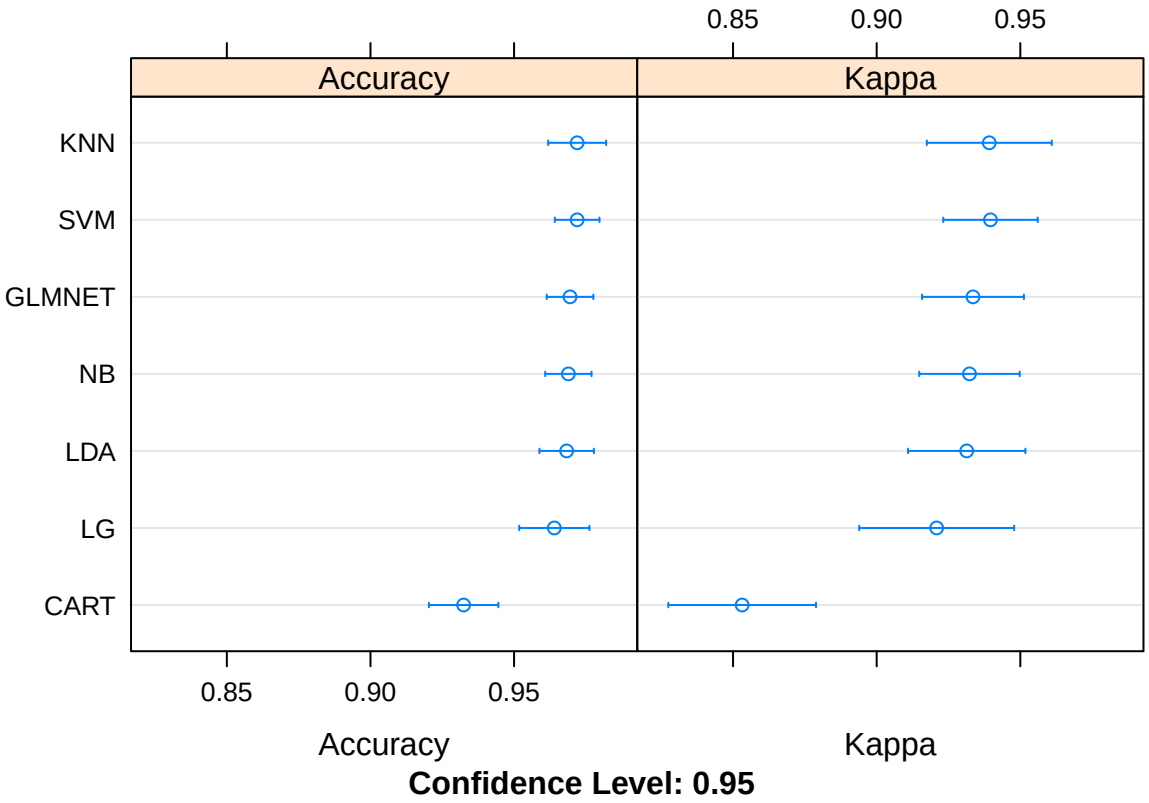
```
                    na.action=na.omit)
# KNN
set.seed(7)
fit.knn <- train(Class~., data=dataset, method="knn", metric=metric,
                 preProc=c("BoxCox"), trControl=trainControl, na.action=na.omit)
# CART
set.seed(7)
fit.cart <- train(Class~., data=dataset, method="rpart", metric=metric,
                  preProc=c("BoxCox"), trControl=trainControl,
                  na.action=na.omit)
# Naive Bayes
set.seed(7)
fit.nb <- train(Class~., data=dataset, method="nb", metric=metric,
                preProc=c("BoxCox"), trControl=trainControl, na.action=na.omit)
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 12

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 32

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 8

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 53

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 15

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 7

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 54

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 41

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 36

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 1

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 6

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 55

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 9
```

```r
# SVM
set.seed(7)
fit.svm <- train(Class~., data=dataset, method="svmRadial", metric=metric,
                 preProc=c("BoxCox"), trControl=trainControl, na.action=na.omit)

# Compare algorithms
transformResults <- resamples(list(LG     = fit.glm,
                                   LDA    = fit.lda,
                                   GLMNET = fit.glmnet,
                                   KNN    = fit.knn,
                                   CART   = fit.cart,
                                   NB     = fit.nb,
                                   SVM    = fit.svm))
summary(transformResults)
```

```
##
## Call:
## summary.resamples(object = transformResults)
##
## Models: LG, LDA, GLMNET, KNN, CART, NB, SVM
## Number of resamples: 30
##
## Accuracy
##             Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## LG     0.8727273 0.9454545 0.9725589 0.9640063 0.9818182    1    0
## LDA    0.8909091 0.9629630 0.9725589 0.9683045 0.9818182    1    0
## GLMNET 0.9259259 0.9629630 0.9636364 0.9694833 0.9818182    1    0
## KNN    0.8727273 0.9631313 0.9814815 0.9719633 0.9818182    1    0
## CART   0.8571429 0.9090909 0.9272727 0.9324311 0.9461851    1    0
## NB     0.9090909 0.9629630 0.9636364 0.9688993 0.9818182    1    0
## SVM    0.9090909 0.9631313 0.9636364 0.9719525 0.9818182    1    0
##
## Kappa
##             Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## LG     0.7286822 0.8808664 0.9402224 0.9208886 0.9602888    1    0
## LDA    0.7701950 0.9195906 0.9407109 0.9313355 0.9602888    1    0
## GLMNET 0.8335901 0.9198692 0.9215407 0.9335337 0.9602888    1    0
## KNN    0.7286822 0.9207048 0.9598811 0.9391947 0.9602888    1    0
## CART   0.7083333 0.8011046 0.8464753 0.8531748 0.8859322    1    0
## NB     0.8062016 0.9198692 0.9215407 0.9323071 0.9602888    1    0
## SVM    0.8107364 0.9207048 0.9215407 0.9396187 0.9602888    1    0
```

```r
dotplot(transformResults)
```



We can see that the accuracy of the previous best algorithm KNN was elevated to 97.14%. We have a new ranking, showing SVM with the most accurate mean accuracy at 97.20%.

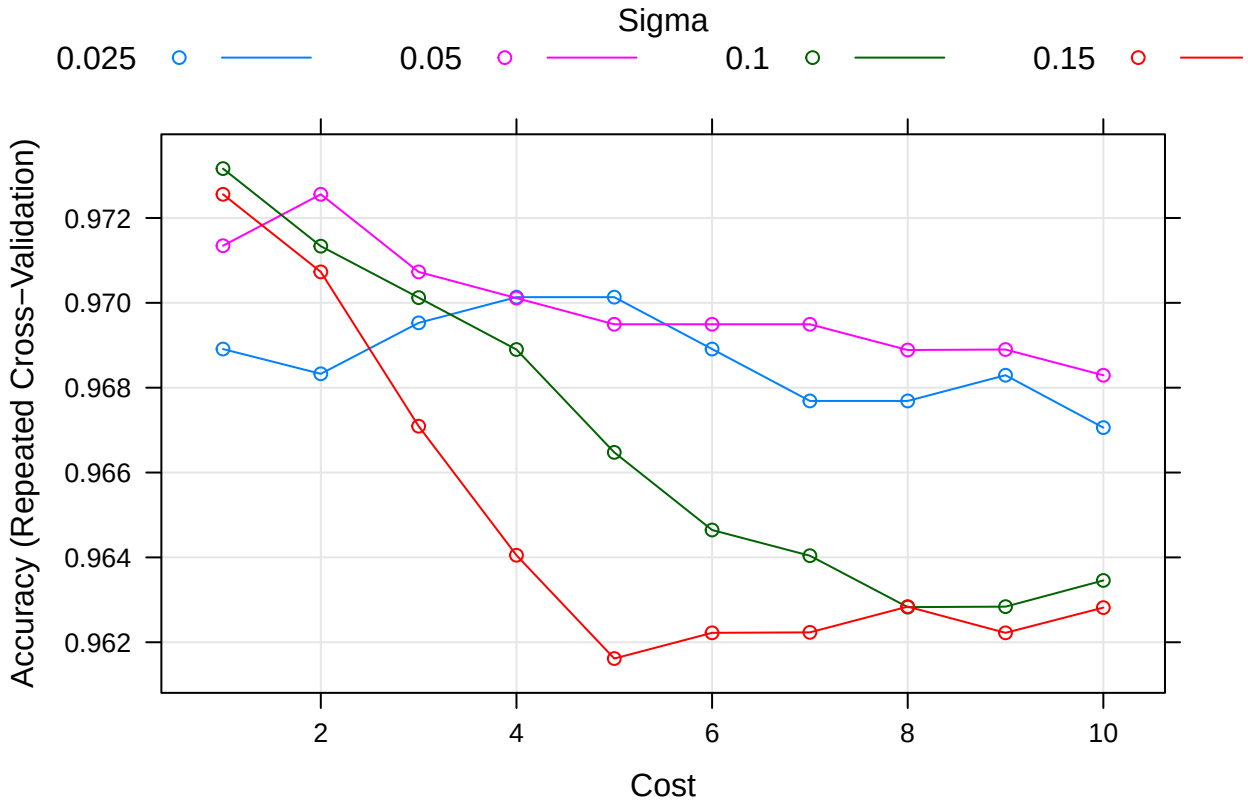
10.7 Tuning SVM

```
# 10-fold cross-validation with 3 repeats
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
set.seed(7)

grid <- expand.grid(.sigma = c(0.025, 0.05, 0.1, 0.15),
                    .C = seq(1, 10, by=1))

fit.svm <- train(Class~., data=dataset, method="svmRadial", metric=metric,
                 tuneGrid=grid,
                 preProc=c("BoxCox"), trControl=trainControl,
                 na.action=na.omit)
print(fit.svm)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 560 samples
##   9 predictor
##   2 classes: 'benign', 'malignant'
##
## Pre-processing: Box-Cox transformation (9)
```

```
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 492, 493, 492, 492, 493, 491, ...
## Resampling results across tuning parameters:
##
##   sigma  C   Accuracy   Kappa
##   0.025   1  0.9689109  0.9329235
##   0.025   2  0.9683269  0.9314403
##   0.025   3  0.9695278  0.9341394
##   0.025   4  0.9701343  0.9354965
##   0.025   5  0.9701343  0.9354965
##   0.025   6  0.9689109  0.9329293
##   0.025   7  0.9676876  0.9302032
##   0.025   8  0.9676876  0.9302032
##   0.025   9  0.9682937  0.9315590
##   0.025  10  0.9670591  0.9288215
##   0.050   1  0.9713464  0.9382360
##   0.050   2  0.9725585  0.9408533
##   0.050   3  0.9707291  0.9368374
##   0.050   4  0.9701118  0.9354950
##   0.050   5  0.9694945  0.9341274
##   0.050   6  0.9694945  0.9341274
##   0.050   7  0.9694945  0.9341274
##   0.050   8  0.9688885  0.9327716
##   0.050   9  0.9688997  0.9328178
##   0.050  10  0.9682937  0.9314271
##   0.100   1  0.9731646  0.9422986
##   0.100   2  0.9713352  0.9382492
##   0.100   3  0.9701231  0.9355412
##   0.100   4  0.9688997  0.9327829
##   0.100   5  0.9664751  0.9275447
##   0.100   6  0.9646457  0.9233957
##   0.100   7  0.9640396  0.9220387
##   0.100   8  0.9628275  0.9192922
##   0.100   9  0.9628387  0.9192111
##   0.100  10  0.9634560  0.9204856
##   0.150   1  0.9725585  0.9409753
##   0.150   2  0.9707291  0.9368926
##   0.150   3  0.9670924  0.9289103
##   0.150   4  0.9640508  0.9220156
##   0.150   5  0.9616154  0.9165448
##   0.150   6  0.9622214  0.9179356
##   0.150   7  0.9622326  0.9177849
##   0.150   8  0.9628387  0.9190402
##   0.150   9  0.9622214  0.9175415
##   0.150  10  0.9628163  0.9189235
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.1 and C = 1.
```

```
plot(fit.svm)
```



We can see that we have made very little difference to the results. The most accurate model had a score of 97.31% (the same as our previously rounded score of 97.20%) using a sigma = 0.1 and C = 1. We could tune further, but I don't expect a payoff.

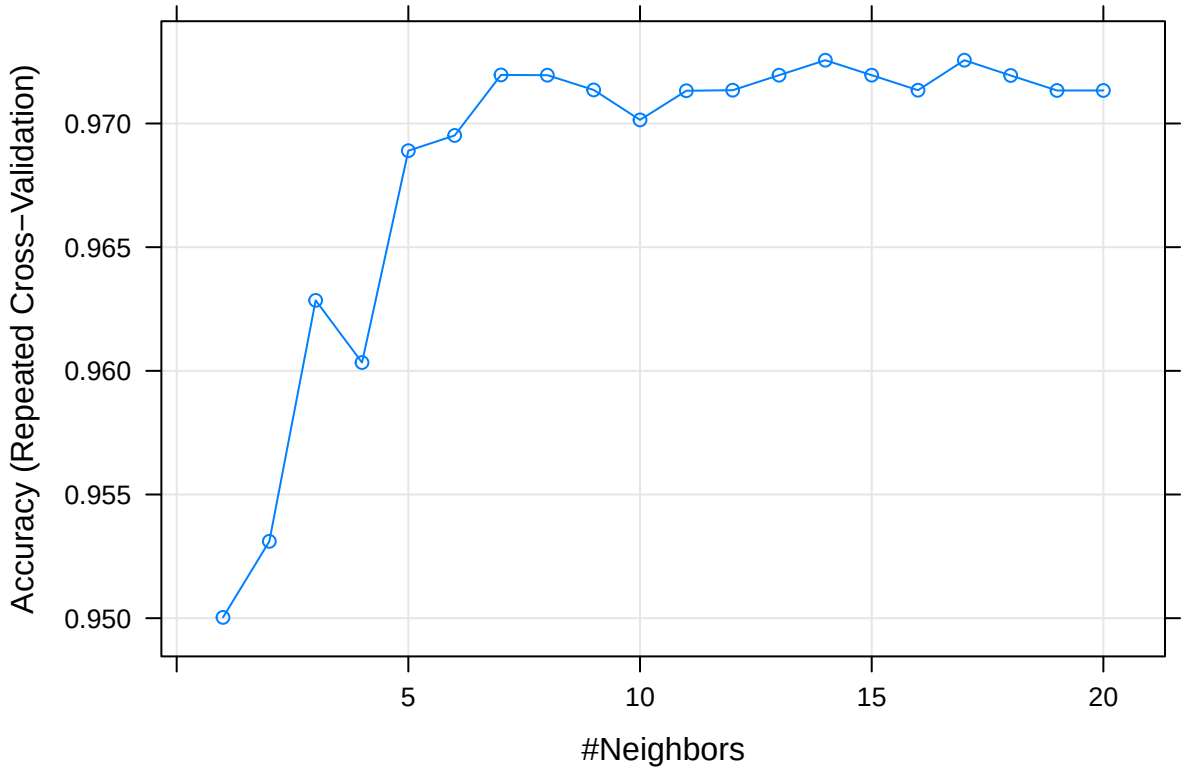
10.8 Tuning KNN

```
# 10-fold cross-validation with 3 repeats
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
set.seed(7)

grid <- expand.grid(.k = seq(1,20, by=1))
fit.knn <- train(Class~., data=dataset, method="knn", metric=metric,
                 tuneGrid=grid,
                 preProc=c("BoxCox"), trControl=trainControl,
                 na.action=na.omit)
print(fit.knn)
```

```
## k-Nearest Neighbors
##
## 560 samples
##   9 predictor
##   2 classes: 'benign', 'malignant'
##
## Pre-processing: Box-Cox transformation (9)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 492, 493, 492, 492, 493, 491, ...
```

```
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9500325  0.8894817
##    2  0.9531069  0.8967630
##    3  0.9628495  0.9188475
##    4  0.9603363  0.9128179
##    5  0.9688997  0.9323375
##    6  0.9695166  0.9339598
##    7  0.9719633  0.9391947
##    8  0.9719525  0.9391787
##    9  0.9713572  0.9378967
##   10  0.9701451  0.9351851
##   11  0.9713244  0.9378279
##   12  0.9713464  0.9378517
##   13  0.9719525  0.9392355
##   14  0.9725585  0.9405913
##   15  0.9719525  0.9392355
##   16  0.9713464  0.9378517
##   17  0.9725585  0.9405913
##   18  0.9719412  0.9392854
##   19  0.9713352  0.9379938
##   20  0.9713352  0.9379938
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 17.
```

```
plot(fit.knn)
```



We can see again that tuning has made little difference, settling on a value of k = 7 with an

accuracy of 97.19%. This is higher than the previous 97.14%, but very similar (or perhaps identical!) to the result achieved by the tuned SVM.

10.9 Ensemble

```
# 10-fold cross-validation with 3 repeats
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"

# Bagged CART
set.seed(7)
fit.treebag <- train(Class~., data=dataset, method="treebag", metric=metric,
                     trControl=trainControl, na.action=na.omit)

# Random Forest
set.seed(7)
fit.rf <- train(Class~., data=dataset, method="rf", metric=metric,
                preProc=c("BoxCox"), trControl=trainControl, na.action=na.omit)

# Stochastic Gradient Boosting
set.seed(7)
fit.gbm <- train(Class~., data=dataset, method="gbm", metric=metric,
                 preProc=c("BoxCox"), trControl=trainControl, verbose=FALSE, na.action=na.omit)

# C5.0
set.seed(7)
fit.c50 <- train(Class~., data=dataset, method="C5.0", metric=metric,
                 preProc=c("BoxCox"), trControl=trainControl, na.action=na.omit)
```

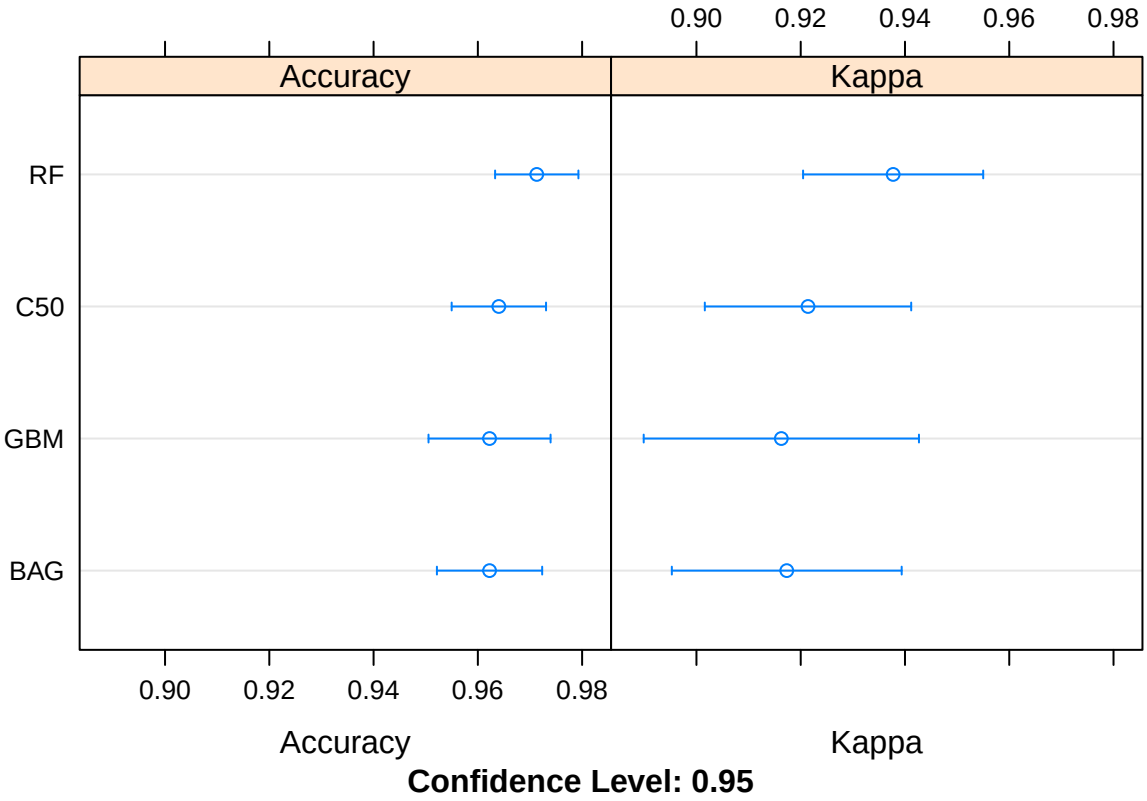
```
## Warning: 'trials' should be <= 7 for this object. Predictions generated
## using 7 trials
```

```
# Compare results
ensembleResults <- resamples(list(BAG = fit.treebag,
                                  RF  = fit.rf,
                                  GBM = fit.gbm,
                                  C50 = fit.c50))
summary(ensembleResults)
```

```
##
## Call:
## summary.resamples(object = ensembleResults)
##
## Models: BAG, RF, GBM, C50
## Number of resamples: 30
##
## Accuracy
##          Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## BAG 0.8727273 0.9454545 0.9632997 0.9622439 0.9818182    1    0
## RF  0.9259259 0.9629630 0.9725589 0.9713015 0.9818182    1    0
## GBM 0.8727273 0.9498316 0.9636364 0.9622439 0.9818182    1    0
## C50 0.8909091 0.9498316 0.9636364 0.9640284 0.9818182    1    0
##
```

```
## Kappa
##          Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## BAG 0.7286822 0.8808664 0.9207048 0.9173235 0.9597332    1    0
## RF  0.8414097 0.9207048 0.9402224 0.9377315 0.9602888    1    0
## GBM 0.7076689 0.8900351 0.9207048 0.9162823 0.9601869    1    0
## C50 0.7646220 0.8903491 0.9207048 0.9214000 0.9602888    1    0
```

```
dotplot(ensembleResults)
```



We see that Random Forest was the most accurate with a score of 97.26%. Very similar to our tuned models above. We could spend time tuning the parameters of Random Forest (e.g. increasing the number of trees) and the other ensemble methods, but I don't expect to see better accuracy scores other than random statistical fluctuations.

10.10 Finalize model

We now need to finalize the model, which really means choose which model we would like to use. For simplicity I would probably select the KNN method, at the expense of the memory required to store the training dataset. SVM would be a good choice to trade-off space and time complexity. I probably would not select the Random Forest algorithm given the complexity of the model. It seems overkill for this dataset, lots of trees with little benefit in Accuracy.

Let's go with the KNN algorithm. This is really simple, as we do not need to store a model. We do need to capture the parameters of the Box-Cox transform though. And we also need to prepare the data by removing the unused Id attribute and converting all of the inputs to numeric format.

The implementation of KNN (knn3()) belongs to the caret package and does not support missing values. We will have to remove the rows with missing values from the training dataset as well as the validation dataset. The code below shows the preparation of the pre-processing parameters using the training dataset.

```
# prepare parameters for data transform
set.seed(7)

datasetNoMissing <- dataset[complete.cases(dataset),]
x <- datasetNoMissing[,1:9]

# transform
preprocessParams <- preProcess(x, method=c("BoxCox"))
x <- predict(preprocessParams, x)
```

10.11 Prepare the validation set

Next we need to prepare the validation dataset for making a prediction. We must:

1. Remove the Id attribute.
2. Remove those rows with missing data.
3. Convert all input attributes to numeric.
4. Apply the Box-Cox transform to the input attributes using parameters prepared on the training dataset.

```
# prepare the validation dataset
set.seed(7)

# remove id column
validation <- validation[,-1]

# remove missing values (not allowed in this implementation of knn)
validation <- validation[complete.cases(validation),]

# convert to numeric
for(i in 1:9) {
    validation[,i] <- as.numeric(as.character(validation[,i]))
}

# transform the validation dataset
validationX <- predict(preprocessParams, validation[,1:9])
```

```
# make predictions
set.seed(7)
# knn3Train(train, test, cl, k = 1, l = 0, prob = TRUE, use.all = TRUE)
# k: number of neighbours considered.
predictions <- knn3Train(x, validationX, datasetNoMissing$Class,
                         k = 9,
                         prob = FALSE)

# convert
confusionMatrix(as.factor(predictions), validation$Class)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  benign malignant
##   benign        87         0
##   malignant      1        48
```

```
##
##                Accuracy : 0.9926
##                  95% CI : (0.9597, 0.9998)
##     No Information Rate : 0.6471
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.984
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9886
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9796
##              Prevalence : 0.6471
##          Detection Rate : 0.6397
##    Detection Prevalence : 0.6397
##       Balanced Accuracy : 0.9943
##
##        'Positive' Class : benign
##
```

We can see that the accuracy of the final model on the validation dataset is 99.26%. This is optimistic because there is only 136 rows, but it does show that we have an accurate standalone model that we could use on other unclassified data.

