A Minimal Book Example

Yihui Xie

2019-05-13

Contents

1 Prerequisites **7**

2 Compare classification algorithms **9**
2.1 Train the models . . . 10
2.2 Compare models . . . 10

3 Temperature modeling using nested dataframes **17**
3.1 Prepare the data . . . 17
3.2 Define the models . . . 21
3.3 Test modeling on one dataset . . . 22
3.4 Making a nested dataframe . . . 25
3.5 Apply multiple models on a nested structure . . . 27
3.6 Using broom package to look at model-statistics . . . 33

4 Standalone Model **35**
4.1 Load libraries . . . 35
4.2 Explore data . . . 35
4.3 Apply tuning parameters for final model . . . 38
4.4 Save model . . . 39
4.5 Use the saved model . . . 39
4.6 Make prediction with new data . . . 39

5 Glass classification **41**

6 Ozone SVM **45**

7 A gentle introduction to support vector machines using R **47**
7.1 Support vector machines in R . . . 47
7.2 SVM on `iris` dataset . . . 47
7.3 SVM with Radial Basis Function kernel. Linear . . . 50
7.4 SVM with Radial Basis Function kernel. Non-linear . . . 51
7.5 Wrapping up . . . 53

8 Multiclass classification. iris **55**
8.1 Peek at the dataset . . . 55
8.2 Levels of the class . . . 56
8.3 class distribution . . . 57
8.4 Visualize the dataset . . . 57
8.5 Evaluate algorithms . . . 61
8.6 Make predictions . . . 63

9 Regression. Boston **65**
9.1 Evaluation . . . 75

9.2 Feature selection . . . 77
9.3 Evaluate Algorithms: Box-Cox Transform . . . 80
9.4 Tune SVM . . . 82
9.5 Ensembling . . . 85
9.6 Finalize the model . . . 89

10 Breast Cancer 109
10.1 clean up . . . 112
10.2 Analyze the class variable . . . 114
10.3 Unimodal visualization . . . 115
10.4 Multimodal visualization . . . 117
10.5 Algorithms Evaluation . . . 119
10.6 Data transform . . . 182
10.7 Tuning SVM . . . 185
10.8 Tuning KNN . . . 187
10.9 Ensemble . . . 189
10.10 Finalize model . . . 190
10.11 Prepare the validation set . . . 191

11 SMS spam. Naive Bayes. Classification 193
11.1 Convert to Document Term Matrix . . . 195
11.2 split in training and test datasets . . . 195
11.3 plot wordcloud . . . 196
11.4 Limit Frequent words . . . 197
11.5 Improve model performance . . . 199

12 Classification Tree: Vehicle example 201
12.1 Load packages . . . 201
12.2 Prepare data . . . 202
12.3 Estimate the decision tree . . . 202
12.4 Assess model . . . 204
12.5 Make predictions . . . 205

13 Bike sharing demand 207
13.1 Step 1. Hypothesis Generation . . . 207
13.2 2. Understanding the Data Set . . . 208
13.3 3. Importing the dataset and Data Exploration . . . 208
13.4 4. Hypothesis Testing (using multivariate analysis) . . . 213
13.5 5. Feature Engineering . . . 223
13.6 6. Model Building . . . 234
13.7 End Notes . . . 238

14 Breast Cancer Wisconsin 241
14.1 Read and process the data . . . 241
14.2 Principal Component Analysis (PCA) . . . 242
14.3 Feature importance . . . 247
14.4 Feature Selection . . . 249
14.5 Model comparison . . . 253
14.6 Create comparison tables . . . 257
14.7 Notes . . . 259

15 Can we predict flu outcome with Machine Learning in R? 261
15.1 The data . . . 262
15.2 Features . . . 267
15.3 Test, train and validation data sets . . . 274

15.4 Comparing Machine Learning algorithms . . . 279
15.5 Comparing accuracy of models . . . 289
15.6 Predicting unknown outcomes . . . 292
15.7 Conclusions . . . 298

16 Dealing with unbalanced data 301
16.1 Read and process the data . . . 301

17 Orange Juice 309
17.1 Introduction . . . 309
17.2 2. Initial Setup – load the package and dataset . . . 309
17.3 3. Data Preparation and Preprocessing . . . 310
17.4 4. How to visualize the importance of variables using featurePlot() . . . 315
17.5 5. How to do feature selection using recursive feature elimination (RFE)? . . . 317
17.6 6. Training and Tuning the model . . . 319
17.7 7. How to do hyperparameter tuning to optimize the model for better performance? . . . 324
17.8 8. How to evaluate performance of multiple machine learning algorithms? . . . 326
17.9 9. Ensembling the predictions . . . 405
17.1010. Conclusion . . . 408

18 Feature selection, Variable Importance 411
18.1 Introduction . . . 411
18.2 1. Boruta . . . 412
18.3 2. Variable Importance from Machine Learning Algorithms . . . 414
18.4 3. Lasso Regression . . . 417
18.5 4. Step wise Forward and Backward Selection . . . 419
18.6 5. Relative Importance from Linear Regression . . . 420
18.7 6. Recursive Feature Elimination (RFE) . . . 422
18.8 7. Genetic Algorithm . . . 423
18.9 8. Simulated Annealing . . . 424
18.109. Information Value and Weights of Evidence . . . 426
18.1110. DALEX Package . . . 428
18.12Conclusion . . . 429

19 Linear Regression. World Happiness 431
19.1 Introduction . . . 431
19.2 A quick exploration of the data . . . 431
19.3 Linear regression with R . . . 434
19.4 Regression summary . . . 434
19.5 Regression analysis . . . 436
19.6 Analysis of colinearity . . . 438
19.7 What drives happiness . . . 438

20 Logistic Regression. Diabetes 441
20.1 Introduction . . . 441
20.2 Exploring the data . . . 441
20.3 Logistic regression with R . . . 443
20.4 A second model . . . 445
20.5 Classification rate and confusion matrix . . . 446
20.6 Plots and decision boundaries . . . 447

21 Linear Regression on Advertising 451

22 Data Visualization. 6 Work with models 457
22.1 6.1 Show several fits at once, with a legend . . . 458

22.2 6.2 Look inside model objects . . . 459
22.3 6.3 Get model-based graphics right . . . 461
22.4 6.4 Generate predictions to graph . . . 462
22.5 6.5 Tidy model objects with `broom` . . . 464
22.6 6.6 Grouped analysis and list-columns . . . 467
22.7 6.7 Plot marginal effects . . . 470
22.8 6.8 Plots from complex surveys . . . 472
22.9 6.9 Where to go next . . . 475

23 Lab 3A: Regression 479
23.1 1. Explore the Data . . . 479
23.2 2. Create Training and Test Sets . . . 481
23.3 3. Predict with Simple Linear Regression . . . 481
23.4 4. Predict with Multiple Regression . . . 483
23.5 5. Predict with Neural Network Regression . . . 485
23.6 6. Evaluate Regression Models . . . 488

24 Regression 3b 489

25 2. Split the Data into Test and Training Sets 493

26 3. Predict with Simple Linear Regression 495

27 4. Predict with Multiple Linear Regression 499

28 6. Predict with Neural Network Regression 501

29 7. Evaluate the Regression Models 505

30 Employee attrition 507
30.1 EMPLOYEE ATTRITION: A MAJOR PROBLEM . . . 507
30.2 EMPLOYEE ATTRITION: MACHINE LEARNING ANALYSIS . . . 508
30.3 MODELING EMPLOYEE ATTRITION . . . 511
30.4 MODEL . . . 512
30.5 PREDICT . . . 513
30.6 PERFORMANCE . . . 513
30.7 LIME . . . 515
30.8 FEATURE IMPORTANCE VISUALIZATION . . . 518
30.9 WHAT FEATURES ARE LINKED TO EMPLOYEE ATTRITION? . . . 520
30.10TRAINING . . . 520
30.11OVERTIME . . . 522
30.12JOB ROLE . . . 524
30.13CONCLUSIONS . . . 528

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

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

```r
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...
```

```r
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

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

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

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

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

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

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

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

2.2 Compare models

```r
# 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
```

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

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

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

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

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

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

```r
# 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)
```

```r
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

```r
# 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

```r
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

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

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

4.2 Explore data

```r
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>
```

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

```r
# 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.
```

```r
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

```r
# 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

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

4.5 Use the saved model

```r
# 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

```r
# 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  4  3  0  0  0
#>    2  9 19  3  4  0  5
#>    3  0  1  1  0  0  0
#>    5  0  0  0  0  0  0
#>    6  0  0  0  0  1  0
#>    7  0  0  0  0  0  4
```

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

```
#>     true
#> pred  1  2  3  5  6  7
#>    1 14  0  0  0  0  0
#>    2 11 16  5  0  0  0
#>    3  1  2  2  0  0  0
#>    5  0  4  0  3  0  1
#>    6  0  1  0  0  1  1
#>    7  0  1  0  1  0  7
```

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  4  3  0  0  0
#>          2  9 19  3  4  0  5
#>          3  0  1  1  0  0  0
#>          5  0  0  0  0  0  0
#>          6  0  0  0  0  1  0
#>          7  0  0  0  0  0  4
#>
#> Overall Statistics
#>
#>                Accuracy : 0.5915
#>                  95% CI : (0.4684, 0.7068)
#>     No Information Rate : 0.3662
#>     P-Value [Acc > NIR] : 9.415e-05
#>
#>                   Kappa : 0.3955
#>
```

```
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
#> Sensitivity            0.6538   0.7917  0.14286  0.00000  1.00000  0.44444
#> Specificity            0.8444   0.5532  0.98438  1.00000  1.00000  1.00000
#> Pos Pred Value         0.7083   0.4750  0.50000      NaN  1.00000  1.00000
#> Neg Pred Value         0.8085   0.8387  0.91304  0.94366  1.00000  0.92537
#> Prevalence             0.3662   0.3380  0.09859  0.05634  0.01408  0.12676
#> Detection Rate         0.2394   0.2676  0.01408  0.00000  0.01408  0.05634
#> Detection Prevalence   0.3380   0.5634  0.02817  0.00000  0.01408  0.05634
#> Balanced Accuracy      0.7491   0.6724  0.56362  0.50000  1.00000  0.72222
```

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

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  1  2  3  5  6  7
#>          1 14  0  0  0  0  0
#>          2 11 16  5  0  0  0
#>          3  1  2  2  0  0  0
#>          5  0  4  0  3  0  1
#>          6  0  1  0  0  1  1
#>          7  0  1  0  1  0  7
#>
#> Overall Statistics
#>
#>                Accuracy : 0.6056
#>                  95% CI : (0.4825, 0.7197)
#>     No Information Rate : 0.3662
#>     P-Value [Acc > NIR] : 3.552e-05
#>
#>                   Kappa : 0.471
#>
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
#> Sensitivity            0.5385   0.6667  0.28571  0.75000  1.00000  0.77778
#> Specificity            1.0000   0.6596  0.95312  0.92537  0.97143  0.96774
#> Pos Pred Value         1.0000   0.5000  0.40000  0.37500  0.33333  0.77778
#> Neg Pred Value         0.7895   0.7949  0.92424  0.98413  1.00000  0.96774
#> Prevalence             0.3662   0.3380  0.09859  0.05634  0.01408  0.12676
#> Detection Rate         0.1972   0.2254  0.02817  0.04225  0.01408  0.09859
#> Detection Prevalence   0.1972   0.4507  0.07042  0.11268  0.04225  0.12676
#> Balanced Accuracy      0.7692   0.6631  0.61942  0.83769  0.98571  0.87276
```

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.

```r
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   0.5471698 0.6545455 0.6956522 0.6907697 0.7254902 0.7777778    0
#> rpart 0.4629630 0.5468750 0.5740741 0.5861252 0.6078431 0.7450980    0
#>
#> Kappa
#>            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> svm   0.3677932 0.5100797 0.5623100 0.5576764 0.6008944 0.6666667    0
#> rpart 0.1516793 0.3604264 0.3929204 0.4055741 0.4590970 0.6283632    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,] 12.09909
```

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

```
##          [,1]
## [1,] 20.33083
```

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)
```

```r
#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
```

```r
dim(testset)
```

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

7.2.2 Build the SVM model

```r
#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
```

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

7.2.6 Confusion matrix and Accuracy

```r
# 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
```

```r
# 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

```r
#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
#> 1          5.1         3.5          1.4         0.2  setosa
#> 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
```

```
library(dplyr)

glimpse(dataset)
```

```
#> Observations: 120
#> Variables: 5
#> $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 5.0, 4.4, 4.9, 5.4,...
#> $ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 2.9, 3.1, 3.7,...
#> $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.5, 1.4, 1.5, 1.5,...
#> $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.2, 0.2, 0.1, 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
#>  Petal.Length       0      120 120 3.78 1.76 1.2 1.6  4.4 5.1   6.7
#>   Petal.Width       0      120 120 1.21 0.77 0.1 0.3  1.3 1.8   2.5
#>  Sepal.Length       0      120 120 5.88 0.82 4.4 5.18 5.8 6.43  7.9
#>   Sepal.Width       0      120 120 3.07 0.45 2   2.8  3   3.4   4.4
#>      hist
#>
#>
#>
#>
```

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)
```

```r
# 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  0.8333333  1.0000      1 0.9750000       1    1    0
#> cart 0.8333333  1.0000      1 0.9833333       1    1    0
#> knn  0.9166667  1.0000      1 0.9916667       1    1    0
#> svm  0.9166667  0.9375      1 0.9750000       1    1    0
#> rf   0.8333333  1.0000      1 0.9833333       1    1    0
#>
#> Kappa
#>       Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
#> lda  0.750 1.00000      1 0.9625       1    1    0
#> cart 0.750 1.00000      1 0.9750       1    1    0
#> knn  0.875 1.00000      1 0.9875       1    1    0
#> svm  0.875 0.90625      1 0.9625       1    1    0
#> rf   0.750 1.00000      1 0.9750       1    1    0
```

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

```r
# 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
#>   0.975     0.9625
```

8.6 Make predictions

```r
# 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          9         2
#>   virginica       0          1         8
#>
#> Overall Statistics
#>
#>                Accuracy : 0.9
#>                  95% CI : (0.7347, 0.9789)
#>     No Information Rate : 0.3333
#>     P-Value [Acc > NIR] : 1.665e-10
#>
#>                   Kappa : 0.85
#>
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: setosa Class: versicolor Class: virginica
#> Sensitivity                 1.0000            0.9000           0.8000
#> Specificity                 1.0000            0.9000           0.9500
#> Pos Pred Value              1.0000            0.8182           0.8889
#> Neg Pred Value              1.0000            0.9474           0.9048
#> Prevalence                  0.3333            0.3333           0.3333
#> Detection Rate              0.3333            0.3000           0.2667
#> Detection Prevalence        0.3333            0.3667           0.3000
#> Balanced Accuracy           1.0000            0.9000           0.8750
```

Chapter 9

Regression. Boston

Comparison of various algorithms.

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

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

```r
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, ...
```

```r
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
```

```r
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
```

```r
library(dplyr)

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

```r
# 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])
}
```

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

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

```r
# 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)
```

```r
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)
```

```r
# 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"
```

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

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

```r
# 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
```

```r
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
```

```r
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.

```r
# 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.
```

```r
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.
```

```r
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

```r
# 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]  Mon May 13 20:57:46 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.

```r
# 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

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

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

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

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

```r
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...
```

```r
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
```

```r
library(skimr)
```

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

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

```r
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
```

```r
# 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

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

```r
# 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

```r
# 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

```r
# 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

```r
# 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.

```r
# 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)
```

```r
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
```

```r
# 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
```

```r
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.
```

```r
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.
```

```r
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.

```r
# 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.

```r
# 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])
```

```r
# 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
```

```r
# 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
```

```r
# 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
```

```r
# convert to lowercase
sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))
```

```r
as.character(sms_corpus[[1]])
```

```
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
```

```r
# 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
```

```r
# remove numbers
sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)
```

```r
# what transformations are available
getTransformations()
```

```
#> [1] "removeNumbers"     "removePunctuation" "removeWords"
#> [4] "stemDocument"      "stripWhitespace"
```

```r
# remove stop words
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords())
```

```r
# remove punctuation
sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)
```

```r
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
```

```r
prop.table(table(sms_train_labels))
```

```
#> sms_train_labels
#>       ham      spam
#> 0.8647158 0.1352842
```

```r
prop.table(table(sms_test_labels))
```

```
#> sms_test_labels
#>       ham      spam
#> 0.8697842 0.1302158
```

```r
# 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

```r
library(wordcloud)
wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE)
```

```r
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

```r
sms_classifier2 <- naiveBayes(sms_train, sms_train_labels,
                              laplace = 1)
```

```r
sms_test_pred2 <- predict(sms_classifier2, sms_test)
```

```r
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

```r
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 ...
```

```r
summary(Vehicle[1])
```

```
#>       Comp
#>  Min.   : 73.00
#>  1st Qu.: 87.00
#>  Median : 93.00
```

```
#>  Mean   : 93.68
#>  3rd Qu.:100.00
#>  Max.   :119.00
```

```r
summary(Vehicle[2])
```

```
#>       Circ
#>  Min.   :33.00
#>  1st Qu.:40.00
#>  Median :44.00
#>  Mean   :44.86
#>  3rd Qu.:49.00
#>  Max.   :59.00
```

```r
attributes(Vehicle$Class)
```

```
#> $levels
#> [1] "bus"  "opel" "saab" "van"
#>
#> $class
#> [1] "factor"
```

12.2 Prepare data

```r
set.seed(107)
N = nrow(Vehicle)
train <- sample(1:N, 500, FALSE)
```

```r
# training and test sets
trainset <- Vehicle[train,]
testset  <- Vehicle[-train,]
```

12.3 Estimate the decision tree

```r
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 ) *
```

```r
# 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 "*".

```r
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%.

```r
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

```r
#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/

13.1 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

13.2 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).

13.2.0.1 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
```

13.2.0.2 Dependent variables

```
registered: number of registered user
casual:     number of non-registered user
count:      number of total rentals (registered + casual)
```

13.3 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

```r
# 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"))
```

```r
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, ...
```

```r
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).

```r
# add variables to test dataset before merging
test$registered=0
test$casual=0
test$count=0

data = rbind(train,test)
```

3. Variable Type Identification

```r
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.

13.3.0.1 histograms

```
# histograms each attribute
par(mfrow=c(2,4))
for(i in 2:9) {
    hist(data[,i], main = names(data)[i])
}
```

13.3.0.2 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])
}
```

13.3.0.3 boxplots

```
# boxplots for each attribute
par(mfrow=c(2,4))
for(i in 2:9) {
    boxplot(data[,i], main=names(data)[i])
}
```

13.3.1 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
```

13.3.1.1 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).

13.4 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:

13.4.1 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.

```r
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
```

```r
class(data$datetime)
```

```
#> [1] "factor"
```

```r
# show hour and day from the variable datetime
head(substr(data$datetime, 12, 13))  # hour
```

```
#> [1] "00" "01" "02" "03" "04" "05"
```

```r
head(substr(data$datetime, 9, 10))   # day
```

```
#> [1] "01" "01" "01" "01" "01" "01"
```

```r
# 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,]
```

13.4.1.1 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
```

13.4.1.2 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.

13.4.1.3 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)")
```

13.4.1.4 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,]
```

13.4.1.5 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.

13.4.2 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.

13.4.2.1 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.

13.4.3 Temperature

Would high or low temperature encourage or disencourage bike riding?

13.4.3.1 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.

13.4.4 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.

13.4.5 Activity by year

13.4.5.1 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,]
```

13.4.5.2 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.

13.4.5.3 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.

13.4.5.4 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.

13.5 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)
```

13.5.1 Build hour bins

```
# for registered users
d = rpart(registered ~ hour, data = train)
fancyRpartPlot(d)
```

```
# for casual users
d = rpart(casual ~ hour, data = train)
fancyRpartPlot(d)
```

Rattle 2019−May−13 21:04:44 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
```

13.5.2 Temperature bins

```
# partition the data by temperature, registered users
f = rpart(registered ~ temp, data=train)
fancyRpartPlot(f)
```

```
# partition the data by temperature,, casual users
f=rpart(casual ~ temp, data=train)
fancyRpartPlot(f)
```

13.5.2.1 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
```

13.5.3 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
```

13.5.4 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))
```

13.5.5 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)
```

13.5.6 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
```

13.5.7 Weekend variable

Created a separate variable for weekend (0/1)

```
data$weekend = 0
data$weekend[data$day=="Sunday" | data$day=="Saturday" ] = 1
```

13.6 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 ...
```

13.6.1 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,]
```

13.6.2 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
```

13.6.2.1 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")
```

13.6.3 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
```

13.6.4 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.

13.7 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

```r
# 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
```

```r
sum(old$count)
```

```
#> Error in eval(expr, envir, enclos): object 'old' not found
```

Chapter 14

Breast Cancer Wisconsin

Source: https://shiring.github.io/machine_learning/2017/01/15/rfe_ga_post

14.1 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"
```

14.1.1 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 ...
```

14.2 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.

14.2.0.1 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())
```

14.2.0.2 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)
```

14.2.1 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)
```

14.3 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)
```

```r
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)
}
```

```r
# train the model
set.seed(27)
imp_1 <- train(classes ~ ., data = bc_data, method = "rf",
               preProcess = c("scale", "center"),
               trControl = control)
```

```r
p1 <- feature_imp(imp_1, title = "Breast cancer dataset 1")
p1
```

14.4 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, ]
```

14.4.1 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 0.579 so flagging column 3
Compare row 7  and column  6 with corr  0.705
  Means:  0.6 vs 0.544 so flagging column 7
Compare row 6  and column  4 with corr  0.72
  Means:  0.58 vs 0.526 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"
```

```r
# 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

14.4.2 Recursive Feature Elimination (RFE)

```r
# 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"                     "single_epithelial_cell_size"
[9] "normal_nucleoli"
```

```r
# 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"
```

```r
# confirm features removed by RFE
outersect(names(bc_data_rfe), names(bc_data_train))
```

```
character(0)
```

No features removed with RFE

14.4.3 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] "uniformity_of_cell_shape"    "marginal_adhesion"
[5] "single_epithelial_cell_size" "bare_nuclei"
[7] "normal_nucleoli"             "mitosis"
```

```r
# 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"     "uniformity_of_cell_shape"
[5] "marginal_adhesion"           "single_epithelial_cell_size"
[7] "bare_nuclei"                 "normal_nucleoli"
[9] "mitosis"
```

```r
# features removed GA
outersect(names(bc_data_ga), names(bc_data_train))
```

```
[1] "bland_chromatin"
```

Two features removed with GA.

14.5 Model comparison

14.5.1 Using all features

```r
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))
```

```r
# 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
```

14.5.2 Compare selection methods

```r
# 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)
```

```r
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   0   1   1  0
111   5   1   1  1
attr(,"intersections")
attr(,"intersections")$`cor:rfe:ga`
[1] "clump_thickness"             "marginal_adhesion"
[3] "single_epithelial_cell_size" "normal_nucleoli"
[5] "mitosis"

attr(,"intersections")$rfe
[1] "bland_chromatin"

attr(,"intersections")$`rfe:ga`
```

```
[1] "uniformity_of_cell_size"  "uniformity_of_cell_shape"
[3] "bare_nuclei"

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.

14.5.3 Correlation

```r
# 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 =
```

```r
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
```

14.5.4 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
```

14.5.5 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         3
  malignant      3        69

               Accuracy : 0.9713
                 95% CI : (0.9386, 0.9894)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9364

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9781
            Specificity : 0.9583
         Pos Pred Value : 0.9781
         Neg Pred Value : 0.9583
             Prevalence : 0.6555
         Detection Rate : 0.6411
   Detection Prevalence : 0.6555
      Balanced Accuracy : 0.9682

       'Positive' Class : benign
```

14.6 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"
  )
```

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.

14.7 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`.

Chapter 15

Can we predict flu outcome with Machine Learning in R?

Among the many nice R packages containing data collections is the outbreaks package. It contains datsets on epidemics and among them is data from the 2013 outbreak of influenza A H7N9 in China as analysed by Kucharski et al. (2014):

> A. Kucharski, H. Mills, A. Pinsent, C. Fraser, M. Van Kerkhove, C. A. Donnelly, and S. Riley. 2014. Distinguishing between reservoir exposure and human-to-human transmission for emerging pathogens using case onset data. PLOS Currents Outbreaks. Mar 7, edition 1. doi: 10.1371/currents.outbreaks.e1473d9bfc99d080ca242139a06c455f.
>
> A. Kucharski, H. Mills, A. Pinsent, C. Fraser, M. Van Kerkhove, C. A. Donnelly, and S. Riley. 2014. Data from: Distinguishing between reservoir exposure and human-to-human transmission for emerging pathogens using case onset data. Dryad Digital Repository. http://dx.doi.org/10.5061/dryad.2g43n.

I will be using their data as an example to show how to use Machine Learning algorithms for predicting disease outcome.

To do so, I selected and extracted features from the raw data, including age, days between onset and outcome, gender, whether the patients were hospitalised, etc. Missing values were imputed and different model algorithms were used to predict outcome (death or recovery). The prediction accuracy, sensitivity and specificity. The thus prepared dataset was devided into training and testing subsets. The test subset contained all cases with an unknown outcome. Before I applied the models to the test data, I further split the training data into validation subsets.

The tested modeling algorithms were similarly successful at predicting the outcomes of the validation data. To decide on final classifications, I compared predictions from all models and defined the outcome "Death" or "Recovery" as a function of all models, whereas classifications with a low prediction probability were flagged as "uncertain". Accounting for this uncertainty led to a 100% correct classification of the validation test set.

The training cases with unknown outcome were then classified based on the same algorithms. From 57 unknown cases, 14 were classified as "Recovery", 10 as "Death" and 33 as uncertain.

In a Part 2, I am looking at how extreme gradient boosting performs on this dataset.

Disclaimer: I am not an expert in Machine Learning. Everything I know, I tought myself during the last months. So, if you see any mistakes or have tips and tricks for improvement, please don't hesitate to let me know! Thanks. :-)

15.1 The data

The dataset contains case ID, date of onset, date of hospitalisation, date of outcome, gender, age, province and of course the outcome: Death or Recovery. I can already see that there are a couple of missing values in the data, which I will deal with later.

```
options(width = 1000)

if (!require("outbreaks")) install.packages("outbreaks")
library(outbreaks)

fluH7N9_china_2013_backup <- fluH7N9_china_2013

fluH7N9_china_2013$age[which(fluH7N9_china_2013$age == "?")] <- NA
fluH7N9_china_2013$case_ID <- paste("case",
                                    fluH7N9_china_2013$case_id,
                                    sep = "_")
head(fluH7N9_china_2013)
```

```
#>   case_id date_of_onset date_of_hospitalisation date_of_outcome outcome gender age province case_ID
#> 1       1    2013-02-19                    <NA>      2013-03-04   Death      m  87 Shanghai  case_1
#> 2       2    2013-02-27              2013-03-03      2013-03-10   Death      m  27 Shanghai  case_2
#> 3       3    2013-03-09              2013-03-19      2013-04-09   Death      f  35    Anhui  case_3
#> 4       4    2013-03-19              2013-03-27            <NA>    <NA>      f  45  Jiangsu  case_4
#> 5       5    2013-03-19              2013-03-30      2013-05-15 Recover      f  48  Jiangsu  case_5
#> 6       6    2013-03-21              2013-03-28      2013-04-26   Death      f  32  Jiangsu  case_6
```

Before I start preparing the data for Machine Learning, I want to get an idea of the distribution of the data points and their different variables by plotting.

Most provinces have only a handful of cases, so I am combining them into the category "other" and keep only Jiangsu, Shanghai and Zhejian and separate provinces.

```
# make tidy data, clean up and simplify
library(tidyr)

# put different type of dates in one variable (Date)
fluH7N9_china_2013_gather <- fluH7N9_china_2013 %>%
  gather(Group, Date, date_of_onset:date_of_outcome)

# convert Group to factor
fluH7N9_china_2013_gather$Group <- factor(fluH7N9_china_2013_gather$Group,
                                          levels = c("date_of_onset",
                                                     "date_of_hospitalisation",
                                                     "date_of_outcome"))

# change the dates description with plyr::mapvalues
library(plyr)
from <- c("date_of_onset", "date_of_hospitalisation", "date_of_outcome")
to <- c("Date of onset", "Date of hospitalisation", "Date of outcome")
fluH7N9_china_2013_gather$Group <- mapvalues(fluH7N9_china_2013_gather$Group,
                                             from = from, to = to)

# change additional provinces to Other
from <- c("Anhui", "Beijing", "Fujian", "Guangdong", "Hebei", "Henan",
          "Hunan", "Jiangxi", "Shandong", "Taiwan")
```

```r
to <- rep("Other", 10)
fluH7N9_china_2013_gather$province <-
  mapvalues(fluH7N9_china_2013_gather$province,
            from = from, to = to)

# convert gender to factor
levels(fluH7N9_china_2013_gather$gender) <-
  c(levels(fluH7N9_china_2013_gather$gender), "unknown")

# replace NA in gender by "unknown
is_na <- is.na(fluH7N9_china_2013_gather$gender)
fluH7N9_china_2013_gather$gender[is_na] <- "unknown"

# convert province to factor
fluH7N9_china_2013_gather$province <- factor(fluH7N9_china_2013_gather$province,
                                             levels = c("Jiangsu",  "Shanghai",
                                                        "Zhejiang", "Other"))
```

```r
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 = 45, 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 = "lightgrey", color = "grey",
                                    size = 1),
    strip.text = element_text(face = "bold", size = 12, color = "black"),
    legend.position = "bottom",
    legend.justification = "top",
    legend.box = "horizontal",
    legend.box.background = element_rect(colour = "grey50"),
    legend.background = element_blank(),
    panel.border = element_rect(color = "grey", fill = NA, size = 0.5)
  )
}
```

```r
ggplot(data = fluH7N9_china_2013_gather,
       aes(x = Date, y = as.numeric(age), fill = outcome)) +
  stat_density2d(aes(alpha = ..level..), geom = "polygon") +
  geom_jitter(aes(color = outcome, shape = gender), size = 1.5) +
  geom_rug(aes(color = outcome)) +
  labs(
    fill = "Outcome",
    color = "Outcome",
    alpha = "Level",
    shape = "Gender",
    x = "Date in 2013",
    y = "Age",
    title = "2013 Influenza A H7N9 cases in China",
    subtitle = "Dataset from 'outbreaks' package (Kucharski et al. 2014)",
```

```
    caption = ""
  ) +
  facet_grid(Group ~ province) +
  my_theme() +
  scale_shape_manual(values = c(15, 16, 17)) +
  scale_color_brewer(palette="Set1", na.value = "grey50") +
  scale_fill_brewer(palette="Set1")
```

This plot shows the dates of onset, hospitalisation and outcome (if known) of each data point. Outcome is marked by color and age shown on the y-axis. Gender is marked by point shape.

The density distribution of date by age for the cases seems to indicate that older people died more frequently in the Jiangsu and Zhejiang province than in Shanghai and in other provinces.

When we look at the distribution of points along the time axis, it suggests that their might be a positive correlation between the likelihood of death and an early onset or early outcome.

I also want to know how many cases there are for each gender and province and compare the genders' age distribution.

```
# more tidy data. first remove age
fluH7N9_china_2013_gather_2 <- fluH7N9_china_2013_gather[, -4] %>%
  gather(group_2, value, gender:province)

# change descriptions
from <- c("m", "f", "unknown", "Other")
to <- c("Male", "Female", "Unknown gender", "Other province")
fluH7N9_china_2013_gather_2$value <- mapvalues(fluH7N9_china_2013_gather_2$value,
                                               from = from, to = to)
# convert to factor
```

```
fluH7N9_china_2013_gather_2$value <- factor(fluH7N9_china_2013_gather_2$value,
                                            levels = c("Female", "Male",
                                                       "Unknown gender",
                                                       "Jiangsu", "Shanghai",
                                                       "Zhejiang", "Other
                                                       province"))

p1 <- ggplot(data = fluH7N9_china_2013_gather_2, aes(x = value,
                                                      fill = outcome,
                                                      color = outcome)) +
  geom_bar(position = "dodge", alpha = 0.7, size = 1) +
  my_theme() +
  scale_fill_brewer(palette="Set1", na.value = "grey50") +
  scale_color_brewer(palette="Set1", na.value = "grey50") +
  labs(
    color = "",
    fill = "",
    x = "",
    y = "Count",
    title = "2013 Influenza A H7N9 cases in China",
    subtitle = "Gender and Province numbers of flu cases",
    caption = ""
  )

p2 <- ggplot(data = fluH7N9_china_2013_gather, aes(x = as.numeric(age),
                                                    fill = outcome,
                                                    color = outcome)) +
  geom_density(alpha = 0.3, size = 1) +
  geom_rug() +
  scale_color_brewer(palette="Set1", na.value = "grey50") +
  scale_fill_brewer(palette="Set1", na.value = "grey50") +
  my_theme() +
  labs(
    color = "",
    fill = "",
    x = "Age",
    y = "Density",
    title = "",
    subtitle = "Age distribution of flu cases",
    caption = ""
  )

library(gridExtra)
library(grid)

grid.arrange(p1, p2, ncol = 2)
```

In the dataset, there are more male than female cases and correspondingly, we see more deaths, recoveries and unknown outcomes in men than in women. This is potentially a problem later on for modeling because the inherent likelihoods for outcome are not directly comparable between the sexes.

Most unknown outcomes were recorded in Zhejiang. Similarly to gender, we don't have an equal distribution of data points across provinces either.

When we look at the age distribution it is obvious that people who died tended to be slightly older than those who recovered. The density curve of unknown outcomes is more similar to that of death than of recovery, suggesting that among these people there might have been more deaths than recoveries.

And lastly, I want to plot how many days passed between onset, hospitalisation and outcome for each case.

```
ggplot(data = fluH7N9_china_2013_gather, aes(x = Date, y = as.numeric(age),
                                             color = outcome)) +
  geom_point(aes(color = outcome, shape = gender), size = 1.5, alpha = 0.6) +
  geom_path(aes(group = case_ID)) +
  facet_wrap( ~ province, ncol = 2) +
  my_theme() +
  scale_shape_manual(values = c(15, 16, 17)) +
  scale_color_brewer(palette="Set1", na.value = "grey50") +
  scale_fill_brewer(palette="Set1") +
  labs(
    color = "Outcome",
    shape = "Gender",
    x = "Date in 2013",
    y = "Age",
    title = "2013 Influenza A H7N9 cases in China",
    subtitle = "Dataset from 'outbreaks' package (Kucharski et al. 2014)",
    caption = "\nTime from onset of flu to outcome."
  )
```

This plot shows that there are many missing values in the dates, so it is hard to draw a general conclusion.

15.2 Features

In Machine Learning-speak features are the variables used for model training. Using the right features dramatically influences the accuracy of the model.

Because we don't have many features, I am keeping age as it is, but I am also generating new features:

- from the date information I am calculating the days between onset and outcome and between onset and hospitalisation
- I am converting gender into numeric values with 1 for female and 0 for male

- similarly, I am converting provinces to binary classifiers (yes == 1, no == 0) for Shanghai, Zhejiang, Jiangsu and other provinces
- the same binary classification is given for whether a case was hospitalised, and whether they had an early onset or early outcome (earlier than the median date)

```
# convert gender, provinces to discrete and numeric values
library(dplyr)

dataset <- fluH7N9_china_2013 %>%
  mutate(hospital = as.factor(ifelse(is.na(date_of_hospitalisation), 0, 1)),
         gender_f = as.factor(ifelse(gender == "f", 1, 0)),
         province_Jiangsu = as.factor(ifelse(province == "Jiangsu", 1, 0)),
         province_Shanghai = as.factor(ifelse(province == "Shanghai", 1, 0)),
         province_Zhejiang = as.factor(ifelse(province == "Zhejiang", 1, 0)),
         province_other = as.factor(ifelse(province == "Zhejiang" |
                                           province == "Jiangsu" |
                                           province == "Shanghai", 0, 1)),
         days_onset_to_outcome = as.numeric(as.character(gsub(" days", "",
                as.Date(as.character(date_of_outcome), format = "%Y-%m-%d") -
                as.Date(as.character(date_of_onset), format = "%Y-%m-%d")))),

         days_onset_to_hospital = as.numeric(as.character(gsub(" days", "",
            as.Date(as.character(date_of_hospitalisation), format = "%Y-%m-%d") -
            as.Date(as.character(date_of_onset), format = "%Y-%m-%d")))),
         age = as.numeric(as.character(age)),
         early_onset = as.factor(ifelse(date_of_onset <
                        summary(date_of_onset)[[3]], 1, 0)),
         early_outcome = as.factor(ifelse(date_of_outcome <
                        summary(date_of_outcome)[[3]], 1, 0))) %>%
  subset(select = -c(2:4, 6, 8))
  # print

rownames(dataset) <- dataset$case_ID
dataset <- dataset[, -c(1,4)]
head(dataset)
```

```
#>        outcome age hospital gender_f province_Jiangsu province_Shanghai province_Zhejiang province_o
#> case_1   Death  87        0        0                0                 1                 0
#> case_2   Death  27        1        0                0                 1                 0
#> case_3   Death  35        1        1                0                 0                 0
#> case_4    <NA>  45        1        1                1                 0                 0
#> case_5 Recover  48        1        1                1                 0                 0
#> case_6   Death  32        1        1                1                 0                 0
```

```
summary(dataset$outcome)
```

```
#>   Death Recover    NA's
#>      32      47      57
```

15.2.1 Imputing missing values

https://www.r-bloggers.com/imputing-missing-data-with-r-mice-package/

When looking at the dataset I created for modeling, it is obvious that we have quite a few missing values.

The missing values from the outcome column are what I want to predict but for the rest I would either have to remove the entire row from the data or impute the missing information. I decided to try the latter with the `mice` package.

```
library(mice)

dataset_impute  <- mice(dataset[, -1],  print = FALSE)  # remove outcome
impute_complete <- mice::complete(dataset_impute, 1) # return 1st imputed dataset
rownames(impute_complete) <- row.names(dataset)
impute_complete
```

```
#>          age hospital gender_f province_Jiangsu province_Shanghai province_Zhejiang province_other da
#> case_1    87        0        0                0                 1                 0              0
#> case_2    27        1        0                0                 1                 0              0
#> case_3    35        1        1                0                 0                 0              1
#> case_4    45        1        1                1                 0                 0              0
#> case_5    48        1        1                1                 0                 0              0
#> case_6    32        1        1                1                 0                 0              0
#> case_7    83        1        0                1                 0                 0              0
#> case_8    38        1        0                0                 0                 1              0
#> case_9    67        1        0                0                 0                 1              0
#> case_10   48        1        0                0                 1                 0              0
#> case_11   64        1        0                0                 0                 1              0
#> case_12   52        0        1                0                 1                 0              0
#> case_13   67        1        1                0                 1                 0              0
#> case_14    4        0        0                0                 1                 0              0
#> case_15   61        0        1                1                 0                 0              0
#> case_16   79        0        0                1                 0                 0              0
#> case_17   74        1        0                0                 1                 0              0
#> case_18   66        1        0                0                 1                 0              0
#> case_19   59        1        0                0                 1                 0              0
#> case_20   55        1        0                0                 0                 0              1
#> case_21   67        1        0                0                 1                 0              0
#> case_22   85        1        0                1                 0                 0              0
#> case_23   25        1        1                1                 0                 0              0
#> case_24   64        0        0                0                 1                 0              0
#> case_25   62        1        0                0                 1                 0              0
#> case_26   77        1        0                0                 1                 0              0
#> case_27   51        1        1                0                 0                 1              0
#> case_28   79        0        0                0                 0                 1              0
#> case_29   76        1        1                0                 1                 0              0
#> case_30   81        0        1                0                 1                 0              0
#> case_31   70        0        0                1                 0                 0              0
#> case_32   74        0        0                1                 0                 0              0
#> case_33   65        0        0                0                 0                 1              0
#> case_34   74        1        0                0                 1                 0              0
#> case_35   83        1        1                0                 1                 0              0
#> case_36   68        0        0                0                 1                 0              0
#> case_37   31        0        0                1                 0                 0              0
#> case_38   56        0        0                1                 0                 0              0
#> case_39   66        1        0                0                 0                 1              0
#> case_40   74        1        0                0                 0                 1              0
#> case_41   54        1        1                0                 0                 1              0
#> case_42   53        1        0                0                 1                 0              0
#> case_43   86        1        0                0                 1                 0              0
```

```
#> case_44    7        1        1                0                0                0              1
#> case_45   56        1        0                0                1                0              0
#> case_46   77        0        1                1                0                0              0
#> case_47   72        0        0                1                0                0              0
#> case_48   65        1        0                0                0                1              0
#> case_49   38        1        0                0                0                1              0
#> case_50   34        1        0                0                0                0              1
#> case_51   65        1        0                0                0                0              1
#> case_52   64        0        1                0                0                1              0
#> case_53   62        0        1                0                0                1              0
#> case_54   75        0        0                0                0                1              0
#> case_55   79        0        0                0                0                1              0
#> case_56   73        1        0                0                1                0              0
#> case_57   54        1        0                0                1                0              0
#> case_58   78        1        0                0                1                0              0
#> case_59   50        0        0                1                0                0              0
#> case_60   26        0        0                1                0                0              0
#> case_61   60        0        0                1                0                0              0
#> case_62   68        0        1                0                0                1              0
#> case_63   60        0        0                0                0                0              1
#> case_64   56        0        0                1                0                0              0
#> case_65   21        0        1                1                0                0              0
#> case_66   72        0        0                1                0                0              0
#> case_67   56        0        0                0                0                1              0
#> case_68   57        0        0                0                0                1              0
#> case_69   62        0        0                0                0                1              0
#> case_70   58        0        1                0                0                1              0
#> case_71   72        0        1                0                0                1              0
#> case_72   47        1        0                0                1                0              0
#> case_73   69        0        0                0                1                0              0
#> case_74   83        0        0                0                1                0              0
#> case_75    2        0        0                0                1                0              0
#> case_76   55        0        0                0                1                0              0
#> case_77    2        0        0                0                1                0              0
#> case_78   89        1        0                0                1                0              0
#> case_79   37        0        1                0                0                1              0
#> case_80   74        0        0                0                0                1              0
#> case_81   86        0        0                0                0                1              0
#> case_82   41        0        0                0                0                1              0
#> case_83   38        0        0                0                0                0              1
#> case_84   26        0        1                1                0                0              0
#> case_85   80        1        1                0                1                0              0
#> case_86   54        0        1                0                0                1              0
#> case_87   69        0        0                0                0                1              0
#> case_88    4        0        0                0                0                0              1
#> case_89   54        1        0                1                0                0              0
#> case_90   43        1        0                0                0                1              0
#> case_91   48        1        0                0                0                1              0
#> case_92   66        1        1                0                0                1              0
#> case_93   56        0        0                0                0                1              0
#> case_94   35        0        1                0                0                1              0
#> case_95   37        0        0                0                0                1              0
#> case_96   43        0        0                1                0                0              0
#> case_97   75        1        1                0                1                0              0
```

```
#> case_98   76        0        0                0                0                1              0
#> case_99   68        0        1                0                0                1              0
#> case_100  58        0        0                0                0                1              0
#> case_101  79        0        1                0                0                1              0
#> case_102  81        0        0                0                0                1              0
#> case_103  68        1        0                1                0                0              0
#> case_104  54        0        1                0                0                1              0
#> case_105  32        0        0                0                0                1              0
#> case_106  36        1        0                0                0                0              1
#> case_107  91        0        0                0                0                0              1
#> case_108  84        0        0                0                0                1              0
#> case_109  62        0        0                0                0                1              0
#> case_110  53        1        0                0                0                0              1
#> case_111  56        0        0                0                0                0              1
#> case_112  69        0        0                0                0                0              1
#> case_113  60        0        1                0                0                1              0
#> case_114  50        0        1                0                0                1              0
#> case_115  38        0        0                0                0                1              0
#> case_116  65        1        0                0                0                0              1
#> case_117  76        0        1                0                0                0              1
#> case_118  49        0        0                1                0                0              0
#> case_119  36        0        0                1                0                0              0
#> case_120  60        0        0                1                0                0              0
#> case_121  64        1        1                0                0                0              1
#> case_122  38        0        0                0                0                1              0
#> case_123  54        1        0                0                0                0              1
#> case_124  80        0        0                0                0                0              1
#> case_125  31        0        1                0                0                0              1
#> case_126  80        1        0                0                0                0              1
#> case_127   4        0        0                0                0                0              1
#> case_128  58        1        0                0                0                0              1
#> case_129  69        1        0                0                0                0              1
#> case_130  69        1        0                0                0                0              1
#> case_131   9        1        0                0                0                0              1
#> case_132  79        1        1                0                0                0              1
#> case_133   6        1        0                0                0                0              1
#> case_134  15        1        0                1                0                0              0
#> case_135  61        1        1                0                0                0              1
#> case_136  51        1        1                0                0                0              1
```

```r
dataset_complete <- merge(dataset[, 1, drop = FALSE],
                          # mice::complete(dataset_impute, 1),
                          impute_complete,
                          by = "row.names", all = TRUE)
rownames(dataset_complete) <- dataset_complete$Row.names
dataset_complete <- dataset_complete[, -1]
dataset_complete
```

```
#>          outcome age hospital gender_f province_Jiangsu province_Shanghai province_Zhejiang province_
#> case_1     Death  87        0        0                0                1                0
#> case_10    Death  48        1        0                0                1                0
#> case_100    <NA>  58        0        0                0                0                1
#> case_101    <NA>  79        0        1                0                0                1
#> case_102    <NA>  81        0        0                0                0                1
#> case_103    <NA>  68        1        0                1                0                0
```

```
#> case_104    <NA>  54        0        1                0                0                1
#> case_105    <NA>  32        0        0                0                0                1
#> case_106 Recover  36        1        0                0                0                0
#> case_107   Death  91        0        0                0                0                0
#> case_108    <NA>  84        0        0                0                0                1
#> case_109    <NA>  62        0        0                0                0                1
#> case_11    Death  64        1        0                0                0                1
#> case_110    <NA>  53        1        0                0                0                0
#> case_111   Death  56        0        0                0                0                0
#> case_112    <NA>  69        0        0                0                0                0
#> case_113    <NA>  60        0        1                0                0                1
#> case_114    <NA>  50        0        1                0                0                1
#> case_115    <NA>  38        0        0                0                0                1
#> case_116 Recover  65        1        0                0                0                0
#> case_117 Recover  76        0        1                0                0                0
#> case_118    <NA>  49        0        0                1                0                0
#> case_119 Recover  36        0        0                1                0                0
#> case_12    Death  52        0        1                0                1                0
#> case_120    <NA>  60        0        0                1                0                0
#> case_121   Death  64        1        1                0                0                0
#> case_122    <NA>  38        0        0                0                0                1
#> case_123   Death  54        1        0                0                0                0
#> case_124 Recover  80        0        0                0                0                0
#> case_125 Recover  31        0        1                0                0                0
#> case_126    <NA>  80        1        0                0                0                0
#> case_127 Recover   4        0        0                0                0                0
#> case_128 Recover  58        1        0                0                0                0
#> case_129 Recover  69        1        0                0                0                0
#> case_13    Death  67        1        1                0                1                0
#> case_130    <NA>  69        1        0                0                0                0
#> case_131 Recover   9        1        0                0                0                0
#> case_132    <NA>  79        1        1                0                0                0
#> case_133 Recover   6        1        0                0                0                0
#> case_134 Recover  15        1        0                1                0                0
#> case_135   Death  61        1        1                0                0                0
#> case_136    <NA>  51        1        1                0                0                0
#> case_14  Recover   4        0        0                0                1                0
#> case_15     <NA>  61        0        1                1                0                0
#> case_16     <NA>  79        0        0                1                0                0
#> case_17    Death  74        1        0                0                1                0
#> case_18  Recover  66        1        0                0                1                0
#> case_19    Death  59        1        0                0                1                0
#> case_2     Death  27        1        0                0                1                0
#> case_20  Recover  55        1        0                0                0                0
#> case_21  Recover  67        1        0                0                1                0
#> case_22     <NA>  85        1        0                1                0                0
#> case_23  Recover  25        1        1                1                0                0
#> case_24    Death  64        0        0                0                1                0
#> case_25  Recover  62        1        0                0                1                0
#> case_26    Death  77        1        0                0                1                0
#> case_27  Recover  51        1        1                0                0                1
#> case_28     <NA>  79        0        0                0                0                1
#> case_29  Recover  76        1        1                0                1                0
#> case_3     Death  35        1        1                0                0                0
```

```
#> case_30  Recover  81        0        1                0                1                0
#> case_31     <NA>  70        0        0                1                0                0
#> case_32     <NA>  74        0        0                1                0                0
#> case_33  Recover  65        0        0                0                0                1
#> case_34    Death  74        1        0                0                1                0
#> case_35    Death  83        1        1                0                1                0
#> case_36  Recover  68        0        0                0                1                0
#> case_37  Recover  31        0        0                1                0                0
#> case_38     <NA>  56        0        0                1                0                0
#> case_39     <NA>  66        1        0                0                0                1
#> case_4      <NA>  45        1        1                1                0                0
#> case_40     <NA>  74        1        0                0                0                1
#> case_41     <NA>  54        1        1                0                0                1
#> case_42     <NA>  53        1        0                0                1                0
#> case_43    Death  86        1        0                0                1                0
#> case_44  Recover   7        1        1                0                0                0
#> case_45    Death  56        1        0                0                1                0
#> case_46    Death  77        0        1                1                0                0
#> case_47     <NA>  72        0        0                1                0                0
#> case_48     <NA>  65        1        0                0                0                1
#> case_49  Recover  38        1        0                0                0                1
#> case_5   Recover  48        1        1                1                0                0
#> case_50  Recover  34        1        0                0                0                0
#> case_51  Recover  65        1        0                0                0                0
#> case_52     <NA>  64        0        1                0                0                1
#> case_53    Death  62        0        1                0                0                1
#> case_54     <NA>  75        0        0                0                0                1
#> case_55  Recover  79        0        0                0                0                1
#> case_56     <NA>  73        1        0                0                1                0
#> case_57  Recover  54        1        0                0                1                0
#> case_58  Recover  78        1        0                0                1                0
#> case_59  Recover  50        0        0                1                0                0
#> case_6     Death  32        1        1                1                0                0
#> case_60  Recover  26        0        0                1                0                0
#> case_61    Death  60        0        0                1                0                0
#> case_62     <NA>  68        0        1                0                0                1
#> case_63     <NA>  60        0        0                0                0                0
#> case_64  Recover  56        0        0                1                0                0
#> case_65  Recover  21        0        1                1                0                0
#> case_66     <NA>  72        0        0                1                0                0
#> case_67     <NA>  56        0        0                0                0                1
#> case_68     <NA>  57        0        0                0                0                1
#> case_69     <NA>  62        0        0                0                0                1
#> case_7     Death  83        1        0                1                0                0
#> case_70     <NA>  58        0        1                0                0                1
#> case_71     <NA>  72        0        1                0                0                1
#> case_72  Recover  47        1        0                0                1                0
#> case_73  Recover  69        0        0                0                1                0
#> case_74  Recover  83        0        0                0                1                0
#> case_75    Death   2        0        0                0                1                0
#> case_76    Death  55        0        0                0                1                0
#> case_77  Recover   2        0        0                0                1                0
#> case_78    Death  89        1        0                0                1                0
#> case_79  Recover  37        0        1                0                0                1
```

```
#> case_8     Death  38        1        0                0                0                1
#> case_80     <NA>  74        0        0                0                0                1
#> case_81    Death  86        0        0                0                0                1
#> case_82  Recover  41        0        0                0                0                1
#> case_83  Recover  38        0        0                0                0                0
#> case_84     <NA>  26        0        1                1                0                0
#> case_85     <NA>  80        1        1                0                1                0
#> case_86     <NA>  54        0        1                0                0                1
#> case_87    Death  69        0        0                0                0                1
#> case_88     <NA>   4        0        0                0                0                0
#> case_89  Recover  54        1        0                1                0                0
#> case_9      <NA>  67        1        0                0                0                1
#> case_90     <NA>  43        1        0                0                0                1
#> case_91  Recover  48        1        0                0                0                1
#> case_92     <NA>  66        1        1                0                0                1
#> case_93     <NA>  56        0        0                0                0                1
#> case_94  Recover  35        0        1                0                0                1
#> case_95     <NA>  37        0        0                0                0                1
#> case_96     <NA>  43        0        0                1                0                0
#> case_97  Recover  75        1        1                0                1                0
#> case_98    Death  76        0        0                0                0                1
#> case_99     <NA>  68        0        1                0                0                1
```

```r
cat("NAs before imput: ", sum(is.na(dataset)), "\n")
```

```
#> NAs before imput:  277
```

```r
summary(dataset$outcome)
```

```
#>   Death Recover    NA's
#>      32      47      57
```

```r
cat("NAs after imput: ", sum(is.na(dataset_complete)), "\n")
```

```
#> NAs after imput:  57
```

```r
summary(dataset_complete$outcome)
```

```
#>   Death Recover    NA's
#>      32      47      57
```