Chapter 11

SMS spam. Naive Bayes. Classification

Dataset: https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/sms_spam.csv

Instructions: Machine Learning with R. Page 104.

```
sms_raw <- read.csv(file.path(data_raw_dir, "sms_spam.csv"), stringsAsFactors = FALSE)
```

```
str(sms_raw)
```

```
#> 'data.frame':    5574 obs. of  2 variables:
#>  $ type: chr  "ham" "ham" "spam" "ham" ...
#>  $ text: chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... C
```

11.0.1 convert type to a factor

```
sms_raw$type <- factor(sms_raw$type)
```

```
str(sms_raw$type)
```

```
#>  Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
```

```
table(sms_raw$type)
```

```
#>
#>  ham spam
#> 4827  747
```

```
library(tm)

sms_corpus <- VCorpus(VectorSource(sms_raw$text))
print(sms_corpus)
```

```
#> <<VCorpus>>
#> Metadata:  corpus specific: 0, document level (indexed): 0
#> Content:  documents: 5574
```

```
inspect(sms_corpus[1:2])
```

```
#> <<VCorpus>>
```

```
#> Metadata:  corpus specific: 0, document level (indexed): 0
#> Content:  documents: 2
#>
#> [[1]]
#> <<PlainTextDocument>>
#> Metadata:  7
#> Content:  chars: 111
#>
#> [[2]]
#> <<PlainTextDocument>>
#> Metadata:  7
#> Content:  chars: 29
```

```
# show some text
as.character(sms_corpus[[1]])
```

```
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
```

```
# show three documents
lapply(sms_corpus[1:3], as.character)
```

```
#> $`1`
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
#>
#> $`2`
#> [1] "Ok lar... Joking wif u oni..."
#>
#> $`3`
#> [1] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive
```

```
# convert to lowercase
sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))
```

```
as.character(sms_corpus[[1]])
```

```
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
```

```
# converted to lowercase
as.character(sms_corpus_clean[[1]])
```

```
#> [1] "go until jurong point, crazy.. available only in bugis n great world la e buffet... cine there
```

```
# remove numbers
sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)
```

```
# what transformations are available
getTransformations()
```

```
#> [1] "removeNumbers"     "removePunctuation" "removeWords"
#> [4] "stemDocument"      "stripWhitespace"
```

```
# remove stop words
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords())
```

```
# remove punctuation
sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)
```

```
library(SnowballC)
wordStem(c("learn", "learned", "learning", "learns"))
```

```
#> [1] "learn" "learn" "learn" "learn"
```

```
# stemming corpus
sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)
```

```
# remove white spaces
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)
```

```
# show what we've got so far
lapply(sms_corpus[1:3], as.character)
```

```
#> $`1`
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
#>
#> $`2`
#> [1] "Ok lar... Joking wif u oni..."
#>
#> $`3`
#> [1] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive
```

```
lapply(sms_corpus_clean[1:3], as.character)
```

```
#> $`1`
#> [1] "go jurong point crazi avail bugi n great world la e buffet cine got amor wat"
#>
#> $`2`
#> [1] "ok lar joke wif u oni"
#>
#> $`3`
#> [1] "free entri wkli comp win fa cup final tkts st may text fa receiv entri questionstd txt ratetc ap
```

11.1 Convert to Document Term Matrix

```
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
sms_dtm
```

```
#> <<DocumentTermMatrix (documents: 5574, terms: 6592)>>
#> Non-/sparse entries: 42608/36701200
#> Sparsity           : 100%
#> Maximal term length: 40
#> Weighting          : term frequency (tf)
```

11.2 split in training and test datasets

```
sms_dtm_train <- sms_dtm[1:4169, ]
sms_dtm_test  <- sms_dtm[4170:5559, ]
```

11.2.1 separate the labels

```
sms_train_labels <- sms_raw[1:4169, ]$type
sms_test_labels  <- sms_raw[4170:5559, ]$type
```

```
prop.table(table(sms_train_labels))
```

```
#> sms_train_labels
#>       ham      spam
#> 0.8647158 0.1352842
```

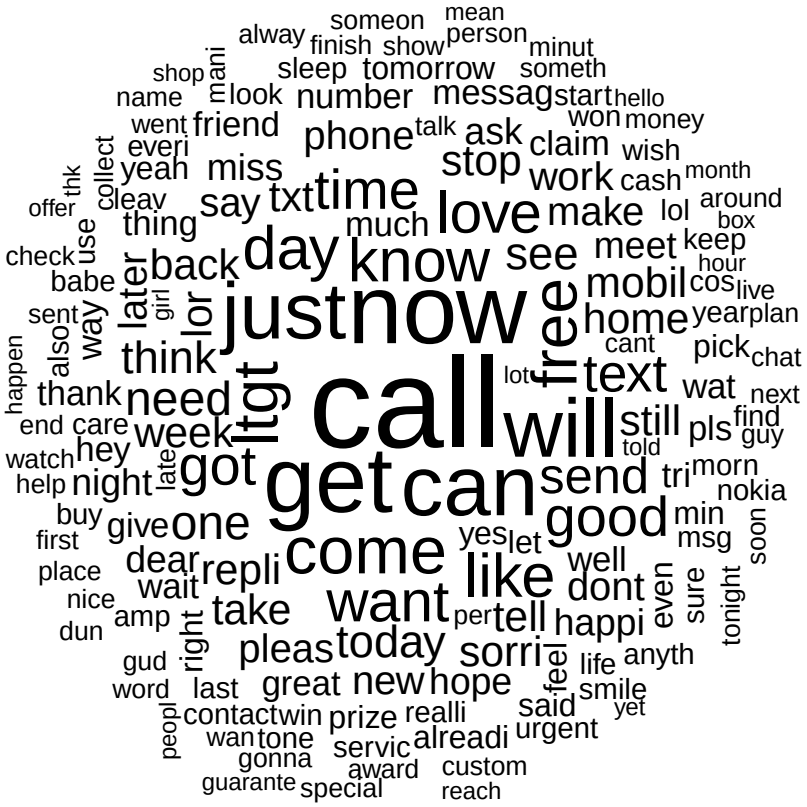
```
prop.table(table(sms_test_labels))
```

```
#> sms_test_labels
#>       ham      spam
#> 0.8697842 0.1302158
```

```
# convert dtm to matrix
sms_mat_train <- as.matrix(t(sms_dtm_train))
dtm.rs <- sort(rowSums(sms_mat_train), decreasing=TRUE)

# dataframe with word-frequency
dtm.df <- data.frame(word = names(dtm.rs), freq = as.integer(dtm.rs),
                     stringsAsFactors = FALSE)
```

11.3 plot wordcloud

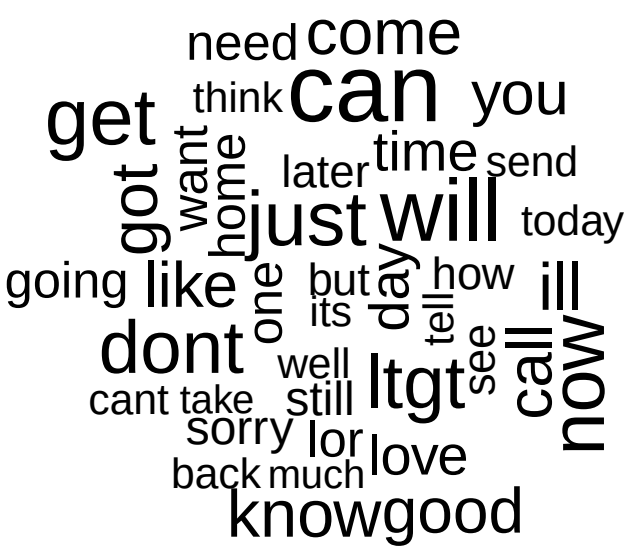
```
library(wordcloud)
wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE)
```



```
spam <- subset(sms_raw, type == "spam")
ham  <- subset(sms_raw, type == "ham")
```

```
wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
```


```
wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```



11.4 Limit Frequent words

```
# words that appear at least in 5 messages
sms_freq_words <- findFreqTerms(sms_dtm_train, 6)
```

```
str(sms_freq_words)
```

```
#>  chr [1:997] "abiola" "abl" "abt" "accept" "access" "account" "across" ...
```

11.4.1 get only frequent words

```
sms_dtm_freq_train<- sms_dtm_train[ , sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
```

11.4.2 function to change value to Yes/No

```
convert_counts <- function(x) {
    x <- ifelse(x > 0, "Yes", "No")
  }
```

```
# change from number to Yes/No
# also the result returns a matrix
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2,
                                        convert_counts)
sms_test  <- apply(sms_dtm_freq_test, MARGIN = 2,
                                       convert_counts)
```

```
# matrix of
# 4169 documents as rows
# 1159 terms as columns
dim(sms_train)
```

```
#> [1] 4169  997
```

```
length(sms_train_labels)
```

```
#> [1] 4169
```

```
# this is how the matrix looks
sms_train[1:10, 10:15]
```

```
#>     Terms
#> Docs add  address admir advanc aft  afternoon
#>   1  "No" "No"    "No"  "No"   "No" "No"
#>   2  "No" "No"    "No"  "No"   "No" "No"
#>   3  "No" "No"    "No"  "No"   "No" "No"
#>   4  "No" "No"    "No"  "No"   "No" "No"
#>   5  "No" "No"    "No"  "No"   "No" "No"
#>   6  "No" "No"    "No"  "No"   "No" "No"
#>   7  "No" "No"    "No"  "No"   "No" "No"
#>   8  "No" "No"    "No"  "No"   "No" "No"
#>   9  "No" "No"    "No"  "No"   "No" "No"
#>   10 "No" "No"    "No"  "No"   "No" "No"
```

```
library(e1071)
sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

```
sms_test_pred <- predict(sms_classifier, sms_test)
```

```
library(gmodels)
CrossTable(sms_test_pred, sms_test_labels,
    prop.chisq = FALSE, prop.t = FALSE,
    dnn = c('predicted', 'actual'))
```

```
#>
```

```
#>
#>    Cell Contents
#> |-------------------------|
#> |                       N |
#> |           N / Row Total |
#> |           N / Col Total |
#> |-------------------------|
#>
#>
#> Total Observations in Table:  1390
#>
#>
#>              | actual
#>    predicted |       ham |      spam | Row Total |
#> -------------|-----------|-----------|-----------|
#>          ham |      1202 |        21 |      1223 |
#>              |     0.983 |     0.017 |     0.880 |
#>              |     0.994 |     0.116 |           |
#> -------------|-----------|-----------|-----------|
#>         spam |         7 |       160 |       167 |
#>              |     0.042 |     0.958 |     0.120 |
#>              |     0.006 |     0.884 |           |
#> -------------|-----------|-----------|-----------|
#> Column Total |      1209 |       181 |      1390 |
#>              |     0.870 |     0.130 |           |
#> -------------|-----------|-----------|-----------|
#>
#>
```

Misclassified: 20+9 (frequency = 5) 25+7 (freq=4) 23+7 (freq=3) 25+8 (freq=2) 21+7 (freq=6)

Decreasing the minimum word frequency doesn't make the model better.

11.5 Improve model performance

```
sms_classifier2 <- naiveBayes(sms_train, sms_train_labels,
                              laplace = 1)
```

```
sms_test_pred2 <- predict(sms_classifier2, sms_test)
```

```
CrossTable(sms_test_pred2, sms_test_labels,
    prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
    dnn = c('predicted', 'actual'))
```

```
#>
#>
#>    Cell Contents
#> |-------------------------|
#> |                       N |
#> |           N / Col Total |
#> |-------------------------|
#>
#>
#> Total Observations in Table:  1390
```

```
#>
#>
#>              | actual
#>    predicted |       ham |      spam | Row Total |
#> -------------|-----------|-----------|-----------|
#>          ham |      1203 |        28 |      1231 |
#>              |     0.995 |     0.155 |           |
#> -------------|-----------|-----------|-----------|
#>         spam |         6 |       153 |       159 |
#>              |     0.005 |     0.845 |           |
#> -------------|-----------|-----------|-----------|
#> Column Total |      1209 |       181 |      1390 |
#>              |     0.870 |     0.130 |           |
#> -------------|-----------|-----------|-----------|
#>
#>
```

Misclassified: 28+7

Chapter 12

Classification Tree: Vehicle example

Dataset: Vehicle (mlbench) Instructions: book "Applied Predictive Modeling Techniques", Lewis, N.D.

12.1 Load packages

```
library(tree)
library(mlbench)

data(Vehicle)
str(Vehicle)
```

```
#> 'data.frame':    846 obs. of  19 variables:
#>  $ Comp        : num  95 91 104 93 85 107 97 90 86 93 ...
#>  $ Circ        : num  48 41 50 41 44 57 43 43 34 44 ...
#>  $ D.Circ      : num  83 84 106 82 70 106 73 66 62 98 ...
#>  $ Rad.Ra      : num  178 141 209 159 205 172 173 157 140 197 ...
#>  $ Pr.Axis.Ra  : num  72 57 66 63 103 50 65 65 61 62 ...
#>  $ Max.L.Ra    : num  10 9 10 9 52 6 6 9 7 11 ...
#>  $ Scat.Ra     : num  162 149 207 144 149 255 153 137 122 183 ...
#>  $ Elong       : num  42 45 32 46 45 26 42 48 54 36 ...
#>  $ Pr.Axis.Rect: num  20 19 23 19 19 28 19 18 17 22 ...
#>  $ Max.L.Rect  : num  159 143 158 143 144 169 143 146 127 146 ...
#>  $ Sc.Var.Maxis: num  176 170 223 160 241 280 176 162 141 202 ...
#>  $ Sc.Var.maxis: num  379 330 635 309 325 957 361 281 223 505 ...
#>  $ Ra.Gyr      : num  184 158 220 127 188 264 172 164 112 152 ...
#>  $ Skew.Maxis  : num  70 72 73 63 127 85 66 67 64 64 ...
#>  $ Skew.maxis  : num  6 9 14 6 9 5 13 3 2 4 ...
#>  $ Kurt.maxis  : num  16 14 9 10 11 9 1 3 14 14 ...
#>  $ Kurt.Maxis  : num  187 189 188 199 180 181 200 193 200 195 ...
#>  $ Holl.Ra     : num  197 199 196 207 183 183 204 202 208 204 ...
#>  $ Class       : Factor w/ 4 levels "bus","opel","saab",..: 4 4 3 4 1 1 1 4 4 3 ...
```

```
summary(Vehicle[1])
```

```
#>       Comp
#>  Min.   : 73.00
#>  1st Qu.: 87.00
#>  Median : 93.00
```

```
#>  Mean   : 93.68
#>  3rd Qu.:100.00
#>  Max.   :119.00
```

```
summary(Vehicle[2])
```

```
#>       Circ
#>  Min.   :33.00
#>  1st Qu.:40.00
#>  Median :44.00
#>  Mean   :44.86
#>  3rd Qu.:49.00
#>  Max.   :59.00
```

```
attributes(Vehicle$Class)
```

```
#> $levels
#> [1] "bus"  "opel" "saab" "van"
#>
#> $class
#> [1] "factor"
```

12.2 Prepare data

```
set.seed(107)
N = nrow(Vehicle)
train <- sample(1:N, 500, FALSE)
```

```
# training and test sets
trainset <- Vehicle[train,]
testset  <- Vehicle[-train,]
```

12.3 Estimate the decision tree

```
fit <- tree(Class ~., data = trainset, split ="deviance")
fit
```

```
#> node), split, n, deviance, yval, (yprob)
#>       * denotes terminal node
#>
#>   1) root 500 1386.000 saab ( 0.248000 0.254000 0.258000 0.240000 )
#>     2) Elong < 41.5 229  489.100 opel ( 0.222707 0.410480 0.366812 0.000000 )
#>       4) Max.L.Ra < 7.5 62   77.560 bus ( 0.806452 0.096774 0.096774 0.000000 )
#>         8) Comp < 95.5 20   43.560 bus ( 0.400000 0.300000 0.300000 0.000000 )
#>          16) Pr.Axis.Ra < 67 12   16.640 opel ( 0.000000 0.500000 0.500000 0.000000 ) *
#>          17) Pr.Axis.Ra > 67 8    0.000 bus ( 1.000000 0.000000 0.000000 0.000000 ) *
#>         9) Comp > 95.5 42    0.000 bus ( 1.000000 0.000000 0.000000 0.000000 ) *
#>       5) Max.L.Ra > 7.5 167  241.800 opel ( 0.005988 0.526946 0.467066 0.000000 )
#>        10) Comp < 107.5 145  205.300 opel ( 0.006897 0.600000 0.393103 0.000000 )
#>          20) Sc.Var.maxis < 720.5 131  179.400 opel ( 0.000000 0.564885 0.435115 0.000000 ) *
#>          21) Sc.Var.maxis > 720.5 14    7.205 opel ( 0.071429 0.928571 0.000000 0.000000 ) *
#>        11) Comp > 107.5 22    8.136 saab ( 0.000000 0.045455 0.954545 0.000000 ) *
```

```
#>     3) Elong > 41.5 271  687.600 van ( 0.269373 0.121771 0.166052 0.442804 )
#>       6) Max.L.Ra < 8.5 200  537.900 bus ( 0.360000 0.165000 0.210000 0.265000 )
#>        12) Sc.Var.maxis < 297.5 86  168.200 van ( 0.000000 0.186047 0.244186 0.569767 )
#>          24) Max.L.Rect < 127.5 25   48.720 saab ( 0.000000 0.280000 0.560000 0.160000 ) *
#>          25) Max.L.Rect > 127.5 61   92.130 van ( 0.000000 0.147541 0.114754 0.737705 ) *
#>        13) Sc.Var.maxis > 297.5 114  228.700 bus ( 0.631579 0.149123 0.184211 0.035088 )
#>          26) D.Circ < 76.5 92  132.300 bus ( 0.782609 0.065217 0.130435 0.021739 )
#>            52) Skew.maxis < 10.5 84   89.720 bus ( 0.857143 0.023810 0.095238 0.023810 )
#>             104) Circ < 40.5 16   36.580 saab ( 0.312500 0.125000 0.500000 0.062500 )
#>               208) Kurt.Maxis < 191 8    6.028 saab ( 0.000000 0.000000 0.875000 0.125000 ) *
#>               209) Kurt.Maxis > 191 8   14.400 bus ( 0.625000 0.250000 0.125000 0.000000 ) *
#>             105) Circ > 40.5 68   10.420 bus ( 0.985294 0.000000 0.000000 0.014706 ) *
#>            53) Skew.maxis > 10.5 8   11.090 saab ( 0.000000 0.500000 0.500000 0.000000 ) *
#>          27) D.Circ > 76.5 22   40.930 opel ( 0.000000 0.500000 0.409091 0.090909 ) *
#>       7) Max.L.Ra > 8.5 71   35.280 van ( 0.014085 0.000000 0.042254 0.943662 )
#>        14) Skew.Maxis < 64.5 7    9.561 van ( 0.000000 0.000000 0.428571 0.571429 ) *
#>        15) Skew.Maxis > 64.5 64   10.300 van ( 0.015625 0.000000 0.000000 0.984375 ) *
```

```
# fit <- tree(Class ~., data = Vehicle[train,], split ="deviance")
# fit
```

We use deviance as the splitting criteria, a common alternative is to use split="gini".

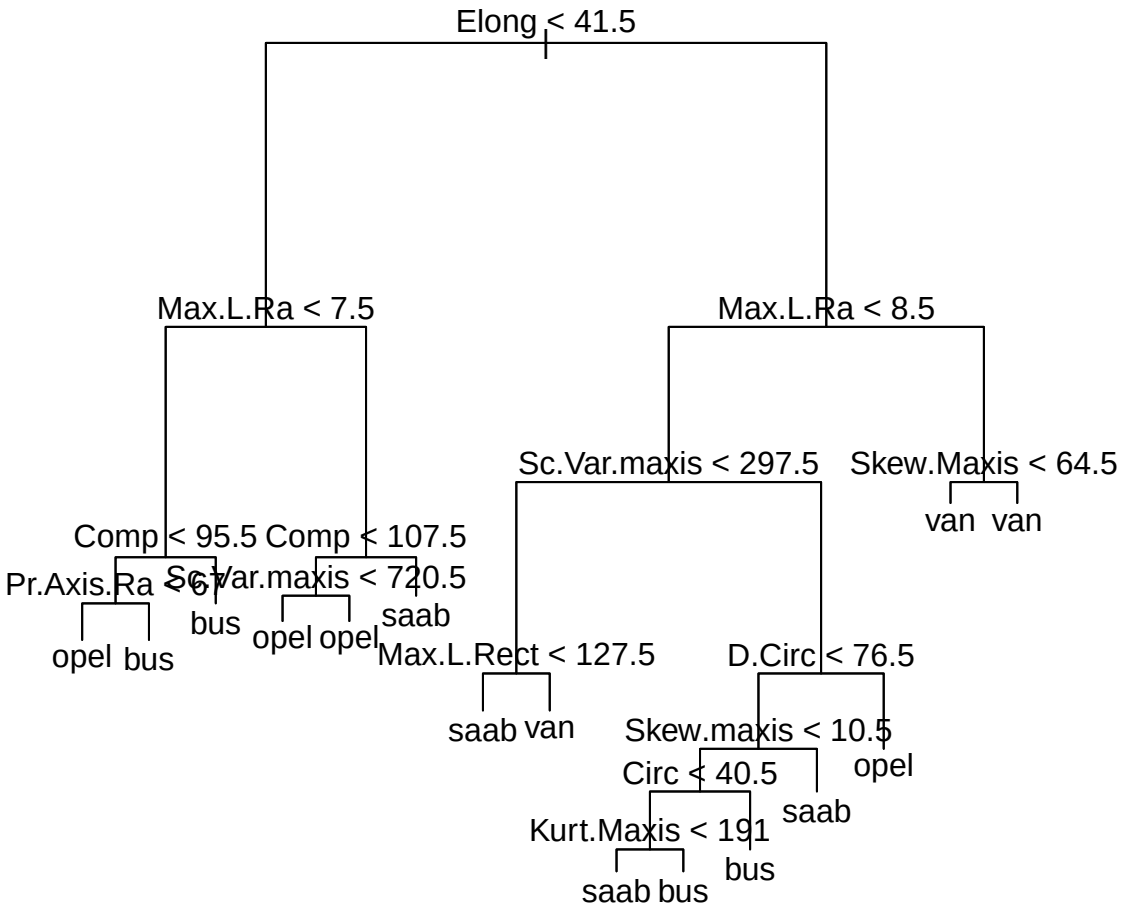
At each branch of the tree (after root) we see in order: 1. The branch number (e.g. in this case 1,2,14 and 15); 2. the split (e.g. Elong $<$ 41.5); 3. the number of samples going along that split (e.g. 229); 4. the deviance associated with that split (e.g. 489.1); 5. the predicted class (e.g. opel); 6. the associated probabilities (e.g. (0.222707 0.410480 0.366812 0.000000); 7. and for a terminal node (or leaf), the symbol "*".

```
summary(fit)
```

```
#>
#> Classification tree:
#> tree(formula = Class ~ ., data = trainset, split = "deviance")
#> Variables actually used in tree construction:
#>  [1] "Elong"        "Max.L.Ra"     "Comp"         "Pr.Axis.Ra"
#>  [5] "Sc.Var.maxis" "Max.L.Rect"   "D.Circ"       "Skew.maxis"
#>  [9] "Circ"         "Kurt.Maxis"   "Skew.Maxis"
#> Number of terminal nodes:  15
#> Residual mean deviance:  0.9381 = 455 / 485
#> Misclassification error rate: 0.232 = 116 / 500
```

Notice that summary(fit) shows: 1. The type of tree, in this case a Classification tree; 2. the formula used to fit the tree; 3. the variables used to fit the tree; 4. the number of terminal nodes in this case 15; 5. the residual mean deviance - 0.9381; 6. the misclassification error rate 0.232 or 23.2%.

```
plot(fit); text(fit)
```



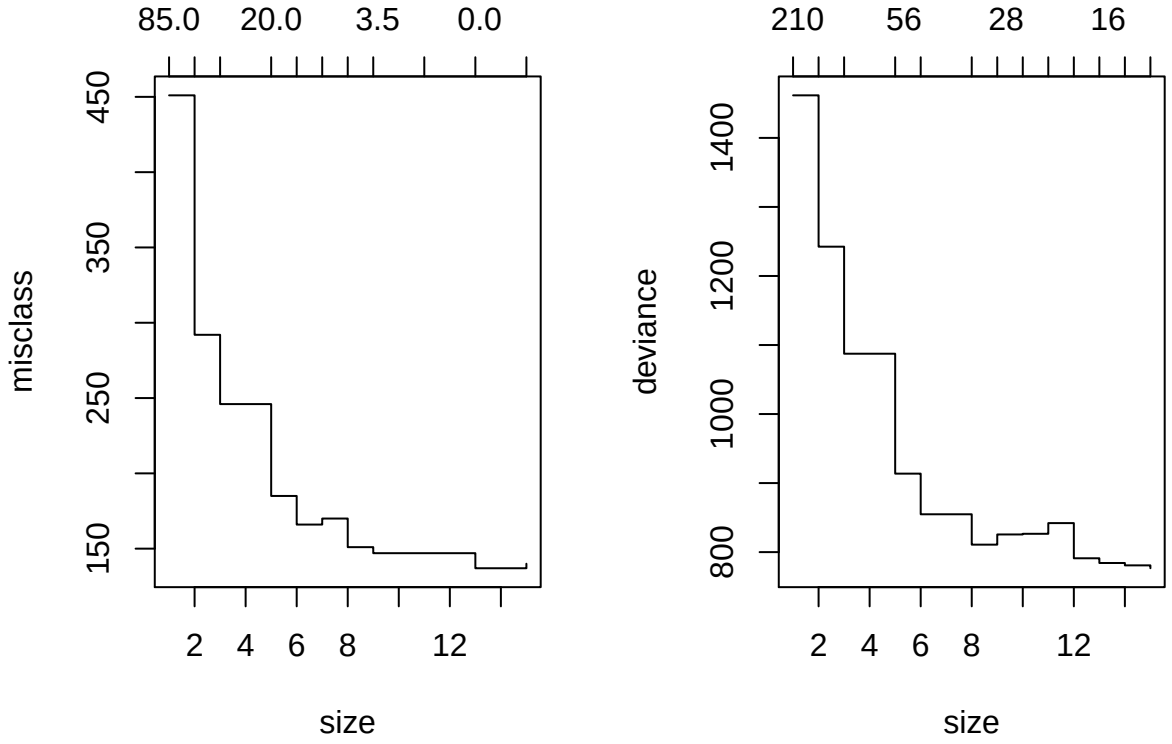
12.4 Assess model

Unfortunately, classification trees have a tendency to overfit the data. One approach to reduce this risk is to use cross-validation. For each hold out sample we fit the model and note at what level the tree gives the best results (using deviance or the misclassification rate). Then we hold out a different sample and repeat. This can be carried out using the `cv.tree()` function. We use a leave-one-out cross-validation using the misclassification rate and deviance (`FUN=prune.misclass`, followed by `FUN=prune.tree`).

```
fitM.cv <- cv.tree(fit, K=346, FUN = prune.misclass)
fitP.cv <- cv.tree(fit, K=346, FUN = prune.tree)
```