15.3 Test, train and validation data sets

For building the model, I am separating the imputed data frame into training and test data. Test data are the 57 cases with unknown outcome.

The training data will be further devided for validation of the models: 70% of the training data will be kept for model building and the remaining 30% will be used for model testing.

I am using the caret package for modeling.

```r
train_index <- which(is.na(dataset_complete$outcome))
train_data <- dataset_complete[-train_index, ]     # 79x12
test_data  <- dataset_complete[train_index, -1]    # 57x11

library(caret)
set.seed(27)
```

```
val_index <- createDataPartition(train_data$outcome, p = 0.7, list=FALSE)
val_train_data <- train_data[val_index, ]
val_test_data  <- train_data[-val_index, ]
val_train_X <- val_train_data[,-1]
val_test_X <- val_test_data[,-1]
```

15.3.1 Decision trees

To get an idea about how each feature contributes to the prediction of the outcome, I first built a decision tree based on the training data using `rpart` and `rattle`.

```
library(rpart)
library(rattle)
library(rpart.plot)
library(RColorBrewer)

set.seed(27)
fit <- rpart(outcome ~ ., data = train_data, method = "class",
             control = rpart.control(xval = 10, minbucket = 2, cp = 0),
             parms = list(split = "information"))

fancyRpartPlot(fit)
```

Rattle 2019−May−13 21:19:28 datascience

This randomly generated decision tree shows that cases with an early outcome were most likely to die when they were 68 or older, when they also had an early onset and when they were sick for fewer than 13 days. If a person was not among the first cases and was younger than 52, they had a good chance of recovering, but if they were 82 or older, they were more likely to die from the flu.

15.3.2 Feature Importance

Not all of the features I created will be equally important to the model. The decision tree already gave me an idea of which features might be most important but I also want to estimate feature importance using a Random Forest approach with repeated cross validation.

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

# train the model
set.seed(27)
model <- train(outcome ~ ., data = train_data, method = "rf",
               preProcess = NULL, trControl = control)

# estimate variable importance
importance <- varImp(model, scale=TRUE)
```

```
from <- c("age", "hospital1", "gender_f1", "province_Jiangsu1",
          "province_Shanghai1", "province_Zhejiang1", "province_other1",
          "days_onset_to_outcome", "days_onset_to_hospital", "early_onset1",
          "early_outcome1" )

to <- c("Age", "Hospital", "Female", "Jiangsu", "Shanghai", "Zhejiang",
        "Other province", "Days onset to outcome", "Days onset to hospital",
        "Early onset", "Early outcome" )

importance_df_1 <- importance$importance
importance_df_1$group <- rownames(importance_df_1)

importance_df_1$group <- mapvalues(importance_df_1$group,
                                   from = from,
                                   to = to)
f = importance_df_1[order(importance_df_1$Overall, decreasing = FALSE), "group"]

importance_df_2 <- importance_df_1
importance_df_2$Overall <- 0

importance_df <- rbind(importance_df_1, importance_df_2)

# setting factor levels
importance_df <- within(importance_df, group <- factor(group, levels = f))
importance_df_1 <- within(importance_df_1, group <- factor(group, levels = f))

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) +
  scale_color_manual(values = rep(brewer.pal(1, "Set1")[1], 11)) +
  my_theme() +
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 0, vjust = 0.5, hjust = 0.5)) +
  labs(
    x = "Importance",
    y = "",
```

```
    title = "2013 Influenza A H7N9 cases in China",
    subtitle = "Scaled feature importance",
    caption = "\nDetermined with Random Forest and
    repeated cross validation (10 repeats, 10 times)"
  )
```

This tells me that age is the most important determining factor for predicting disease outcome, followed by days between onset an outcome, early outcome and days between onset and hospitalisation.

15.3.3 Feature Plot

Before I start actually building models, I want to check whether the distribution of feature values is comparable between training, validation and test datasets.

```
# tidy dataframe of 11 variables for plotting features for all datasets
# dataset_complete: 136x12. test + val_train + val_test
dataset_complete_gather <- dataset_complete %>%
  mutate(set = ifelse(rownames(dataset_complete) %in% rownames(test_data),
                      "Test Data",
               ifelse(rownames(dataset_complete) %in% rownames(val_train_data),
                      "Validation Train Data",
               ifelse(rownames(dataset_complete) %in% rownames(val_test_data),
                      "Validation Test Data", "NA"))),
         case_ID = rownames(.)) %>%
  gather(group, value, age:early_outcome)

# map values in group to more readable
from <- c("age", "hospital", "gender_f", "province_Jiangsu", "province_Shanghai",
          "province_Zhejiang", "province_other", "days_onset_to_outcome",
          "days_onset_to_hospital", "early_onset",  "early_outcome" )
```

```
to <- c("Age", "Hospital", "Female", "Jiangsu", "Shanghai", "Zhejiang",
        "Other province", "Days onset to outcome", "Days onset to hospital",
        "Early onset", "Early outcome" )
dataset_complete_gather$group <- mapvalues(dataset_complete_gather$group,
                        from = from,
                        to = to)
```

15.3.4 Plot distribution of features in each dataset

```
# plot features all datasets
ggplot(data = dataset_complete_gather,
       aes(x = as.numeric(value), fill = outcome, color = outcome)) +
  geom_density(alpha = 0.2) +
  geom_rug() +
  scale_color_brewer(palette="Set1", na.value = "grey50") +
  scale_fill_brewer(palette="Set1", na.value = "grey50") +
  my_theme() +
  facet_wrap(set ~ group, ncol = 11, scales = "free") +
  labs(
    x = "Value",
    y = "Density",
    title = "2013 Influenza A H7N9 cases in China",
    subtitle = "Features for classifying outcome",
    caption = "\nDensity distribution of all features used for
    classification of flu outcome."
  )
```

15.3.5 Plot 3 features vs outcome, all datasets

```
# plot three groups vs outcome
ggplot(subset(dataset_complete_gather,
              group == "Age" |
              group == "Days onset to hospital" |
              group == "Days onset to outcome"),
       aes(x=outcome, y=as.numeric(value), fill=set)) +
  geom_boxplot() +
```

```
my_theme() +
scale_fill_brewer(palette="Set1", type = "div ") +
facet_wrap( ~ group, ncol = 3, scales = "free") +
labs(
  fill = "",
  x = "Outcome",
  y = "Value",
  title = "2013 Influenza A H7N9 cases in China",
  subtitle = "Features for classifying outcome",
  caption = "\nBoxplot of the features age, days from onset to
  hospitalisation and days from onset to outcome."
)
```

Luckily, the distributions looks reasonably similar between the validation and test data, except for a few outliers.

15.4 Comparing Machine Learning algorithms

Before I try to predict the outcome of the unknown cases, I am testing the models' accuracy with the validation datasets on a couple of algorithms. I have chosen only a few more well known algorithms, but `caret` implements many more.

I have chosen to not do any preprocessing because I was worried that the different data distributions with

continuous variables (e.g. age) and binary variables (i.e. 0, 1 classification of e.g. hospitalisation) would lead to problems.

- Random Forest
- GLM net
- k-Nearest Neighbors
- Penalized Discriminant Analysis
- Stabilized Linear Discriminant Analysis
- Nearest Shrunken Centroids
- Single C5.0 Tree
- Partial Least Squares

```
train_control <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 10,
                              verboseIter = FALSE)
```

15.4.1 Random Forest

Random Forests predictions are based on the generation of multiple classification trees.

This model classified 14 out of 23 cases correctly.

```
set.seed(27)
model_rf <- caret::train(outcome ~ .,
                             data = val_train_data,
                             method = "rf",
                             preProcess = NULL,
                             trControl = train_control)
model_rf
```

```
#> Random Forest
#>
#> 56 samples
#> 11 predictors
#>  2 classes: 'Death', 'Recover'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 10 times)
#> Summary of sample sizes: 50, 51, 51, 50, 50, 50, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
#>    2    0.6212381  0.1937524
#>    6    0.6573333  0.2574674
#>   11    0.6691429  0.2742098
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 11.
```

```
confusionMatrix(predict(model_rf, val_test_data[, -1]), val_test_data$outcome)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
```

```
#>    Death       4       3
#>    Recover     5      11
#>
#>                Accuracy : 0.6522
#>                  95% CI : (0.4273, 0.8362)
#>     No Information Rate : 0.6087
#>     P-Value [Acc > NIR] : 0.4216
#>
#>                   Kappa : 0.2397
#>
#>  Mcnemar's Test P-Value : 0.7237
#>
#>             Sensitivity : 0.4444
#>             Specificity : 0.7857
#>          Pos Pred Value : 0.5714
#>          Neg Pred Value : 0.6875
#>              Prevalence : 0.3913
#>          Detection Rate : 0.1739
#>    Detection Prevalence : 0.3043
#>       Balanced Accuracy : 0.6151
#>
#>        'Positive' Class : Death
#>
```

15.4.2 GLM net

Lasso or elastic net regularization for generalized linear model regression are based on linear regression models and is useful when we have feature correlation in our model.

This model classified 13 out of 23 cases correctly.

```r
set.seed(27)
model_glmnet <- caret::train(outcome ~ .,
                             data = val_train_data,
                             method = "glmnet",
                             preProcess = NULL,
                             trControl = train_control)
model_glmnet
```

```
#> glmnet
#>
#> 56 samples
#> 11 predictors
#>  2 classes: 'Death', 'Recover'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 10 times)
#> Summary of sample sizes: 50, 51, 51, 50, 50, 50, ...
#> Resampling results across tuning parameters:
#>
#>   alpha  lambda        Accuracy   Kappa
#>   0.10   0.0003711537  0.6046667  0.1785573
#>   0.10   0.0037115374  0.6046667  0.1799734
#>   0.10   0.0371153744  0.6274286  0.2276455
```

```
#>   0.55   0.0003711537  0.6046667  0.1785573
#>   0.55   0.0037115374  0.6097619  0.1896572
#>   0.55   0.0371153744  0.6369048  0.2422699
#>   1.00   0.0003711537  0.6046667  0.1785573
#>   1.00   0.0037115374  0.6077619  0.1849603
#>   1.00   0.0371153744  0.6517619  0.2656089
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were alpha = 1 and lambda = 0.03711537.
```

```
confusionMatrix(predict(model_glmnet, val_test_data[, -1]), val_test_data$outcome)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       3       4
#>    Recover     6      10
#>
#>                Accuracy : 0.5652
#>                  95% CI : (0.3449, 0.7681)
#>     No Information Rate : 0.6087
#>     P-Value [Acc > NIR] : 0.7418
#>
#>                   Kappa : 0.0496
#>
#>  Mcnemar's Test P-Value : 0.7518
#>
#>             Sensitivity : 0.3333
#>             Specificity : 0.7143
#>          Pos Pred Value : 0.4286
#>          Neg Pred Value : 0.6250
#>              Prevalence : 0.3913
#>          Detection Rate : 0.1304
#>    Detection Prevalence : 0.3043
#>       Balanced Accuracy : 0.5238
#>
#>        'Positive' Class : Death
#>
```

15.4.3 k-Nearest Neighbors

k-nearest neighbors predicts based on point distances with predefined constants.

This model classified 14 out of 23 cases correctly.

```
set.seed(27)
model_kknn <- caret::train(outcome ~ .,
                           data = val_train_data,
                           method = "kknn",
                           preProcess = NULL,
                           trControl = train_control)
model_kknn
```

```
#> k-Nearest Neighbors
```

```
#>
#> 56 samples
#> 11 predictors
#>  2 classes: 'Death', 'Recover'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 10 times)
#> Summary of sample sizes: 50, 51, 51, 50, 50, 50, ...
#> Resampling results across tuning parameters:
#>
#>   kmax  Accuracy   Kappa
#>   5     0.5690476  0.09233238
#>   7     0.5430476  0.03497524
#>   9     0.5387619  0.02352395
#>
#> Tuning parameter 'distance' was held constant at a value of 2
#> Tuning parameter 'kernel' was held constant at a value of optimal
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were kmax = 5, distance = 2 and kernel = optimal.
```

```r
confusionMatrix(predict(model_kknn, val_test_data[, -1]), val_test_data$outcome)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       4       3
#>    Recover     5      11
#>
#>                Accuracy : 0.6522
#>                  95% CI : (0.4273, 0.8362)
#>     No Information Rate : 0.6087
#>     P-Value [Acc > NIR] : 0.4216
#>
#>                   Kappa : 0.2397
#>
#>  Mcnemar's Test P-Value : 0.7237
#>
#>             Sensitivity : 0.4444
#>             Specificity : 0.7857
#>          Pos Pred Value : 0.5714
#>          Neg Pred Value : 0.6875
#>              Prevalence : 0.3913
#>          Detection Rate : 0.1739
#>    Detection Prevalence : 0.3043
#>       Balanced Accuracy : 0.6151
#>
#>        'Positive' Class : Death
#>
```

15.4.4 Penalized Discriminant Analysis

Penalized Discriminant Analysis is the penalized linear discriminant analysis and is also useful for when we have highly correlated features.

This model classified 14 out of 23 cases correctly.

```
set.seed(27)
model_pda <- caret::train(outcome ~ .,
                          data = val_train_data,
                          method = "pda",
                          preProcess = NULL,
                          trControl = train_control)
model_pda
```

```
#> Penalized Discriminant Analysis
#>
#> 56 samples
#> 11 predictors
#>  2 classes: 'Death', 'Recover'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 10 times)
#> Summary of sample sizes: 50, 51, 51, 50, 50, 50, ...
#> Resampling results across tuning parameters:
#>
#>   lambda  Accuracy   Kappa
#>   0e+00         NaN        NaN
#>   1e-04   0.5920000  0.1687676
#>   1e-01   0.5954286  0.1746618
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was lambda = 0.1.
```

```
confusionMatrix(predict(model_pda, val_test_data[, -1]), val_test_data$outcome)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       2       4
#>    Recover     7      10
#>
#>                Accuracy : 0.5217
#>                  95% CI : (0.3059, 0.7318)
#>     No Information Rate : 0.6087
#>     P-Value [Acc > NIR] : 0.8570
#>
#>                   Kappa : -0.0675
#>
#>  Mcnemar's Test P-Value : 0.5465
#>
#>             Sensitivity : 0.22222
#>             Specificity : 0.71429
#>          Pos Pred Value : 0.33333
#>          Neg Pred Value : 0.58824
#>              Prevalence : 0.39130
#>          Detection Rate : 0.08696
#>    Detection Prevalence : 0.26087
#>       Balanced Accuracy : 0.46825
```

```
#>
#>        'Positive' Class : Death
#>
```

15.4.5 Stabilized Linear Discriminant Analysis

Stabilized Linear Discriminant Analysis is designed for high-dimensional data and correlated co-variables.

This model classified 15 out of 23 cases correctly.

```r
set.seed(27)
model_slda <- caret::train(outcome ~ .,
                           data = val_train_data,
                           method = "slda",
                           preProcess = NULL,
                           trControl = train_control)
model_slda
```

```
#> Stabilized Linear Discriminant Analysis
#>
#> 56 samples
#> 11 predictors
#>  2 classes: 'Death', 'Recover'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 10 times)
#> Summary of sample sizes: 50, 51, 51, 50, 50, 50, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.6210952  0.2067368
```

```r
confusionMatrix(predict(model_slda, val_test_data[, -1]), val_test_data$outcome)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       2       3
#>    Recover     7      11
#>
#>                Accuracy : 0.5652
#>                  95% CI : (0.3449, 0.7681)
#>     No Information Rate : 0.6087
#>     P-Value [Acc > NIR] : 0.7418
#>
#>                   Kappa : 0.0086
#>
#>  Mcnemar's Test P-Value : 0.3428
#>
#>             Sensitivity : 0.22222
#>             Specificity : 0.78571
#>          Pos Pred Value : 0.40000
#>          Neg Pred Value : 0.61111
#>              Prevalence : 0.39130
```

```
#>          Detection Rate : 0.08696
#>    Detection Prevalence : 0.21739
#>       Balanced Accuracy : 0.50397
#>
#>        'Positive' Class : Death
#>
```

15.4.6 Nearest Shrunken Centroids

Nearest Shrunken Centroids computes a standardized centroid for each class and shrinks each centroid toward the overall centroid for all classes.

This model classified 15 out of 23 cases correctly.

```
set.seed(27)
model_pam <- caret::train(outcome ~ .,
                          data = val_train_data,
                          method = "pam",
                          preProcess = NULL,
                          trControl = train_control)
```

```
#> 12345678910111213141516171819202122232425262728293011111111111111111111111111111111111111111111111111111
```

```
model_pam
```

```
#> Nearest Shrunken Centroids
#>
#> 56 samples
#> 11 predictors
#>  2 classes: 'Death', 'Recover'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 10 times)
#> Summary of sample sizes: 50, 51, 51, 50, 50, 50, ...
#> Resampling results across tuning parameters:
#>
#>   threshold   Accuracy   Kappa
#>   0.09704703  0.6868571   0.345706472
#>   1.40718188  0.6003333   0.088499703
#>   2.71731673  0.5886667  -0.003333333
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was threshold = 0.09704703.
```

```
confusionMatrix(predict(model_pam, val_test_data[, -1]), val_test_data$outcome)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       4       3
#>    Recover     5      11
#>
#>                Accuracy : 0.6522
#>                  95% CI : (0.4273, 0.8362)
#>     No Information Rate : 0.6087
```

```
#>     P-Value [Acc > NIR] : 0.4216
#>
#>                   Kappa : 0.2397
#>
#>  Mcnemar's Test P-Value : 0.7237
#>
#>             Sensitivity : 0.4444
#>             Specificity : 0.7857
#>          Pos Pred Value : 0.5714
#>          Neg Pred Value : 0.6875
#>              Prevalence : 0.3913
#>          Detection Rate : 0.1739
#>    Detection Prevalence : 0.3043
#>       Balanced Accuracy : 0.6151
#>
#>        'Positive' Class : Death
#>
```

15.4.7 Single C5.0 Tree

C5.0 is another tree-based modeling algorithm.

This model classified 15 out of 23 cases correctly.

```r
set.seed(27)
model_C5.0Tree <- caret::train(outcome ~ .,
                               data = val_train_data,
                               method = "C5.0Tree",
                               preProcess = NULL,
                               trControl = train_control)
model_C5.0Tree
```

```
#> Single C5.0 Tree
#>
#> 56 samples
#> 11 predictors
#>  2 classes: 'Death', 'Recover'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 10 times)
#> Summary of sample sizes: 50, 51, 51, 50, 50, 50, ...
#> Resampling results:
#>
#>   Accuracy   Kappa
#>   0.5878095  0.1095544
```

```r
confusionMatrix(predict(model_C5.0Tree, val_test_data[, -1]), val_test_data$outcome)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       6       3
#>    Recover     3      11
#>
```

```
#>                Accuracy : 0.7391
#>                  95% CI : (0.5159, 0.8977)
#>     No Information Rate : 0.6087
#>     P-Value [Acc > NIR] : 0.1421
#>
#>                   Kappa : 0.4524
#>
#>  Mcnemar's Test P-Value : 1.0000
#>
#>             Sensitivity : 0.6667
#>             Specificity : 0.7857
#>          Pos Pred Value : 0.6667
#>          Neg Pred Value : 0.7857
#>              Prevalence : 0.3913
#>          Detection Rate : 0.2609
#>    Detection Prevalence : 0.3913
#>       Balanced Accuracy : 0.7262
#>
#>        'Positive' Class : Death
#>
```

15.4.8 Partial Least Squares

modeling with correlated features.

This model classified 15 out of 23 cases correctly.