The results are plotted out side by side in Figure 1.2. The jagged lines shows where the minimum deviance / misclassification occurred with the cross-validated tree. Since the cross validated misclassification and deviance both reach their minimum close to the number of branches in the original fitted tree there is little to be gained from pruning this tree

```
par(mfrow = c(1, 2))
plot(fitM.cv)
plot(fitP.cv)
```



12.5 Make predictions

We use the validation data set and the fitted decision tree to predict vehicle classes; then we display the confusion matrix and calculate the error rate of the fitted tree. Overall, the model has an error rate of 32%.

```
testLabels <- Vehicle$Class[-train]
testLabels
```

```
#>   [1] bus  bus  saab saab bus  saab opel bus  saab bus  saab opel van  van
#>  [15] van  bus  van  saab bus  bus  van  opel opel opel bus  bus  van  bus
#>  [29] van  bus  opel van  bus  bus  opel bus  bus  van  bus  bus  opel opel
#>  [43] van  saab bus  bus  saab van  van  van  bus  saab van  saab opel opel
#>  [57] opel saab bus  van  opel opel opel saab bus  van  opel bus  van  bus
#>  [71] opel saab bus  bus  opel van  saab bus  opel opel van  van  bus  bus
#>  [85] opel van  saab van  van  saab van  saab van  bus  opel bus  saab bus
#>  [99] opel van  saab opel bus  van  saab van  bus  bus  bus  saab van  saab
#> [113] van  bus  bus  saab opel bus  saab opel van  van  bus  saab saab bus
#> [127] van  opel van  bus  saab van  bus  saab opel bus  opel opel van  opel
#> [141] van  bus  opel van  bus  bus  opel opel saab van  van  van  bus  bus
#> [155] opel bus  bus  saab opel bus  van  saab opel van  saab bus  saab opel
#> [169] van  bus  van  bus  opel opel saab van  opel opel bus  opel opel opel
#> [183] opel opel opel bus  opel van  opel opel saab opel van  opel saab saab
#> [197] van  saab saab saab opel bus  saab saab van  van  saab bus  van  saab
#> [211] opel bus  saab bus  saab opel opel saab saab saab saab van  saab saab
#> [225] opel van  bus  van  opel opel bus  van  saab van  opel bus  opel opel
#> [239] van  van  van  saab bus  saab bus  saab bus  opel bus  van  opel bus
#> [253] opel bus  saab van  opel saab bus  opel bus  bus  bus  van  bus  van
#> [267] bus  bus  opel saab saab bus  van  bus  saab opel van  opel saab opel
#> [281] bus  saab saab saab saab van  opel van  van  saab opel bus  saab bus
```

```
#> [295] saab bus  saab van  saab van  saab saab van  opel saab saab bus  saab
#> [309] opel bus  saab bus  saab van  van  saab van  bus  saab van  saab saab
#> [323] bus  opel opel opel bus  opel saab bus  saab van  saab opel saab opel
#> [337] opel opel van  bus  bus  bus  saab opel saab van
#> Levels: bus opel saab van
```

```
# Confusion Matrix
pred <- predict(fit, newdata = testset)
# find column whih has the maximum of all rows
pred.class <- colnames(pred)[max.col(pred, ties.method = c("random"))]
cm <- table(testLabels, pred.class,
      dnn = c("Observed Class", "Predicted Class"))
cm
```

```
#>               Predicted Class
#> Observed Class bus opel saab van
#>           bus   86    1    3   4
#>           opel   1   55   20   9
#>           saab   4   55   23   6
#>           van    2    2    5  70
```

```
# Sensitivity
sum(diag(cm)) / sum(cm)
```

```
#> [1] 0.6763006
```

```
# pred <- predict(fit, newdata = Vehicle[-train,])
# pred.class <- colnames(pred)[max.col(pred, ties.method = c("random"))]
# table(Vehicle$Class[-train], pred.class,
#       dnn = c("Observed Class", "Predicted Class"))
```

```
error_rate = (1 - sum(pred.class == testset) / nrow(testset))
round(error_rate, 3)
```

```
#> [1] 0.324
```

```
# error_rate = (1 - sum(pred.class == Vehicle$Class[-train])/346)
# round(error_rate,3)
```

Chapter 13

Bike sharing demand

```
#loading the required libraries
library(rpart)
library(rattle)
library(rpart.plot)
library(RColorBrewer)
library(randomForest)
library(corrplot)
library(dplyr)
```

Source: https://www.analyticsvidhya.com/blog/2015/06/solution-kaggle-competition-bike-sharing-demand/

Chapter 14

Step 1. Hypothesis Generation

Before exploring the data to understand the relationship between variables, I'd recommend you to focus on hypothesis generation first. Now, this might sound counter-intuitive for solving a data science problem, but if there is one thing I have learnt over years, it is this. Before exploring data, you should spend some time thinking about the business problem, gaining the domain knowledge and may be gaining first hand experience of the problem (only if I could travel to North America!)

How does it help? This practice usually helps you form better features later on, which are not biased by the data available in the dataset. At this stage, you are expected to posses structured thinking i.e. a thinking process which takes into consideration all the possible aspects of a particular problem.

Here are some of the hypothesis which I thought could influence the demand of bikes:

- **Hourly trend**: There must be high demand during office timings. Early morning and late evening can have different trend (cyclist) and low demand during 10:00 pm to 4:00 am.
- **Daily Trend**: Registered users demand more bike on weekdays as compared to weekend or holiday.
- **Rain**: The demand of bikes will be lower on a rainy day as compared to a sunny day. Similarly, higher humidity will cause to lower the demand and vice versa.
- **Temperature**: Would high or low temperature encourage or disencourage bike riding?
- **Pollution**: If the pollution level in a city starts soaring, people may start using Bike (it may be influenced by government / company policies or increased awareness).
- **Time**: Total demand should have higher contribution of registered user as compared to casual because registered user base would increase over time.
- **Traffic**: It can be positively correlated with Bike demand. Higher traffic may force people to use bike as compared to other road transport medium like car, taxi etc

Chapter 15

2. Understanding the Data Set

The dataset shows hourly rental data for two years (2011 and 2012). The training data set is for the **first 19 days of each month**. The test dataset is from **20th day to month's end**. We are required to predict the total count of bikes rented during each hour covered by the test set.

In the training data set, they have separately given bike demand by registered, casual users and sum of both is given as count.

Training data set has 12 variables (see below) and Test has 9 (excluding registered, casual and count).

15.0.1 Independent variables

```
datetime:   date and hour in "mm/dd/yyyy hh:mm" format
season:     Four categories-> 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday:    whether the day is a holiday or not (1/0)
workingday: whether the day is neither a weekend nor holiday (1/0)
weather:    Four Categories of weather
            1-> Clear, Few clouds, Partly cloudy, Partly cloudy
            2-> Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
            3-> Light Snow and Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
            4-> Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp:       hourly temperature in Celsius
atemp:      "feels like" temperature in Celsius
humidity:   relative humidity
windspeed:  wind speed
```

15.0.2 Dependent variables

```
registered: number of registered user
casual:     number of non-registered user
count:      number of total rentals (registered + casual)
```

Chapter 16

3. Importing the dataset and Data Exploration

For this solution, I have used R (R Studio 0.99.442) in Windows Environment.

Below are the steps to import and perform data exploration. If you are new to this concept, you can refer this guide on Data Exploration in R

1. Import Train and Test Data Set

```
# https://www.kaggle.com/c/bike-sharing-demand/data
train = read.csv(file.path(data_raw_dir, "bike_train.csv"))
test = read.csv(file.path(data_raw_dir, "bike_test.csv"))
```

```
glimpse(train)
```

```
#> Observations: 10,886
#> Variables: 12
#> $ datetime   <fct> 2011-01-01 00:00:00, 2011-01-01 01:00:00, 2011-01-0...
#> $ season     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ holiday    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ workingday <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ weather    <int> 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, ...
#> $ temp       <dbl> 9.84, 9.02, 9.02, 9.84, 9.84, 9.84, 9.02, 8.20, 9.8...
#> $ atemp      <dbl> 14.395, 13.635, 13.635, 14.395, 14.395, 12.880, 13....
#> $ humidity   <int> 81, 80, 80, 75, 75, 75, 80, 86, 75, 76, 76, 81, 77,...
#> $ windspeed  <dbl> 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 6.0032, 0.0...
#> $ casual     <int> 3, 8, 5, 3, 0, 0, 2, 1, 1, 8, 12, 26, 29, 47, 35, 4...
#> $ registered <int> 13, 32, 27, 10, 1, 1, 0, 2, 7, 6, 24, 30, 55, 47, 7...
#> $ count      <int> 16, 40, 32, 13, 1, 1, 2, 3, 8, 14, 36, 56, 84, 94, ...
```

```
glimpse(test)
```

```
#> Observations: 6,493
#> Variables: 9
#> $ datetime   <fct> 2011-01-20 00:00:00, 2011-01-20 01:00:00, 2011-01-2...
#> $ season     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ holiday    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ workingday <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ weather    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, ...
#> $ temp       <dbl> 10.66, 10.66, 10.66, 10.66, 10.66, 9.84, 9.02, 9.02...
```

```
#> $ atemp      <dbl> 11.365, 13.635, 13.635, 12.880, 12.880, 11.365, 10....
#> $ humidity   <int> 56, 56, 56, 56, 56, 60, 60, 55, 55, 52, 48, 45, 42,...
#> $ windspeed  <dbl> 26.0027, 0.0000, 0.0000, 11.0014, 11.0014, 15.0013,...
```

2. Combine both Train and Test Data set (to understand the distribution of independent variable together).

```
# add variables to test dataset before merging
test$registered=0
test$casual=0
test$count=0

data = rbind(train,test)
```

3. Variable Type Identification

```
str(data)
```

```
#> 'data.frame':    17379 obs. of  12 variables:
#>  $ datetime  : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 6 7 8 9 10 ...
#>  $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
#>  $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
#>  $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
#>  $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
#>  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
#>  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
#>  $ humidity  : int  81 80 80 75 75 75 80 86 75 76 ...
#>  $ windspeed : num  0 0 0 0 0 ...
#>  $ casual    : num  3 8 5 3 0 0 2 1 1 8 ...
#>  $ registered: num  13 32 27 10 1 1 0 2 7 6 ...
#>  $ count     : num  16 40 32 13 1 1 2 3 8 14 ...
```

4. Find missing values in the dataset if any

```
table(is.na(data))
```

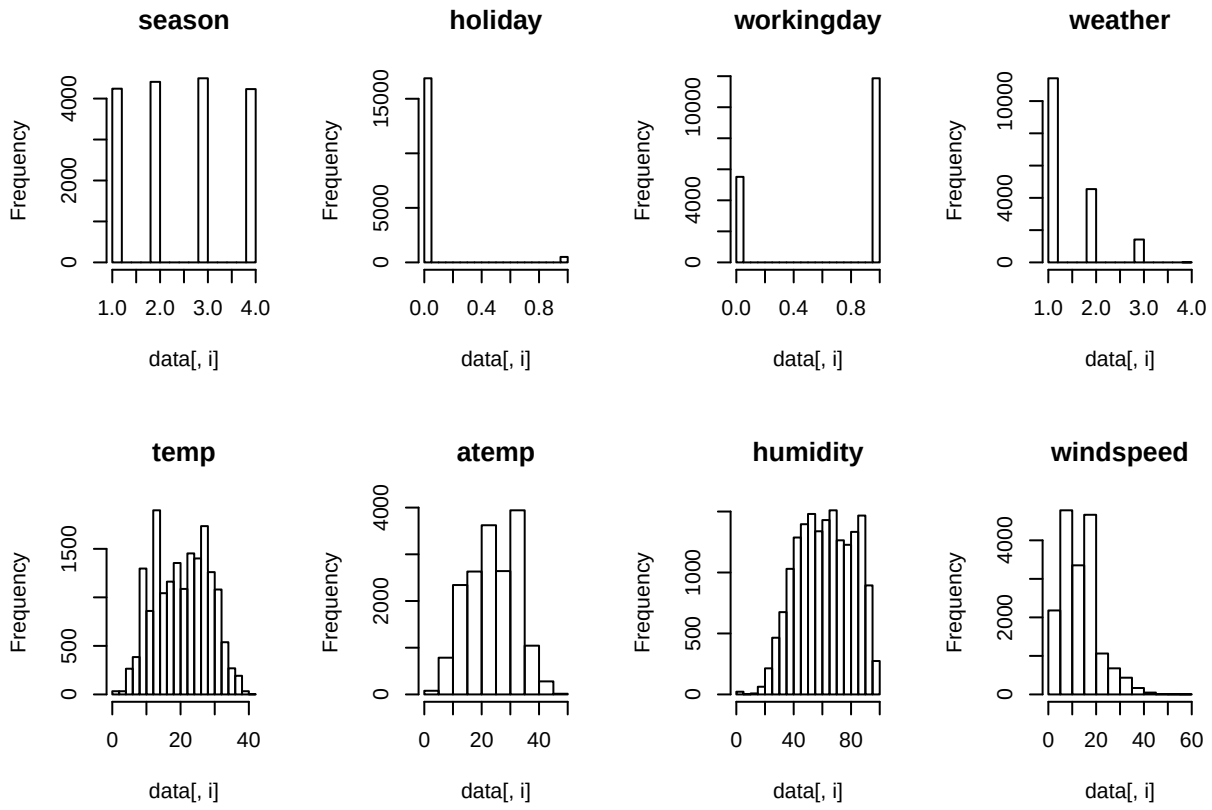
```
#>
#>   FALSE
#> 208548
```

No NAs in the dataset.

5. Understand the distribution of numerical variables and generate a frequency table for numeric variables. Analyze the distribution.

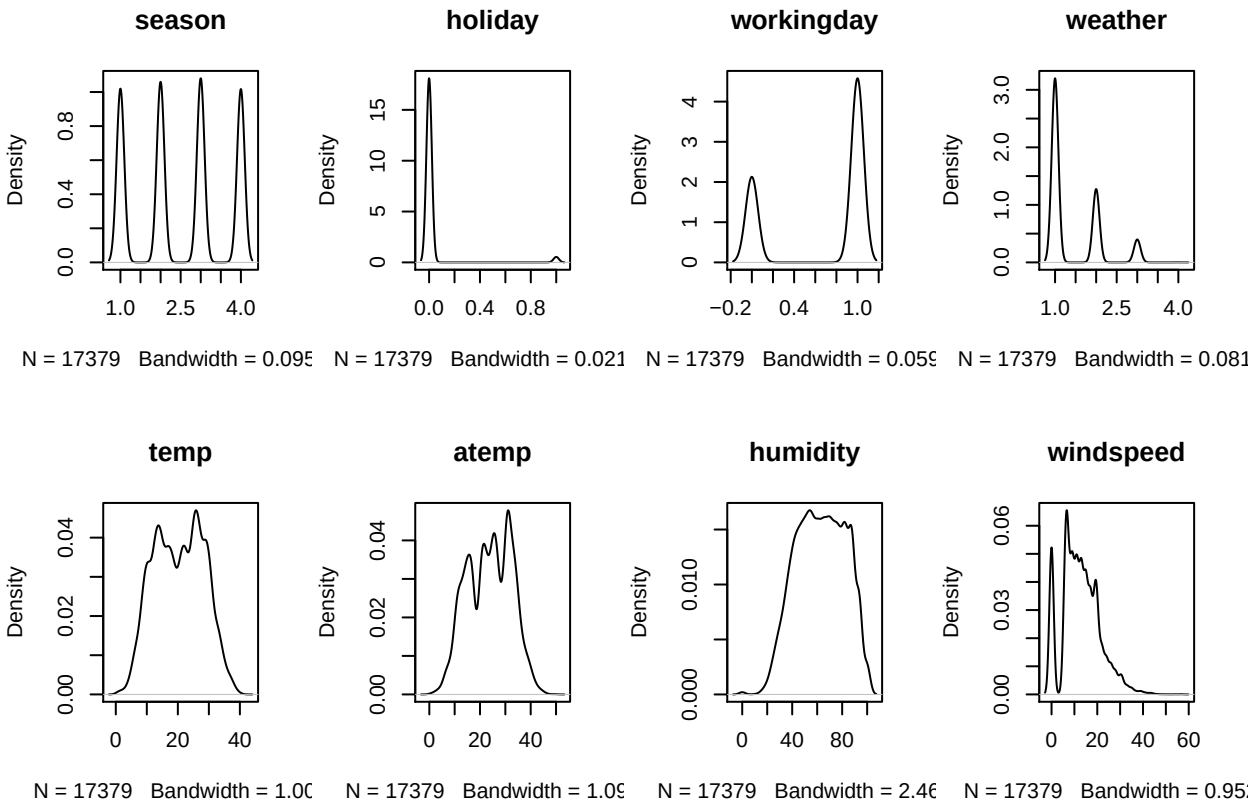
16.0.1 histograms

```
# histograms each attribute
par(mfrow=c(2,4))
for(i in 2:9) {
    hist(data[,i], main = names(data)[i])
}
```



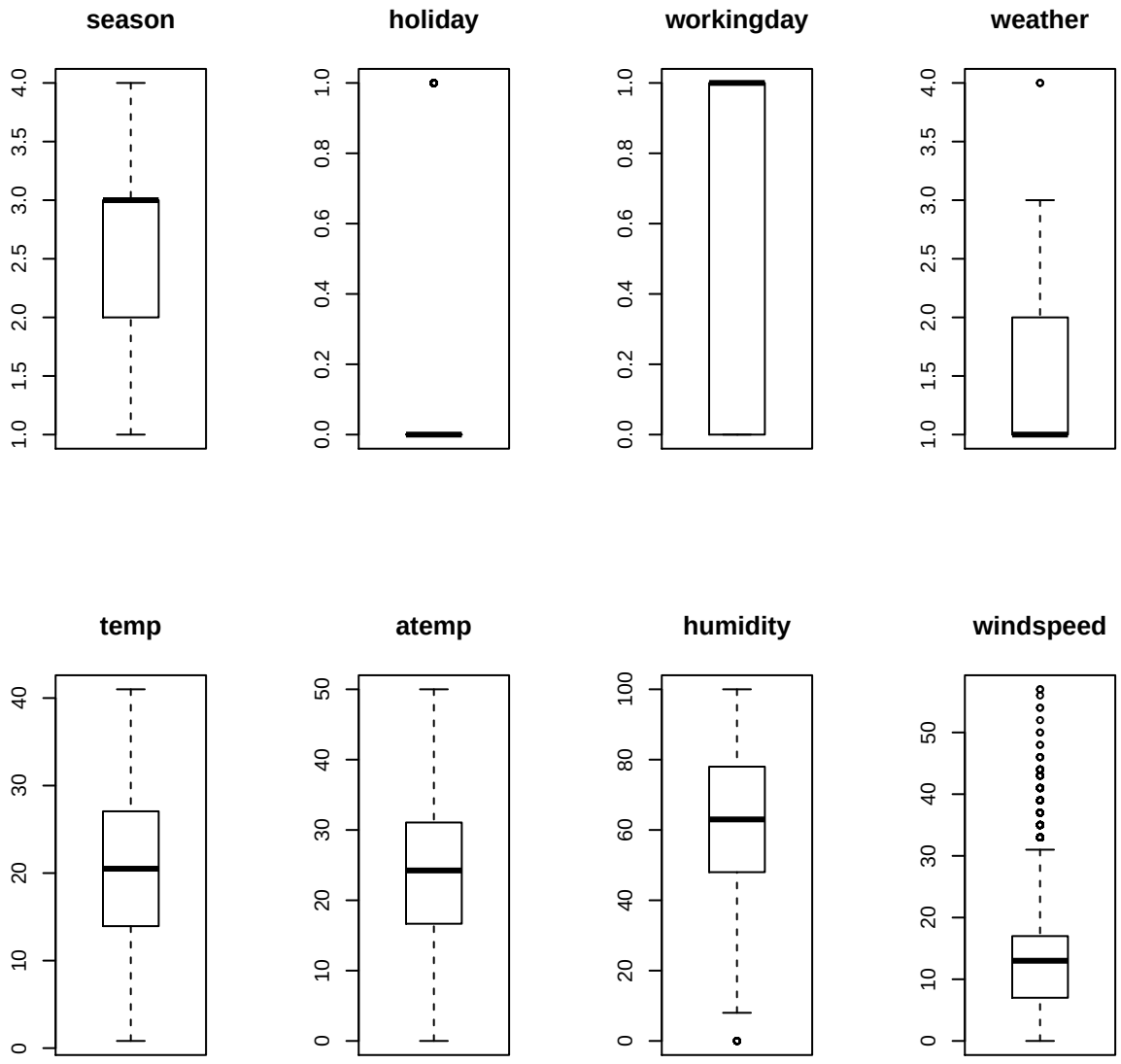
16.0.2 density plots

```
# density plot for each attribute
par(mfrow=c(2,4))
for(i in 2:9) {
    plot(density(data[,i]), main=names(data)[i])
}
```



16.0.3 boxplots

```
# boxplots for each attribute
par(mfrow=c(2,4))
for(i in 2:9) {
    boxplot(data[,i], main=names(data)[i])
}
```



16.1 Unique values of discrete variables

```
# the discrete variables in this case are integers
ints <- unlist(lapply(data, is.integer))
names(data)[ints]
```

```
#> [1] "season"     "holiday"    "workingday" "weather"    "humidity"
```

Humidity should not be an integer or discrete variable; it is a continuous or numeric variable.

```
# convert humidity to numeric
data$humidity <- as.numeric(data$humidity)
```

```
# list unique values of integer variables
ints <- unlist(lapply(data, is.integer))
int_vars <- names(data)[ints]

sapply(int_vars, function(x) unique(data[x]))
```

```
#> $season.season
#> [1] 1 2 3 4
#>
#> $holiday.holiday
#> [1] 0 1
#>
#> $workingday.workingday
#> [1] 0 1
#>
#> $weather.weather
#> [1] 1 2 3 4
```

16.1.1 Inferences

1. The variables `season`, `holiday`, `workingday` and `weather` are discrete (integer).
2. Activity is even through all seasons.
3. Most of the activitity happens during non-holidays.
4. Activity doubles during the working days.
5. Activity happens mostly during clear (1) weather.
6. temp, atemp and humidity are continuous variables (numeric).