```r
set.seed(27)
model_pls <- caret::train(outcome ~ .,
                              data = val_train_data,
                              method = "pls",
                              preProcess = NULL,
                              trControl = train_control)
model_pls
```

```
#> Partial Least Squares
#>
#> 56 samples
#> 11 predictors
#>  2 classes: 'Death', 'Recover'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 10 times)
#> Summary of sample sizes: 50, 51, 51, 50, 50, 50, ...
#> Resampling results across tuning parameters:
#>
#>   ncomp  Accuracy   Kappa
#>   1      0.6153810  0.1924316
#>   2      0.6490952  0.2661078
#>   3      0.6270000  0.2245740
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was ncomp = 2.
```

```r
confusionMatrix(predict(model_pls, val_test_data[, -1]), val_test_data$outcome)
```

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       4       4
#>    Recover     5      10
#>
#>                Accuracy : 0.6087
#>                  95% CI : (0.3854, 0.8029)
#>     No Information Rate : 0.6087
#>     P-Value [Acc > NIR] : 0.5901
#>
#>                   Kappa : 0.1619
#>
#>  Mcnemar's Test P-Value : 1.0000
#>
#>             Sensitivity : 0.4444
#>             Specificity : 0.7143
#>          Pos Pred Value : 0.5000
#>          Neg Pred Value : 0.6667
#>              Prevalence : 0.3913
#>          Detection Rate : 0.1739
#>    Detection Prevalence : 0.3478
#>       Balanced Accuracy : 0.5794
#>
#>        'Positive' Class : Death
#>
```

15.5 Comparing accuracy of models

All models were similarly accurate.

15.5.1 Summary Accuracy and Kappa

```r
# Create a list of models
models <- list(rf       = model_rf,
               glmnet   = model_glmnet,
               kknn     = model_kknn,
               pda      = model_pda,
               slda     = model_slda,
               pam      = model_pam,
               C5.0Tree = model_C5.0Tree,
               pls      = model_pls)

# Resample the models
resample_results <- resamples(models)

# Generate a summary
summary(resample_results, metric = c("Kappa", "Accuracy"))
```

```
#>
#> Call:
#> summary.resamples(object = resample_results, metric = c("Kappa", "Accuracy"))
#>
#> Models: rf, glmnet, kknn, pda, slda, pam, C5.0Tree, pls
#> Number of resamples: 100
#>
#> Kappa
#>                Min.    1st Qu.    Median       Mean   3rd Qu. Max. NA's
#> rf       -0.6666667  0.0000000 0.3333333 0.27420979 0.5519481 1.00    0
#> glmnet   -0.6666667  0.0000000 0.2678571 0.26560887 0.6153846 1.00    0
#> kknn     -0.8000000 -0.2214286 0.0000000 0.09233238 0.4000000 1.00    0
#> pda      -0.6666667 -0.1538462 0.1666667 0.17466185 0.5454545 1.00    0
#> slda     -0.6666667  0.0000000 0.1818182 0.20673683 0.5454545 1.00    0
#> pam      -0.6666667  0.0000000 0.3333333 0.34570647 0.6666667 1.00    0
#> C5.0Tree -0.6666667 -0.1538462 0.0000000 0.10955439 0.4041667 0.72    0
#> pls      -0.9230769  0.0000000 0.2857143 0.26610780 0.5519481 1.00    0
#>
#> Accuracy
#>               Min. 1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> rf       0.2000000   0.500 0.6666667 0.6691429 0.8000000 1.0000000    0
#> glmnet   0.1666667   0.500 0.6666667 0.6517619 0.8333333 1.0000000    0
#> kknn     0.0000000   0.400 0.6000000 0.5690476 0.6666667 1.0000000    0
#> pda      0.1666667   0.400 0.6000000 0.5954286 0.8000000 1.0000000    0
#> slda     0.1666667   0.500 0.6000000 0.6210952 0.8000000 1.0000000    0
#> pam      0.1666667   0.575 0.6666667 0.6868571 0.8333333 1.0000000    0
#> C5.0Tree 0.1666667   0.500 0.6000000 0.5878095 0.7142857 0.8571429    0
#> pls      0.0000000   0.500 0.6666667 0.6490952 0.8000000 1.0000000    0
```

```
bwplot(resample_results , metric = c("Kappa","Accuracy"))
```

15.5.2 Combined results of predicting validation test samples

To compare the predictions from all models, I summed up the prediction probabilities for Death and Recovery from all models and calculated the log2 of the ratio between the summed probabilities for Recovery by the summed probabilities for Death. All cases with a log2 ratio bigger than 1.5 were defined as Recover, cases with a log2 ratio below -1.5 were defined as Death, and the remaining cases were defined as uncertain.

```
results <- data.frame(
  randomForest = predict(model_rf, newdata = val_test_data[, -1], type="prob"),
  glmnet = predict(model_glmnet, newdata = val_test_data[, -1], type="prob"),
  kknn = predict(model_kknn, newdata = val_test_data[, -1], type="prob"),
  pda = predict(model_pda, newdata = val_test_data[, -1], type="prob"),
  slda = predict(model_slda, newdata = val_test_data[, -1], type="prob"),
  pam = predict(model_pam, newdata = val_test_data[, -1], type="prob"),
  C5.0Tree = predict(model_C5.0Tree, newdata = val_test_data[, -1], type="prob"),
  pls = predict(model_pls, newdata = val_test_data[, -1], type="prob"))

results$sum_Death <- rowSums(results[, grep("Death", colnames(results))])
results$sum_Recover <- rowSums(results[, grep("Recover", colnames(results))])
results$log2_ratio <- log2(results$sum_Recover/results$sum_Death)
results$true_outcome <- val_test_data$outcome
results$pred_outcome <- ifelse(results$log2_ratio > 1.5, "Recover",
               ifelse(results$log2_ratio < -1.5, "Death", "uncertain"))
results$prediction <- ifelse(results$pred_outcome == results$true_outcome, "CORRECT",
               ifelse(results$pred_outcome == "uncertain", "uncertain", "wrong"))
results[, -c(1:16)]
```

```
#>          sum_Death sum_Recover  log2_ratio true_outcome pred_outcome prediction
#> case_123 2.2615972    5.738403  1.34330721        Death    uncertain  uncertain
#> case_127 1.2246065    6.775394  2.46798652      Recover      Recover    CORRECT
#> case_128 2.5044146    5.495585  1.13379975      Recover    uncertain  uncertain
#> case_14  2.2277804    5.772220  1.37351914      Recover    uncertain  uncertain
#> case_19  3.8985137    4.101486  0.07322260        Death    uncertain  uncertain
#> case_2   3.9673468    4.032653  0.02355485        Death    uncertain  uncertain
#> case_20  3.4039928    4.596007  0.43315310      Recover    uncertain  uncertain
#> case_21  3.8232306    4.176769  0.12759531      Recover    uncertain  uncertain
#> case_34  5.9616109    2.038389 -1.54827270        Death        Death    CORRECT
#> case_37  0.9633813    7.036619  2.86870346      Recover      Recover    CORRECT
#> case_5   1.2508213    6.749179  2.43183632      Recover      Recover    CORRECT
#> case_51  3.6457908    4.354209  0.25617893      Recover    uncertain  uncertain
#> case_55  3.4877525    4.512248  0.37154857      Recover    uncertain  uncertain
#> case_6   2.0907461    5.909254  1.49895812        Death    uncertain  uncertain
#> case_61  4.9874015    3.012599 -0.72727991        Death    uncertain  uncertain
#> case_65  0.9036931    7.096307  2.97316368      Recover      Recover    CORRECT
#> case_74  5.6347036    2.365296 -1.25231876      Recover    uncertain  uncertain
#> case_78  4.3010148    3.698985 -0.21754754        Death    uncertain  uncertain
#> case_79  1.4720485    6.527952  2.14880514      Recover      Recover    CORRECT
#> case_8   2.1806628    5.819337  1.41608815        Death    uncertain  uncertain
#> case_87  3.2784093    4.721591  0.52627703        Death    uncertain  uncertain
#> case_91  1.4933131    6.506687  2.12340642      Recover      Recover    CORRECT
#> case_94  2.8615037    5.138496  0.84457277      Recover    uncertain  uncertain
```

All predictions based on all models were either correct or uncertain.

15.6 Predicting unknown outcomes

The above models will now be used to predict the outcome of cases with unknown fate.

```
train_control <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 10,
                              verboseIter = FALSE)

set.seed(27)
model_rf <- caret::train(outcome ~ .,
                             data = train_data,
                             method = "rf",
                             preProcess = NULL,
                             trControl = train_control)
model_glmnet <- caret::train(outcome ~ .,
                             data = train_data,
                             method = "glmnet",
                             preProcess = NULL,
                             trControl = train_control)
model_kknn <- caret::train(outcome ~ .,
                             data = train_data,
                             method = "kknn",
                             preProcess = NULL,
                             trControl = train_control)
model_pda <- caret::train(outcome ~ .,
                             data = train_data,
                             method = "pda",
                             preProcess = NULL,
                             trControl = train_control)
model_slda <- caret::train(outcome ~ .,
                             data = train_data,
                             method = "slda",
                             preProcess = NULL,
                             trControl = train_control)
model_pam <- caret::train(outcome ~ .,
                             data = train_data,
                             method = "pam",
                             preProcess = NULL,
                             trControl = train_control)
```

```
#> 12345678910111213141516171819202122232425262728293011111111111111111111111111111111111111111111111111111
```

```
model_C5.0Tree <- caret::train(outcome ~ .,
                             data = train_data,
                             method = "C5.0Tree",
                             preProcess = NULL,
                             trControl = train_control)
model_pls <- caret::train(outcome ~ .,
                             data = train_data,
                             method = "pls",
                             preProcess = NULL,
                             trControl = train_control)

models <- list(rf = model_rf,
```

```
                glmnet   = model_glmnet,
                kknn     = model_kknn,
                pda      = model_pda,
                slda     = model_slda,
                pam      = model_pam,
                C5.0Tree = model_C5.0Tree,
                pls      = model_pls)

# Resample the models
resample_results <- resamples(models)

bwplot(resample_results , metric = c("Kappa","Accuracy"))
```

Here again, the accuracy is similar in all models.

15.6.1 Final results

The final results are calculated as described above.

```
results <- data.frame(
  randomForest = predict(model_rf, newdata = test_data, type="prob"),
  glmnet = predict(model_glmnet, newdata = test_data, type="prob"),
  kknn = predict(model_kknn, newdata = test_data, type="prob"),
  pda = predict(model_pda, newdata = test_data, type="prob"),
  slda = predict(model_slda, newdata = test_data, type="prob"),
  pam = predict(model_pam, newdata = test_data, type="prob"),
  C5.0Tree = predict(model_C5.0Tree, newdata = test_data, type="prob"),
  pls = predict(model_pls, newdata = test_data, type="prob"))
```

```
results$sum_Death <- rowSums(results[, grep("Death", colnames(results))])
results$sum_Recover <- rowSums(results[, grep("Recover", colnames(results))])
results$log2_ratio <- log2(results$sum_Recover/results$sum_Death)
results$predicted_outcome <- ifelse(results$log2_ratio > 1.5, "Recover",
                                    ifelse(results$log2_ratio < -1.5, "Death",
                                           "uncertain"))
results[, -c(1:16)]
```

```
#>          sum_Death sum_Recover log2_ratio predicted_outcome
#> case_100  1.967720    6.032280  1.6161782           Recover
#> case_101  2.647819    5.352181  1.0153224         uncertain
#> case_102  2.939504    5.060496  0.7837061         uncertain
#> case_103  4.761670    3.238330 -0.5562173         uncertain
#> case_104  1.701923    6.298077  1.8877459           Recover
#> case_105  1.476635    6.523365  2.1433032           Recover
#> case_108  6.138719    1.861281 -1.7216422             Death
#> case_109  2.170930    5.829070  1.4249527         uncertain
#> case_110  2.040455    5.959545  1.5463110           Recover
#> case_112  3.490092    4.509908  0.3698330         uncertain
#> case_113  2.591667    5.408333  1.0613036         uncertain
#> case_114  3.048364    4.951636  0.6998699         uncertain
#> case_115  1.414963    6.585037  2.2184275           Recover
#> case_118  2.888458    5.111542  0.8234593         uncertain
#> case_120  5.473646    2.526354 -1.1154450         uncertain
#> case_122  1.611154    6.388846  1.9874611           Recover
#> case_126  4.606299    3.393701 -0.4407484         uncertain
#> case_130  3.514324    4.485676  0.3520781         uncertain
#> case_132  4.733816    3.266184 -0.5353977         uncertain
#> case_136  3.101313    4.898687  0.6595162         uncertain
#> case_15   5.572913    2.427087 -1.1992059         uncertain
#> case_16   6.123160    1.876840 -1.7059708             Death
#> case_22   6.411340    1.588660 -2.0128157             Death
#> case_28   6.127011    1.872989 -1.7098410             Death
#> case_31   5.790278    2.209722 -1.3897678         uncertain
#> case_32   6.024659    1.975341 -1.6087782             Death
#> case_38   5.296652    2.703348 -0.9703335         uncertain
#> case_39   4.872198    3.127802 -0.6394235         uncertain
#> case_4    4.778177    3.221823 -0.5685831         uncertain
#> case_40   2.782043    5.217957  0.9073405         uncertain
#> case_41   5.357210    2.642790 -1.0194200         uncertain
#> case_42   3.408183    4.591817  0.4300626         uncertain
#> case_47   5.893493    2.106507 -1.4842699         uncertain
#> case_48   2.411625    5.588375  1.2124234         uncertain
#> case_52   5.574206    2.425794 -1.2003092         uncertain
#> case_54   6.021572    1.978428 -1.6057853             Death
#> case_56   5.771738    2.228262 -1.3730872         uncertain
#> case_62   5.391787    2.608213 -1.0477019         uncertain
#> case_63   3.217351    4.782649  0.5719365         uncertain
#> case_66   5.583041    2.416959 -1.2078584         uncertain
#> case_67   1.987706    6.012294  1.5968109           Recover
#> case_68   4.170499    3.829501 -0.1230636         uncertain
#> case_69   2.083688    5.916312  1.5055592           Recover
#> case_70   2.059378    5.940622  1.5284057           Recover
#> case_71   2.484213    5.515787  1.1507782         uncertain
```

```
#> case_80   5.871840    2.128160 -1.4642058         uncertain
#> case_84   4.204050    3.795950 -0.1473187         uncertain
#> case_85   2.956773    5.043227  0.7703236         uncertain
#> case_86   2.129425    5.870575  1.4630381         uncertain
#> case_88   1.077039    6.922961  2.6843185           Recover
#> case_9    2.637667    5.362333  1.0235986         uncertain
#> case_90   1.782990    6.217010  1.8019218           Recover
#> case_92   2.514391    5.485609  1.1254427         uncertain
#> case_93   1.987706    6.012294  1.5968109           Recover
#> case_95   4.645061    3.354939 -0.4694111         uncertain
#> case_96   4.624548    3.375452 -0.4542317         uncertain
#> case_99   4.819786    3.180214 -0.5998456         uncertain
```

```
write.table(results, "results_prediction_unknown_outcome_ML_part1.txt",
            col.names = T, sep = "\t")
```

```
results %>%
  filter(predicted_outcome == "Recover") %>%
  select(-c(1:16))
```

```
#>    sum_Death sum_Recover log2_ratio predicted_outcome
#> 1   1.967720    6.032280   1.616178           Recover
#> 2   1.701923    6.298077   1.887746           Recover
#> 3   1.476635    6.523365   2.143303           Recover
#> 4   2.040455    5.959545   1.546311           Recover
#> 5   1.414963    6.585037   2.218427           Recover
#> 6   1.611154    6.388846   1.987461           Recover
#> 7   1.987706    6.012294   1.596811           Recover
#> 8   2.083688    5.916312   1.505559           Recover
#> 9   2.059378    5.940622   1.528406           Recover
#> 10  1.077039    6.922961   2.684319           Recover
#> 11  1.782990    6.217010   1.801922           Recover
#> 12  1.987706    6.012294   1.596811           Recover
```

15.6.2 Predicted outcome

```
results %>%
  group_by(predicted_outcome) %>%
  summarize(n = n())
```

```
#> # A tibble: 3 x 2
#>   predicted_outcome     n
#>   <chr>             <int>
#> 1 Death                 6
#> 2 Recover              12
#> 3 uncertain            39
```

From 57 cases, 14 were defined as Recover, 3 as Death and 40 as uncertain.

```
results_combined <- merge(results[, -c(1:16)],
                          fluH7N9_china_2013[which(fluH7N9_china_2013$case_ID
                                                   %in% rownames(results)), ],
                          by.x = "row.names", by.y = "case_ID")
# results_combined <- results_combined[, -c(2, 3, 8, 9)]
results_combined <- results_combined[, -c(1, 2, 3, 9, 10)]
```

```
results_combined
```

```
#>    log2_ratio predicted_outcome case_id date_of_onset date_of_hospitalisation gender age  province
#> 1   1.6161782           Recover     100    2013-04-16                    <NA>      m  58  Zhejiang
#> 2   1.0153224         uncertain     101    2013-04-13                    <NA>      f  79  Zhejiang
#> 3   0.7837061         uncertain     102    2013-04-12                    <NA>      m  81  Zhejiang
#> 4  -0.5562173         uncertain     103    2013-04-13              2013-04-19      m  68   Jiangsu
#> 5   1.8877459           Recover     104    2013-04-16                    <NA>      f  54  Zhejiang
#> 6   2.1433032           Recover     105    2013-04-14                    <NA>      m  32  Zhejiang
#> 7  -1.7216422             Death     108    2013-04-15                    <NA>      m  84  Zhejiang
#> 8   1.4249527         uncertain     109    2013-04-15                    <NA>      m  62  Zhejiang
#> 9   1.5463110           Recover     110    2013-04-12              2013-04-16      m  53    Taiwan
#> 10  0.3698330         uncertain     112    2013-04-17                    <NA>      m  69   Jiangxi
#> 11  1.0613036         uncertain     113    2013-04-15                    <NA>      f  60  Zhejiang
#> 12  0.6998699         uncertain     114    2013-04-18                    <NA>      f  50  Zhejiang
#> 13  2.2184275           Recover     115    2013-04-17                    <NA>      m  38  Zhejiang
#> 14  0.8234593         uncertain     118    2013-04-17                    <NA>      m  49   Jiangsu
#> 15 -1.1154450         uncertain     120    2013-03-08                    <NA>      m  60   Jiangsu
#> 16  1.9874611           Recover     122    2013-04-18                    <NA>      m  38  Zhejiang
#> 17 -0.4407484         uncertain     126    2013-04-17              2013-04-27      m  80    Fujian
#> 18  0.3520781         uncertain     130    2013-04-29              2013-04-30      m  69    Fujian
#> 19 -0.5353977         uncertain     132    2013-05-03              2013-05-03      f  79   Jiangxi
#> 20  0.6595162         uncertain     136    2013-07-27              2013-07-28      f  51 Guangdong
#> 21 -1.1992059         uncertain      15    2013-03-20                    <NA>      f  61   Jiangsu
#> 22 -1.7059708             Death      16    2013-03-21                    <NA>      m  79   Jiangsu
#> 23 -2.0128157             Death      22    2013-03-28              2013-04-01      m  85   Jiangsu
#> 24 -1.7098410             Death      28    2013-03-29                    <NA>      m  79  Zhejiang
#> 25 -1.3897678         uncertain      31    2013-03-29                    <NA>      m  70   Jiangsu
#> 26 -1.6087782             Death      32    2013-04-02                    <NA>      m  74   Jiangsu
#> 27 -0.9703335         uncertain      38    2013-04-03                    <NA>      m  56   Jiangsu
#> 28 -0.6394235         uncertain      39    2013-04-08              2013-04-08      m  66  Zhejiang
#> 29 -0.5685831         uncertain       4    2013-03-19              2013-03-27      f  45   Jiangsu
#> 30  0.9073405         uncertain      40    2013-04-06              2013-04-11      m  74  Zhejiang
#> 31 -1.0194200         uncertain      41    2013-04-06              2013-04-12      f  54  Zhejiang
#> 32  0.4300626         uncertain      42    2013-04-03              2013-04-10      m  53  Shanghai
#> 33 -1.4842699         uncertain      47    2013-04-01                    <NA>      m  72   Jiangsu
#> 34  1.2124234         uncertain      48    2013-04-03              2013-04-09      m  65  Zhejiang
#> 35 -1.2003092         uncertain      52    2013-04-06                    <NA>      f  64  Zhejiang
#> 36 -1.6057853             Death      54    2013-04-06                    <NA>      m  75  Zhejiang
#> 37 -1.3730872         uncertain      56    2013-04-05              2013-04-11      m  73  Shanghai
#> 38 -1.0477019         uncertain      62    2013-04-03                    <NA>      f  68  Zhejiang
#> 39  0.5719365         uncertain      63    2013-04-10                    <NA>      m  60     Anhui
#> 40 -1.2078584         uncertain      66          <NA>                    <NA>      m  72   Jiangsu
#> 41  1.5968109           Recover      67    2013-04-12                    <NA>      m  56  Zhejiang
#> 42 -0.1230636         uncertain      68    2013-04-10                    <NA>      m  57  Zhejiang
#> 43  1.5055592           Recover      69    2013-04-10                    <NA>      m  62  Zhejiang
#> 44  1.5284057           Recover      70    2013-04-11                    <NA>      f  58  Zhejiang
#> 45  1.1507782         uncertain      71    2013-04-10                    <NA>      f  72  Zhejiang
#> 46 -1.4642058         uncertain      80    2013-04-08                    <NA>      m  74  Zhejiang
#> 47 -0.1473187         uncertain      84          <NA>                    <NA>      f  26   Jiangsu
#> 48  0.7703236         uncertain      85    2013-04-09              2013-04-16      f  80  Shanghai
#> 49  1.4630381         uncertain      86    2013-04-13                    <NA>      f  54  Zhejiang
#> 50  2.6843185           Recover      88          <NA>                    <NA>      m   4   Beijing
#> 51  1.0235986         uncertain       9    2013-03-25              2013-03-25      m  67  Zhejiang
```

```
#> 52  1.8019218          Recover      90    2013-04-12              2013-04-15      m  43  Zhejiang
#> 53  1.1254427        uncertain      92    2013-04-10              2013-04-17      f  66  Zhejiang
#> 54  1.5968109          Recover      93    2013-04-11                    <NA>      m  56  Zhejiang
#> 55 -0.4694111        uncertain      95    2013-03-30                    <NA>      m  37  Zhejiang
#> 56 -0.4542317        uncertain      96    2013-04-07                    <NA>      m  43   Jiangsu
#> 57 -0.5998456        uncertain      99    2013-04-12                    <NA>      f  68  Zhejiang
```

```r
# tidy dataframe for plotting
results_combined_gather <- results_combined %>%
  gather(group_dates, date, date_of_onset:date_of_hospitalisation)

results_combined_gather$group_dates <- factor(results_combined_gather$group_dates,
                          levels = c("date_of_onset", "date_of_hospitalisation"))

results_combined_gather$group_dates <- mapvalues(results_combined_gather$group_dates,
                            from = c("date_of_onset", "date_of_hospitalisation"),
                              to = c("Date of onset", "Date of hospitalisation"))

results_combined_gather$gender <- mapvalues(results_combined_gather$gender,
                                            from = c("f", "m"),
                                             to = c("Female", "Male"))
levels(results_combined_gather$gender) <- c(levels(results_combined_gather$gender),
                                            "unknown")
results_combined_gather$gender[is.na(results_combined_gather$gender)] <- "unknown"
```

```r
ggplot(data = results_combined_gather, aes(x = date, y = log2_ratio,
                                           color = predicted_outcome)) +
  geom_jitter(aes(size = as.numeric(age)), alpha = 0.3) +
  geom_rug() +
  facet_grid(gender ~ group_dates) +
  labs(
    color = "Predicted outcome",
    size = "Age",
    x = "Date in 2013",
    y = "log2 ratio of prediction Recover vs Death",
    title = "2013 Influenza A H7N9 cases in China",
    subtitle = "Predicted outcome",
    caption = ""
  ) +
  my_theme() +
  scale_shape_manual(values = c(15, 16, 17)) +
  scale_color_brewer(palette="Set1") +
  scale_fill_brewer(palette="Set1")
```

The comparison of date of onset, data of hospitalisation, gender and age with predicted outcome shows that predicted deaths were associated with older age than predicted recoveries. Date of onset does not show an obvious bias in either direction.

15.7 Conclusions

This dataset posed a couple of difficulties to begin with, like unequal distribution of data points across variables and missing data. This makes the modeling inherently prone to flaws. However, real life data isn't perfect either, so I went ahead and tested the modeling success anyway.

By accounting for uncertain classification with low predictions probability, the validation data could be classified accurately. However, for a more accurate model, these few cases don't give enough information to reliably predict the outcome. More cases, more information (i.e. more features) and fewer missing data would improve the modeling outcome.

Also, this example is only applicable for this specific case of flu. In order to be able to draw more general conclusions about flu outcome, other cases and additional information, for example on medical parameters like preexisting medical conditions, disase parameters, demographic information, etc. would be necessary.

All in all, this dataset served as a nice example of the possibilities (and pitfalls) of machine learning applications and showcases a basic workflow for building prediction models with R.

For a comparison of feature selection methods see here.

If you see any mistakes or have tips and tricks for improvement, please don't hesitate to let me know! Thanks. :-)

```
sessionInfo()
```

```
#> R version 3.5.3 (2019-03-11)
#> Platform: x86_64-pc-linux-gnu (64-bit)
```

```
#> Running under: Ubuntu 18.04.2 LTS
#>
#> Matrix products: default
#> BLAS: /opt/R/3.5.3/lib/R/lib/libRblas.so
#> LAPACK: /opt/R/3.5.3/lib/R/lib/libRlapack.so
#>
#> locale:
#>  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8        LC_COLLATE=en_U
#>
#> attached base packages:
#> [1] grid      stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#>  [1] RColorBrewer_1.1-2 rpart.plot_3.0.7   rattle_5.2.0       rpart_4.1-15       caret_6.0-84       r
#>
#> loaded via a namespace (and not attached):
#>  [1] nlme_3.1-139        lubridate_1.7.4     rprojroot_1.3-2     C50_0.1.2           tools_3.5.3
#> [50] mitml_0.3-7         crayon_1.3.4        splines_3.5.3       zeallot_0.1.0       knitr_1.22
```

Chapter 16

Dealing with unbalanced data

Source: https://shiring.github.io/machine_learning/2017/04/02/unbalanced

```
library(caret)
library(mice)
library(ggplot2)
```

In my last post, where I shared the code that I used to produce an example analysis to go along with my webinar on building meaningful models for disease prediction, I mentioned that it is advised to consider over- or under-sampling when you have unbalanced data sets. Because my focus in this webinar was on evaluating model performance, I did not want to add an additional layer of complexity and therefore did not further discuss how to specifically deal with unbalanced data.

But because I had gotten a few questions regarding this, I thought it would be worthwhile to explain over- and under-sampling techniques in more detail and show how you can very easily implement them with `caret`.

16.1 Read and process the data

```
bc_data <- read.table(file.path(data_raw_dir, "breast-cancer-wisconsin.data"),
                      header = FALSE, sep = ",")

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")

bc_data$classes <- ifelse(bc_data$classes == "2", "benign",
                          ifelse(bc_data$classes == "4", "malignant", NA))

bc_data[bc_data == "?"] <- NA

# how many NAs are in the data
length(which(is.na(bc_data)))
```

```
[1] 16
```

```
# impute missing data

# skip columns: sample_code_number and classes
bc_data[,2:10] <- apply(bc_data[, 2:10], 2, function(x) as.numeric(as.character(x)))

# impute but stay 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)
```

16.1.1 Unbalanced data

In this context, unbalanced data refers to classification problems where we have unequal instances for different classes. Having unbalanced data is actually very common in general, but it is especially prevalent when working with disease data where we usually have more healthy control samples than disease cases. Even more extreme unbalance is seen with fraud detection, where e.g. most credit card uses are okay and only very few will be fraudulent. In the example I used for my webinar, a *breast cancer* dataset, we had about twice as many benign than malignant samples.

```
# how many benign and malignant cases are there?
summary(bc_data$classes)
```

```
   benign malignant
      458       241
```

16.1.1.1 Why is unbalanced data a problem in machine learning?

Most machine learning classification algorithms are sensitive to unbalance in the predictor classes. Let's consider an even more extreme example than our breast cancer dataset: assume we had 10 malignant vs 90 benign samples. A machine learning model that has been trained and tested on such a dataset could now predict "benign" for all samples and still gain a very high accuracy. An unbalanced dataset will bias the prediction model towards the more common class!

16.1.1.2 How to balance data for modeling

The basic theoretical concepts behind over- and under-sampling are very simple:

With under-sampling, we randomly select a subset of samples from the class with more instances to match the number of samples coming from each class. In our example, we would randomly pick 241 out of the 458 benign cases. The main disadvantage of under-sampling is that we lose potentially relevant information from the left-out samples.

With oversampling, we randomly duplicate samples from the class with fewer instances or we generate additional instances based on the data that we have, so as to match the number of samples in each class. While we avoid losing information with this approach, we also run the risk of overfitting our model as we are more likely to get the same samples in the training and in the test data, i.e. the test data is no longer independent from training data. This would lead to an overestimation of our model's performance and generalizability.

In reality though, we should not simply perform over- or under-sampling on our training data and then run the model. We need to account for cross-validation and perform over- or under-sampling on each fold independently to get an honest estimate of model performance!

16.1.1.3 Modeling the original unbalanced data

Here is the same model I used in my webinar example: I randomly divide the data into training and test sets (stratified by class) and perform Random Forest modeling with 10 x 10 repeated cross-validation. Final model performance is then measured on the test set.

```
set.seed(42)
index <- createDataPartition(bc_data$classes, p = 0.7, list = FALSE)
train_data <- bc_data[index, ]
test_data  <- bc_data[-index, ]
```

```
set.seed(42)
model_rf <- caret::train(classes ~ .,
                         data = train_data,
                         method = "rf",
                         preProcess = c("scale", "center"),
                         trControl = trainControl(method = "repeatedcv",
                                                  number = 10,
                                                  repeats = 10,
                                                  verboseIter = FALSE))
```

```
final <- data.frame(actual = test_data$classes,
                    predict(model_rf,
                            newdata = test_data,
                            type = "prob"))

final$predict <- ifelse(final$benign > 0.5, "benign", "malignant")
```

```
final_predict <- as.factor(final$predict)
test_data_classes <- as.factor(test_data$classes)

cm_original <- confusionMatrix(final_predict, test_data_classes)
cm_original$byClass['Sensitivity']
```

```
Sensitivity
  0.9708029
```

16.1.1.4 Under-sampling

Luckily, `caret` makes it very easy to incorporate over- and under-sampling techniques with cross-validation resampling. We can simply add the sampling option to our `trainControl` and choose down for under- (also called `down`-) sampling. The rest stays the same as with our original model.

```
set.seed(42)
ctrl <- trainControl(method = "repeatedcv",
                     number = 10,
                     repeats = 10,
                     verboseIter = FALSE,
                     sampling = "down")
```

```
model_rf_under <- caret::train(classes ~ .,
                           data = train_data,
                           method = "rf",
                           preProcess = c("scale", "center"),
                           trControl = ctrl)
```

```
final_under <- data.frame(actual = test_data$classes,
                     predict(model_rf_under,
                             newdata = test_data,
                             type = "prob"))

final_under$predict <- ifelse(final_under$benign > 0.5, "benign", "malignant")
```

```
final_under_predict <- as.factor(final_under$predict)
test_data_classes <- test_data$classes

cm_under <- confusionMatrix(final_under_predict, test_data_classes)
cm_under$byClass['Sensitivity']
```

```
Sensitivity
  0.9708029
```

16.1.1.5 Oversampling

For over- (also called up-) sampling we simply specify sampling = "up".

```
set.seed(42)
ctrl <- trainControl(method = "repeatedcv",
                     number = 10,
                     repeats = 10,
                     verboseIter = FALSE,
                     sampling = "up")


model_rf_over <- caret::train(classes ~ .,
                           data = train_data,
                           method = "rf",
                           preProcess = c("scale", "center"),
                           trControl = ctrl)
```

```
final_over <- data.frame(actual = test_data$classes,
                          predict(model_rf_over,
                                  newdata = test_data,
                                  type = "prob"))

final_over$predict <- ifelse(final_over$benign > 0.5, "benign", "malignant")
```

```
final_over_predict <- as.factor(final_over$predict)
test_data_classes <- test_data$classes

cm_over <- confusionMatrix(final_over_predict, test_data_classes)
cm_over$byClass['Sensitivity']
```

```
Sensitivity
  0.9708029
```

16.1.2 ROSE

Besides over- and under-sampling, there are hybrid methods that combine under-sampling with the generation of additional data. Two of the most popular are ROSE and SMOTE.

> From Nicola Lunardon, Giovanna Menardi and Nicola Torelli's "ROSE: A Package for Binary Imbalanced Learning" (R Journal, 2014, Vol. 6 Issue 1, p. 79): "The ROSE package provides functions to deal with binary classification problems in the presence of imbalanced classes. Artificial balanced samples are generated according to a smoothed bootstrap approach and allow for aiding both the phases of estimation and accuracy evaluation of a binary classifier in the presence of a rare class. Functions that implement more traditional remedies for the class imbalance and different metrics to evaluate accuracy are also provided. These are estimated by holdout, bootstrap, or cross-validation methods."

You implement them the same way as before, this time choosing sampling = "rose"…

```
set.seed(42)
ctrl <- trainControl(method = "repeatedcv",
                     number = 10,
                     repeats = 10,
                     verboseIter = FALSE,
                     sampling = "rose")

model_rf_rose <- caret::train(classes ~ .,
                              data = train_data,
                              method = "rf",
                              preProcess = c("scale", "center"),
                              trControl = ctrl)
```

```
final_rose <- data.frame(actual = test_data$classes,
                         predict(model_rf_rose,
                                 newdata = test_data,
                                 type = "prob"))

final_rose$predict <- ifelse(final_rose$benign > 0.5, "benign", "malignant")
```

```
cm_rose <- confusionMatrix(as.factor(final_rose$predict),
                           as.factor(test_data$classes))
cm_rose$byClass['Sensitivity']
```

```
Sensitivity
  0.9562044
```

16.1.3 SMOTE

… or by choosing sampling = "smote" in the `trainControl` settings.

> From Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall and W. Philip Kegelmeyer's "SMOTE: Synthetic Minority Over-sampling Technique" (Journal of Artificial Intelligence Research, 2002, Vol. 16, pp. 321–357): "This paper shows that a combination of our method of over-sampling the minority (abnormal) class and under-sampling the majority (normal) class can achieve better classifier performance (in ROC space) than only under-sampling the majority class. This paper also shows that a combination of our method of over-sampling the minority class and under-sampling the majority class can achieve better classifier performance (in ROC space) than varying the loss ratios in Ripper or class priors in Naive Bayes. Our method of over-sampling the minority class involves creating synthetic minority class examples."

```
set.seed(42)
ctrl <- trainControl(method = "repeatedcv",
                     number = 10,
                     repeats = 10,
                     verboseIter = FALSE,
                     sampling = "smote")

model_rf_smote <- caret::train(classes ~ .,
                               data = train_data,
                               method = "rf",
                               preProcess = c("scale", "center"),
                               trControl = ctrl)
```

```
final_smote <- data.frame(actual = test_data$classes,
                          predict(model_rf_smote,
                                  newdata = test_data,
                                  type = "prob"))

final_smote$predict <- ifelse(final_smote$benign > 0.5, "benign", "malignant")
```

```
cm_smote <- confusionMatrix(as.factor(final_smote$predict),
                            as.factor(test_data$classes))
cm_smote$byClass['Sensitivity']
```

```
Sensitivity
  0.9708029
```

16.1.3.1 Predictions

Now let's compare the predictions of all these models:

```
models <- list(
                original = model_rf,
                under = model_rf_under,
                over = model_rf_over,
                smote = model_rf_smote,
                rose = model_rf_rose)

resampling <- resamples(models)
bwplot(resampling)
```

```
library(dplyr)
comparison <- data.frame(model = names(models),
                         Sensitivity = rep(NA, length(models)),
                         Specificity = rep(NA, length(models)),
                         Precision   = rep(NA, length(models)),
                         Recall      = rep(NA, length(models)),
                         F1          = rep(NA, length(models)))

for (name in names(models)) {
    cm_model <- get(paste0("cm_", name))
    comparison[comparison$model==name, ] <- filter(comparison, model==name) %>%
    mutate(Sensitivity = cm_model$byClass["Sensitivity"],
           Specificity = cm_model$byClass["Specificity"],
           Precision   = cm_model$byClass["Precision"],
           Recall      = cm_model$byClass["Recall"],
           F1          = cm_model$byClass["F1"]
  )
}

print(comparison)
```

```
     model Sensitivity Specificity Precision    Recall        F1
1 original   0.9708029   0.9722222 0.9851852 0.9708029 0.9779412
2    under   0.9708029   0.9861111 0.9925373 0.9708029 0.9815498
3     over   0.9708029   0.9861111 0.9925373 0.9708029 0.9815498
4    smote   0.9708029   0.9861111 0.9925373 0.9708029 0.9815498
5     rose   0.9562044   0.9861111 0.9924242 0.9562044 0.9739777
```

```
library(tidyr)
comparison %>%
  gather(x, y, Sensitivity:F1) %>%
```

```
ggplot(aes(x = x, y = y, color = model)) +
  geom_jitter(width = 0.2, alpha = 0.5, size = 3)
```

With this small dataset, we can already see how the different techniques can influence model performance. **Sensitivity** (or **recall**) describes the proportion of benign cases that have been predicted correctly, while **specificity** describes the proportion of malignant cases that have been predicted correctly. **Precision describes** the true positives, i.e. the proportion of benign predictions that were actual from benign samples. **F1** is the weighted average of precision and sensitivity/ recall.

Here, all four methods improved specificity and precision compared to the original model. Under-sampling, over-sampling and ROSE additionally improved precision and the F1 score.

This post shows a simple example of how to correct for unbalance in datasets for machine learning. For more advanced instructions and potential caveats with these techniques, check out the excellent `caret` documentation.

If you are interested in more machine learning posts, check out the category listing for machine_learning on my blog.

Chapter 17

Orange Juice

Source: https://www.machinelearningplus.com/machine-learning/caret-package/ Data: https://raw.githubusercontent.com/selva86/datasets/master/orange_juice_withmissing.csv

17.1 Introduction

A lot of exciting stuff ahead. To make it simpler, this tutorial is structured to cover the following 5 topics:

1. Data Preparation and Preprocessing
2. Visualize the importance of variables
3. Feature Selection using RFE
4. Training and Tuning the model
5. Ensembling the predictions

17.2 2. Initial Setup – load the package and dataset

For this tutorial, I am going to use a modified version of the Orange Juice Data, originally made available in the ISLR package.

The goal of this dataset is to predict which of the two brands of orange juices did the customers buy.

The predictor variables are characteristics of the customer and the product itself. It contains 1070 rows with 18 columns. The response variable is 'Purchase' which takes either the value 'CH'(citrus hill) or 'MM'(minute maid).

I have chosen a lightweight dataset so the focus is on getting familiar with the usage of caret package, instead of spending time on training models on large data.

Let's import the dataset and see it's structure and starting few rows.

```
library(caret)
library(readr)
library(tictoc)

# write_csv(orange, path = "orange.csv")
orange <- read_csv(file = file.path(data_raw_dir, "orange.csv"))    # 1070x18
```

```r
# Structure of the dataframe
dplyr::glimpse(orange)
```

```
Observations: 1,070
Variables: 18
$ Purchase       <chr> "CH", "CH", "CH", "MM", "CH", "CH", "CH", "CH",...
$ WeekofPurchase <dbl> 237, 239, 245, 227, 228, 230, 232, 234, 235, 23...
$ StoreID        <dbl> 1, 1, 1, 1, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,...
$ PriceCH        <dbl> 1.75, 1.75, 1.86, 1.69, 1.69, 1.69, 1.69, 1.75,...
$ PriceMM        <dbl> 1.99, 1.99, 2.09, 1.69, 1.69, 1.99, 1.99, 1.99,...
$ DiscCH         <dbl> 0.00, 0.00, 0.17, 0.00, 0.00, 0.00, 0.00, 0.00,...
$ DiscMM         <dbl> 0.00, 0.30, 0.00, 0.00, 0.00, 0.00, 0.40, 0.40,...
$ SpecialCH      <dbl> 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,...
$ SpecialMM      <dbl> 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,...
$ LoyalCH        <dbl> 0.500000, 0.600000, 0.680000, 0.400000, 0.95653...
$ SalePriceMM    <dbl> 1.99, 1.69, 2.09, 1.69, 1.69, 1.99, 1.59, 1.59,...
$ SalePriceCH    <dbl> 1.75, 1.75, 1.69, 1.69, 1.69, 1.69, 1.69, 1.75,...
$ PriceDiff      <dbl> 0.24, -0.06, 0.40, 0.00, 0.00, 0.30, -0.10, -0....
$ Store7         <chr> "No", "No", "No", "No", "Yes", "Yes", "Yes", "Y...
$ PctDiscMM      <dbl> 0.000000, 0.150754, 0.000000, 0.000000, 0.00000...
$ PctDiscCH      <dbl> 0.000000, 0.000000, 0.091398, 0.000000, 0.00000...
$ ListPriceDiff  <dbl> 0.24, 0.24, 0.23, 0.00, 0.00, 0.30, 0.30, 0.24,...
$ STORE          <dbl> 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
```

```r
# See top 6 rows and 10 columns
head(orange[, 1:10])
```

```
# A tibble: 6 x 10
  Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
  <chr>             <dbl>   <dbl>   <dbl>   <dbl>  <dbl>  <dbl>     <dbl>
1 CH                  237       1    1.75    1.99   0       0           0
2 CH                  239       1    1.75    1.99   0       0.3         0
3 CH                  245       1    1.86    2.09   0.17    0           0
4 MM                  227       1    1.69    1.69   0       0           0
5 CH                  228       7    1.69    1.69   0       0           0
6 CH                  230       7    1.69    1.99   0       0           0
# ... with 2 more variables: SpecialMM <dbl>, LoyalCH <dbl>
```

17.3 3. Data Preparation and Preprocessing

17.3.1 3.1. How to split the dataset into training and validation?

The dataset is ready.

The first step is to split it into training(80%) and test(20%) datasets using caret's `createDataPartition` function.

The advantage of using `createDataPartition()` over the traditional random `sample()` is, it preserves the proportion of the categories in `Y` variable, that can be disturbed if you sample randomly.

Let me quickly refresh why are splitting the dataset into training and test data.

The reason is when building models the algorithm should see only the training data to learn the relationship between `X` and `Y`.

This learned information forms what is called a machine learning model.

The model is then used to predict the Y in test data by looking only the `X` values of test data. Finally, the predicted values of `Y` is compared to the known `Y` from test dataset to evaluate how good the model really is.

Alright, let's create the training and test datasets.

```
# Create the training and test datasets
set.seed(100)

# Step 1: Get row numbers for the training data
trainRowNumbers <- createDataPartition(orange$Purchase, p=0.8, list=FALSE)

# Step 2: Create the training  dataset
trainData <- orange[trainRowNumbers,]

# Step 3: Create the test dataset
testData <- orange[-trainRowNumbers,]

# Store X and Y for later use.
x = trainData[, 2:18]
y = trainData$Purchase
```

`createDataPartition()` takes as input the `Y` variable in the source dataset and the percentage data that should go into training as the p argument. It returns the row numbers that should form the training dataset.

Plus, you need to set `list=F`, to prevent returning the result as a list.

17.3.2 3.2. Descriptive statistics

Before moving to missing value imputation and feature preprocessing, let's observe the descriptive statistics of each column in the training dataset.

The `skimr` package provides a nice solution to show key descriptive stats for each column.

The `skimr::skim_to_wide()` produces a nice dataframe containing the descriptive stats of each of the columns. The dataframe output includes a nice histogram drawn without any plotting help.

```
library(skimr)
skimmed <- skim_to_wide(trainData)
skimmed[, c(1:5, 9:11, 13, 15:16)]
```

```
# A tibble: 18 x 11
   type  variable missing complete n     n_unique mean  sd    p25   p75
   <chr> <chr>    <chr>   <chr>    <chr> <chr>    <chr> <chr> <chr> <chr>
 1 char~ Purchase 0       857      857   2        <NA>  <NA>  <NA>  <NA>
 2 char~ Store7   0       857      857   2        <NA>  <NA>  <NA>  <NA>
 3 nume~ DiscCH   2       855      857   <NA>     "  0~ " 0.~ "  0~ "  0~
 4 nume~ DiscMM   3       854      857   <NA>     "  0~ " 0.~ "  0~ "  0~
 5 nume~ ListPri~ 0       857      857   <NA>     "  0~ " 0.~ "  0~ "  0~
 6 nume~ LoyalCH  5       852      857   <NA>     "  0~ " 0.~ "  0~ "  0~
 7 nume~ PctDisc~ 2       855      857   <NA>     "  0~ " 0.~ "  0~ "  0~
 8 nume~ PctDisc~ 2       855      857   <NA>     "  0~ " 0.~ "  0~ "  0~
 9 nume~ PriceCH  1       856      857   <NA>     "  1~ " 0.~ "  1~ "  1~
10 nume~ PriceDi~ 1       856      857   <NA>     "  0~ " 0.~ "  0~ "  0~
11 nume~ PriceMM  1       856      857   <NA>     "  2~ " 0.~ "  1~ "  2~
```

```
12 nume~ SalePri~ 1       856      857   <NA>     "  1~ " 0.~ "  1~ "  1~
13 nume~ SalePri~ 3       854      857   <NA>     "  1~ " 0.~ "  1~ "  2~
14 nume~ Special~ 2       855      857   <NA>     "  0~ " 0.~ "  0~ "  0~
15 nume~ Special~ 4       853      857   <NA>     "  0~ " 0.~ "  0~ "  0~
16 nume~ STORE    2       855      857   <NA>     "  1~ " 1.~ "  0~ "  3~
17 nume~ StoreID  1       856      857   <NA>     "  3~ " 2.~ "  2~ "  7~
18 nume~ WeekofP~ 0       857      857   <NA>     "254~ "15.~ "239~ "268~
# ... with 1 more variable: p100 <chr>
```

Notice the number of missing values for each feature, mean, median, proportion split of categories in the factor variables, percentiles and the histogram in the last column.

We will examine how important each of these features is in predicting the **response** (Purchase) in section 4, once we are done with the data preprocessing.

17.3.3 3.3. How to impute missing values using preProcess()?

We've seen that the dataset has few missing values across all columns, we may to do well to impute it. Impute, means to fill it up with some meaningful values.

If the feature is a continuous variable, it is a common practice to replace the missing values with the mean of the column. And if it's a categorical variable, replace the missings with the most frequently occurring value, aka, the mode.

But this is quite a basic and a rather rudimentary approach.

Instead what can be done is, you can actually predict the missing values by considering the rest of the available variables as predictors. A popular algorithm to do imputation is the **k-Nearest Neighbors**. This can be quickly and easily be done using caret.