Chapter 17

4. Hypothesis Testing (using multivariate analysis)

Till now, we have got a fair understanding of the data set. Now, let's test the hypothesis which we had generated earlier. Here I have added some additional hypothesis from the dataset. Let's test them one by one:

17.1 Hourly trend

> *There must be high demand during office timings. Early morning and late evening can have different trend (cyclist) and low demand during 10:00 pm to 4:00 am.*

We don't have the variable 'hour' with us. But we can extract it using the datetime column.

```
head(data$datetime)
```

```
#> [1] 2011-01-01 00:00:00 2011-01-01 01:00:00 2011-01-01 02:00:00
#> [4] 2011-01-01 03:00:00 2011-01-01 04:00:00 2011-01-01 05:00:00
#> 17379 Levels: 2011-01-01 00:00:00 2011-01-01 01:00:00 ... 2012-12-31 23:00:00
```

```
class(data$datetime)
```

```
#> [1] "factor"
```

```
# show hour and day from the variable datetime
head(substr(data$datetime, 12, 13))  # hour
```

```
#> [1] "00" "01" "02" "03" "04" "05"
```

```
head(substr(data$datetime, 9, 10))   # day
```

```
#> [1] "01" "01" "01" "01" "01" "01"
```

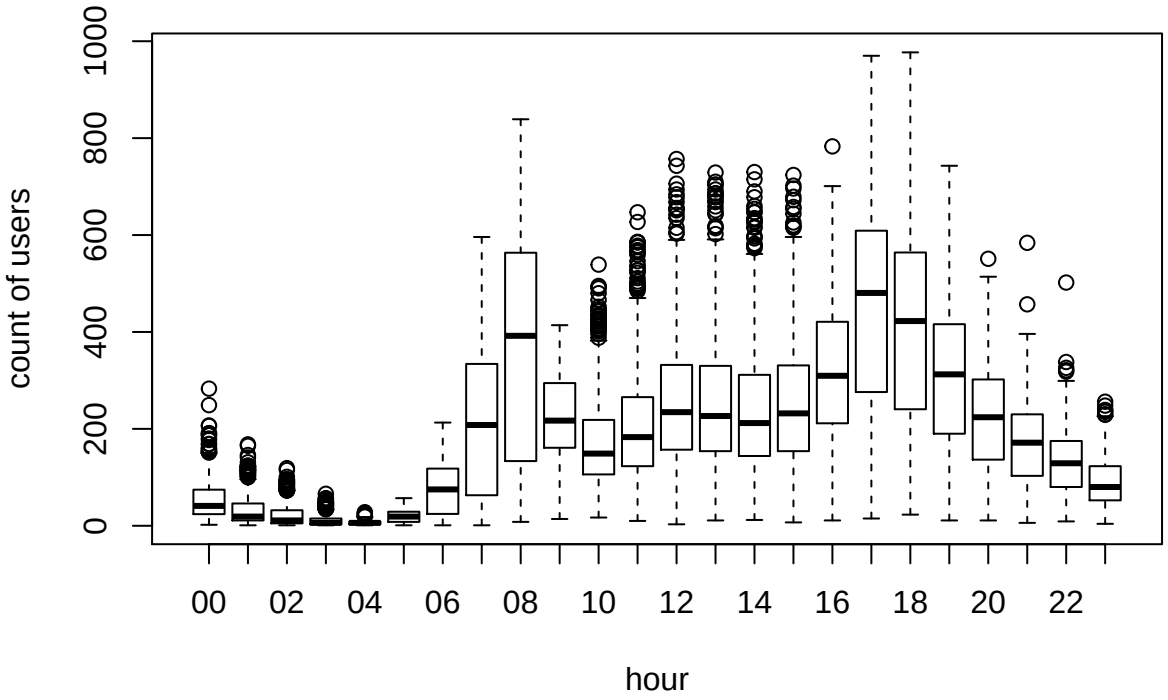
```
# extracting hour
data$hour = substr(data$datetime,12,13)
data$hour = as.factor(data$hour)
head(data$hour)
```

```
#> [1] 00 01 02 03 04 05
#> 24 Levels: 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 ... 23
```

```
### dividing again in train and test
# the train dataset is for the first 19 days
train = data[as.integer(substr(data$datetime, 9, 10)) < 20,]

# the test dataset is from day 20 to the end of the month
test = data[as.integer(substr(data$datetime, 9, 10)) > 19,]
```

17.1.1 boxplot count vs hour in training set

```
boxplot(train$count ~ train$hour, xlab="hour", ylab="count of users")
```



Rides increase from 6 am to 6pm, during office hours.

```
# casual users
casual <- data[data$casual > 0, ]
registered <- data[data$registered > 0, ]

dim(casual)
```

```
#> [1] 9900   13
```

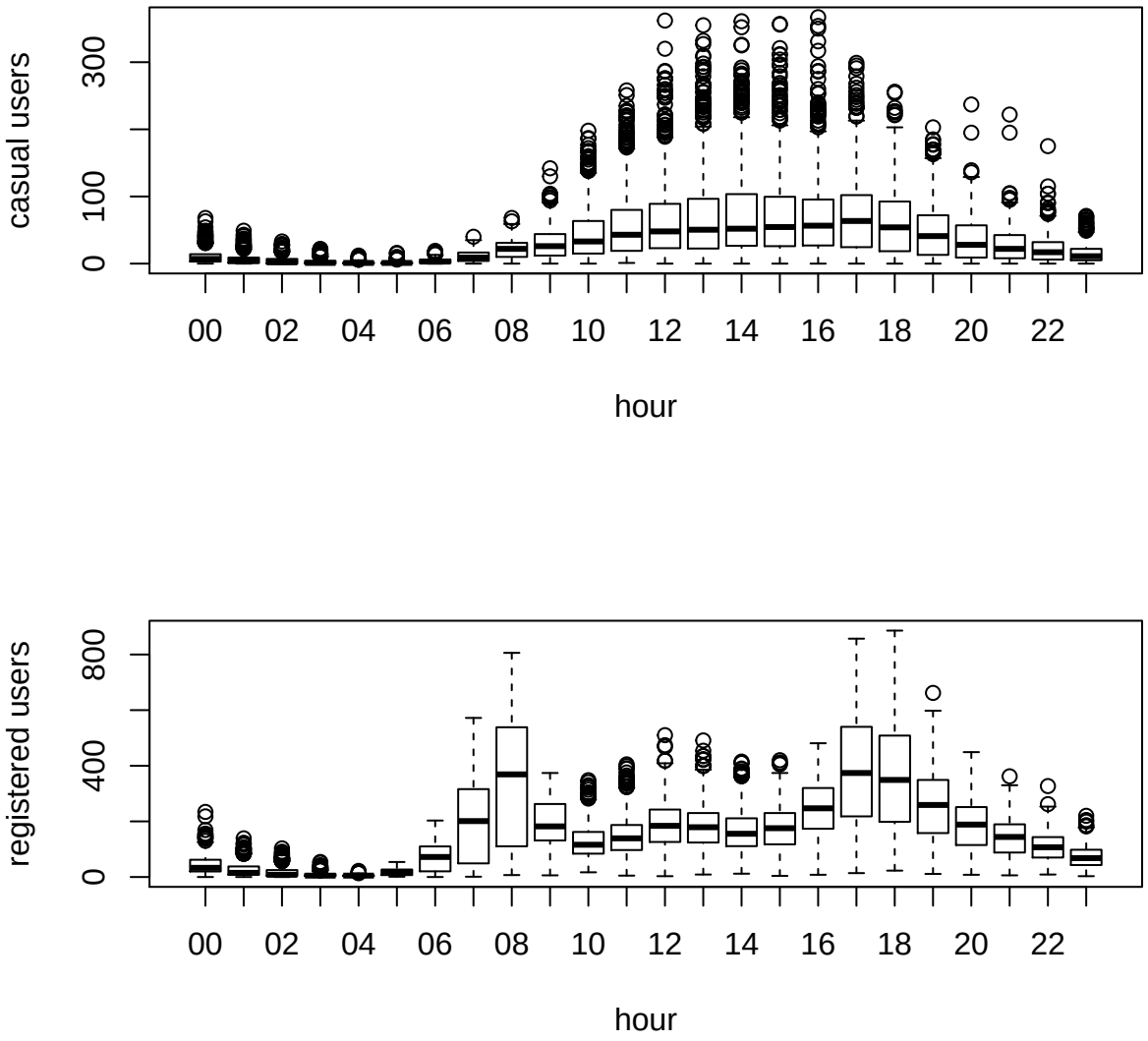
```
dim(registered)
```

```
#> [1] 10871    13
```

17.1.2 Boxplot hourly: casual vs registered users in the training set

```
# by hour: casual vs registered users
par(mfrow=c(2,1))
```

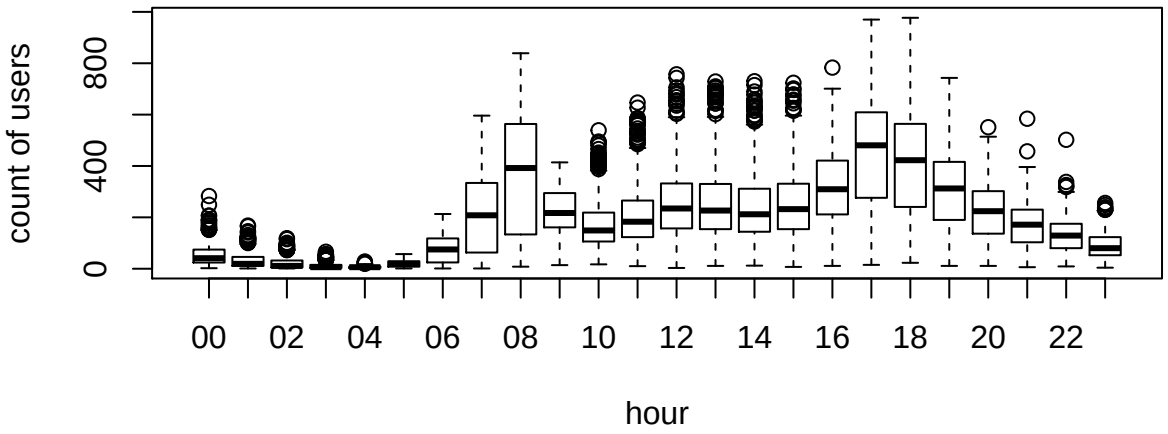
```
boxplot(train$casual ~ train$hour, xlab="hour", ylab="casual users")
boxplot(train$registered ~ train$hour, xlab="hour", ylab="registered users")
```

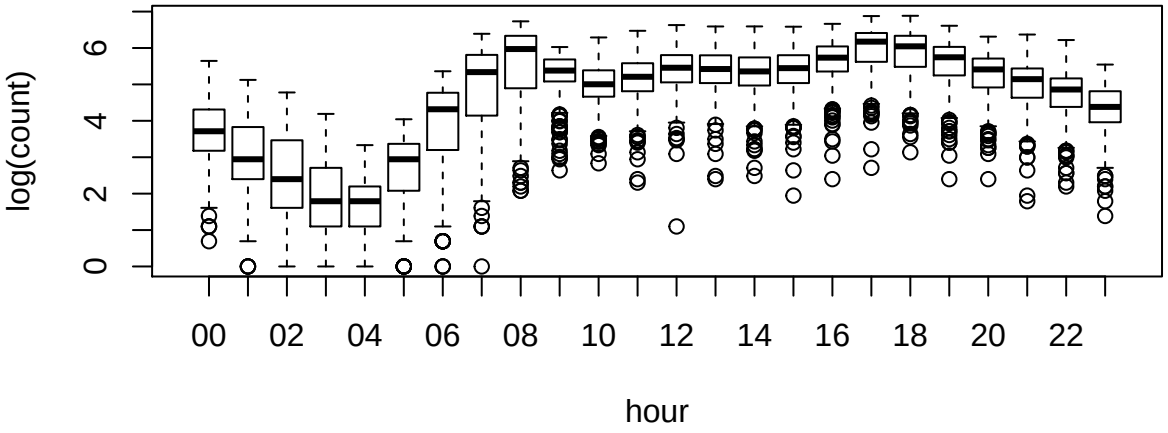


Casual and Registered users have different distributions. Casual users tend to rent more during office hours.

17.1.3 outliers in the training set

```
par(mfrow=c(2,1))
boxplot(train$count ~ train$hour, xlab="hour", ylab="count of users")
boxplot(log(train$count) ~ train$hour,xlab="hour",ylab="log(count)")
```





17.2 Daily trend

Registered users demand more bike on weekdays as compared to weekend or holiday.

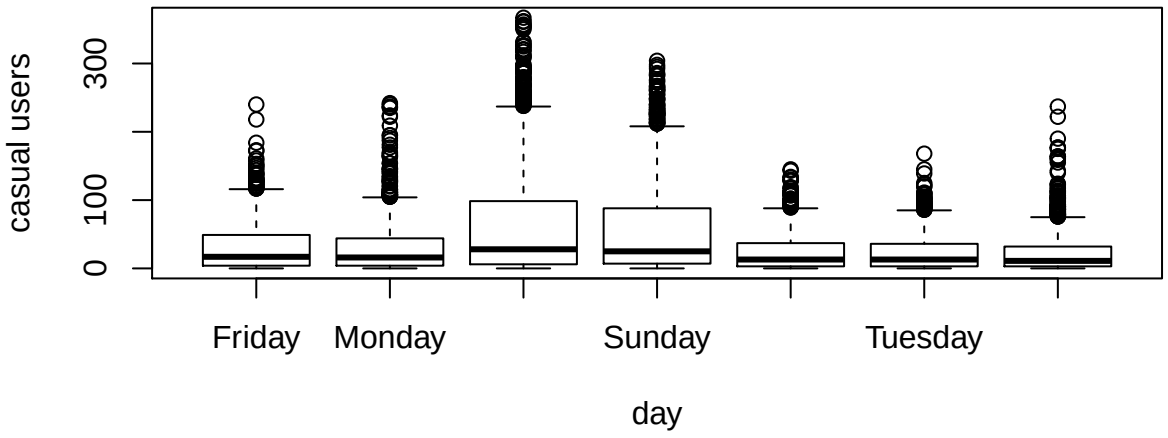
```
# extracting days of week
date <- substr(data$datetime, 1, 10)
days <- weekdays(as.Date(date))
data$day <- days
```

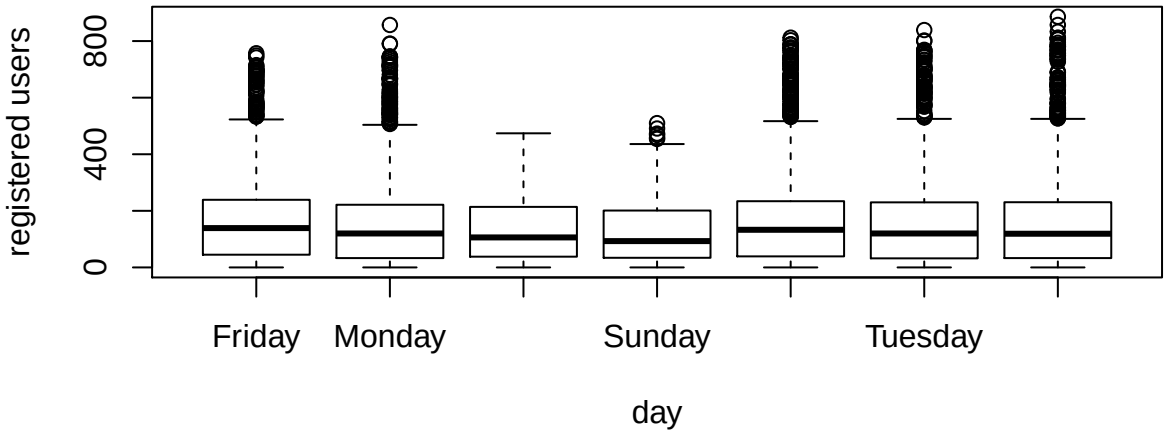
```
# split the dataset again at day 20 of the month, before and after
train = data[as.integer(substr(data$datetime,9,10)) < 20,]
test  = data[as.integer(substr(data$datetime,9,10)) > 19,]
```

17.2.1 Boxplot daily trend: casual vs registered users, training set

```
# creating boxplots for rentals with different variables to see the variation
par(mfrow=c(2,1))
```

```
boxplot(train$casual ~ train$day, xlab="day", ylab="casual users")
boxplot(train$registered ~ train$day, xlab="day", ylab="registered users")
```





Demand of casual users increases during the weekend, contrary of registered users.

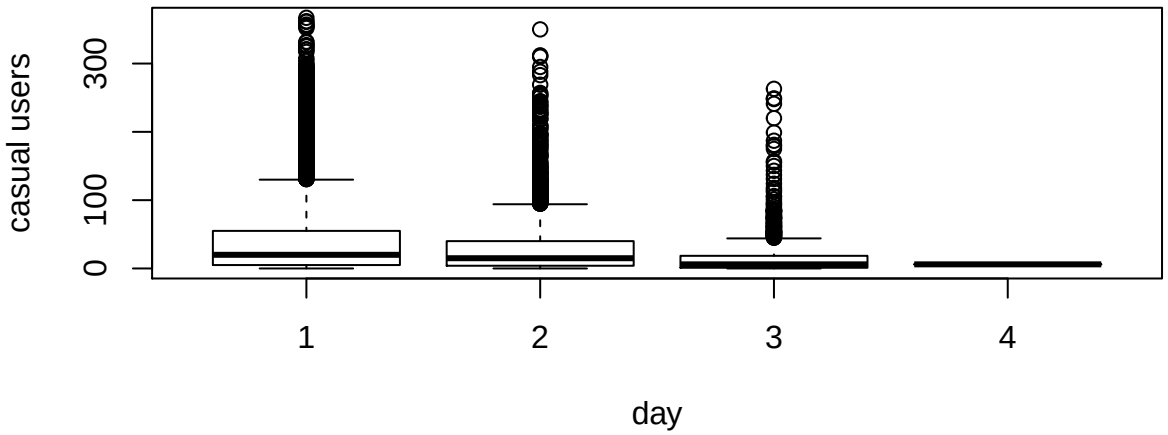
17.3 Rain

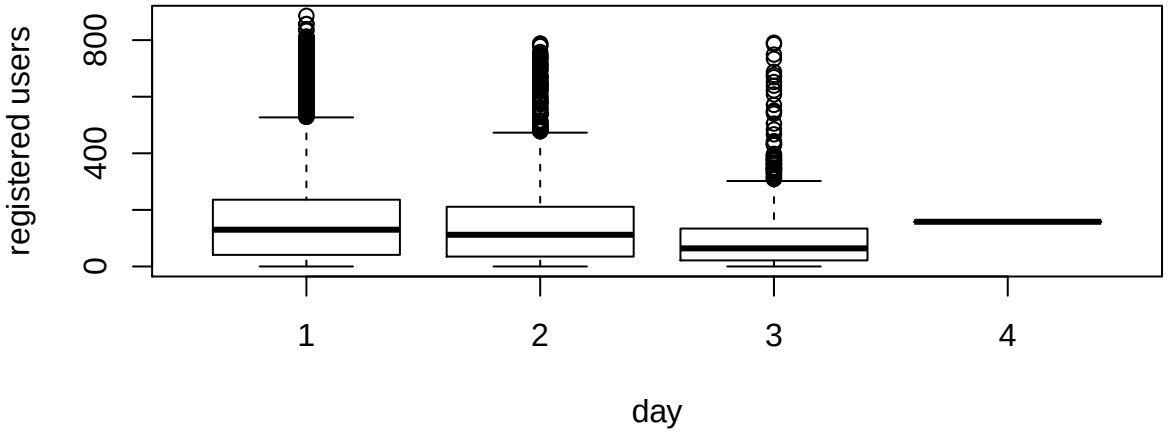
The demand of bikes will be lower on a rainy day as compared to a sunny day. Similarly, higher humidity will cause to lower the demand and vice versa.

We use the variable weather (1 to 4) to analyze riding under rain conditions.

17.3.1 Boxplot of rain effect on bike riding, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$weather, xlab="day", ylab="casual users")
boxplot(train$registered ~ train$weather, xlab="day", ylab="registered users")
```





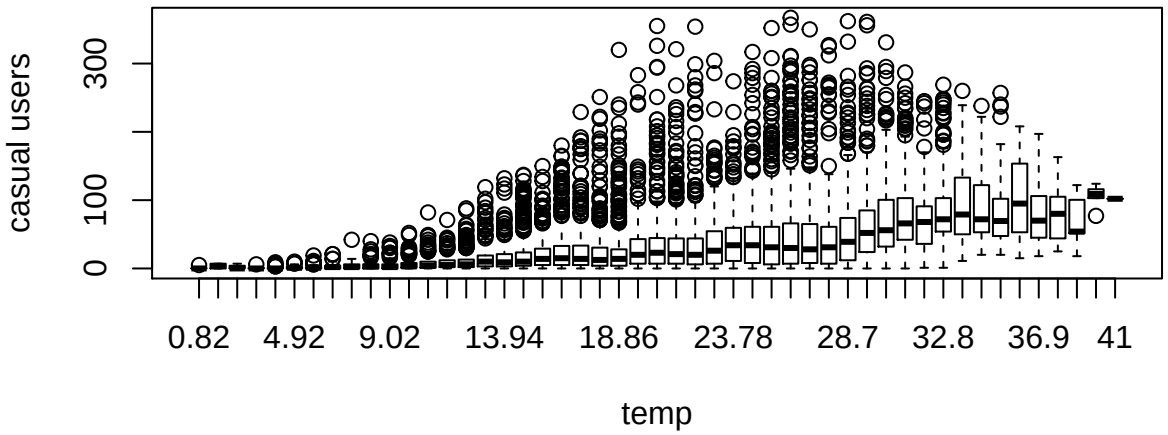
Registered used tend to ride even with rain.

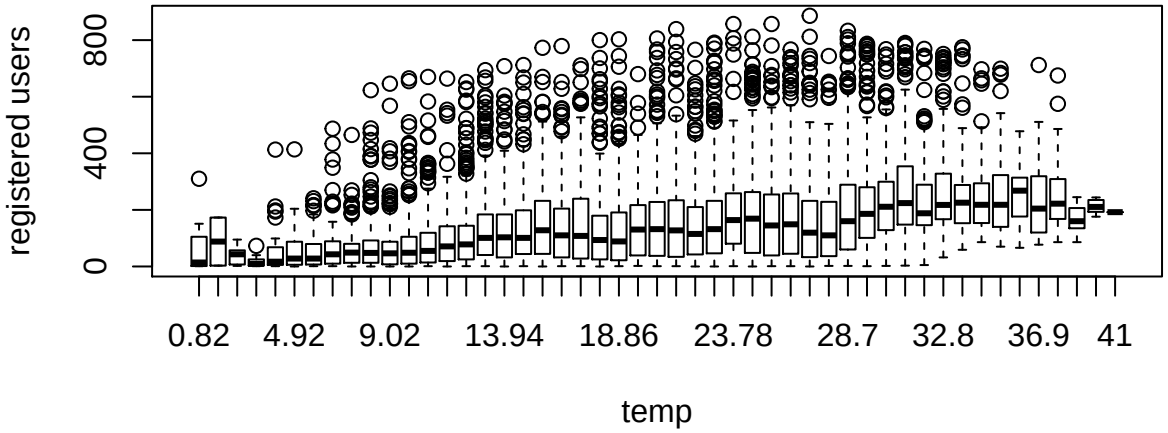
17.4 Temperature

Would high or low temperature encourage or disencourage bike riding?

17.4.1 boxplot of temperature effect, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$temp, xlab="temp", ylab="casual users")
boxplot(train$registered ~ train$temp, xlab="temp", ylab="registered users")
```





Casual users tend to ride with milder temperatures while registered users ride even at low temperatures.

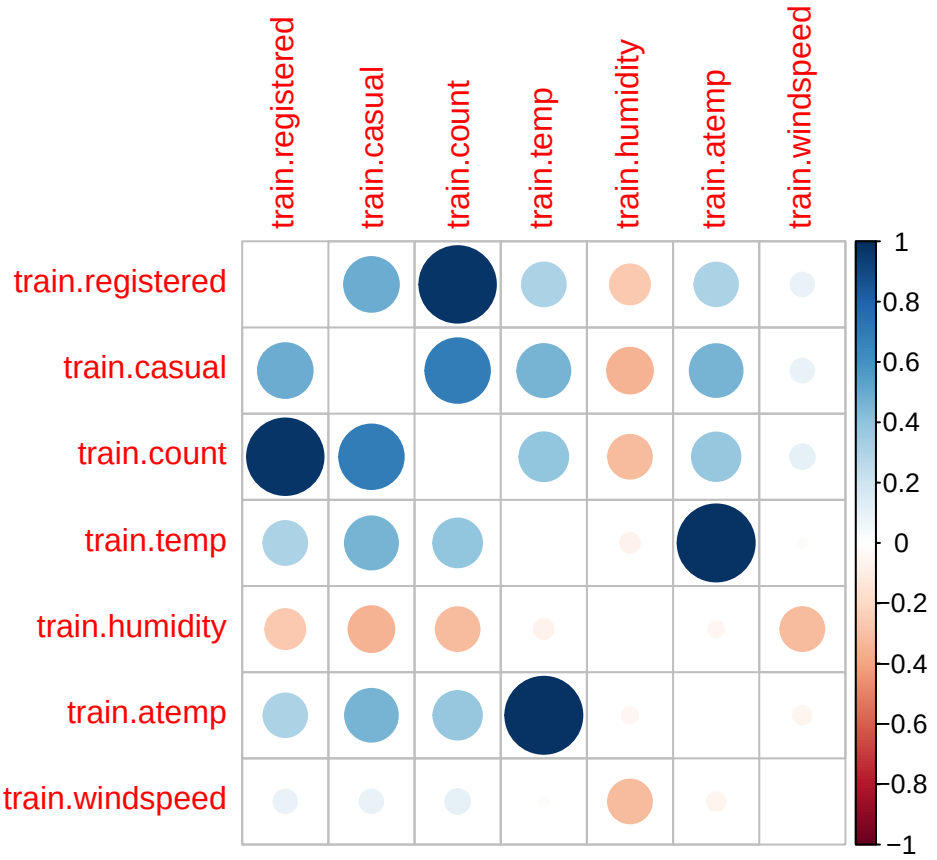
17.5 Correlation

```
sub = data.frame(train$registered, train$casual, train$count, train$temp,
                 train$humidity, train$atemp, train$windspeed)
cor(sub)
```

```
#>                  train.registered train.casual train.count  train.temp
#> train.registered       1.00000000   0.49724969   0.9709481  0.31857128
#> train.casual           0.49724969   1.00000000   0.6904136  0.46709706
#> train.count            0.97094811   0.69041357   1.0000000  0.39445364
#> train.temp             0.31857128   0.46709706   0.3944536  1.00000000
#> train.humidity        -0.26545787  -0.34818690  -0.3173715 -0.06494877
#> train.atemp            0.31463539   0.46206654   0.3897844  0.98494811
#> train.windspeed        0.09105166   0.09227619   0.1013695 -0.01785201
#>                  train.humidity train.atemp train.windspeed
#> train.registered    -0.26545787  0.31463539      0.09105166
```

```
#> train.casual        -0.34818690  0.46206654      0.09227619
#> train.count         -0.31737148  0.38978444      0.10136947
#> train.temp          -0.06494877  0.98494811     -0.01785201
#> train.humidity       1.00000000 -0.04353571     -0.31860699
#> train.atemp         -0.04353571  1.00000000     -0.05747300
#> train.windspeed     -0.31860699 -0.05747300      1.00000000
```

```
# do not show the diagonal
corrplot(cor(sub), diag = FALSE)
```



1. correlation between `casual` and atemp, temp.
2. Strong correlation between temp and atemp.