Because, caret offers a nice convenient preProcess function that can predict missing values besides other preprocessing.

To predict the missing values with k-Nearest Neighbors using `preProcess()`:

You need to set the `method=knnImpute` for k-Nearest Neighbors and apply it on the training data. This creates a preprocess model. Then use `predict()` on the created preprocess model by setting the newdata argument on the same training data.

Caret also provides `bagImpute` as an alternative imputation algorithm.

```r
# Create the knn imputation model on the training data
preProcess_missingdata_model <- preProcess(trainData, method='knnImpute')
preProcess_missingdata_model
```

```
Created from 828 samples and 18 variables

Pre-processing:
  - centered (16)
  - ignored (2)
  - 5 nearest neighbor imputation (16)
  - scaled (16)
```

The above output shows the various preprocessing steps done in the process of knn imputation.

That is, it has centered (subtract by mean) 16 variables, ignored 2, used `k=5` (considered 5 nearest neighbors) to predict the missing values and finally scaled (divide by standard deviation) 16 variables.

Let's now use this model to predict the missing values in `trainData`.

```r
# Use the imputation model to predict the values of missing data points
library(RANN)  # required for knnInpute

trainData <- predict(preProcess_missingdata_model, newdata = trainData)
anyNA(trainData)
```

```
[1] FALSE
```

All the missing values are successfully imputed.

```r
# confirmed no more missing data
skimmed <- skim_to_wide(trainData)
skimmed[, c(1:5, 9:11, 13, 15:16)]
```

```
# A tibble: 18 x 11
   type  variable missing complete n     n_unique mean  sd    p25   p75
   <chr> <chr>    <chr>   <chr>    <chr> <chr>    <chr> <chr> <chr> <chr>
 1 char~ Purchase 0       857      857   2        <NA>  <NA>  <NA>  <NA>
 2 char~ Store7   0       857      857   2        <NA>  <NA>  <NA>  <NA>
 3 nume~ DiscCH   0       857      857   <NA>     "   ~ 1     -0.45 -0.45
 4 nume~ DiscMM   0       857      857   <NA>     "   ~ 1     -0.58 " 0.~
 5 nume~ ListPri~ 0       857      857   <NA>     "   ~ 1     "-0.~ " 0.~
 6 nume~ LoyalCH  0       857      857   <NA>     "   ~ 1     -0.78 " 0.~
 7 nume~ PctDisc~ 0       857      857   <NA>     "   ~ 1     -0.45 -0.45
 8 nume~ PctDisc~ 0       857      857   <NA>     "   ~ 1     -0.59 " 0.~
 9 nume~ PriceCH  0       857      857   <NA>     "-3e~ 1     -0.75 " 1.~
10 nume~ PriceDi~ 0       857      857   <NA>     "   ~ 1     -0.56 " 0.~
11 nume~ PriceMM  0       857      857   <NA>     "   ~ 1     -0.69 " 0.~
12 nume~ SalePri~ 0       857      857   <NA>     "   ~ 1     -0.43 " 0.~
13 nume~ SalePri~ 0       857      857   <NA>     "   ~ 1     "-1.~ " 0.~
14 nume~ Special~ 0       857      857   <NA>     "   ~ 1     -0.43 -0.43
15 nume~ Special~ 0       857      857   <NA>     "   ~ 1     -0.42 -0.42
16 nume~ STORE    0       857      857   <NA>     "   ~ 1     -1.15 " 0.~
17 nume~ StoreID  0       857      857   <NA>     "   ~ 1     -0.85 " 1.~
18 nume~ WeekofP~ 0       857      857   <NA>     "   ~ 1     -0.97 " 0.~
# ... with 1 more variable: p100 <chr>
```

We confirm that there no more missing rows.

17.3.4 3.4. How to create One-Hot Encoding (dummy variables)?

Let me first explain what is one-hot encoding and why it is required.

Suppose if you have a categorical column as one of the features, it needs to be converted to numeric in order for it to be used by the machine learning algorithms.

Just replacing the categories with a number may not be meaningful especially if there is no intrinsic ordering amongst the categories.

So what you can do instead is to convert the categorical variable with as many binary (1 or 0) variables as there are categories.

An important aspect you should be careful about here is, in real-world environments, you might get new values of categorical variables in the new scoring data. So, you should ensure the `dummyVars` model is built on the training data alone and that model is in turn used to create the dummy vars on the test data.

In `caret`, one-hot-encodings can be created using `dummyVars()`. Just pass in all the features to `dummyVars()` as the training data and all the factor columns will automatically be converted to one-hot-encodings.

```
# One-Hot Encoding
# Creating dummy variables is converting a categorical variable to as many binary variables as here are
dummies_model <- dummyVars(Purchase ~ ., data = trainData)

# Create the dummy variables using predict.
# The Y variable (Purchase) will not be present in trainData_mat.
trainData_mat <- predict(dummies_model, newdata = trainData)

# # Convert to dataframe
trainData <- data.frame(trainData_mat)

# # See the structure of the new dataset
str(trainData)
```

```
'data.frame':   857 obs. of  18 variables:
 $ WeekofPurchase: num  -1.1 -1.74 -1.68 -1.29 -1.04 ...
 $ StoreID       : num  -1.29 -1.29 1.33 1.33 1.33 ...
 $ PriceCH       : num  -1.14 -1.73 -1.73 -1.14 -1.14 ...
 $ PriceMM       : num  -0.688 -2.898 -2.898 -0.688 -0.688 ...
 $ DiscCH        : num  -0.452 -0.452 -0.452 -0.452 -0.452 ...
 $ DiscMM        : num  -0.582 -0.582 -0.582 1.341 1.341 ...
 $ SpecialCH     : num  -0.429 -0.429 -0.429 2.329 -0.429 ...
 $ SpecialMM     : num  -0.42 -0.42 -0.42 -0.42 -0.42 ...
 $ LoyalCH       : num  -0.205 -0.525 1.256 1.324 1.35 ...
 $ SalePriceMM   : num  0.113 -1.101 -1.101 -1.506 -1.506 ...
 $ SalePriceCH   : num  -0.431 -0.844 -0.844 -0.431 -0.431 ...
 $ PriceDiff     : num  0.341 -0.563 -0.563 -1.165 -1.165 ...
 $ Store7No      : num  1 1 0 0 0 0 0 0 0 1 ...
 $ Store7Yes     : num  0 0 1 1 1 1 1 1 1 0 ...
 $ PctDiscMM     : num  -0.588 -0.588 -0.588 1.447 1.447 ...
 $ PctDiscCH     : num  -0.448 -0.448 -0.448 -0.448 -0.448 ...
 $ ListPriceDiff : num  0.211 -1.988 -1.988 0.211 0.211 ...
 $ STORE         : num  -0.457 -0.457 -1.15 -1.15 -1.15 ...
```

In the above case, we had one categorical variable, `Store7` with 2 categories. It was one-hot-encoded to produce two new columns – `Store7.No` and `Store7.Yes`.

17.3.5 3.5. How to preprocess to transform the data?

With the missing values handled and the factors one-hot-encoded, our training dataset is now ready to undergo variable transformations if required.

So what type of preprocessing are available in caret?

- **range**: Normalize values so it ranges between 0 and 1
- **center**: Subtract Mean
- **scale**: Divide by standard deviation
- **BoxCox**: Remove skewness leading to normality. Values must be > 0
- **YeoJohnson**: Like BoxCox, but works for negative values.
- **expoTrans**: Exponential transformation, works for negative values.
- **pca**: Replace with principal components
- **ica**: Replace with independent components

- **spatialSign**: Project the data to a unit circle

For our problem, let's convert all the numeric variables to range between 0 and 1, by setting `method=range` in `preProcess()`.

```
tic()
preProcess_range_model <- preProcess(trainData, method = 'range')
trainData <- predict(preProcess_range_model, newdata = trainData)

# Append the Y variable
trainData$Purchase <- y

apply(trainData[, 1:10], 2, FUN =
          function(x) {
              c('min'= min(x), 'max' = max(x)) })
```

```
    WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
min              0       0       0       0      0      0         0
max              1       1       1       1      1      1         1
    SpecialMM LoyalCH SalePriceMM
min         0       0           0
max         1       1           1
```

```
toc()
```

```
0.059 sec elapsed
```

All the predictor now range between 0 and 1.

17.4 4. How to visualize the importance of variables using featurePlot()

Now that the preprocessing is complete, let's visually examine how the predictors influence the `Y` (Purchase).

In this problem, the `X` variables are numeric whereas the `Y` is categorical.

So how to gauge if a given `X` is an important predictor of `Y`?

A simple common sense approach is, if you group the X variable by the categories of Y, a significant mean shift amongst the X's groups is a strong indicator (if not the only indicator) that X will have a significant role to help predict Y.

It is possible to watch this shift visually using box plots and density plots.

In fact, caret's `featurePlot()` function makes it so convenient.

Simply set the X and Y parameters and set `plot='box'`. You can additionally adjust the label font size (using `strip`) and the `scales` to be free as I have done in the below plot.

This plot initially was of the `box` type, but it is now giving out a `NULL` value.

```
x = trainData[, 1:18]
y = trainData$Purchase

trellis.par.set(theme = col.whitebg(), warn = FALSE)
featurePlot(x, y, "ellipse")
```

```
NULL
```

```
featurePlot(x, y, "strip", jitter = TRUE)
```

```
NULL
```

```
featurePlot(x, y, "box")
```

```
NULL
```

```
featurePlot(x, y, "pairs")
```

```
featurePlot(x = trainData[, 1:18],
            y = trainData$Purchase,
            plot = "box",
            strip=strip.custom(par.strip.text=list(cex=.7)),
            scales = list(x = list(relation="free"),
                          y = list(relation="free")))
```

```
NULL
```

Let me quickly refresh how to interpret a boxplot.

Each subplot in the above figure has two boxplots (in blue) inside it, one each for each of the Y categories, CH and MM. The top of the box represents the 25th %ile and the bottom of the box represents the 75th %ile. The black dot inside the box is the mean.

The blue box represents the region where most of the regular data point lie.

The subplots also show many blue dots lying outside the top and bottom dashed lines called whiskers. These dots are formally considered as extreme values.

So, What did you observe in the above figure?

Consider for example, `LoyalCHs` subplot, which measures the loyalty score of the customer to the CH brand. The mean and the placement of the two boxes are glaringly different.

Just by seeing that, I am pretty sure, `LoyalCH` is going to be a significant predictor of Y.

What other predictors do you notice have significant mean differences?

Let's do a similar exercise with density plots.

In this case, for a variable to be important, I would expect the density curves to be significantly different for the 2 classes, both in terms of the height (`kurtosis`) and placement (`skewness`).

Take a look at the density curves of the two categories for '`LoyalCH`', '`STORE`', '`StoreID`', '`WeekofPurchase`'. Are they different?

```
featurePlot(x = trainData[, 1:18],
            y = trainData$Purchase,
            plot = "density",
            strip=strip.custom(par.strip.text=list(cex=.7)),
            scales = list(x = list(relation="free"),
                          y = list(relation="free")))
```

```
NULL
```

Having visualised the relationships between X and Y, We can only say which variables are likely to be important to predict Y. It may not be wise to conclude which variables are NOT important.

Because sometimes, variables with uninteresting pattern can help explain certain aspects of Y that the visually important variables may not.

So to be safe, let's not arrive at conclusions about excluding variables prematurely.

17.5 5. How to do feature selection using recursive feature elimination (RFE)?

Most machine learning algorithms are able to determine what features are important to predict the Y. But in some scenarios, you might be need to be careful to include only variables that may be significantly important and makes strong business sense.

This is quite common in banking, economics and financial institutions.

Or you might just be doing an exploratory analysis to determine important predictors and report it as a metric in your analytics dashboard.

Or if you are using a traditional algorithm like like linear or logistic regression, determining what variable to feed to the model is in the hands of the practitioner.

Given such requirements, you might need a rigorous way to determine the important variables first before feeding them to the ML algorithm.

A good choice of selecting the important features is the recursive feature elimination (RFE).

So how does recursive feature elimination work?

RFE works in 3 broad steps:

Step 1: Build a ML model on a training dataset and estimate the feature importances on the test dataset.

Step 2: Keeping priority to the most important variables, iterate through by building models of given subset sizes, that is, subgroups of most important predictors determined from step 1. Ranking of the predictors is recalculated in each iteration.

Step 3: The model performances are compared across different subset sizes to arrive at the optimal number and list of final predictors.

It can be implemented using the rfe() function and you have the flexibility to control what algorithm rfe uses and how it cross validates by defining the `rfeControl()`.

```
set.seed(100)
options(warn=-1)

subsets <- c(1:5, 10, 15, 18)

ctrl <- rfeControl(functions = rfFuncs,
                   method = "repeatedcv",
                   repeats = 5,
                   verbose = FALSE)

lmProfile <- rfe(x=trainData[, 1:18], y=trainData$Purchase,
                 sizes = subsets,
                 rfeControl = ctrl)
```

```
Error in {: task 1 failed - "non-numeric argument to binary operator"
```

```
lmProfile
```

```
Error in eval(expr, envir, enclos): object 'lmProfile' not found
```

In the above code, we call the `rfe()` which implements the recursive feature elimination.

Apart from the `x` and `y` datasets, RFE also takes two important parameters.

- **sizes**
- **rfeControl**

The `sizes` determines what all model sizes (the number of most important features) the `rfe` should consider. In above case, it iterates models of size 1 to 5, 10, 15 and 18.

The `rfeControl` parameter on the other hand receives the output of the `rfeControl()` as values. If you look at the call to `rfeControl()` we set what type of algorithm and what cross validation method should be used.

In above case, the cross validation method is `repeatedcv` which implements k-Fold cross validation repeated 5 times, which is rigorous enough for our case.

Once `rfe()` is run, the output shows the `accuracy` and `kappa` (and their standard deviation) for the different model sizes we provided. The final selected model subset size is marked with a * in the rightmost Selected column.

From the above output, a model size of 3 with `LoyalCH`, `PriceDiff` and `StoreID` seems to achieve the optimal accuracy.

That means, out of 18 other features, a model with just 3 features outperformed many other larger model. Interesting isn't it! Can you explain why?

However, it is not a mandate that only including these 3 variables will always give high accuracy over larger sized models.

Thats because, the `rfe()` we just implemented is particular to random forest based rfFuncs

Since ML algorithms have their own way of learning the relationship between the x and y, it is not wise to neglect the other predictors, especially when there is evidence that there is information contained in rest of the variables to explain the relationship between `x` and `y`.

Plus also, since the training dataset isn't large enough, the other predictors may not have had the chance to show its worth.

In the next step, we will build the actual randomForest model on `trainData`.

17.6 6. Training and Tuning the model

17.6.1 6.1. How to train the model and interpret the results?

Now comes the important stage where you actually build the machine learning model.

To know what models caret supports, run the following:

```
# See available algorithms in caret
modelnames <- paste(names(getModelInfo()), collapse=',  ')
modelnames
```

```
[1] "ada,  AdaBag,  AdaBoost.M1,  adaboost,  amdai,  ANFIS,  avNNet,  awnb,  awtan,  bag,  bagEarth,  ba
```

Each of those is a machine learning algorithm `caret` supports.

Yes, it's a huge list!

And if you want to know more details like the hyperparameters and if it can be used of regression or classification problem, then do a`modelLookup(algo)`.

Once you have chosen an algorithm, building the model is fairly easy using the `train()` function.

Let's train a Multivariate Adaptive Regression Splines (MARS) model by setting the `method='earth'`.

The MARS algorithm was named as 'earth' in R because of a possible trademark conflict with Salford Systems. May be a rumor. Or not.

```
modelLookup('earth')
```

```
  model parameter          label forReg forClass probModel
1 earth    nprune         #Terms   TRUE     TRUE      TRUE
2 earth    degree Product Degree   TRUE     TRUE      TRUE
```

```
# Set the seed for reproducibility
set.seed(100)

# Train the model using randomForest and predict on the training data itself.
model_mars = train(Purchase ~ .,
                   data = trainData,
                   method = 'earth')

fitted <- predict(model_mars)
```

But you may ask **how is using train() different from using the algorithm's function directly?**

The difference is, besides building the model `train()` does multiple other things like:

1. Cross validating the model
2. Tune the hyper parameters for optimal model performance
3. Choose the optimal model based on a given evaluation metric
4. Preprocess the predictors (what we did so far using preProcess()) The train function also accepts the arguments used by the algorithm specified in the method argument.

Now let's see what the `train()` has generated.

```
model_mars
```

```
Multivariate Adaptive Regression Spline

857 samples
 18 predictor
  2 classes: 'CH', 'MM'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 857, 857, 857, 857, 857, 857, ...
Resampling results across tuning parameters:

  nprune  Accuracy   Kappa
   2      0.8013184  0.5746285
  10      0.8102610  0.5987447
  19      0.8103685  0.5986923

Tuning parameter 'degree' was held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were nprune = 19 and degree = 1.
```

You can see what is the Accuracy and Kappa for various combinations of the hyper parameters – `interaction.depth` and `n.trees`. And it says 'Resampling: Bootstrapped (25 reps)' with a summary of sample sizes.

Looks like `train()` has already done a basic cross validation and hyper parameter tuning. And that is the default behaviour.

The chosen model and its parameters is reported in the last 2 lines of the output.

When we used `model_mars` to predict the Y, this final model was automatically used by `predict()` to compute the predictions.

Plotting the model shows how the various iterations of hyperparameter search performed.

```
plot(model_mars, main="Model Accuracies with MARS")
```

17.6.2 6.2 How to compute variable importance?

Excellent, since MARS supports computing variable importances, let's extract the variable importances using `varImp()` to understand which variables came out to be useful.

```
varimp_mars <- varImp(model_mars)
plot(varimp_mars, main="Variable Importance with MARS")
```

As suspected, LoyalCH was the most used variable, followed by PriceDiff and StoreID.

17.6.3 6.3. Prepare the test dataset and predict

A default MARS model has been selected.

Now in order to use the model to predict on new data, the data has to be preprocessed and transformed just the way we did on the training data.

Thanks to `caret`, all the information required for pre-processing is stored in the respective `preProcess` model and `dummyVar` model.

If you recall, we did the pre-processing in the following sequence:

```
Missing Value imputation -> One-Hot Encoding -> Range Normalization
```

You need to pass the `testData` through these models in the same sequence:

```
preProcess_missingdata_model -> dummies_model -> preProcess_range_model
```

```r
# Step 1: Impute missing values
testData2 <- predict(preProcess_missingdata_model, testData)

# Step 2: Create one-hot encodings (dummy variables)
testData3 <- predict(dummies_model, testData2)

# Step 3: Transform the features to range between 0 and 1
testData4 <- predict(preProcess_range_model, testData3)
```

```r
# View
head(testData4[, 1:10])
```

```
  WeekofPurchase StoreID PriceCH   PriceMM DiscCH DiscMM SpecialCH
1     0.23529412       0   0.150 0.5000000   0.00  0.375         0
2     0.35294118       0   0.425 0.6666667   0.34  0.000         0
3     0.05882353       1   0.000 0.5000000   0.00  0.000         0
4     0.09803922       1   0.000 0.5000000   0.00  0.500         1
5     0.15686275       1   0.150 0.5000000   0.00  0.500         0
6     0.96078431       1   0.750 0.7333333   0.00  0.675         0
  SpecialMM   LoyalCH SalePriceMM
1         1 0.6000352   0.4545455
2         0 0.6800414   0.8181818
3         1 0.9652913   0.7272727
4         1 0.9722459   0.3636364
5         0 0.9822616   0.3636364
6         1 0.9927734   0.3636364
```

17.6.4 6.4. Predict on testData

The test dataset is prepared. Let's predict the Y.

```r
# Predict on testData
predicted <- predict(model_mars, testData4)
predicted
```

```
  [1] CH CH CH CH CH MM CH CH CH CH CH CH MM CH CH CH CH CH CH CH MM CH CH
 [24] CH CH CH CH CH CH CH CH CH MM CH CH CH CH CH CH CH CH CH CH CH CH MM
 [47] MM MM CH CH MM MM CH CH CH CH MM CH MM MM MM MM MM MM MM MM CH CH
 [70] CH MM CH MM CH MM CH MM MM MM MM MM MM MM CH MM CH MM MM CH MM MM CH
 [93] CH CH CH MM MM CH CH MM MM MM CH CH MM CH CH CH MM MM MM CH CH CH CH
[116] MM MM CH CH CH CH CH CH CH CH CH CH CH CH CH CH CH CH CH CH CH MM MM CH
[139] MM MM MM MM MM MM CH MM MM MM MM MM MM CH CH CH MM MM MM MM CH CH CH
[162] CH MM MM CH CH CH CH CH CH CH CH CH CH CH CH CH MM CH CH MM CH MM MM
[185] MM CH CH MM MM MM CH CH CH MM MM MM MM MM CH CH CH CH CH CH CH CH CH
[208] CH CH CH MM CH CH
Levels: CH MM
```

17.6.5 6.5. Confusion Matrix

The confusion matrix is a tabular representation to compare the predictions (data) vs the actuals (reference). By setting `mode='everything` pretty much most classification evaluation metrics are computed.

```r
# Compute the confusion matrix
confusionMatrix(reference = testData$Purchase,
                data = predicted,
                mode='everything', positive = 'MM')
```

```
Error: `data` and `reference` should be factors with the same levels.
```

You have an overall accuracy of 82%.

17.7 7. How to do hyperparameter tuning to optimize the model for better performance?

There are two main ways to do hyper parameter tuning using the `train()` function:

1. Set the `tuneLength`
2. Define and set the `tuneGrid`

`tuneLength` corresponds to the number of unique values for the tuning parameters `caret` will consider while forming the hyper parameter combinations.

`caret` will automatically determine the values each parameter should take.

Alternately, if you want to explicitly control what values should be considered for each parameter, then, you can define the `tuneGrid` and pass it to `train()`.

Let's see an example of both these approaches but first let's setup the `trainControl()`.

17.7.1 7.1. Setting up the `trainControl()`

The `train()` function takes a `trControl` argument that accepts the output of `trainControl()`.

Inside `trainControl()` you can control how the `train()` will:

1, Cross validation method to use. 2. How the results should be summarised using a summary function

Cross validation `method` can be one amongst:

- 'boot': Bootstrap sampling
- 'boot632': Bootstrap sampling with 63.2% bias correction applied
- 'optimism_boot': The optimism bootstrap estimator
- 'boot_all': All boot methods.
- 'cv': k-Fold cross validation
- 'repeatedcv': Repeated k-Fold cross validation
- 'oob': Out of Bag cross validation
- 'LOOCV': Leave one out cross validation
- 'LGOCV': Leave group out cross validation

The `summaryFunction` can be `twoClassSummary` if Y is binary class or `multiClassSummary` if the Y has more than 2 categories.

By setting the `classProbs=T` the probability scores are generated instead of directly predicting the class based on a predetermined cutoff of 0.5.

```
# Define the training control
fitControl <- trainControl(
    method = 'cv',                   # k-fold cross validation
    number = 5,                      # number of folds
    savePredictions = 'final',       # saves predicts. optimum tuning parameter
    classProbs = T,                  # should class probabilities be returned
    summaryFunction = twoClassSummary  # results summary function
)
```

17.7.2 7.2 Hyper Parameter Tuning using `tuneLength`

Let's take the `train()` function we used before, plus, additionally set the `tuneLength`, `trControl` and `metric`.

```
# Step 1: Tune hyper parameters by setting tuneLength
set.seed(100)
model_mars2 = train(Purchase ~ .,
                    data=trainData,
                    method='earth',
                    tuneLength = 5,
                    metric='ROC',
                    trControl = fitControl)
model_mars2
```

```
Multivariate Adaptive Regression Spline

857 samples
 18 predictor
  2 classes: 'CH', 'MM'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 685, 685, 687, 686, 685
Resampling results across tuning parameters:

  nprune  ROC        Sens       Spec
   2      0.8745398  0.8700916  0.7006784
   6      0.8912361  0.8719414  0.7334238
  10      0.8879988  0.8623626  0.7423790
  14      0.8879988  0.8623626  0.7423790
  19      0.8879846  0.8661722  0.7483492

Tuning parameter 'degree' was held constant at a value of 1
ROC was used to select the optimal model using the largest value.
The final values used for the model were nprune = 6 and degree = 1.
```

```
# Step 2: Predict on testData and Compute the confusion matrix
predicted2 <- predict(model_mars2, testData4)
confusionMatrix(reference = testData$Purchase,
                data = predicted2,
                mode='everything', positive='MM')
```

```
Error: `data` and `reference` should be factors with the same levels.
```

17.7.3 7.3. Hyper Parameter Tuning using `tuneGrid`

Alternately, you can set the `tuneGrid` instead of `tuneLength`.

```
# Step 1: Define the tuneGrid
marsGrid <-  expand.grid(nprune = c(2, 4, 6, 8, 10),
                         degree = c(1, 2, 3))

# Step 2: Tune hyper parameters by setting tuneGrid
set.seed(100)
model_mars3 = train(Purchase ~ .,
                    data = trainData,
                    method = 'earth',
                    metric = 'ROC',
```

```
                     tuneGrid = marsGrid,
                     trControl = fitControl)
model_mars3
```

```
Multivariate Adaptive Regression Spline

857 samples
 18 predictor
  2 classes: 'CH', 'MM'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 685, 685, 687, 686, 685
Resampling results across tuning parameters:

  degree  nprune  ROC        Sens       Spec
  1        2      0.8745398  0.8700916  0.7006784
  1        4      0.8924657  0.8662454  0.7394844
  1        6      0.8912361  0.8719414  0.7334238
  1        8      0.8886974  0.8661722  0.7334238
  1       10      0.8879988  0.8623626  0.7423790
  2        2      0.8745398  0.8700916  0.7006784
  2        4      0.8925044  0.8758425  0.7395296
  2        6      0.8918962  0.8681868  0.7545002
  2        8      0.8911809  0.8624359  0.7634102
  2       10      0.8947444  0.8604579  0.7784261
  3        2      0.8582783  0.8777106  0.6618725
  3        4      0.8919687  0.8681502  0.7514699
  3        6      0.8889365  0.8509341  0.7665310
  3        8      0.8802930  0.8432784  0.7396653
  3       10      0.8790021  0.8452015  0.7486205

ROC was used to select the optimal model using the largest value.
The final values used for the model were nprune = 10 and degree = 2.
```

```
# Step 3: Predict on testData and Compute the confusion matrix
predicted3 <- predict(model_mars3, testData4)
confusionMatrix(reference = testData$Purchase,
                data = predicted3,
                mode ='everything',
                positive = 'MM')
```

```
Error: `data` and `reference` should be factors with the same levels.
```

17.8 8. How to evaluate performance of multiple machine learning algorithms?

Caret provides the `resamples()` function where you can provide multiple machine learning models and collectively evaluate them.

Let's first train some more algorithms.

17.8.1 8.1. Training Adaboost

```
set.seed(100)

# Train the model using adaboost
model_adaboost = train(Purchase ~ ., data = trainData,
                       method = 'adaboost',
                       tuneLength = 2,
                       trControl = fitControl)
model_adaboost
```

```
AdaBoost Classification Trees

857 samples
 18 predictor
  2 classes: 'CH', 'MM'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 685, 685, 687, 686, 685
Resampling results across tuning parameters:

  nIter  method         ROC        Sens       Spec
   50    Adaboost.M1    0.8657598  0.8070330  0.7635007
   50    Real adaboost  0.6989310  0.8337363  0.7395749
  100    Adaboost.M1    0.8638447  0.8127839  0.7515604
  100    Real adaboost  0.6978488  0.8432784  0.7245590

ROC was used to select the optimal model using the largest value.
The final values used for the model were nIter = 50 and method
 = Adaboost.M1.
```

17.8.2 8.2. Training Random Forest

```
set.seed(100)

# Train the model using rf
model_rf = train(Purchase ~ ., data = trainData,
                 method = 'rf',
                 tuneLength = 5,
                 trControl = fitControl)
model_rf
```

```
Random Forest

857 samples
 18 predictor
  2 classes: 'CH', 'MM'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 685, 685, 687, 686, 685
Resampling results across tuning parameters:
```

```
  mtry  ROC        Sens       Spec
   2    0.8693670  0.8834249  0.6496156
   6    0.8771441  0.8624359  0.7335595
  10    0.8775297  0.8490476  0.7365445
  14    0.8765595  0.8490659  0.7305292
  18    0.8752593  0.8452198  0.7425147

ROC was used to select the optimal model using the largest value.
The final value used for the model was mtry = 10.
```

17.8.3 8.3. Training xgBoost Dart

Ref.: https://xgboost.readthedocs.io/en/latest/tutorials/dart.html Kaggle: http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/ Papers: http://proceedings.mlr.press/v38/korlakaivinayak15.pdf https://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf. XGBoost: A Scalable Tree Boosting System

> *"We propose a novel way of employing dropouts in MART, resulting in the DART algorithm. We evaluate DART on ranking, regression and classification tasks, using large scale, publicly available datasets, and show that DART outperforms MART in each of the tasks, with a significant margin. We also show that DART overcomes the issue of over-specialization to a considerable extent."*

```r
tic()
set.seed(100)

# Train the model using MARS
model_xgbDART = train(Purchase ~ .,
                      data = trainData,
                      method = 'xgbDART',
                      tuneLength=5,
                      trControl = fitControl,
                      verbose = F)
toc()
```

```
777.271 sec elapsed
```

```r
model_xgbDART
```

```
eXtreme Gradient Boosting

857 samples
 18 predictor
  2 classes: 'CH', 'MM'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 685, 685, 687, 686, 685
Resampling results across tuning parameters:

  max_depth  eta  rate_drop  skip_drop  subsample  colsample_bytree
  1          0.3  0.01       0.05       0.500      0.6
  1          0.3  0.01       0.05       0.500      0.6
  1          0.3  0.01       0.05       0.500      0.6
  1          0.3  0.01       0.05       0.500      0.6
```

1	0.3	0.01	0.05	0.500	0.6
1	0.3	0.01	0.05	0.500	0.8
1	0.3	0.01	0.05	0.500	0.8
1	0.3	0.01	0.05	0.500	0.8
1	0.3	0.01	0.05	0.500	0.8
1	0.3	0.01	0.05	0.500	0.8
1	0.3	0.01	0.05	0.625	0.6
1	0.3	0.01	0.05	0.625	0.6
1	0.3	0.01	0.05	0.625	0.6
1	0.3	0.01	0.05	0.625	0.6
1	0.3	0.01	0.05	0.625	0.6
1	0.3	0.01	0.05	0.625	0.8
1	0.3	0.01	0.05	0.625	0.8
1	0.3	0.01	0.05	0.625	0.8
1	0.3	0.01	0.05	0.625	0.8
1	0.3	0.01	0.05	0.625	0.8
1	0.3	0.01	0.05	0.750	0.6
1	0.3	0.01	0.05	0.750	0.6
1	0.3	0.01	0.05	0.750	0.6
1	0.3	0.01	0.05	0.750	0.6
1	0.3	0.01	0.05	0.750	0.6
1	0.3	0.01	0.05	0.750	0.8
1	0.3	0.01	0.05	0.750	0.8
1	0.3	0.01	0.05	0.750	0.8
1	0.3	0.01	0.05	0.750	0.8
1	0.3	0.01	0.05	0.750	0.8
1	0.3	0.01	0.05	0.875	0.6
1	0.3	0.01	0.05	0.875	0.6
1	0.3	0.01	0.05	0.875	0.6
1	0.3	0.01	0.05	0.875	0.6
1	0.3	0.01	0.05	0.875	0.6
1	0.3	0.01	0.05	0.875	0.8
1	0.3	0.01	0.05	0.875	0.8
1	0.3	0.01	0.05	0.875	0.8
1	0.3	0.01	0.05	0.875	0.8
1	0.3	0.01	0.05	0.875	0.8
1	0.3	0.01	0.05	1.000	0.6
1	0.3	0.01	0.05	1.000	0.6
1	0.3	0.01	0.05	1.000	0.6
1	0.3	0.01	0.05	1.000	0.6
1	0.3	0.01	0.05	1.000	0.6
1	0.3	0.01	0.05	1.000	0.8
1	0.3	0.01	0.05	1.000	0.8
1	0.3	0.01	0.05	1.000	0.8
1	0.3	0.01	0.05	1.000	0.8
1	0.3	0.01	0.05	1.000	0.8
1	0.3	0.01	0.95	0.500	0.6
1	0.3	0.01	0.95	0.500	0.6
1	0.3	0.01	0.95	0.500	0.6
1	0.3	0.01	0.95	0.500	0.6
1	0.3	0.01	0.95	0.500	0.6
1	0.3	0.01	0.95	0.500	0.8
1	0.3	0.01	0.95	0.500	0.8
1	0.3	0.01	0.95	0.500	0.8

1	0.3	0.01	0.95	0.500	0.8
1	0.3	0.01	0.95	0.500	0.8
1	0.3	0.01	0.95	0.625	0.6
1	0.3	0.01	0.95	0.625	0.6
1	0.3	0.01	0.95	0.625	0.6
1	0.3	0.01	0.95	0.625	0.6
1	0.3	0.01	0.95	0.625	0.6
1	0.3	0.01	0.95	0.625	0.8
1	0.3	0.01	0.95	0.625	0.8
1	0.3	0.01	0.95	0.625	0.8
1	0.3	0.01	0.95	0.625	0.8
1	0.3	0.01	0.95	0.625	0.8
1	0.3	0.01	0.95	0.750	0.6
1	0.3	0.01	0.95	0.750	0.6
1	0.3	0.01	0.95	0.750	0.6
1	0.3	0.01	0.95	0.750	0.6
1	0.3	0.01	0.95	0.750	0.6
1	0.3	0.01	0.95	0.750	0.8
1	0.3	0.01	0.95	0.750	0.8
1	0.3	0.01	0.95	0.750	0.8
1	0.3	0.01	0.95	0.750	0.8
1	0.3	0.01	0.95	0.750	0.8
1	0.3	0.01	0.95	0.875	0.6
1	0.3	0.01	0.95	0.875	0.6
1	0.3	0.01	0.95	0.875	0.6
1	0.3	0.01	0.95	0.875	0.6
1	0.3	0.01	0.95	0.875	0.6
1	0.3	0.01	0.95	0.875	0.8
1	0.3	0.01	0.95	0.875	0.8
1	0.3	0.01	0.95	0.875	0.8
1	0.3	0.01	0.95	0.875	0.8
1	0.3	0.01	0.95	0.875	0.8
1	0.3	0.01	0.95	1.000	0.6
1	0.3	0.01	0.95	1.000	0.6
1	0.3	0.01	0.95	1.000	0.6
1	0.3	0.01	0.95	1.000	0.6
1	0.3	0.01	0.95	1.000	0.6
1	0.3	0.01	0.95	1.000	0.8
1	0.3	0.01	0.95	1.000	0.8
1	0.3	0.01	0.95	1.000	0.8
1	0.3	0.01	0.95	1.000	0.8
1	0.3	0.01	0.95	1.000	0.8
1	0.3	0.50	0.05	0.500	0.6
1	0.3	0.50	0.05	0.500	0.6
1	0.3	0.50	0.05	0.500	0.6
1	0.3	0.50	0.05	0.500	0.6
1	0.3	0.50	0.05	0.500	0.6
1	0.3	0.50	0.05	0.500	0.8
1	0.3	0.50	0.05	0.500	0.8
1	0.3	0.50	0.05	0.500	0.8
1	0.3	0.50	0.05	0.500	0.8
1	0.3	0.50	0.05	0.500	0.8
1	0.3	0.50	0.05	0.625	0.6
1	0.3	0.50	0.05	0.625	0.6

1	0.3	0.50	0.05	0.625	0.6
1	0.3	0.50	0.05	0.625	0.6
1	0.3	0.50	0.05	0.625	0.6
1	0.3	0.50	0.05	0.625	0.8
1	0.3	0.50	0.05	0.625	0.8
1	0.3	0.50	0.05	0.625	0.8
1	0.3	0.50	0.05	0.625	0.8
1	0.3	0.50	0.05	0.625	0.8
1	0.3	0.50	0.05	0.750	0.6
1	0.3	0.50	0.05	0.750	0.6
1	0.3	0.50	0.05	0.750	0.6
1	0.3	0.50	0.05	0.750	0.6
1	0.3	0.50	0.05	0.750	0.6
1	0.3	0.50	0.05	0.750	0.8
1	0.3	0.50	0.05	0.750	0.8
1	0.3	0.50	0.05	0.750	0.8
1	0.3	0.50	0.05	0.750	0.8
1	0.3	0.50	0.05	0.750	0.8
1	0.3	0.50	0.05	0.875	0.6
1	0.3	0.50	0.05	0.875	0.6
1	0.3	0.50	0.05	0.875	0.6
1	0.3	0.50	0.05	0.875	0.6
1	0.3	0.50	0.05	0.875	0.6
1	0.3	0.50	0.05	0.875	0.8
1	0.3	0.50	0.05	0.875	0.8
1	0.3	0.50	0.05	0.875	0.8
1	0.3	0.50	0.05	0.875	0.8
1	0.3	0.50	0.05	0.875	0.8
1	0.3	0.50	0.05	1.000	0.6
1	0.3	0.50	0.05	1.000	0.6
1	0.3	0.50	0.05	1.000	0.6
1	0.3	0.50	0.05	1.000	0.6
1	0.3	0.50	0.05	1.000	0.6
1	0.3	0.50	0.05	1.000	0.8
1	0.3	0.50	0.05	1.000	0.8
1	0.3	0.50	0.05	1.000	0.8
1	0.3	0.50	0.05	1.000	0.8
1	0.3	0.50	0.05	1.000	0.8
1	0.3	0.50	0.95	0.500	0.6
1	0.3	0.50	0.95	0.500	0.6
1	0.3	0.50	0.95	0.500	0.6
1	0.3	0.50	0.95	0.500	0.6
1	0.3	0.50	0.95	0.500	0.6
1	0.3	0.50	0.95	0.500	0.8
1	0.3	0.50	0.95	0.500	0.8
1	0.3	0.50	0.95	0.500	0.8
1	0.3	0.50	0.95	0.500	0.8
1	0.3	0.50	0.95	0.500	0.8
1	0.3	0.50	0.95	0.625	0.6
1	0.3	0.50	0.95	0.625	0.6
1	0.3	0.50	0.95	0.625	0.6
1	0.3	0.50	0.95	0.625	0.6
1	0.3	0.50	0.95	0.625	0.6
1	0.3	0.50	0.95	0.625	0.8

1	0.3	0.50	0.95	0.625	0.8
1	0.3	0.50	0.95	0.625	0.8
1	0.3	0.50	0.95	0.625	0.8
1	0.3	0.50	0.95	0.625	0.8
1	0.3	0.50	0.95	0.750	0.6
1	0.3	0.50	0.95	0.750	0.6
1	0.3	0.50	0.95	0.750	0.6
1	0.3	0.50	0.95	0.750	0.6
1	0.3	0.50	0.95	0.750	0.6
1	0.3	0.50	0.95	0.750	0.8
1	0.3	0.50	0.95	0.750	0.8
1	0.3	0.50	0.95	0.750	0.8
1	0.3	0.50	0.95	0.750	0.8
1	0.3	0.50	0.95	0.750	0.8
1	0.3	0.50	0.95	0.875	0.6
1	0.3	0.50	0.95	0.875	0.6
1	0.3	0.50	0.95	0.875	0.6
1	0.3	0.50	0.95	0.875	0.6
1	0.3	0.50	0.95	0.875	0.6
1	0.3	0.50	0.95	0.875	0.8
1	0.3	0.50	0.95	0.875	0.8
1	0.3	0.50	0.95	0.875	0.8
1	0.3	0.50	0.95	0.875	0.8
1	0.3	0.50	0.95	0.875	0.8
1	0.3	0.50	0.95	1.000	0.6
1	0.3	0.50	0.95	1.000	0.6
1	0.3	0.50	0.95	1.000	0.6
1	0.3	0.50	0.95	1.000	0.6
1	0.3	0.50	0.95	1.000	0.6
1	0.3	0.50	0.95	1.000	0.8
1	0.3	0.50	0.95	1.000	0.8
1	0.3	0.50	0.95	1.000	0.8
1	0.3	0.50	0.95	1.000	0.8
1	0.3	0.50	0.95	1.000	0.8
1	0.4	0.01	0.05	0.500	0.6
1	0.4	0.01	0.05	0.500	0.6
1	0.4	0.01	0.05	0.500	0.6
1	0.4	0.01	0.05	0.500	0.6
1	0.4	0.01	0.05	0.500	0.6
1	0.4	0.01	0.05	0.500	0.8
1	0.4	0.01	0.05	0.500	0.8
1	0.4	0.01	0.05	0.500	0.8
1	0.4	0.01	0.05	0.500	0.8
1	0.4	0.01	0.05	0.500	0.8
1	0.4	0.01	0.05	0.625	0.6
1	0.4	0.01	0.05	0.625	0.6
1	0.4	0.01	0.05	0.625	0.6
1	0.4	0.01	0.05	0.625	0.6
1	0.4	0.01	0.05	0.625	0.6
1	0.4	0.01	0.05	0.625	0.8
1	0.4	0.01	0.05	0.625	0.8
1	0.4	0.01	0.05	0.625	0.8
1	0.4	0.01	0.05	0.625	0.8
1	0.4	0.01	0.05	0.625	0.8

1	0.4	0.01	0.05	0.750	0.6
1	0.4	0.01	0.05	0.750	0.6
1	0.4	0.01	0.05	0.750	0.6
1	0.4	0.01	0.05	0.750	0.6
1	0.4	0.01	0.05	0.750	0.6
1	0.4	0.01	0.05	0.750	0.8
1	0.4	0.01	0.05	0.750	0.8
1	0.4	0.01	0.05	0.750	0.8
1	0.4	0.01	0.05	0.750	0.8
1	0.4	0.01	0.05	0.750	0.8
1	0.4	0.01	0.05	0.875	0.6
1	0.4	0.01	0.05	0.875	0.6
1	0.4	0.01	0.05	0.875	0.6
1	0.4	0.01	0.05	0.875	0.6
1	0.4	0.01	0.05	0.875	0.6
1	0.4	0.01	0.05	0.875	0.8
1	0.4	0.01	0.05	0.875	0.8
1	0.4	0.01	0.05	0.875	0.8
1	0.4	0.01	0.05	0.875	0.8
1	0.4	0.01	0.05	0.875	0.8
1	0.4	0.01	0.05	1.000	0.6
1	0.4	0.01	0.05	1.000	0.6
1	0.4	0.01	0.05	1.000	0.6
1	0.4	0.01	0.05	1.000	0.6
1	0.4	0.01	0.05	1.000	0.6
1	0.4	0.01	0.05	1.000	0.8
1	0.4	0.01	0.05	1.000	0.8
1	0.4	0.01	0.05	1.000	0.8
1	0.4	0.01	0.05	1.000	0.8
1	0.4	0.01	0.05	1.000	0.8
1	0.4	0.01	0.95	0.500	0.6
1	0.4	0.01	0.95	0.500	0.6
1	0.4	0.01	0.95	0.500	0.6
1	0.4	0.01	0.95	0.500	0.6
1	0.4	0.01	0.95	0.500	0.6
1	0.4	0.01	0.95	0.500	0.8
1	0.4	0.01	0.95	0.500	0.8
1	0.4	0.01	0.95	0.500	0.8
1	0.4	0.01	0.95	0.500	0.8
1	0.4	0.01	0.95	0.500	0.8
1	0.4	0.01	0.95	0.625	0.6
1	0.4	0.01	0.95	0.625	0.6
1	0.4	0.01	0.95	0.625	0.6
1	0.4	0.01	0.95	0.625	0.6
1	0.4	0.01	0.95	0.625	0.6
1	0.4	0.01	0.95	0.625	0.8
1	0.4	0.01	0.95	0.625	0.8
1	0.4	0.01	0.95	0.625	0.8
1	0.4	0.01	0.95	0.625	0.8
1	0.4	0.01	0.95	0.625	0.8
1	0.4	0.01	0.95	0.750	0.6
1	0.4	0.01	0.95	0.750	0.6
1	0.4	0.01	0.95	0.750	0.6
1	0.4	0.01	0.95	0.750	0.6

1	0.4	0.01	0.95	0.750	0.6
1	0.4	0.01	0.95	0.750	0.8
1	0.4	0.01	0.95	0.750	0.8
1	0.4	0.01	0.95	0.750	0.8
1	0.4	0.01	0.95	0.750	0.8
1	0.4	0.01	0.95	0.750	0.8
1	0.4	0.01	0.95	0.875	0.6
1	0.4	0.01	0.95	0.875	0.6
1	0.4	0.01	0.95	0.875	0.6
1	0.4	0.01	0.95	0.875	0.6
1	0.4	0.01	0.95	0.875	0.6
1	0.4	0.01	0.95	0.875	0.8
1	0.4	0.01	0.95	0.875	0.8
1	0.4	0.01	0.95	0.875	0.8
1	0.4	0.01	0.95	0.875	0.8
1	0.4	0.01	0.95	0.875	0.8
1	0.4	0.01	0.95	1.000	0.6
1	0.4	0.01	0.95	1.000	0.6
1	0.4	0.01	0.95	1.000	0.6
1	0.4	0.01	0.95	1.000	0.6
1	0.4	0.01	0.95	1.000	0.6
1	0.4	0.01	0.95	1.000	0.8
1	0.4	0.01	0.95	1.000	0.8
1	0.4	0.01	0.95	1.000	0.8
1	0.4	0.01	0.95	1.000	0.8
1	0.4	0.01	0.95	1.000	0.8
1	0.4	0.50	0.05	0.500	0.6
1	0.4	0.50	0.05	0.500	0.6
1	0.4	0.50	0.05	0.500	0.6
1	0.4	0.50	0.05	0.500	0.6
1	0.4	0.50	0.05	0.500	0.6
1	0.4	0.50	0.05	0.500	0.8
1	0.4	0.50	0.05	0.500	0.8
1	0.4	0.50	0.05	0.500	0.8
1	0.4	0.50	0.05	0.500	0.8
1	0.4	0.50	0.05	0.500	0.8
1	0.4	0.50	0.05	0.625	0.6
1	0.4	0.50	0.05	0.625	0.6
1	0.4	0.50	0.05	0.625	0.6
1	0.4	0.50	0.05	0.625	0.6
1	0.4	0.50	0.05	0.625	0.6
1	0.4	0.50	0.05	0.625	0.8
1	0.4	0.50	0.05	0.625	0.8
1	0.4	0.50	0.05	0.625	0.8
1	0.4	0.50	0.05	0.625	0.8
1	0.4	0.50	0.05	0.625	0.8
1	0.4	0.50	0.05	0.750	0.6
1	0.4	0.50	0.05	0.750	0.6
1	0.4	0.50	0.05	0.750	0.6
1	0.4	0.50	0.05	0.750	0.6
1	0.4	0.50	0.05	0.750	0.6
1	0.4	0.50	0.05	0.750	0.8
1	0.4	0.50	0.05	0.750	0.8
1	0.4	0.50	0.05	0.750	0.8