17.6 Activity by year

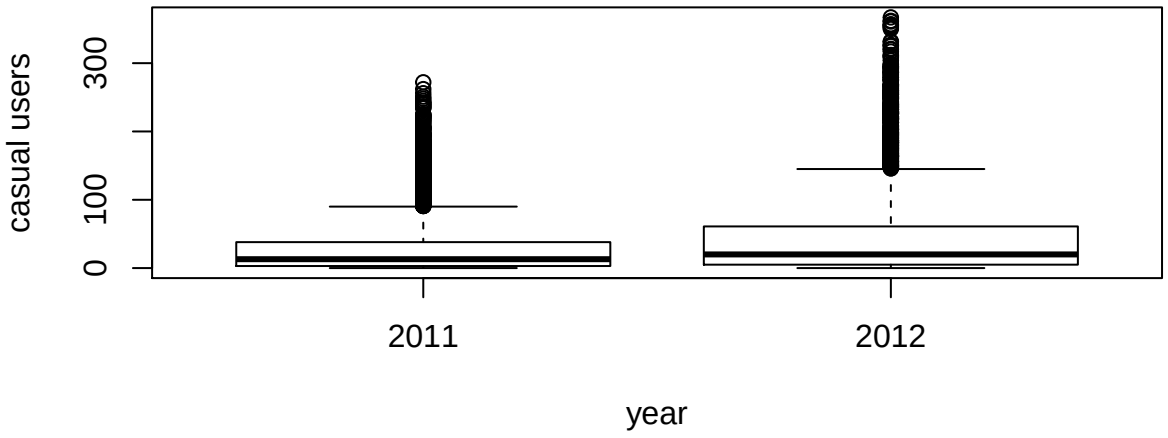
17.6.1 Year extraction

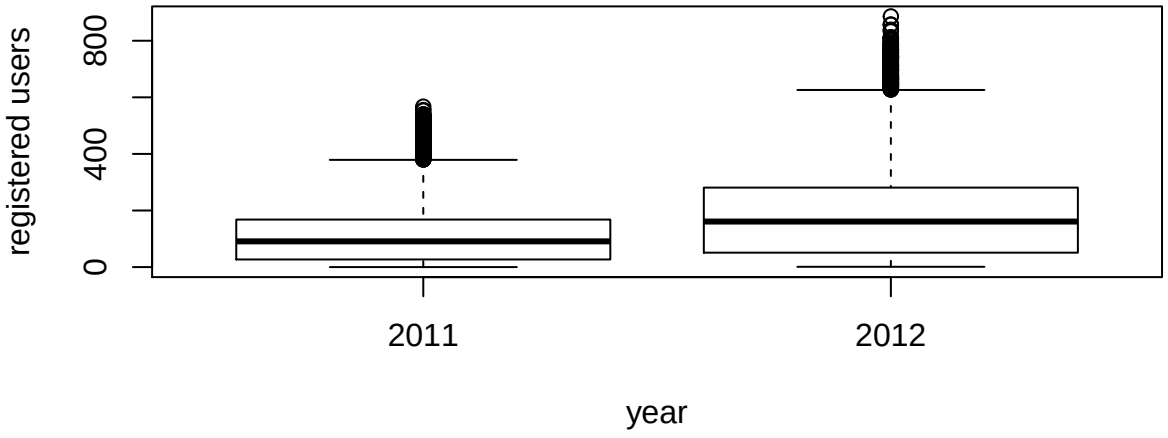
```
# extracting year
data$year = substr(data$datetime, 1, 4)
data$year = as.factor(data$year)
```

```
# ignore the division of data again and again, this could have been done together also
train = data[as.integer(substr(data$datetime,9,10)) < 20,]
test = data[as.integer(substr(data$datetime,9,10)) > 19,]
```

17.6.2 Trend by year, training set

```
par(mfrow=c(2,1))
# again some boxplots with different variables
# these boxplots give important information about the dependent variable with respect to the independen
boxplot(train$casual ~ train$year, xlab="year", ylab="casual users")
boxplot(train$registered ~ train$year, xlab="year", ylab="registered users")
```

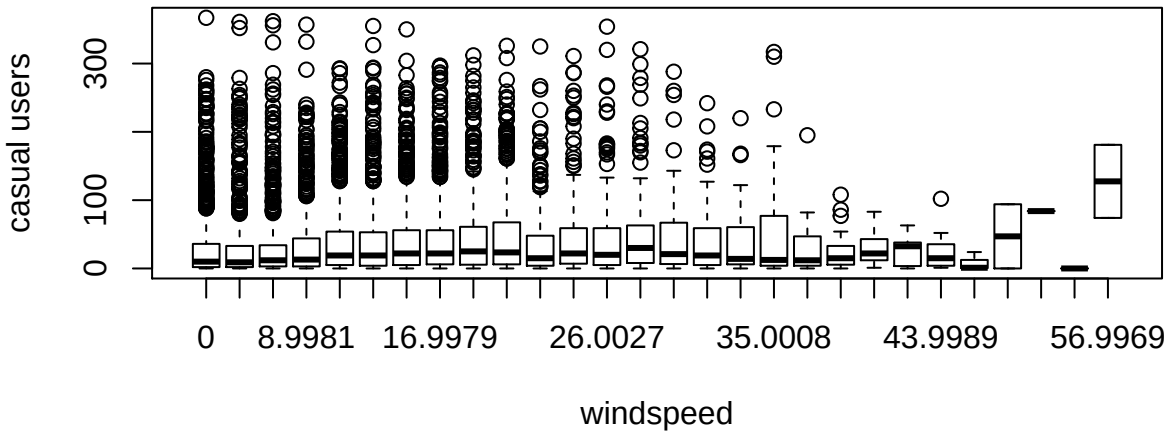


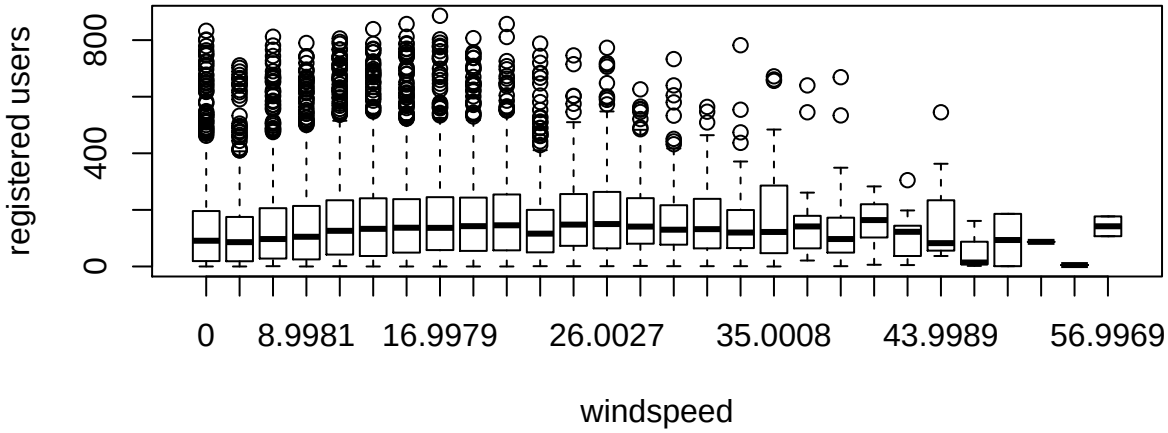


Activity increased in 2012.

17.6.3 trend by windspeed, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$windspeed, xlab="windspeed", ylab="casual users")
boxplot(train$registered ~ train$windspeed, xlab="windspeed", ylab="registered users")
```

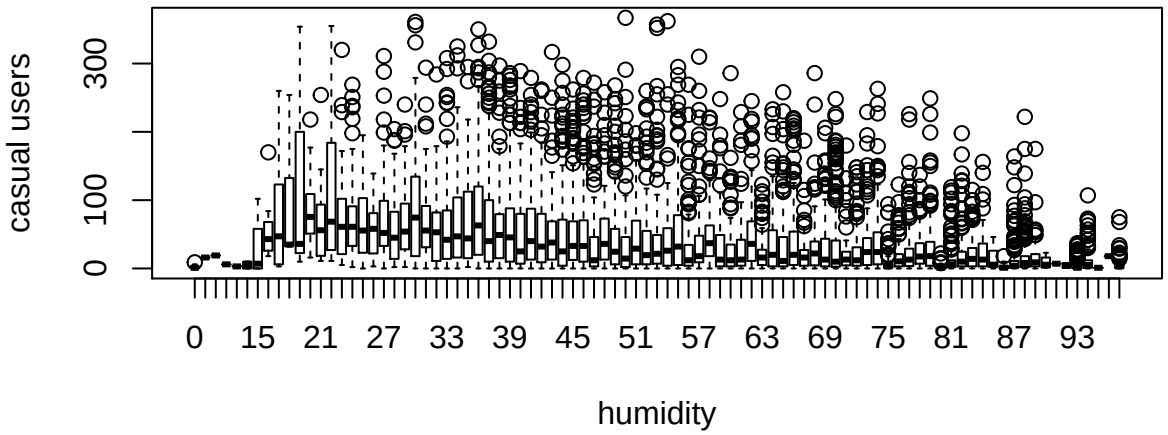


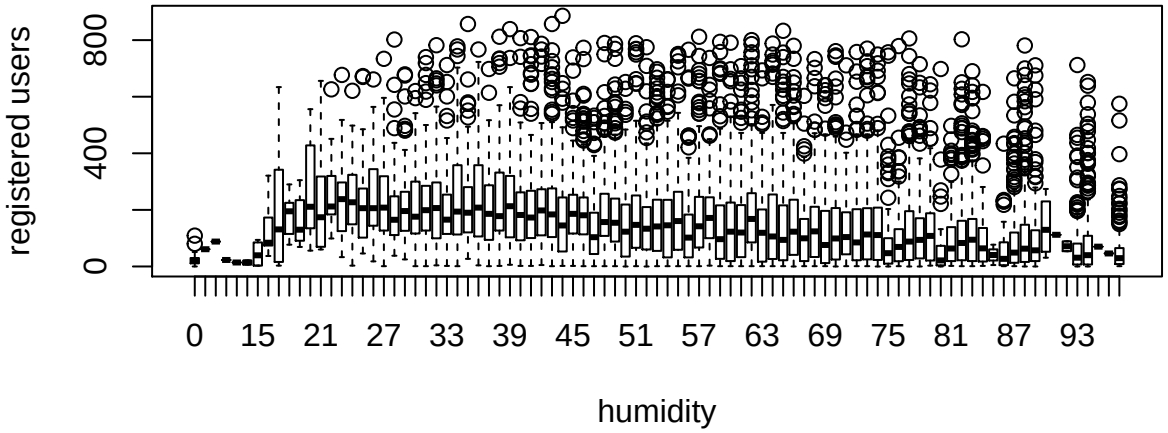


Casual users ride even with stron winds.

17.6.4 trend by humidity, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$humidity, xlab="humidity", ylab="casual users")
boxplot(train$registered ~ train$humidity, xlab="humidity", ylab="registered users")
```





Casual users prefer not to ride with humid weather.

Chapter 18

5. Feature Engineering

```
# factoring some variables from integer
data$season     <- as.factor(data$season)
data$weather    <- as.factor(data$weather)
data$holiday    <- as.factor(data$holiday)
data$workingday <- as.factor(data$workingday)
```

```
# new column
data$hour <- as.integer(data$hour)
```

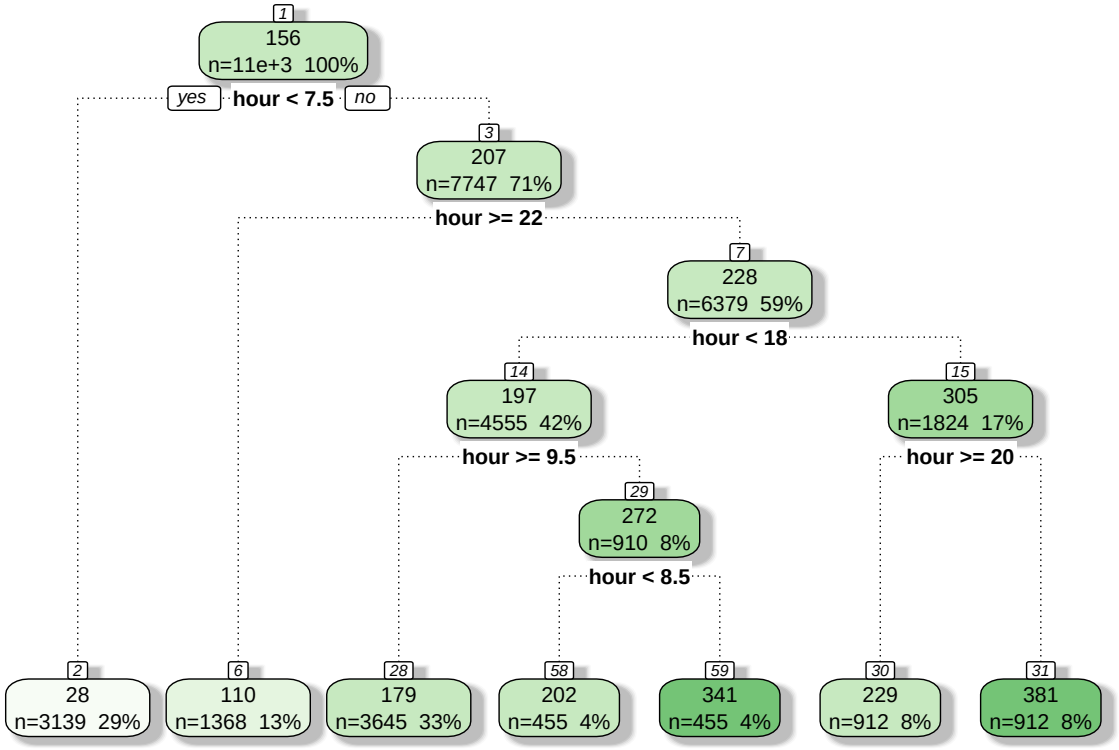
```
# created this variable to divide a day into parts, but did not finally use it
data$day_part <- 0
```

```
# split in training and test sets again
train <- data[as.integer(substr(data$datetime, 9, 10)) < 20,]
test  <- data[as.integer(substr(data$datetime, 9, 10)) > 19,]
```

```
# combine the sets
data <- rbind(train, test)
```

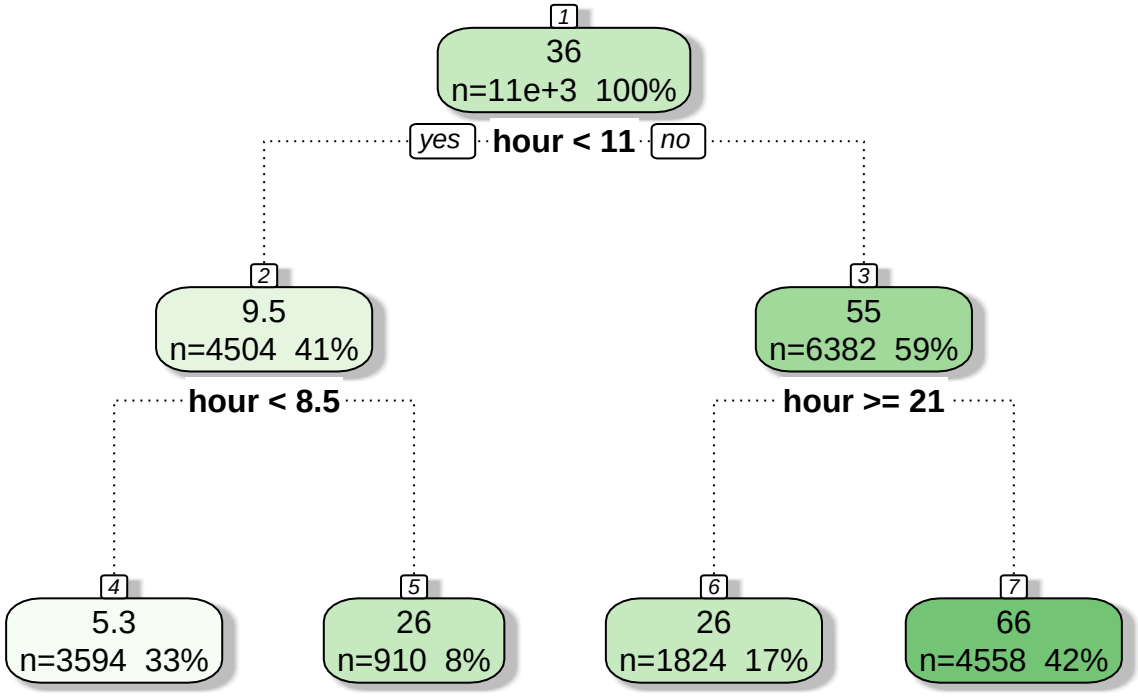
18.1 Build hour bins

```
# for registered users
d = rpart(registered ~ hour, data = train)
fancyRpartPlot(d)
```



Rattle 2019−May−12 23:23:37 datascience

```r
# for casual users
d = rpart(casual ~ hour, data = train)
fancyRpartPlot(d)
```



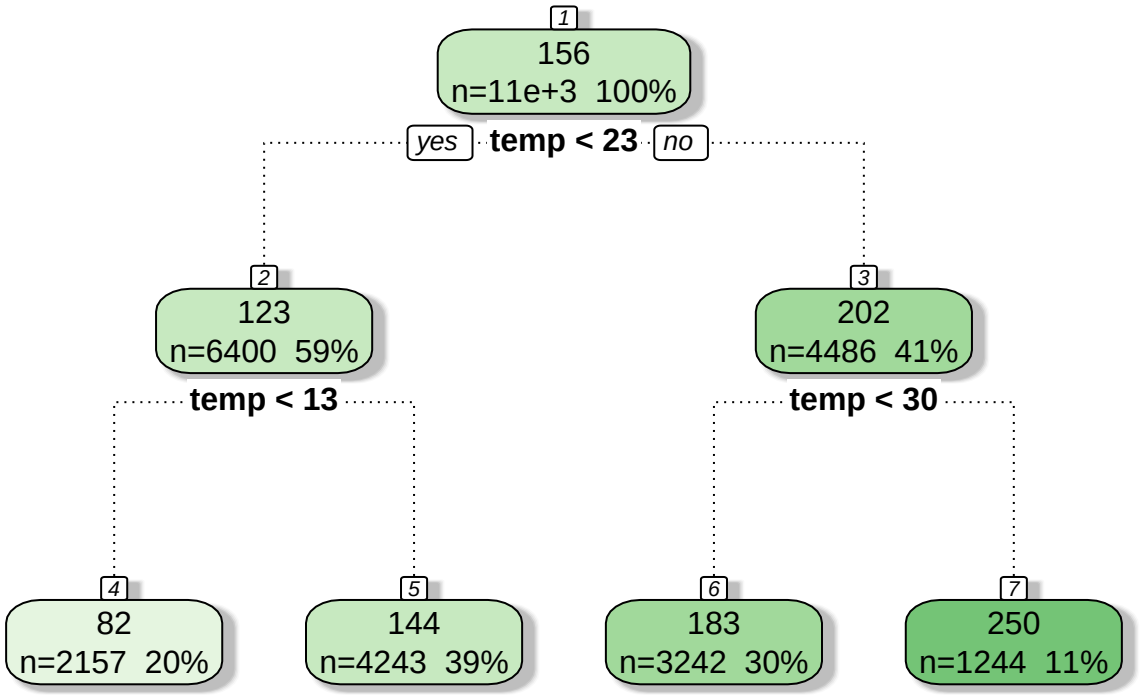
Rattle 2019−May−12 23:23:38 datascience

```
# Assign the timings according to tree
# fill the hour bins
data = rbind(train,test)

# create hour buckets for registered users
# 0,1,2,3,4,5,6,7 < 7.5
# 22,23,24 >=22
# 10,11,12,13,14,15,16,17: h>=9.5 & h<18
# h<9.5 & h<8.5 : 8
# h<9.5 & h>=8.5 : 9
# h>=20: 20,21
# h < 20: 18,19

data$dp_reg = 0
data$dp_reg[data$hour < 8] = 1
data$dp_reg[data$hour >= 22] = 2
data$dp_reg[data$hour > 9 & data$hour < 18] = 3
data$dp_reg[data$hour == 8] = 4
data$dp_reg[data$hour == 9] = 5
data$dp_reg[data$hour == 20 | data$hour == 21] = 6
data$dp_reg[data$hour == 19 | data$hour == 18] = 7
```

```
# casual users
# h<11, h<8.5: 0,1,2,3,4,5,6,7,8
# h>=8.5 & h<11: 9, 10
# h >=11 & h>=21: 21,22,23,24
# h >=11 & h<21: 11,12,13,14,15,16,17,18,19,20
data$dp_cas = 0
data$dp_cas[data$hour < 11 & data$hour >= 8] = 1
data$dp_cas[data$hour == 9 | data$hour == 10] = 2
data$dp_cas[data$hour >= 11 & data$hour < 21] = 3
data$dp_cas[data$hour >= 21] = 4
```

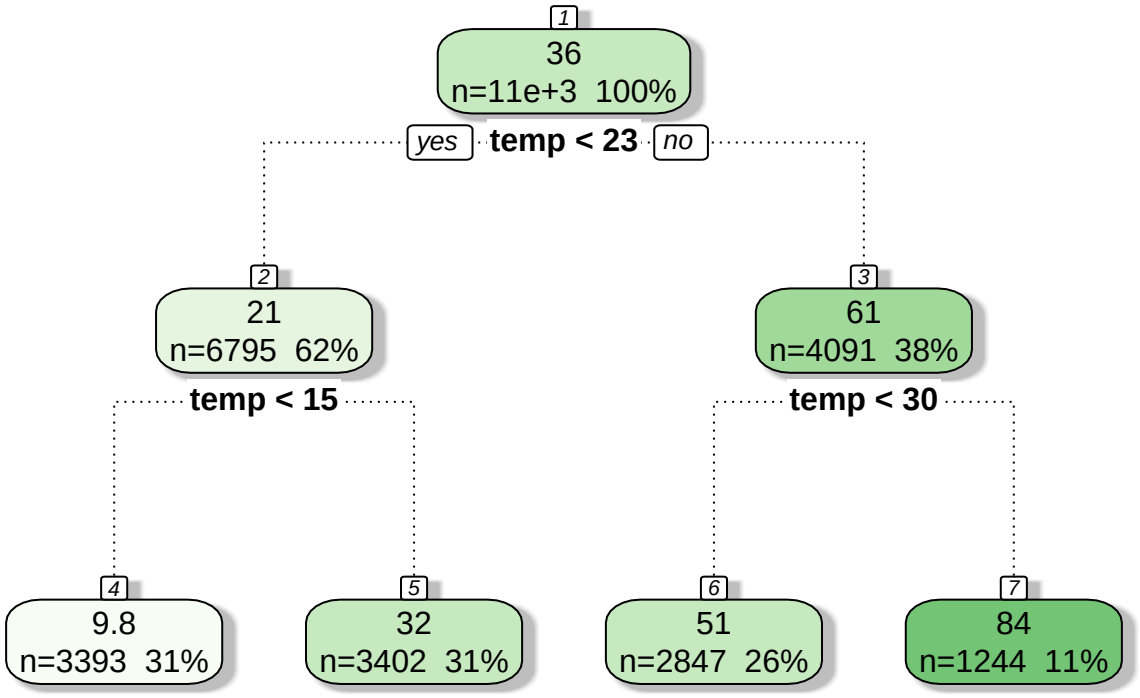
18.2 Temperature bins

```
# partition the data by temperature, registered users
f = rpart(registered ~ temp, data=train)
fancyRpartPlot(f)
```



Rattle 2019−May−12 23:23:38 datascience

```
# partition the data by temperature,, casual users
f=rpart(casual ~ temp, data=train)
fancyRpartPlot(f)
```



Rattle 2019−May−12 23:23:39 datascience

18.2.1 Assign temperature ranges accoding to trees

```
data$temp_reg = 0
data$temp_reg[data$temp < 13] = 1
data$temp_reg[data$temp >= 13 & data$temp < 23] = 2
data$temp_reg[data$temp >= 23 & data$temp < 30] = 3
data$temp_reg[data$temp >= 30] = 4
```

```
data$temp_cas = 0
data$temp_cas[data$temp < 15] = 1
data$temp_cas[data$temp >= 15 & data$temp < 23] = 2
data$temp_cas[data$temp >= 23 & data$temp < 30] = 3
data$temp_cas[data$temp >= 30] = 4
```

18.3 Year bins by quarter

```
# add new variable with the month number
data$month <- substr(data$datetime, 6, 7)
data$month <- as.integer(data$month)
```

```
# bin by quarter manually
data$year_part[data$year=='2011']                 = 1
data$year_part[data$year=='2011' & data$month>3] = 2
data$year_part[data$year=='2011' & data$month>6] = 3
data$year_part[data$year=='2011' & data$month>9] = 4
data$year_part[data$year=='2012']                 = 5
data$year_part[data$year=='2012' & data$month>3] = 6
data$year_part[data$year=='2012' & data$month>6] = 7
data$year_part[data$year=='2012' & data$month>9] = 8
table(data$year_part)
```

```
#>
#>    1    2    3    4    5    6    7    8
#> 2067 2183 2192 2203 2176 2182 2208 2168
```

18.4 Day Type

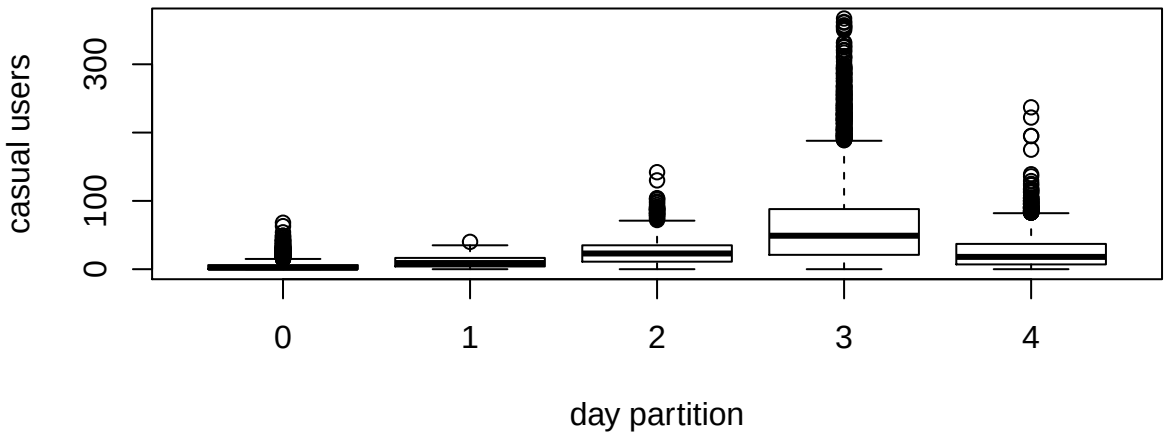
Created a variable having categories like “weekday”, “weekend” and “holiday”.

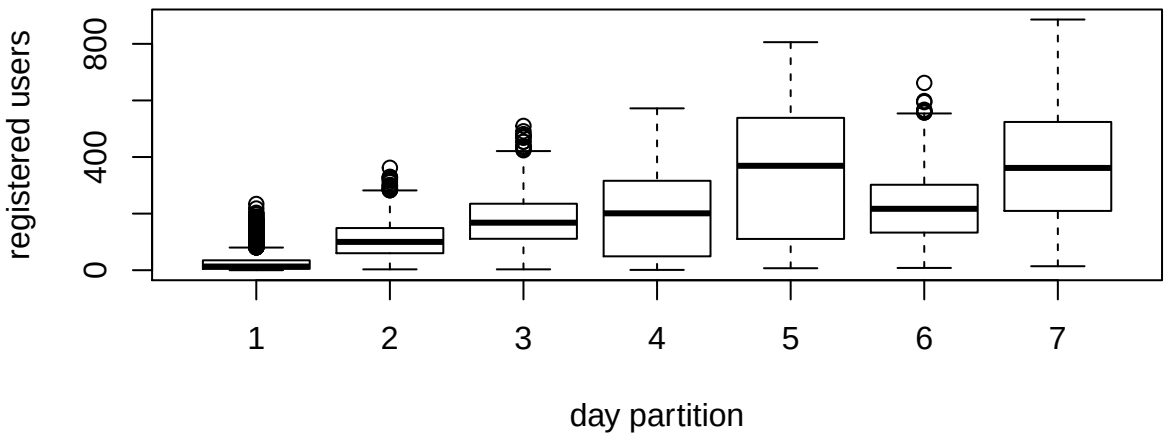
```
# creating another variable day_type which may affect our accuracy as weekends and weekdays are importa
data$day_type = 0
data$day_type[data$holiday==0 & data$workingday==0] = "weekend"
data$day_type[data$holiday==1]                      = "holiday"
data$day_type[data$holiday==0 & data$workingday==1] = "working day"
```

```
# split dataset again
train = data[as.integer(substr(data$datetime,9,10)) < 20,]
test = data[as.integer(substr(data$datetime,9,10)) > 19,]
```

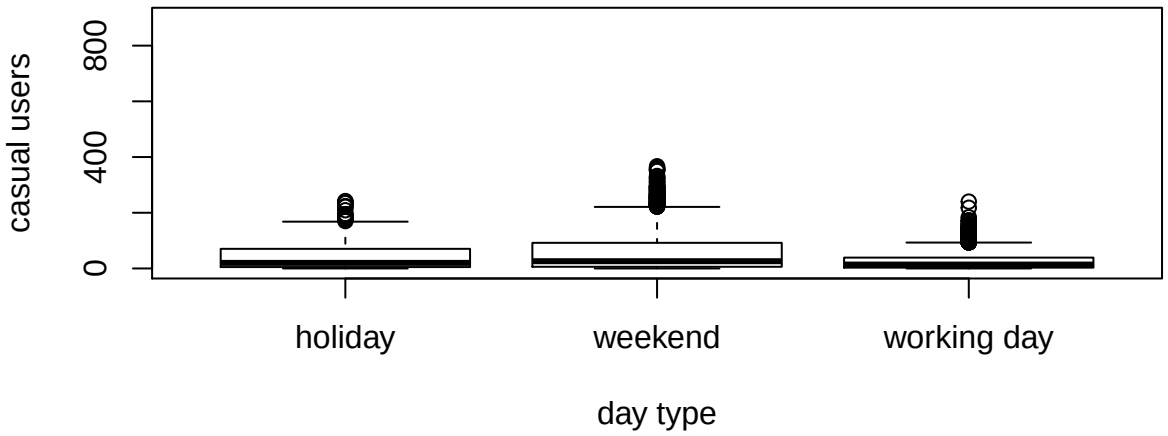
```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$dp_cas, xlab = "day partition", ylab="casual users")
```

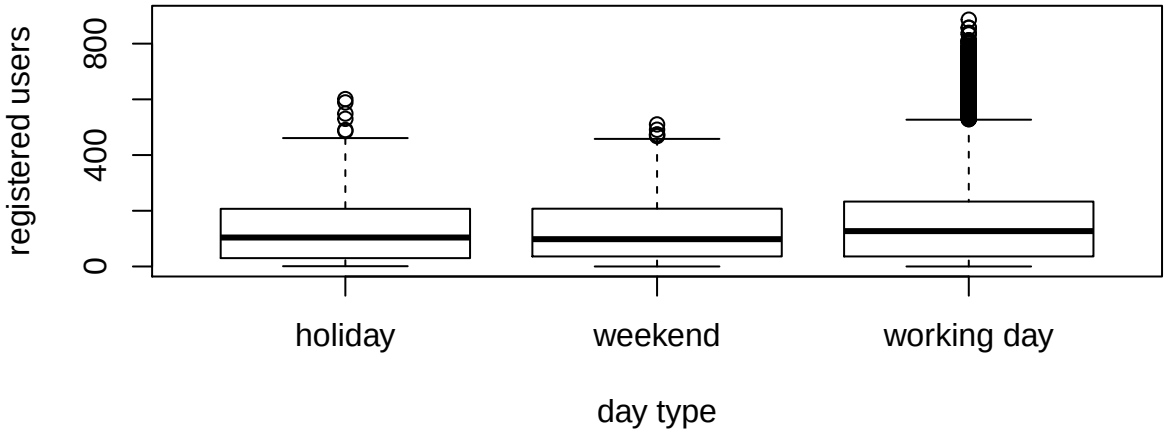
```
boxplot(train$registered ~ train$dp_reg, xlab = "day partition", ylab="registered users")
```



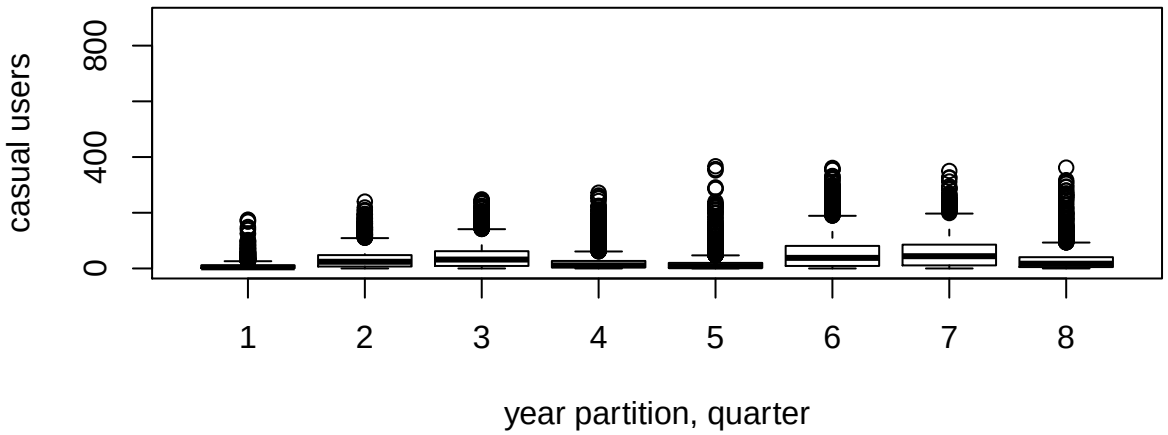


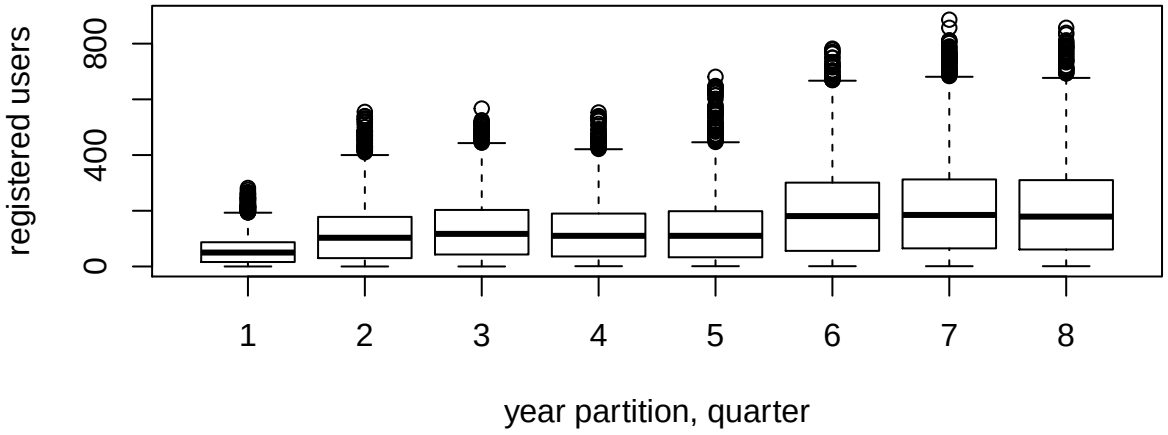

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$day_type, xlab = "day type",
        ylab="casual users", ylim = c(0,900))
boxplot(train$registered ~ train$day_type, xlab = "day type",
        ylab="registered users", ylim = c(0,900))
```





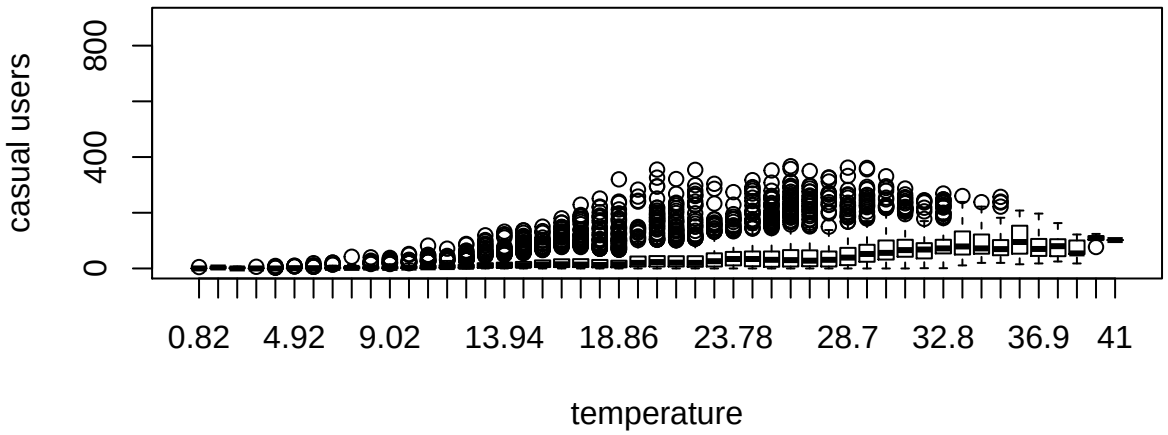

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$year_part, xlab = "year partition, quarter",
        ylab="casual users", ylim = c(0,900))
boxplot(train$registered ~ train$year_part, xlab = "year partition, quarter",
        ylab="registered users", ylim = c(0,900))
```

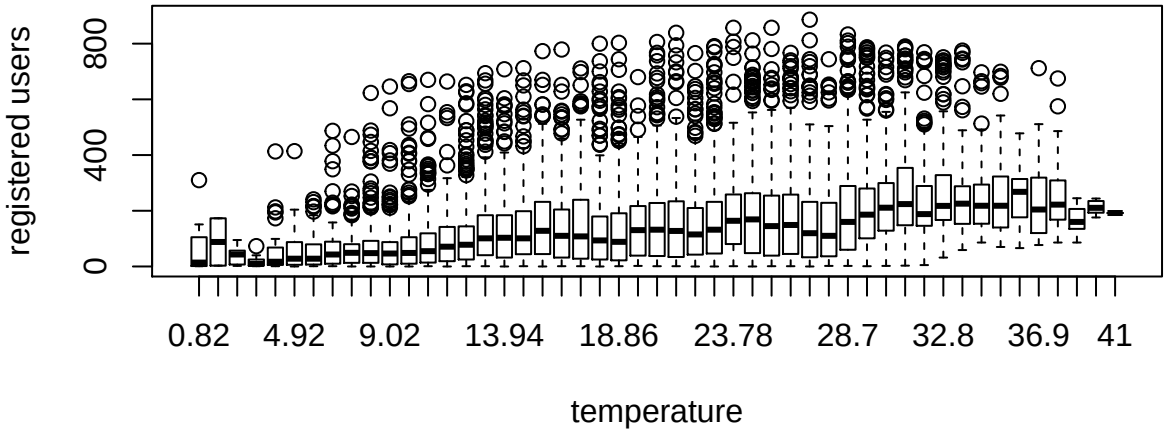




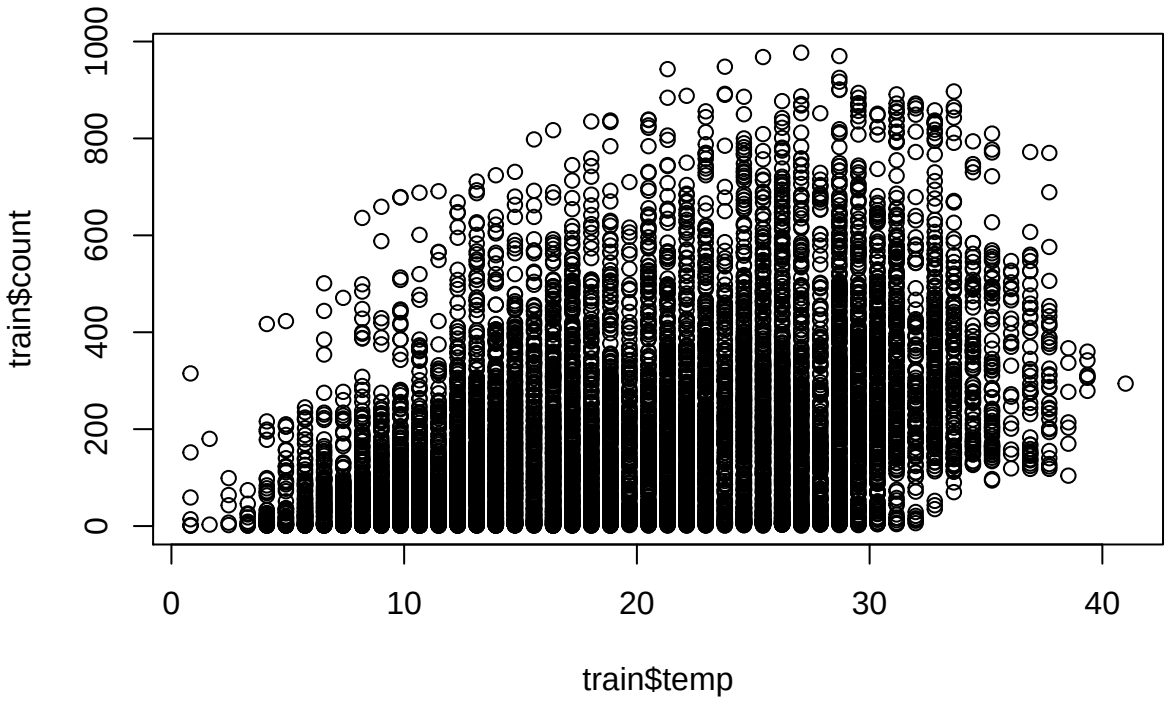
18.5 Temperatures

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$temp, xlab = "temperature",
        ylab="casual users", ylim = c(0,900))
boxplot(train$registered ~ train$temp, xlab = "temperature",
        ylab="registered users", ylim = c(0,900))
```






```
plot(train$temp, train$count)
```




```
data <- rbind(train, test)
# data$month <- substr(data$datetime, 6, 7)
# data$month <- as.integer(data$month)
```

18.6 Imputting missing data to wind speed

```
# dividing total data depending on windspeed to impute/predict the missing values
table(data$windspeed == 0)
```

```
#>
#> FALSE  TRUE
#> 15199  2180
```

```
    # FALSE  TRUE
    # 15199  2180

k = data$windspeed == 0

wind_0 = subset(data, k)     # windspeed is zero
wind_1 = subset(data, !k)    # windspeed not zero
```

```
# predicting missing values in windspeed using a random forest model
# this is a different approach to impute missing values rather than
# just using the mean or median or some other statistic for imputation

set.seed(415)
fit <- randomForest(windspeed ~ season + weather + humidity + month + temp +
                        year + atemp,
                    data = wind_1,
                    importance = TRUE,
```

```
                    ntree = 250)

pred = predict(fit, wind_0)
wind_0$windspeed = pred       # fill with wind speed predictions
```

```
# recompose the whole dataset
data = rbind(wind_0, wind_1)
```

```
# how many zero values now?
sum(data$windspeed == 0)
```

```
#> [1] 0
```

18.7 Weekend variable

Created a separate variable for weekend (0/1)

```
data$weekend = 0
data$weekend[data$day=="Sunday" | data$day=="Saturday" ] = 1
```

Chapter 19

6. Model Building

As this was our first attempt, we applied decision tree, conditional inference tree and random forest algorithms and found that random forest is performing the best. You can also go with regression, boosted regression, neural network and find which one is working well for you.

Before executing the random forest model code, I have followed following steps:

Convert discrete variables into factor (weather, season, hour, holiday, working day, month, day)

```
str(data)
```

```
#> 'data.frame':    17379 obs. of  24 variables:
#>  $ datetime  : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 7 8 9 10 65 ...
#>  $ season    : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ holiday   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ workingday: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
#>  $ weather   : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
#>  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
#>  $ humidity  : num  81 80 80 75 75 80 86 75 76 47 ...
#>  $ windspeed : num  9.03 9.05 9.05 9.15 9.15 ...
#>  $ casual    : num  3 8 5 3 0 2 1 1 8 8 ...
#>  $ registered: num  13 32 27 10 1 0 2 7 6 102 ...
#>  $ count     : num  16 40 32 13 1 2 3 8 14 110 ...
#>  $ hour      : int  1 2 3 4 5 7 8 9 10 20 ...
#>  $ day       : chr  "Saturday" "Saturday" "Saturday" "Saturday" ...
#>  $ year      : Factor w/ 2 levels "2011","2012": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_part  : num  0 0 0 0 0 0 0 0 0 0 ...
#>  $ dp_reg    : num  1 1 1 1 1 1 4 5 3 6 ...
#>  $ dp_cas    : num  0 0 0 0 0 0 1 2 2 3 ...
#>  $ temp_reg  : num  1 1 1 1 1 1 1 1 2 1 ...
#>  $ temp_cas  : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ month     : int  1 1 1 1 1 1 1 1 1 1 ...
#>  $ year_part : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_type  : chr  "weekend" "weekend" "weekend" "weekend" ...
#>  $ weekend   : num  1 1 1 1 1 1 1 1 1 0 ...
```

19.1 Convert to factors

```
# converting all relevant categorical variables into factors to feed to our random forest model
data$season     = as.factor(data$season)
data$holiday    = as.factor(data$holiday)
data$workingday = as.factor(data$workingday)
data$weather    = as.factor(data$weather)
data$hour       = as.factor(data$hour)
data$month      = as.factor(data$month)
data$day_part   = as.factor(data$dp_cas)
data$day_type   = as.factor(data$dp_reg)
data$day        = as.factor(data$day)
data$temp_cas   = as.factor(data$temp_cas)
data$temp_reg   = as.factor(data$temp_reg)
```

```
str(data)
```

```
#> 'data.frame':    17379 obs. of  24 variables:
#>  $ datetime  : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 7 8 9 10 65 ...
#>  $ season    : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ holiday   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ workingday: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
#>  $ weather   : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
#>  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
#>  $ humidity  : num  81 80 80 75 75 80 86 75 76 47 ...
#>  $ windspeed : num  9.03 9.05 9.05 9.15 9.15 ...
#>  $ casual    : num  3 8 5 3 0 2 1 1 8 8 ...
#>  $ registered: num  13 32 27 10 1 0 2 7 6 102 ...
#>  $ count     : num  16 40 32 13 1 2 3 8 14 110 ...
#>  $ hour      : Factor w/ 24 levels "1","2","3","4",..: 1 2 3 4 5 7 8 9 10 20 ...
#>  $ day       : Factor w/ 7 levels "Friday","Monday",..: 3 3 3 3 3 3 3 3 3 2 ...
#>  $ year      : Factor w/ 2 levels "2011","2012": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_part  : Factor w/ 5 levels "0","1","2","3",..: 1 1 1 1 1 1 2 3 3 4 ...
#>  $ dp_reg    : num  1 1 1 1 1 1 4 5 3 6 ...
#>  $ dp_cas    : num  0 0 0 0 0 0 1 2 2 3 ...
#>  $ temp_reg  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 2 1 ...
#>  $ temp_cas  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ month     : Factor w/ 12 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
#>  $ year_part : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_type  : Factor w/ 7 levels "1","2","3","4",..: 1 1 1 1 1 1 4 5 3 6 ...
#>  $ weekend   : num  1 1 1 1 1 1 1 1 1 0 ...
```

- As we know that dependent variables have natural outliers so we will predict log of dependent variables.

- Predict bike demand registered and casual users separately. $y1 = \log(casual + 1)$ and $y2 = \log(registered + 1)$, Here we have added 1 to deal with zero values in the casual and registered columns.

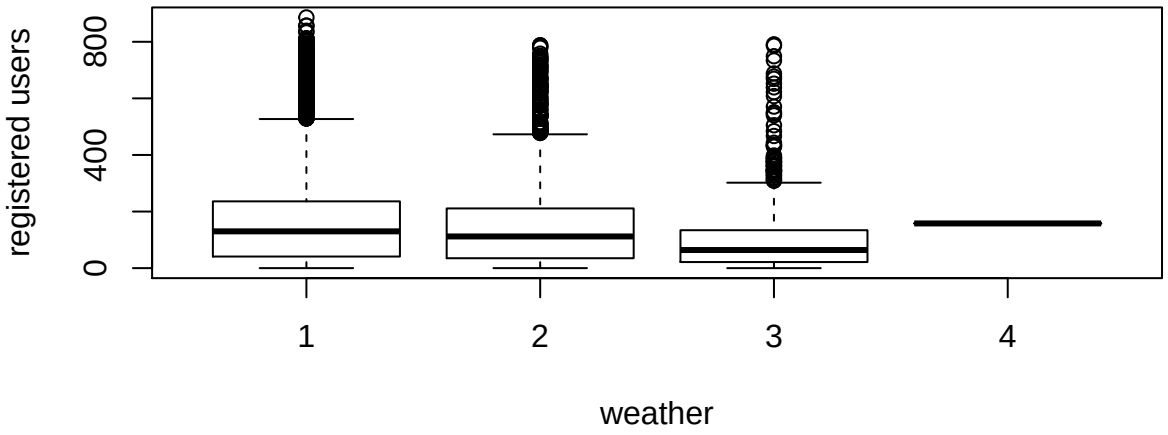
```
# separate again as train and test set
train = data[as.integer(substr(data$datetime, 9, 10)) < 20,]
test = data[as.integer(substr(data$datetime, 9, 10)) > 19,]
```

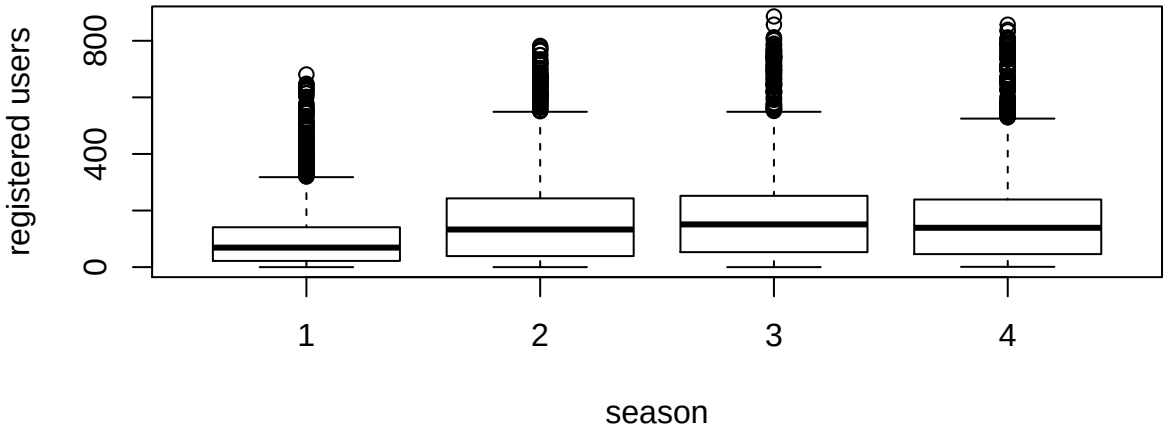
19.2 Log transform

```
# log transformation for some skewed variables,
# which can be seen from their distribution
train$reg1   = train$registered + 1
train$cas1   = train$casual + 1
train$logcas = log(train$cas1)
train$logreg = log(train$reg1)
test$logreg  = 0
test$logcas  = 0
```

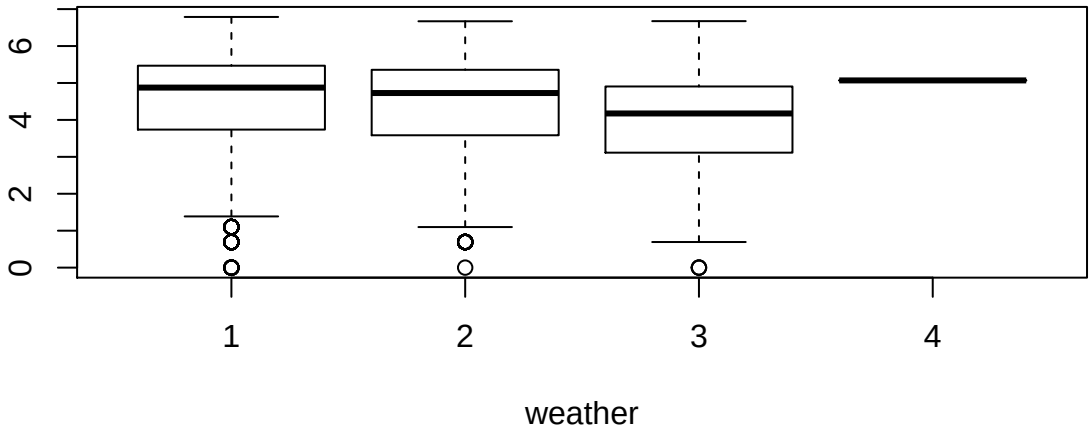
19.2.1 Plot by weather, by season

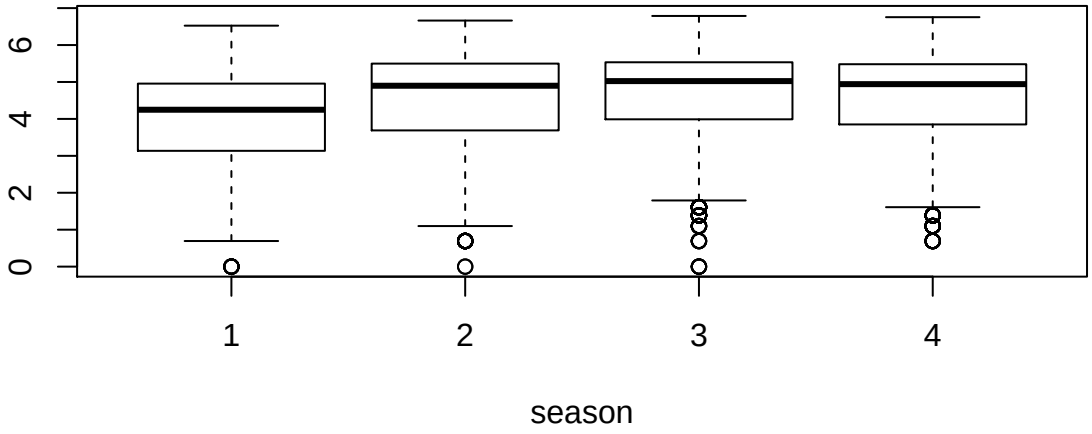
```
# cartesian plot
par(mfrow=c(2,1))
boxplot(train$registered ~ train$weather, xlab="weather", ylab="registered users")
boxplot(train$registered ~ train$season, xlab="season", ylab="registered users")
```





```
# semilog plot
par(mfrow=c(2,1))
boxplot(train$logreg ~ train$weather, xlab = "weather")
boxplot(train$logreg ~ train$season, xlab = "season")
```





19.3 Predicting for registered and casual users, test dataset

```
# final model building using random forest
# note that we build different models for predicting for
# registered and casual users
# this was seen as giving best result after a lot of experimentation
set.seed(415)
fit1 <- randomForest(logreg ~ hour + workingday + day + holiday + day_type +
                         temp_reg + humidity + atemp + windspeed + season +
                         weather + dp_reg + weekend + year + year_part,
                     data = train,
                     importance = TRUE,
                     ntree = 250)
```

```
pred1 = predict(fit1, test)
test$logreg = pred1
```

```
# casual users
set.seed(415)
fit2 <- randomForest(logcas ~ hour + day_type + day + humidity + atemp +
                         temp_cas + windspeed + season + weather + holiday +
                         workingday + dp_cas + weekend + year + year_part,
                     data = train, importance = TRUE, ntree = 250)

pred2 = predict(fit2, test)
test$logcas = pred2
```

19.4 Preparing and exporting results

```
# creating the final submission file
# reverse log conversion
test$registered <- exp(test$logreg) - 1
test$casual     <- exp(test$logcas) - 1
test$count      <- test$casual + test$registered

r <- data.frame(datetime = test$datetime,
                casual = test$casual,
                registered = test$registered)

print(sum(r$casual))
```

```
#> [1] 205803.7
```

```
print(sum(r$registered))
```

```
#> [1] 962834.2
```

```
s <- data.frame(datetime = test$datetime, count = test$count)
write.csv(s, file =file.path(data_out_dir, "bike-submit.csv"), row.names = FALSE)

# sum(cas+reg) = 1168638
# month number now is correct
```

After following the steps mentioned above, you can score 0.38675 on Kaggle leaderboard i.e. top 5 percentile of total participants. As you might have seen, we have not applied any extraordinary science in getting to this level. But, the real competition starts here. I would like to see, if I can improve this further by use of more features and some more advanced modeling techniques.

Chapter 20

End Notes

In this article, we have looked at structured approach of problem solving and how this method can help you to improve performance. I would recommend you to generate hypothesis before you deep dive in the data set as this technique will not limit your thought process. You can improve your performance by applying advanced techniques (or ensemble methods) and understand your data trend better.

You can find the complete solution here : GitHub Link

```
# this is the older submission. months were incomplete
old <- read.csv(file = file.path(data_raw_dir, "bike-submit-old.csv"))
```

```
#> Error in file(file, "rt"): cannot open the connection
```

```
sum(old$count)
```

```
#> Error in eval(expr, envir, enclos): object 'old' not found
```

Chapter 21

Breast Cancer Wisconsin

Source: https://shiring.github.io/machine_learning/2017/01/15/rfe_ga_post

21.1 Read and process the data

```
bc_data <- read.table(file.path(data_raw_dir, "breast-cancer-wisconsin.data"),
                      header = FALSE, sep = ",")
```

```
# assign the column names
colnames(bc_data) <- c("sample_code_number", "clump_thickness",
                       "uniformity_of_cell_size", "uniformity_of_cell_shape",
                       "marginal_adhesion", "single_epithelial_cell_size",
                       "bare_nuclei", "bland_chromatin", "normal_nucleoli",
                       "mitosis", "classes")

# change classes from numeric to character
bc_data$classes <- ifelse(bc_data$classes == "2", "benign",
                          ifelse(bc_data$classes == "4", "malignant", NA))

# if query sign make NA
bc_data[bc_data == "?"] <- NA

# how many NAs are in the data
length(which(is.na(bc_data)))
```

```
[1] 16
```

```
names(bc_data)
```

```
 [1] "sample_code_number"          "clump_thickness"
 [3] "uniformity_of_cell_size"     "uniformity_of_cell_shape"
 [5] "marginal_adhesion"           "single_epithelial_cell_size"
 [7] "bare_nuclei"                 "bland_chromatin"
 [9] "normal_nucleoli"             "mitosis"
[11] "classes"
```

21.1.1 Missing data

```
# impute missing data
library(mice)

# skip these columns: sample_code_number and classes
# convert to numeric
bc_data[,2:10] <- apply(bc_data[, 2:10], 2, function(x) as.numeric(as.character(x)))

# impute but mute
dataset_impute <- mice(bc_data[, 2:10],  print = FALSE)

# bind "classes" with the rest. skip "sample_code_number"
bc_data <- cbind(bc_data[, 11, drop = FALSE],
                 mice::complete(dataset_impute, action =1))

bc_data$classes <- as.factor(bc_data$classes)

# how many benign and malignant cases are there?
summary(bc_data$classes)
```

```
   benign malignant
      458       241
```

```
# confirm NAs have been removed
length(which(is.na(bc_data)))
```

```
[1] 0
```

```
str(bc_data)
```

```
'data.frame':   699 obs. of  10 variables:
 $ classes                    : Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 1 1 1 1 ...
 $ clump_thickness            : num  5 5 3 6 4 8 1 2 2 4 ...
 $ uniformity_of_cell_size    : num  1 4 1 8 1 10 1 1 1 2 ...
 $ uniformity_of_cell_shape   : num  1 4 1 8 1 10 1 2 1 1 ...
 $ marginal_adhesion          : num  1 5 1 1 3 8 1 1 1 1 ...
 $ single_epithelial_cell_size: num  2 7 2 3 2 7 2 2 2 2 ...
 $ bare_nuclei                : num  1 10 2 4 1 10 10 1 1 1 ...
 $ bland_chromatin            : num  3 3 3 3 3 9 3 3 1 2 ...
 $ normal_nucleoli            : num  1 2 1 7 1 7 1 1 1 1 ...
 $ mitosis                    : num  1 1 1 1 1 1 1 1 5 1 ...
```

21.2 Principal Component Analysis (PCA)

To get an idea about the dimensionality and variance of the datasets, I am first looking at PCA plots for samples and features. The first two principal components (PCs) show the two components that explain the majority of variation in the data.

After defining my custom `ggplot2` theme, I am creating a function that performs the PCA (using the `pcaGoPromoter` package), calculates ellipses of the data points (with the `ellipse` package) and produces the plot with `ggplot2`. Some of the features in datasets 2 and 3 are not very distinct and overlap in the PCA plots, therefore I am also plotting hierarchical clustering dendrograms.

21.2.0.1 theme

```
# plotting theme

library(ggplot2)

my_theme <- function(base_size = 12, base_family = "sans"){
  theme_minimal(base_size = base_size, base_family = base_family) +
  theme(
    axis.text = element_text(size = 12),
    axis.text.x = element_text(angle = 0, vjust = 0.5, hjust = 0.5),
    axis.title = element_text(size = 14),
    panel.grid.major = element_line(color = "grey"),
    panel.grid.minor = element_blank(),
    panel.background = element_rect(fill = "aliceblue"),
    strip.background = element_rect(fill = "navy", color = "navy", size = 1),
    strip.text = element_text(face = "bold", size = 12, color = "white"),
    legend.position = "right",
    legend.justification = "top",
    legend.background = element_blank(),
    panel.border = element_rect(color = "grey", fill = NA, size = 0.5)
  )
}

theme_set(my_theme())
```

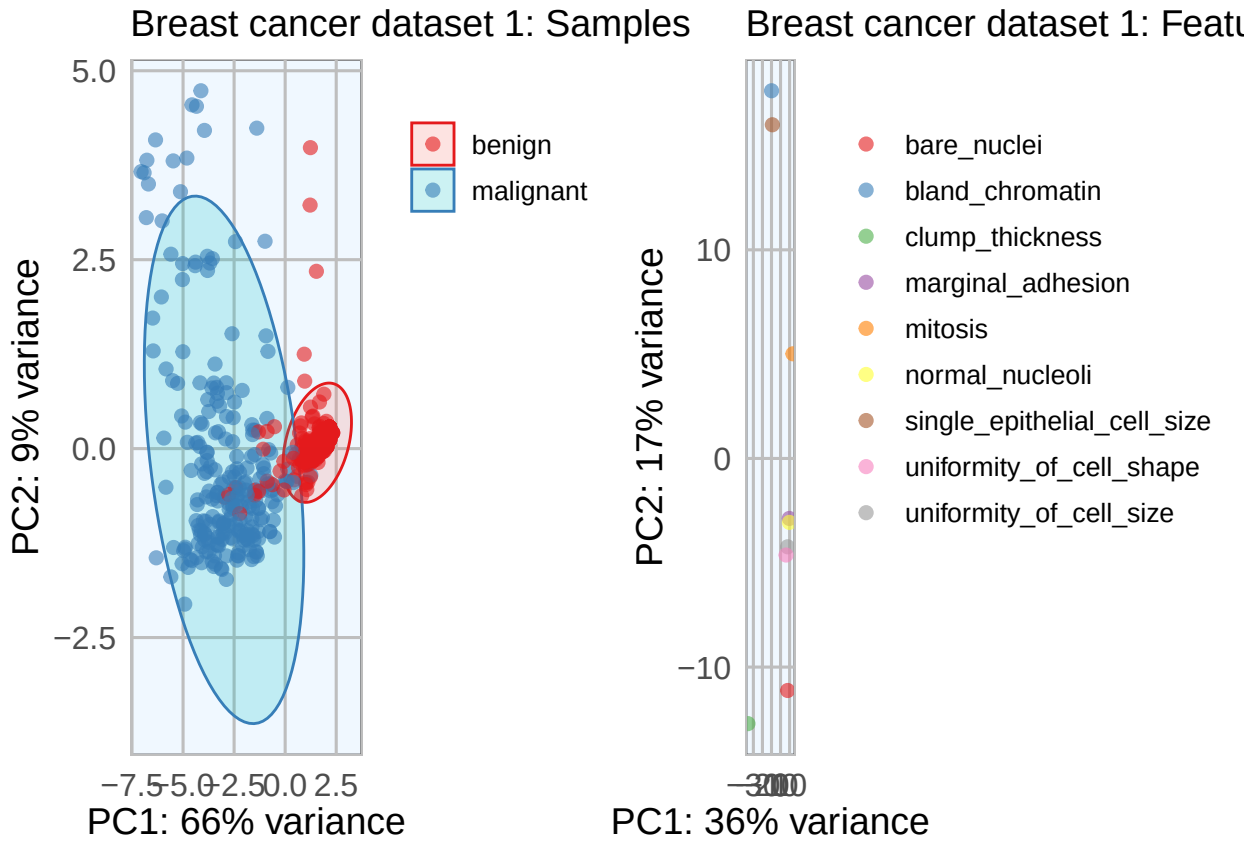
21.2.0.2 PCA function

```
# function for PCA plotting
library(pcaGoPromoter)                  # install from BioConductor
library(ellipse)

pca_func <- function(data, groups, title, print_ellipse = TRUE) {

  # perform pca and extract scores for all principal components: PC1:PC9
  pcaOutput <- pca(data, printDropped = FALSE, scale = TRUE, center = TRUE)
  pcaOutput2 <- as.data.frame(pcaOutput$scores)

  # define groups for plotting. will group the classes
  pcaOutput2$groups <- groups

  # when plotting samples calculate ellipses for plotting
  # (when plotting features, there are no replicates)
  if (print_ellipse) {
    # group and summarize by classes: benign, malignant
    # centroids w/3 columns: groups, PC1, PC2
    centroids <- aggregate(cbind(PC1, PC2) ~ groups, pcaOutput2, mean)
    # bind for the two groups (classes)
    # conf.rgn w/3 columns: groups, PC1, PC2
    conf.rgn  <- do.call(rbind, lapply(unique(pcaOutput2$groups), function(t)
      data.frame(groups = as.character(t),
                 # ellipse data for PC1 and PC2
```

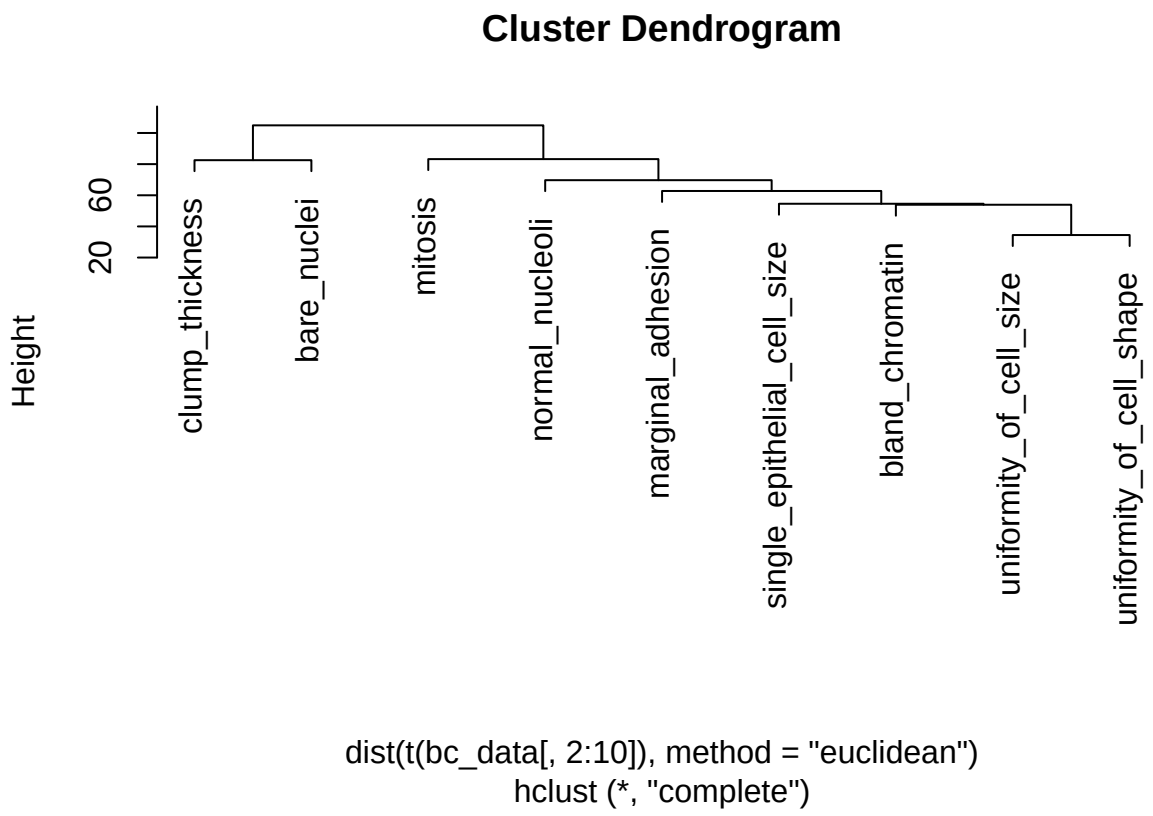
```
                ellipse(cov(pcaOutput2[pcaOutput2$groups == t, 1:2]),
                      centre = as.matrix(centroids[centroids$groups == t, 2:3]),
                      level = 0.95),
                stringsAsFactors = FALSE)))

  plot <- ggplot(data = pcaOutput2, aes(x = PC1, y = PC2,
                                        group = groups,
                                        color = groups)) +
    geom_polygon(data = conf.rgn, aes(fill = groups), alpha = 0.2) + # ellipses
    geom_point(size = 2, alpha = 0.6) +
    scale_color_brewer(palette = "Set1") +
    labs(title = title,
         color = "",
         fill = "",
         x = paste0("PC1: ", round(pcaOutput$pov[1], digits = 2) * 100,
                    "% variance"),
         y = paste0("PC2: ", round(pcaOutput$pov[2], digits = 2) * 100,
                    "% variance"))

} else {

  # if < 10 groups (e.g. the predictor classes) have colors from RColorBrewer
  if (length(unique(pcaOutput2$groups)) <= 10) {

    plot <- ggplot(data = pcaOutput2, aes(x = PC1, y = PC2,
                                          group = groups,
                                          color = groups)) +
      geom_point(size = 2, alpha = 0.6) +
      scale_color_brewer(palette = "Set1") +
      labs(title = title,
           color = "",
           fill = "",
           x = paste0("PC1: ", round(pcaOutput$pov[1], digits = 2) * 100,
                      "% variance"),
           y = paste0("PC2: ", round(pcaOutput$pov[2], digits = 2) * 100,
                      "% variance"))

  } else {
    # otherwise use the default rainbow colors
    plot <- ggplot(data = pcaOutput2, aes(x = PC1, y = PC2,
                                          group = groups, color = groups)) +
      geom_point(size = 2, alpha = 0.6) +
      labs(title = title,
           color = "",
           fill = "",
           x = paste0("PC1: ", round(pcaOutput$pov[1], digits = 2) * 100,
                      "% variance"),
           y = paste0("PC2: ", round(pcaOutput$pov[2], digits = 2) * 100,
                      "% variance"))
  }
}

return(plot)
```

```
}
```

```
library(gridExtra)
library(grid)

# plot all data. one row is a feature
p1 <- pca_func(data = t(bc_data[, 2:10]),
               groups = as.character(bc_data$classes),
               title = "Breast cancer dataset 1: Samples")

# plot features only. features as columns
p2 <- pca_func(data = bc_data[, 2:10],
               groups = as.character(colnames(bc_data[, 2:10])),
               title = "Breast cancer dataset 1: Features", print_ellipse = FALSE)

grid.arrange(p1, p2, ncol = 2)
```

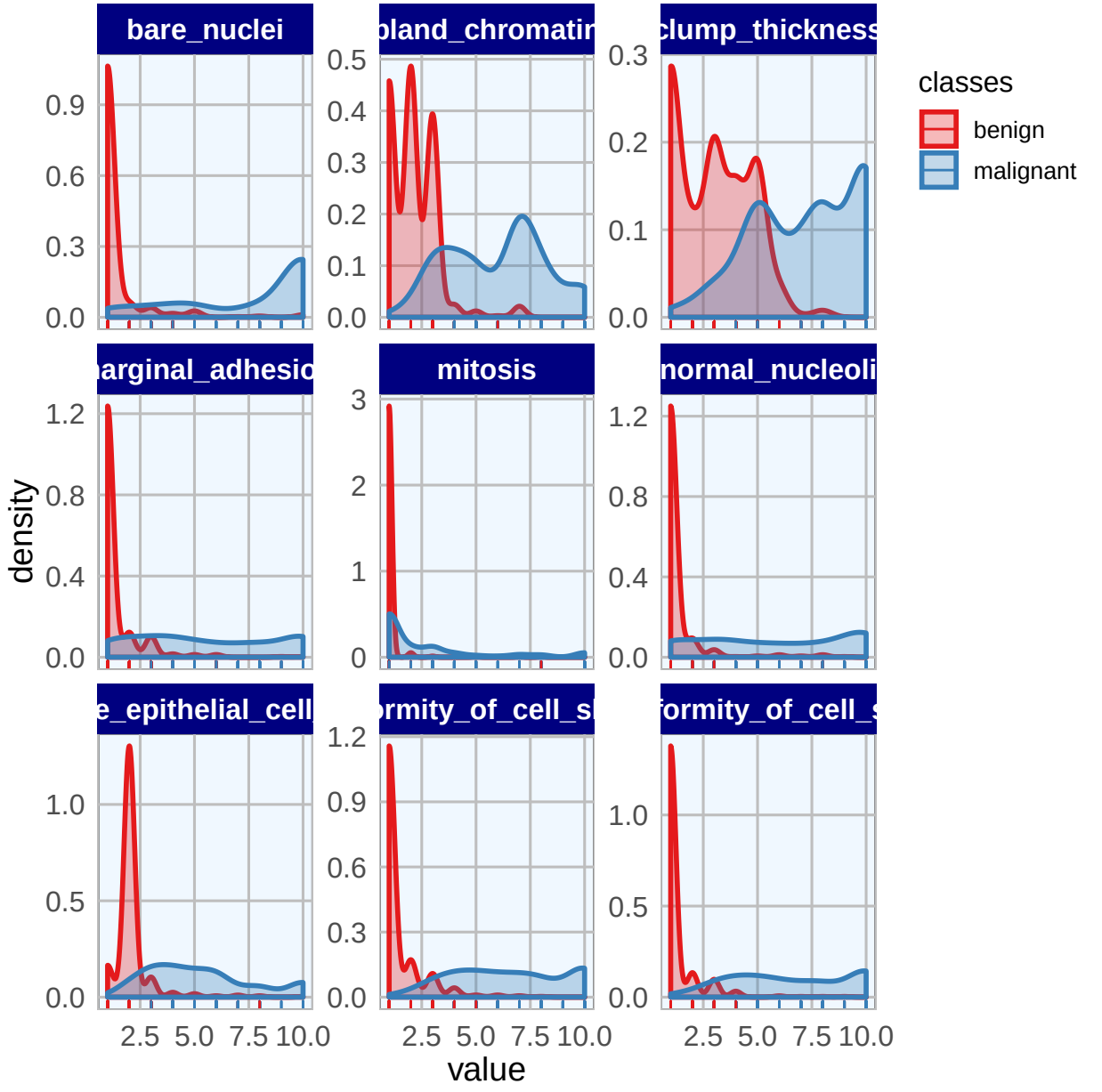



```
h_1 <- hclust(dist(t(bc_data[, 2:10]), method = "euclidean"), method = "complete")
plot(h_1)
```



21.2.1 density plots vs class

```
# density plot showing the feature vs classes
library(tidyr)

# gather data. from column clump_thickness to mitosis
bc_data_gather <- bc_data %>%
  gather(measure, value, clump_thickness:mitosis)

ggplot(data = bc_data_gather, aes(x = value, fill = classes, color = classes)) +
  geom_density(alpha = 0.3, size = 1) +
  geom_rug() +
  scale_fill_brewer(palette = "Set1") +
  scale_color_brewer(palette = "Set1") +
  facet_wrap( ~ measure, scales = "free_y", ncol = 3)
```



21.3 Feature importance

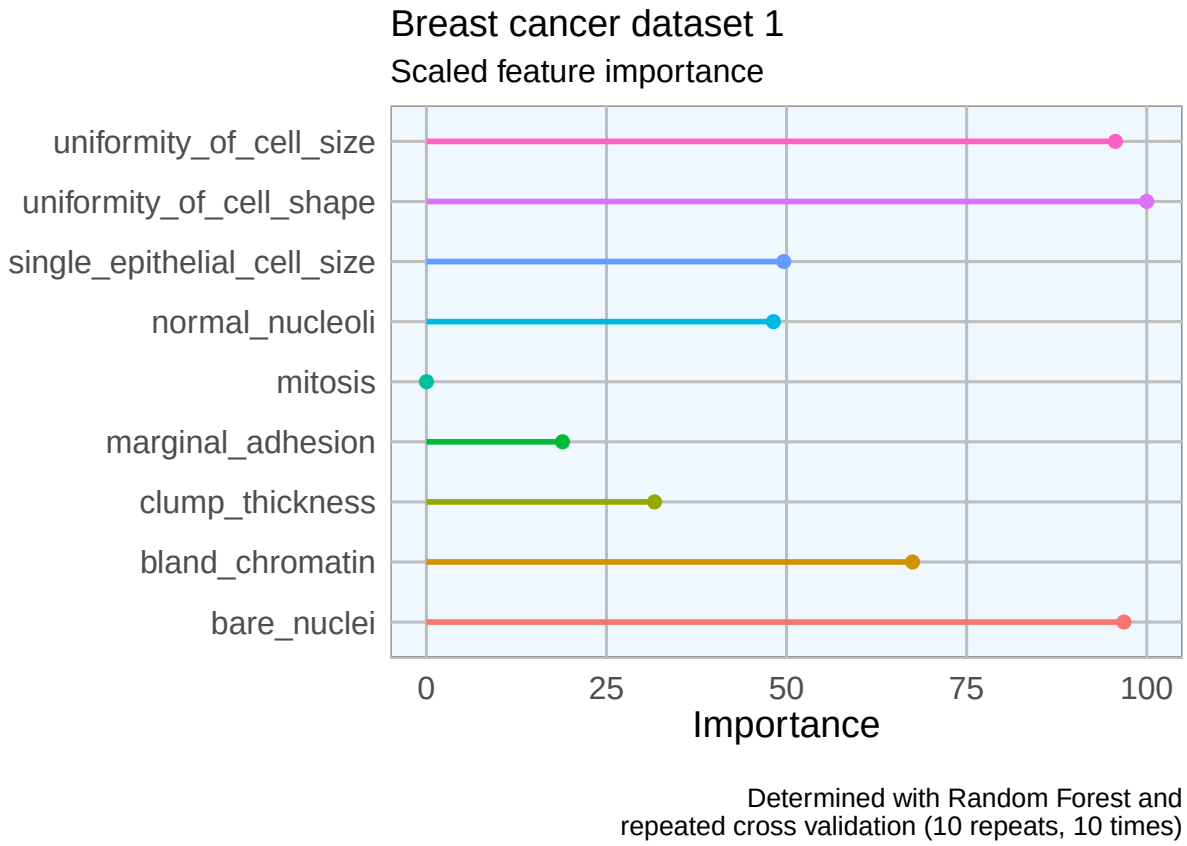
To get an idea about the feature's respective importances, I'm running Random Forest models with 10 x 10 cross validation using the `caret` package. If I wanted to use feature importance to select features for modeling, I would need to perform it on the training data instead of on the complete dataset. But here, I only want to use it to get acquainted with my data. I am again defining a function that estimates the feature importance and produces a plot.

```
library(caret)
# library(doParallel) # parallel processing
# registerDoParallel()

# prepare training scheme
control <- trainControl(method = "repeatedcv", number = 10, repeats = 10)
```

```
feature_imp <- function(model, title) {
  # estimate variable importance
  importance <- varImp(model, scale = TRUE)
  # prepare dataframes for plotting
  importance_df_1 <- importance$importance
  importance_df_1$group <- rownames(importance_df_1)

  importance_df_2 <- importance_df_1
  importance_df_2$Overall <- 0
  importance_df <- rbind(importance_df_1, importance_df_2)

  plot <- ggplot() +
    geom_point(data = importance_df_1, aes(x = Overall,
                                           y = group,
                                           color = group), size = 2) +
    geom_path(data = importance_df, aes(x = Overall,
                                        y = group,
                                        color = group,
                                        group = group), size = 1) +
    theme(legend.position = "none") +
    labs(
      x = "Importance",
      y = "",
      title = title,
      subtitle = "Scaled feature importance",
      caption = "\nDetermined with Random Forest and
      repeated cross validation (10 repeats, 10 times)"
    )
  return(plot)
}
```

```
# train the model
set.seed(27)
imp_1 <- train(classes ~ ., data = bc_data, method = "rf",
               preProcess = c("scale", "center"),
               trControl = control)
```

```
p1 <- feature_imp(imp_1, title = "Breast cancer dataset 1")
p1
```



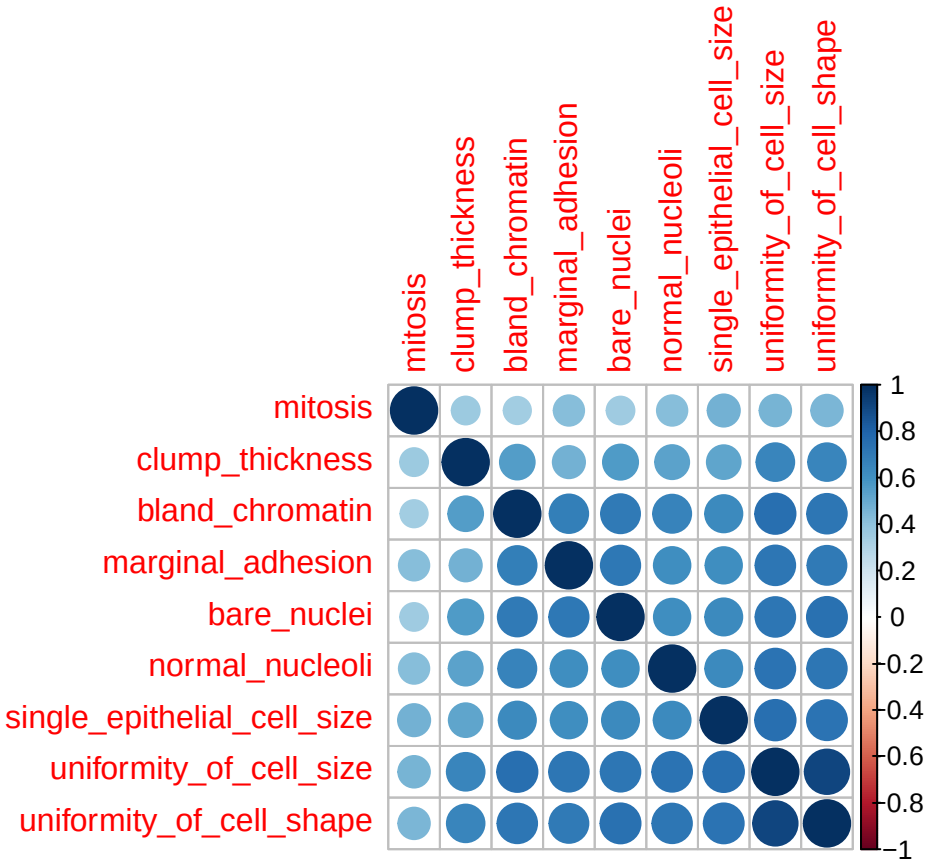
21.4 Feature Selection

1. By correlation
2. By Recursive Feature Elimination
3. By Genetic Algorithm

```
set.seed(27)
bc_data_index <- createDataPartition(bc_data$classes, p = 0.7, list = FALSE)
bc_data_train <- bc_data[bc_data_index, ]
bc_data_test  <- bc_data[-bc_data_index, ]
```

21.4.1 Correlation

```
library(corrplot)

# calculate correlation matrix
corMatMy <- cor(bc_data_train[, -1])
corrplot(corMatMy, order = "hclust")
```




```
# Apply correlation filter at 0.70,
highlyCor <- colnames(bc_data_train[, -1])[findCorrelation(corMatMy,
                                                           cutoff = 0.7,
                                                           verbose = TRUE)]
```

```
Compare row 2  and column  3 with corr  0.913
  Means:  0.716 vs 0.601 so flagging column 2
Compare row 3  and column  7 with corr  0.725
  Means:  0.677 vs 0.578 so flagging column 3
Compare row 7  and column  6 with corr  0.702
  Means:  0.6 vs 0.544 so flagging column 7
Compare row 6  and column  4 with corr  0.715
  Means:  0.579 vs 0.525 so flagging column 6
All correlations <= 0.7
```

```
# which variables are flagged for removal?
highlyCor
```

```
[1] "uniformity_of_cell_size"  "uniformity_of_cell_shape"
[3] "bland_chromatin"          "bare_nuclei"
```

```
# then we remove these variables
bc_data_cor <- bc_data_train[, which(!colnames(bc_data_train) %in% highlyCor)]
names(bc_data_cor)
```

```
[1] "classes"                    "clump_thickness"
[3] "marginal_adhesion"          "single_epithelial_cell_size"
[5] "normal_nucleoli"            "mitosis"
```

```
# confirm features were removed
outersect <- function(x, y) {
  sort(c(setdiff(x, y),
         setdiff(y, x)))
}

outersect(names(bc_data_cor), names(bc_data_train))
```

```
[1] "bare_nuclei"              "bland_chromatin"
[3] "uniformity_of_cell_shape" "uniformity_of_cell_size"
```

Four features removed

21.4.2 Recursive Feature Elimination (RFE)

```
# ensure the results are repeatable
set.seed(7)

# define the control using a random forest selection function with cross validation
control <- rfeControl(functions = rfFuncs, method = "cv", number = 10)

# run the RFE algorithm
results_1 <- rfe(x = bc_data_train[, -1],
                 y = bc_data_train$classes,
                 sizes = c(1:9),
                 rfeControl = control)

# chosen features
predictors(results_1)
```

```
[1] "bare_nuclei"                "uniformity_of_cell_size"
[3] "clump_thickness"            "uniformity_of_cell_shape"
[5] "bland_chromatin"            "marginal_adhesion"
[7] "mitosis"                    "normal_nucleoli"
[9] "single_epithelial_cell_size"
```

```
# subset the chosen features
sel_cols <- which(colnames(bc_data_train) %in% predictors(results_1))
bc_data_rfe <- bc_data_train[, c(1, sel_cols)]
names(bc_data_rfe)
```

```
 [1] "classes"                    "clump_thickness"
 [3] "uniformity_of_cell_size"    "uniformity_of_cell_shape"
 [5] "marginal_adhesion"          "single_epithelial_cell_size"
 [7] "bare_nuclei"                "bland_chromatin"
 [9] "normal_nucleoli"            "mitosis"
```

```
# confirm features removed by RFE
outersect(names(bc_data_rfe), names(bc_data_train))
```

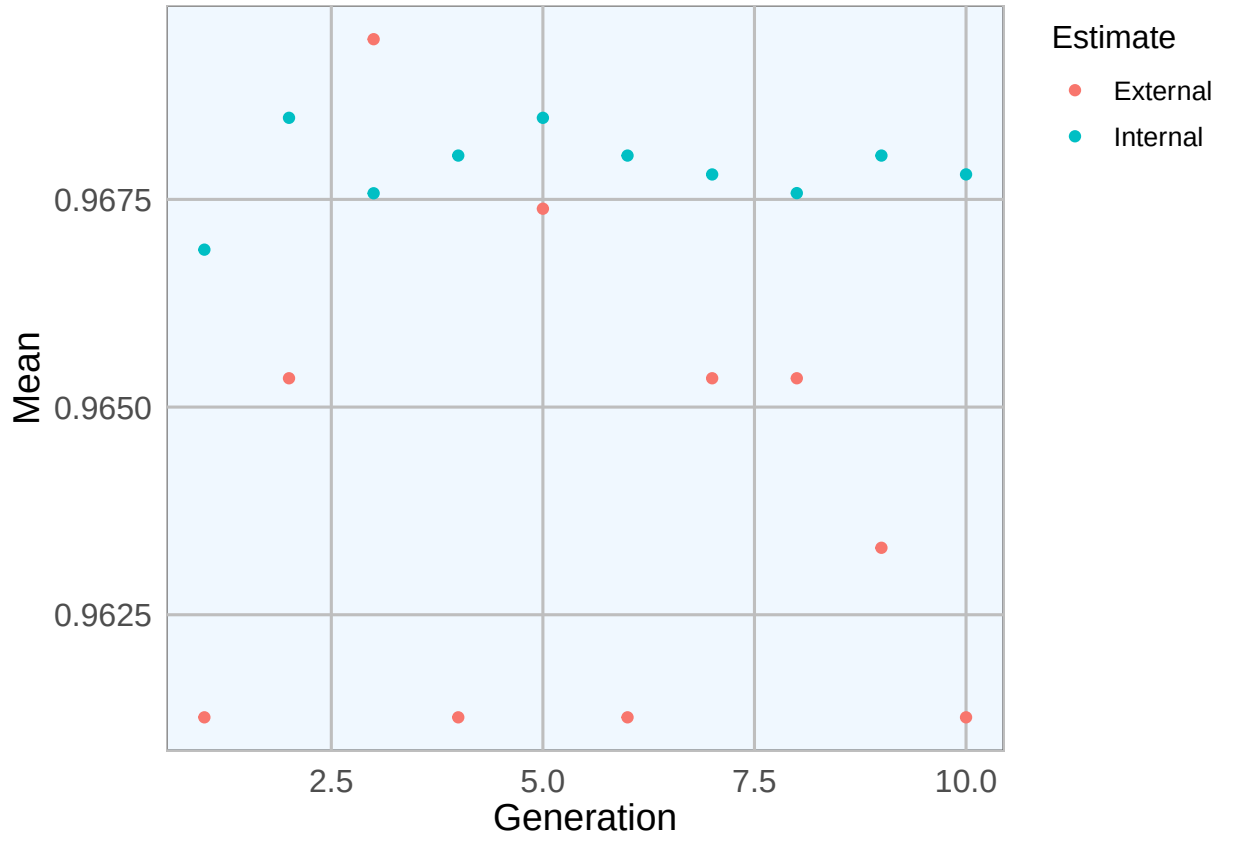
```
character(0)
```

No features removed with RFE

21.4.3 Genetic Algorithm (GA)

```
library(dplyr)

ga_ctrl <- gafsControl(functions = rfGA, # Assess fitness with RF
                       method = "cv",    # 10 fold cross validation
                       genParallel = TRUE, # Use parallel programming
                       allowParallel = TRUE)

lev <- c("malignant", "benign")     # Set the levels

set.seed(27)
model_1 <- gafs(x = bc_data_train[, -1], y = bc_data_train$classes,
                  iters = 10, # generations of algorithm
                  popSize = 5, # population size for each generation
                  levels = lev,
                  gafsControl = ga_ctrl)
```

```
plot(model_1) # Plot mean fitness (AUC) by generation
```




```
# features
model_1$ga$final
```

```
[1] "clump_thickness"         "uniformity_of_cell_size"
[3] "marginal_adhesion"       "bare_nuclei"
[5] "bland_chromatin"         "normal_nucleoli"
[7] "mitosis"
```

```
# select features
sel_cols_ga <- which(colnames(bc_data_train) %in% model_1$ga$final)
bc_data_ga <- bc_data_train[, c(1, sel_cols_ga)]
names(bc_data_ga)
```

```
[1] "classes"                "clump_thickness"
[3] "uniformity_of_cell_size" "marginal_adhesion"
[5] "bare_nuclei"            "bland_chromatin"
[7] "normal_nucleoli"        "mitosis"
```

```
# features removed GA
outersect(names(bc_data_ga), names(bc_data_train))
```

```
[1] "single_epithelial_cell_size" "uniformity_of_cell_shape"
```

Two features removed with GA.

21.5 Model comparison

21.5.1 Using all features

```
set.seed(27)
model_bc_data_all <- train(classes ~ .,
                           data = bc_data_train,
                           method = "rf",
                           preProcess = c("scale", "center"),
                           trControl = trainControl(method = "repeatedcv",
                                                    number = 5, repeats = 10,
                                                    verboseIter = FALSE))
```

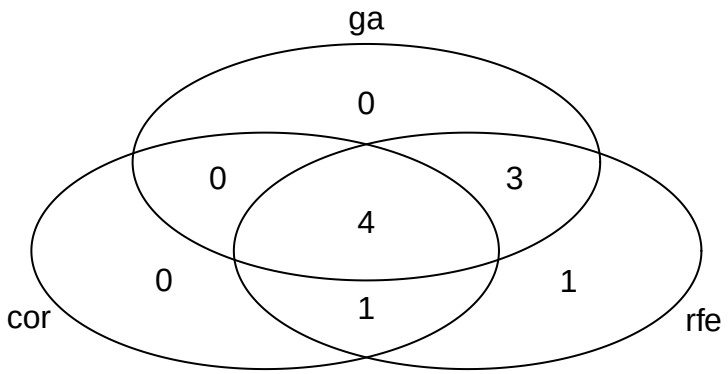
```
# confusion matrix
cm_all_1 <- confusionMatrix(predict(model_bc_data_all, bc_data_test[, -1]), bc_data_test$classes)
cm_all_1
```

```
Confusion Matrix and Statistics

           Reference
Prediction  benign malignant
  benign       134         5
  malignant      3        67

               Accuracy : 0.9617
                 95% CI : (0.926, 0.9833)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9147

 Mcnemar's Test P-Value : 0.7237

            Sensitivity : 0.9781
            Specificity : 0.9306
         Pos Pred Value : 0.9640
```

```
         Neg Pred Value : 0.9571
             Prevalence : 0.6555
         Detection Rate : 0.6411
   Detection Prevalence : 0.6651
      Balanced Accuracy : 0.9543

       'Positive' Class : benign
```

21.5.2 Compare selection methods

```
# compare features selected by the three methods
library(gplots)

venn_list <- list(cor = colnames(bc_data_cor)[-1],
                  rfe = colnames(bc_data_rfe)[-1],
                  ga  = colnames(bc_data_ga)[-1])

venn <- venn(venn_list)
```




```
venn
```

```
    num cor rfe ga
000   0   0   0  0
001   0   0   0  1
010   1   0   1  0
011   3   0   1  1
100   0   1   0  0
101   0   1   0  1
110   1   1   1  0
111   4   1   1  1
attr(,"intersections")
attr(,"intersections")$`cor:rfe:ga`
[1] "clump_thickness"   "marginal_adhesion" "normal_nucleoli"
[4] "mitosis"

attr(,"intersections")$rfe
[1] "uniformity_of_cell_shape"

attr(,"intersections")$`rfe:ga`
[1] "uniformity_of_cell_size" "bare_nuclei"
[3] "bland_chromatin"
```

```
attr(,"intersections")$`cor:rfe`
[1] "single_epithelial_cell_size"

attr(,"class")
[1] "venn"
```

4 out of 10 features were chosen by all three methods; the biggest overlap is seen between GA and RFE with 7 features. RFE and GA both retained 8 features for modeling, compared to only 5 based on the correlation method.

21.5.3 Correlation

```
# correlation
set.seed(127)
model_bc_data_cor <- train(classes ~ .,
                 data = bc_data_cor,
                 method = "rf",
                 preProcess = c("scale", "center"),
                 trControl = trainControl(method = "repeatedcv", number = 5, repeats = 10, verboseIter =
```

```
cm_cor_1 <- confusionMatrix(predict(model_bc_data_cor, bc_data_test[, -1]), bc_data_test$classes)
cm_cor_1
```

```
Confusion Matrix and Statistics

           Reference
Prediction  benign malignant
  benign       132         6
  malignant      5        66

               Accuracy : 0.9474
                 95% CI : (0.9078, 0.9734)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8831

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9635
            Specificity : 0.9167
         Pos Pred Value : 0.9565
         Neg Pred Value : 0.9296
             Prevalence : 0.6555
         Detection Rate : 0.6316
   Detection Prevalence : 0.6603
      Balanced Accuracy : 0.9401

       'Positive' Class : benign
```

21.5.4 Recursive Feature Elimination

```
set.seed(127)
model_bc_data_rfe <- train(classes ~ .,
                           data = bc_data_rfe,
                           method = "rf",
                           preProcess = c("scale", "center"),
                           trControl = trainControl(method = "repeatedcv",
                                                    number = 5, repeats = 10,
                                                    verboseIter = FALSE))
```

```
cm_rfe_1 <- confusionMatrix(predict(model_bc_data_rfe, bc_data_test[, -1]), bc_data_test$classes)
cm_rfe_1
```

```
Confusion Matrix and Statistics

           Reference
Prediction  benign malignant
  benign       134         5
  malignant      3        67

               Accuracy : 0.9617
                 95% CI : (0.926, 0.9833)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9147

 Mcnemar's Test P-Value : 0.7237

            Sensitivity : 0.9781
            Specificity : 0.9306
         Pos Pred Value : 0.9640
         Neg Pred Value : 0.9571
             Prevalence : 0.6555
         Detection Rate : 0.6411
   Detection Prevalence : 0.6651
      Balanced Accuracy : 0.9543

       'Positive' Class : benign
```

21.5.5 GA

```
set.seed(127)
model_bc_data_ga <- train(classes ~ .,
                          data = bc_data_ga,
                          method = "rf",
                          preProcess = c("scale", "center"),
                          trControl = trainControl(method = "repeatedcv",
                                                   number = 5, repeats = 10,
                                                   verboseIter = FALSE))
```

```
cm_ga_1 <- confusionMatrix(predict(model_bc_data_ga, bc_data_test[, -1]), bc_data_test$classes)
cm_ga_1
```

```
Confusion Matrix and Statistics

           Reference
Prediction  benign malignant
  benign       134         4
  malignant      3        68

               Accuracy : 0.9665
                 95% CI : (0.9322, 0.9864)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9256

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9781
            Specificity : 0.9444
         Pos Pred Value : 0.9710
         Neg Pred Value : 0.9577
             Prevalence : 0.6555
         Detection Rate : 0.6411
   Detection Prevalence : 0.6603
      Balanced Accuracy : 0.9613

       'Positive' Class : benign
```

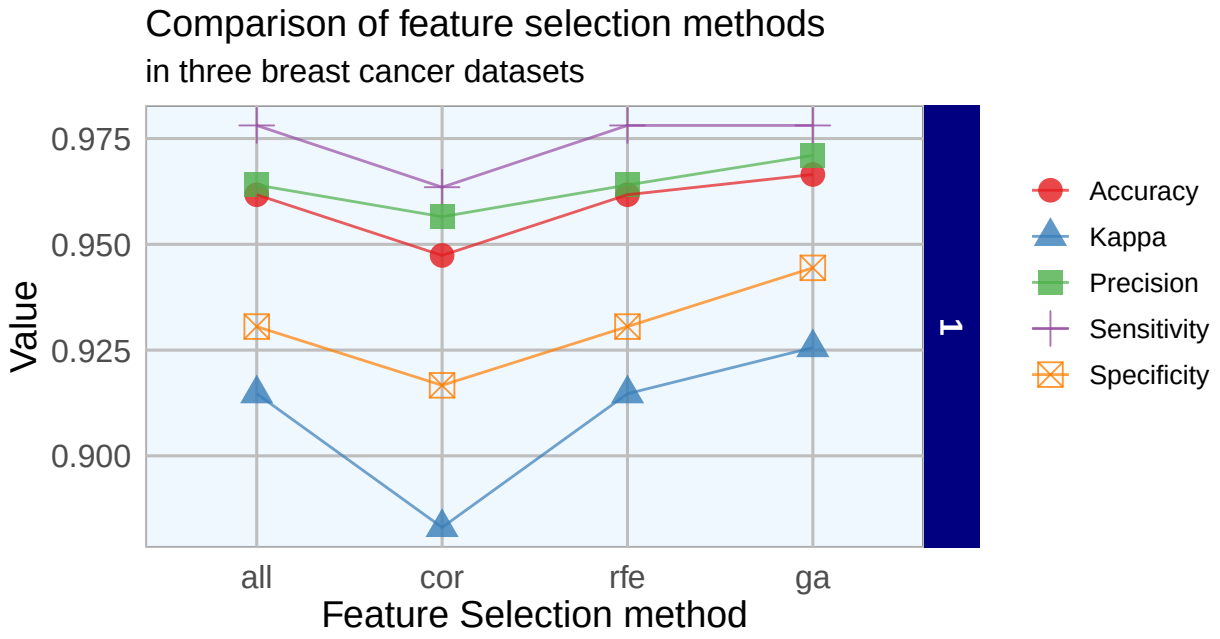
21.6 Create comparison tables

```
# take "overall" variable only from Confusion Matrix
overall <- data.frame(dataset = 1,
           model = rep(c("all", "cor", "rfe", "ga"), 1),
           rbind(cm_all_1$overall,
                 cm_cor_1$overall,
                 cm_rfe_1$overall,
                 cm_ga_1$overall)
)

# convert to tidy data
library(tidyr)
overall_gather <- overall[, 1:4] %>%     # take the first columns:
  gather(measure, value, Accuracy:Kappa) # dataset, model, Accuracy and Kappa
```

```
# take "byClass" variable only from Confusion Matrix
byClass <- data.frame(dataset = 1,
                      model = rep(c("all", "cor", "rfe", "ga"), 1),
                      rbind(cm_all_1$byClass,
                      cm_cor_1$byClass,
```

```
                      cm_rfe_1$byClass,
                      cm_ga_1$byClass)
)

# convert to tidy data
byClass_gather <- byClass[, c(1:4, 7)] %>%      # select columns: dataset, model
  gather(measure, value, Sensitivity:Precision) # Sensitiv, Specific, Precis
```

```
# join the two tables
overall_byClass_gather <- rbind(overall_gather, byClass_gather)
overall_byClass_gather <- within(
  overall_byClass_gather, model <- factor(model,
                                          levels = c("all", "cor", "rfe", "ga")))
                                          # convert to factor

ggplot(overall_byClass_gather, aes(x = model, y = value, color = measure,
                                   shape = measure, group = measure)) +
  geom_point(size = 4, alpha = 0.8) +
  geom_path(alpha = 0.7) +
  scale_colour_brewer(palette = "Set1") +
  facet_grid(dataset ~ ., scales = "free_y") +
  labs(
    x = "Feature Selection method",
    y = "Value",
    color = "",
    shape = "",
    title = "Comparison of feature selection methods",
    subtitle = "in three breast cancer datasets",
    caption = "\nBreast Cancer Wisconsin (Diagnostic) Data Sets: 1, 2 & 3
    Street et al., 1993;
    all: no feature selection
    cor: features with correlation > 0.7 removed
    rfe: Recursive Feature Elimination
    ga: Genetic Algorithm"
  )
```



Breast Cancer Wisconsin (Diagnostic) Data Sets: 1, 2 & 3
Street et al., 1993;
all: no feature selection
cor: features with correlation > 0.7 removed
rfe: Recursive Feature Elimination
ga: Genetic Algorithm

1. Less accurate: selection of features by correlation
2. More accurate: genetic algorithm
3. Including all features is more accurate to removing features by correlation.

21.7 Notes

`pcaGoPromoter` is a BioConductor package. Its dependencies are `BioGenerics`, `AnnotationDbi` and `BioStrings`, which at their turn require `DBI` and `RSQLite` packages from CRAN. Install first those from CRAN, and then move to install `pcaGoPromoter`.