1	0.4	0.50	0.05	0.750	0.8
1	0.4	0.50	0.05	0.750	0.8
1	0.4	0.50	0.05	0.875	0.6
1	0.4	0.50	0.05	0.875	0.6
1	0.4	0.50	0.05	0.875	0.6
1	0.4	0.50	0.05	0.875	0.6
1	0.4	0.50	0.05	0.875	0.6
1	0.4	0.50	0.05	0.875	0.8
1	0.4	0.50	0.05	0.875	0.8
1	0.4	0.50	0.05	0.875	0.8
1	0.4	0.50	0.05	0.875	0.8
1	0.4	0.50	0.05	0.875	0.8
1	0.4	0.50	0.05	1.000	0.6
1	0.4	0.50	0.05	1.000	0.6
1	0.4	0.50	0.05	1.000	0.6
1	0.4	0.50	0.05	1.000	0.6
1	0.4	0.50	0.05	1.000	0.6
1	0.4	0.50	0.05	1.000	0.8
1	0.4	0.50	0.05	1.000	0.8
1	0.4	0.50	0.05	1.000	0.8
1	0.4	0.50	0.05	1.000	0.8
1	0.4	0.50	0.05	1.000	0.8
1	0.4	0.50	0.95	0.500	0.6
1	0.4	0.50	0.95	0.500	0.6
1	0.4	0.50	0.95	0.500	0.6
1	0.4	0.50	0.95	0.500	0.6
1	0.4	0.50	0.95	0.500	0.6
1	0.4	0.50	0.95	0.500	0.8
1	0.4	0.50	0.95	0.500	0.8
1	0.4	0.50	0.95	0.500	0.8
1	0.4	0.50	0.95	0.500	0.8
1	0.4	0.50	0.95	0.500	0.8
1	0.4	0.50	0.95	0.625	0.6
1	0.4	0.50	0.95	0.625	0.6
1	0.4	0.50	0.95	0.625	0.6
1	0.4	0.50	0.95	0.625	0.6
1	0.4	0.50	0.95	0.625	0.6
1	0.4	0.50	0.95	0.625	0.8
1	0.4	0.50	0.95	0.625	0.8
1	0.4	0.50	0.95	0.625	0.8
1	0.4	0.50	0.95	0.625	0.8
1	0.4	0.50	0.95	0.625	0.8
1	0.4	0.50	0.95	0.750	0.6
1	0.4	0.50	0.95	0.750	0.6
1	0.4	0.50	0.95	0.750	0.6
1	0.4	0.50	0.95	0.750	0.6
1	0.4	0.50	0.95	0.750	0.6
1	0.4	0.50	0.95	0.750	0.8
1	0.4	0.50	0.95	0.750	0.8
1	0.4	0.50	0.95	0.750	0.8
1	0.4	0.50	0.95	0.750	0.8
1	0.4	0.50	0.95	0.750	0.8
1	0.4	0.50	0.95	0.875	0.6
1	0.4	0.50	0.95	0.875	0.6

1	0.4	0.50	0.95	0.875	0.6
1	0.4	0.50	0.95	0.875	0.6
1	0.4	0.50	0.95	0.875	0.6
1	0.4	0.50	0.95	0.875	0.8
1	0.4	0.50	0.95	0.875	0.8
1	0.4	0.50	0.95	0.875	0.8
1	0.4	0.50	0.95	0.875	0.8
1	0.4	0.50	0.95	0.875	0.8
1	0.4	0.50	0.95	1.000	0.6
1	0.4	0.50	0.95	1.000	0.6
1	0.4	0.50	0.95	1.000	0.6
1	0.4	0.50	0.95	1.000	0.6
1	0.4	0.50	0.95	1.000	0.6
1	0.4	0.50	0.95	1.000	0.8
1	0.4	0.50	0.95	1.000	0.8
1	0.4	0.50	0.95	1.000	0.8
1	0.4	0.50	0.95	1.000	0.8
1	0.4	0.50	0.95	1.000	0.8
2	0.3	0.01	0.05	0.500	0.6
2	0.3	0.01	0.05	0.500	0.6
2	0.3	0.01	0.05	0.500	0.6
2	0.3	0.01	0.05	0.500	0.6
2	0.3	0.01	0.05	0.500	0.6
2	0.3	0.01	0.05	0.500	0.8
2	0.3	0.01	0.05	0.500	0.8
2	0.3	0.01	0.05	0.500	0.8
2	0.3	0.01	0.05	0.500	0.8
2	0.3	0.01	0.05	0.500	0.8
2	0.3	0.01	0.05	0.625	0.6
2	0.3	0.01	0.05	0.625	0.6
2	0.3	0.01	0.05	0.625	0.6
2	0.3	0.01	0.05	0.625	0.6
2	0.3	0.01	0.05	0.625	0.6
2	0.3	0.01	0.05	0.625	0.8
2	0.3	0.01	0.05	0.625	0.8
2	0.3	0.01	0.05	0.625	0.8
2	0.3	0.01	0.05	0.625	0.8
2	0.3	0.01	0.05	0.625	0.8
2	0.3	0.01	0.05	0.750	0.6
2	0.3	0.01	0.05	0.750	0.6
2	0.3	0.01	0.05	0.750	0.6
2	0.3	0.01	0.05	0.750	0.6
2	0.3	0.01	0.05	0.750	0.6
2	0.3	0.01	0.05	0.750	0.8
2	0.3	0.01	0.05	0.750	0.8
2	0.3	0.01	0.05	0.750	0.8
2	0.3	0.01	0.05	0.750	0.8
2	0.3	0.01	0.05	0.750	0.8
2	0.3	0.01	0.05	0.875	0.6
2	0.3	0.01	0.05	0.875	0.6
2	0.3	0.01	0.05	0.875	0.6
2	0.3	0.01	0.05	0.875	0.6
2	0.3	0.01	0.05	0.875	0.6
2	0.3	0.01	0.05	0.875	0.8

2	0.3	0.01	0.05	0.875	0.8
2	0.3	0.01	0.05	0.875	0.8
2	0.3	0.01	0.05	0.875	0.8
2	0.3	0.01	0.05	0.875	0.8
2	0.3	0.01	0.05	1.000	0.6
2	0.3	0.01	0.05	1.000	0.6
2	0.3	0.01	0.05	1.000	0.6
2	0.3	0.01	0.05	1.000	0.6
2	0.3	0.01	0.05	1.000	0.6
2	0.3	0.01	0.05	1.000	0.8
2	0.3	0.01	0.05	1.000	0.8
2	0.3	0.01	0.05	1.000	0.8
2	0.3	0.01	0.05	1.000	0.8
2	0.3	0.01	0.05	1.000	0.8
2	0.3	0.01	0.95	0.500	0.6
2	0.3	0.01	0.95	0.500	0.6
2	0.3	0.01	0.95	0.500	0.6
2	0.3	0.01	0.95	0.500	0.6
2	0.3	0.01	0.95	0.500	0.6
2	0.3	0.01	0.95	0.500	0.8
2	0.3	0.01	0.95	0.500	0.8
2	0.3	0.01	0.95	0.500	0.8
2	0.3	0.01	0.95	0.500	0.8
2	0.3	0.01	0.95	0.500	0.8
2	0.3	0.01	0.95	0.625	0.6
2	0.3	0.01	0.95	0.625	0.6
2	0.3	0.01	0.95	0.625	0.6
2	0.3	0.01	0.95	0.625	0.6
2	0.3	0.01	0.95	0.625	0.6
2	0.3	0.01	0.95	0.625	0.8
2	0.3	0.01	0.95	0.625	0.8
2	0.3	0.01	0.95	0.625	0.8
2	0.3	0.01	0.95	0.625	0.8
2	0.3	0.01	0.95	0.625	0.8
2	0.3	0.01	0.95	0.750	0.6
2	0.3	0.01	0.95	0.750	0.6
2	0.3	0.01	0.95	0.750	0.6
2	0.3	0.01	0.95	0.750	0.6
2	0.3	0.01	0.95	0.750	0.6
2	0.3	0.01	0.95	0.750	0.8
2	0.3	0.01	0.95	0.750	0.8
2	0.3	0.01	0.95	0.750	0.8
2	0.3	0.01	0.95	0.750	0.8
2	0.3	0.01	0.95	0.750	0.8
2	0.3	0.01	0.95	0.875	0.6
2	0.3	0.01	0.95	0.875	0.6
2	0.3	0.01	0.95	0.875	0.6
2	0.3	0.01	0.95	0.875	0.6
2	0.3	0.01	0.95	0.875	0.6
2	0.3	0.01	0.95	0.875	0.8
2	0.3	0.01	0.95	0.875	0.8
2	0.3	0.01	0.95	0.875	0.8
2	0.3	0.01	0.95	0.875	0.8
2	0.3	0.01	0.95	0.875	0.8

2	0.3	0.01	0.95	1.000	0.6
2	0.3	0.01	0.95	1.000	0.6
2	0.3	0.01	0.95	1.000	0.6
2	0.3	0.01	0.95	1.000	0.6
2	0.3	0.01	0.95	1.000	0.6
2	0.3	0.01	0.95	1.000	0.8
2	0.3	0.01	0.95	1.000	0.8
2	0.3	0.01	0.95	1.000	0.8
2	0.3	0.01	0.95	1.000	0.8
2	0.3	0.01	0.95	1.000	0.8
2	0.3	0.50	0.05	0.500	0.6
2	0.3	0.50	0.05	0.500	0.6
2	0.3	0.50	0.05	0.500	0.6
2	0.3	0.50	0.05	0.500	0.6
2	0.3	0.50	0.05	0.500	0.6
2	0.3	0.50	0.05	0.500	0.8
2	0.3	0.50	0.05	0.500	0.8
2	0.3	0.50	0.05	0.500	0.8
2	0.3	0.50	0.05	0.500	0.8
2	0.3	0.50	0.05	0.500	0.8
2	0.3	0.50	0.05	0.625	0.6
2	0.3	0.50	0.05	0.625	0.6
2	0.3	0.50	0.05	0.625	0.6
2	0.3	0.50	0.05	0.625	0.6
2	0.3	0.50	0.05	0.625	0.6
2	0.3	0.50	0.05	0.625	0.8
2	0.3	0.50	0.05	0.625	0.8
2	0.3	0.50	0.05	0.625	0.8
2	0.3	0.50	0.05	0.625	0.8
2	0.3	0.50	0.05	0.625	0.8
2	0.3	0.50	0.05	0.750	0.6
2	0.3	0.50	0.05	0.750	0.6
2	0.3	0.50	0.05	0.750	0.6
2	0.3	0.50	0.05	0.750	0.6
2	0.3	0.50	0.05	0.750	0.6
2	0.3	0.50	0.05	0.750	0.8
2	0.3	0.50	0.05	0.750	0.8
2	0.3	0.50	0.05	0.750	0.8
2	0.3	0.50	0.05	0.750	0.8
2	0.3	0.50	0.05	0.750	0.8
2	0.3	0.50	0.05	0.875	0.6
2	0.3	0.50	0.05	0.875	0.6
2	0.3	0.50	0.05	0.875	0.6
2	0.3	0.50	0.05	0.875	0.6
2	0.3	0.50	0.05	0.875	0.6
2	0.3	0.50	0.05	0.875	0.8
2	0.3	0.50	0.05	0.875	0.8
2	0.3	0.50	0.05	0.875	0.8
2	0.3	0.50	0.05	0.875	0.8
2	0.3	0.50	0.05	0.875	0.8
2	0.3	0.50	0.05	1.000	0.6
2	0.3	0.50	0.05	1.000	0.6
2	0.3	0.50	0.05	1.000	0.6
2	0.3	0.50	0.05	1.000	0.6

2	0.3	0.50	0.05	1.000	0.6
2	0.3	0.50	0.05	1.000	0.8
2	0.3	0.50	0.05	1.000	0.8
2	0.3	0.50	0.05	1.000	0.8
2	0.3	0.50	0.05	1.000	0.8
2	0.3	0.50	0.05	1.000	0.8
2	0.3	0.50	0.95	0.500	0.6
2	0.3	0.50	0.95	0.500	0.6
2	0.3	0.50	0.95	0.500	0.6
2	0.3	0.50	0.95	0.500	0.6
2	0.3	0.50	0.95	0.500	0.6
2	0.3	0.50	0.95	0.500	0.8
2	0.3	0.50	0.95	0.500	0.8
2	0.3	0.50	0.95	0.500	0.8
2	0.3	0.50	0.95	0.500	0.8
2	0.3	0.50	0.95	0.500	0.8
2	0.3	0.50	0.95	0.625	0.6
2	0.3	0.50	0.95	0.625	0.6
2	0.3	0.50	0.95	0.625	0.6
2	0.3	0.50	0.95	0.625	0.6
2	0.3	0.50	0.95	0.625	0.6
2	0.3	0.50	0.95	0.625	0.8
2	0.3	0.50	0.95	0.625	0.8
2	0.3	0.50	0.95	0.625	0.8
2	0.3	0.50	0.95	0.625	0.8
2	0.3	0.50	0.95	0.625	0.8
2	0.3	0.50	0.95	0.750	0.6
2	0.3	0.50	0.95	0.750	0.6
2	0.3	0.50	0.95	0.750	0.6
2	0.3	0.50	0.95	0.750	0.6
2	0.3	0.50	0.95	0.750	0.6
2	0.3	0.50	0.95	0.750	0.8
2	0.3	0.50	0.95	0.750	0.8
2	0.3	0.50	0.95	0.750	0.8
2	0.3	0.50	0.95	0.750	0.8
2	0.3	0.50	0.95	0.750	0.8
2	0.3	0.50	0.95	0.875	0.6
2	0.3	0.50	0.95	0.875	0.6
2	0.3	0.50	0.95	0.875	0.6
2	0.3	0.50	0.95	0.875	0.6
2	0.3	0.50	0.95	0.875	0.6
2	0.3	0.50	0.95	0.875	0.8
2	0.3	0.50	0.95	0.875	0.8
2	0.3	0.50	0.95	0.875	0.8
2	0.3	0.50	0.95	0.875	0.8
2	0.3	0.50	0.95	0.875	0.8
2	0.3	0.50	0.95	1.000	0.6
2	0.3	0.50	0.95	1.000	0.6
2	0.3	0.50	0.95	1.000	0.6
2	0.3	0.50	0.95	1.000	0.6
2	0.3	0.50	0.95	1.000	0.6
2	0.3	0.50	0.95	1.000	0.8
2	0.3	0.50	0.95	1.000	0.8
2	0.3	0.50	0.95	1.000	0.8

2	0.3	0.50	0.95	1.000	0.8
2	0.3	0.50	0.95	1.000	0.8
2	0.4	0.01	0.05	0.500	0.6
2	0.4	0.01	0.05	0.500	0.6
2	0.4	0.01	0.05	0.500	0.6
2	0.4	0.01	0.05	0.500	0.6
2	0.4	0.01	0.05	0.500	0.6
2	0.4	0.01	0.05	0.500	0.8
2	0.4	0.01	0.05	0.500	0.8
2	0.4	0.01	0.05	0.500	0.8
2	0.4	0.01	0.05	0.500	0.8
2	0.4	0.01	0.05	0.500	0.8
2	0.4	0.01	0.05	0.625	0.6
2	0.4	0.01	0.05	0.625	0.6
2	0.4	0.01	0.05	0.625	0.6
2	0.4	0.01	0.05	0.625	0.6
2	0.4	0.01	0.05	0.625	0.6
2	0.4	0.01	0.05	0.625	0.8
2	0.4	0.01	0.05	0.625	0.8
2	0.4	0.01	0.05	0.625	0.8
2	0.4	0.01	0.05	0.625	0.8
2	0.4	0.01	0.05	0.625	0.8
2	0.4	0.01	0.05	0.750	0.6
2	0.4	0.01	0.05	0.750	0.6
2	0.4	0.01	0.05	0.750	0.6
2	0.4	0.01	0.05	0.750	0.6
2	0.4	0.01	0.05	0.750	0.6
2	0.4	0.01	0.05	0.750	0.8
2	0.4	0.01	0.05	0.750	0.8
2	0.4	0.01	0.05	0.750	0.8
2	0.4	0.01	0.05	0.750	0.8
2	0.4	0.01	0.05	0.750	0.8
2	0.4	0.01	0.05	0.875	0.6
2	0.4	0.01	0.05	0.875	0.6
2	0.4	0.01	0.05	0.875	0.6
2	0.4	0.01	0.05	0.875	0.6
2	0.4	0.01	0.05	0.875	0.6
2	0.4	0.01	0.05	0.875	0.8
2	0.4	0.01	0.05	0.875	0.8
2	0.4	0.01	0.05	0.875	0.8
2	0.4	0.01	0.05	0.875	0.8
2	0.4	0.01	0.05	0.875	0.8
2	0.4	0.01	0.05	1.000	0.6
2	0.4	0.01	0.05	1.000	0.6
2	0.4	0.01	0.05	1.000	0.6
2	0.4	0.01	0.05	1.000	0.6
2	0.4	0.01	0.05	1.000	0.6
2	0.4	0.01	0.05	1.000	0.8
2	0.4	0.01	0.05	1.000	0.8
2	0.4	0.01	0.05	1.000	0.8
2	0.4	0.01	0.05	1.000	0.8
2	0.4	0.01	0.05	1.000	0.8
2	0.4	0.01	0.95	0.500	0.6
2	0.4	0.01	0.95	0.500	0.6

2	0.4	0.01	0.95	0.500	0.6
2	0.4	0.01	0.95	0.500	0.6
2	0.4	0.01	0.95	0.500	0.6
2	0.4	0.01	0.95	0.500	0.8
2	0.4	0.01	0.95	0.500	0.8
2	0.4	0.01	0.95	0.500	0.8
2	0.4	0.01	0.95	0.500	0.8
2	0.4	0.01	0.95	0.500	0.8
2	0.4	0.01	0.95	0.625	0.6
2	0.4	0.01	0.95	0.625	0.6
2	0.4	0.01	0.95	0.625	0.6
2	0.4	0.01	0.95	0.625	0.6
2	0.4	0.01	0.95	0.625	0.6
2	0.4	0.01	0.95	0.625	0.8
2	0.4	0.01	0.95	0.625	0.8
2	0.4	0.01	0.95	0.625	0.8
2	0.4	0.01	0.95	0.625	0.8
2	0.4	0.01	0.95	0.625	0.8
2	0.4	0.01	0.95	0.750	0.6
2	0.4	0.01	0.95	0.750	0.6
2	0.4	0.01	0.95	0.750	0.6
2	0.4	0.01	0.95	0.750	0.6
2	0.4	0.01	0.95	0.750	0.6
2	0.4	0.01	0.95	0.750	0.8
2	0.4	0.01	0.95	0.750	0.8
2	0.4	0.01	0.95	0.750	0.8
2	0.4	0.01	0.95	0.750	0.8
2	0.4	0.01	0.95	0.750	0.8
2	0.4	0.01	0.95	0.875	0.6
2	0.4	0.01	0.95	0.875	0.6
2	0.4	0.01	0.95	0.875	0.6
2	0.4	0.01	0.95	0.875	0.6
2	0.4	0.01	0.95	0.875	0.6
2	0.4	0.01	0.95	0.875	0.8
2	0.4	0.01	0.95	0.875	0.8
2	0.4	0.01	0.95	0.875	0.8
2	0.4	0.01	0.95	0.875	0.8
2	0.4	0.01	0.95	0.875	0.8
2	0.4	0.01	0.95	1.000	0.6
2	0.4	0.01	0.95	1.000	0.6
2	0.4	0.01	0.95	1.000	0.6
2	0.4	0.01	0.95	1.000	0.6
2	0.4	0.01	0.95	1.000	0.6
2	0.4	0.01	0.95	1.000	0.8
2	0.4	0.01	0.95	1.000	0.8
2	0.4	0.01	0.95	1.000	0.8
2	0.4	0.01	0.95	1.000	0.8
2	0.4	0.01	0.95	1.000	0.8
2	0.4	0.50	0.05	0.500	0.6
2	0.4	0.50	0.05	0.500	0.6
2	0.4	0.50	0.05	0.500	0.6
2	0.4	0.50	0.05	0.500	0.6
2	0.4	0.50	0.05	0.500	0.6
2	0.4	0.50	0.05	0.500	0.8

2	0.4	0.50	0.05	0.500	0.8
2	0.4	0.50	0.05	0.500	0.8
2	0.4	0.50	0.05	0.500	0.8
2	0.4	0.50	0.05	0.500	0.8
2	0.4	0.50	0.05	0.625	0.6
2	0.4	0.50	0.05	0.625	0.6
2	0.4	0.50	0.05	0.625	0.6
2	0.4	0.50	0.05	0.625	0.6
2	0.4	0.50	0.05	0.625	0.6
2	0.4	0.50	0.05	0.625	0.8
2	0.4	0.50	0.05	0.625	0.8
2	0.4	0.50	0.05	0.625	0.8
2	0.4	0.50	0.05	0.625	0.8
2	0.4	0.50	0.05	0.625	0.8
2	0.4	0.50	0.05	0.750	0.6
2	0.4	0.50	0.05	0.750	0.6
2	0.4	0.50	0.05	0.750	0.6
2	0.4	0.50	0.05	0.750	0.6
2	0.4	0.50	0.05	0.750	0.6
2	0.4	0.50	0.05	0.750	0.8
2	0.4	0.50	0.05	0.750	0.8
2	0.4	0.50	0.05	0.750	0.8
2	0.4	0.50	0.05	0.750	0.8
2	0.4	0.50	0.05	0.750	0.8
2	0.4	0.50	0.05	0.875	0.6
2	0.4	0.50	0.05	0.875	0.6
2	0.4	0.50	0.05	0.875	0.6
2	0.4	0.50	0.05	0.875	0.6
2	0.4	0.50	0.05	0.875	0.6
2	0.4	0.50	0.05	0.875	0.8
2	0.4	0.50	0.05	0.875	0.8
2	0.4	0.50	0.05	0.875	0.8
2	0.4	0.50	0.05	0.875	0.8
2	0.4	0.50	0.05	0.875	0.8
2	0.4	0.50	0.05	1.000	0.6
2	0.4	0.50	0.05	1.000	0.6
2	0.4	0.50	0.05	1.000	0.6
2	0.4	0.50	0.05	1.000	0.6
2	0.4	0.50	0.05	1.000	0.6
2	0.4	0.50	0.05	1.000	0.8
2	0.4	0.50	0.05	1.000	0.8
2	0.4	0.50	0.05	1.000	0.8
2	0.4	0.50	0.05	1.000	0.8
2	0.4	0.50	0.05	1.000	0.8
2	0.4	0.50	0.95	0.500	0.6
2	0.4	0.50	0.95	0.500	0.6
2	0.4	0.50	0.95	0.500	0.6
2	0.4	0.50	0.95	0.500	0.6
2	0.4	0.50	0.95	0.500	0.6
2	0.4	0.50	0.95	0.500	0.8
2	0.4	0.50	0.95	0.500	0.8
2	0.4	0.50	0.95	0.500	0.8
2	0.4	0.50	0.95	0.500	0.8
2	0.4	0.50	0.95	0.500	0.8

2	0.4	0.50	0.95	0.625	0.6
2	0.4	0.50	0.95	0.625	0.6
2	0.4	0.50	0.95	0.625	0.6
2	0.4	0.50	0.95	0.625	0.6
2	0.4	0.50	0.95	0.625	0.6
2	0.4	0.50	0.95	0.625	0.8
2	0.4	0.50	0.95	0.625	0.8
2	0.4	0.50	0.95	0.625	0.8
2	0.4	0.50	0.95	0.625	0.8
2	0.4	0.50	0.95	0.625	0.8
2	0.4	0.50	0.95	0.750	0.6
2	0.4	0.50	0.95	0.750	0.6
2	0.4	0.50	0.95	0.750	0.6
2	0.4	0.50	0.95	0.750	0.6
2	0.4	0.50	0.95	0.750	0.6
2	0.4	0.50	0.95	0.750	0.8
2	0.4	0.50	0.95	0.750	0.8
2	0.4	0.50	0.95	0.750	0.8
2	0.4	0.50	0.95	0.750	0.8
2	0.4	0.50	0.95	0.750	0.8
2	0.4	0.50	0.95	0.875	0.6
2	0.4	0.50	0.95	0.875	0.6
2	0.4	0.50	0.95	0.875	0.6
2	0.4	0.50	0.95	0.875	0.6
2	0.4	0.50	0.95	0.875	0.6
2	0.4	0.50	0.95	0.875	0.8
2	0.4	0.50	0.95	0.875	0.8
2	0.4	0.50	0.95	0.875	0.8
2	0.4	0.50	0.95	0.875	0.8
2	0.4	0.50	0.95	0.875	0.8
2	0.4	0.50	0.95	1.000	0.6
2	0.4	0.50	0.95	1.000	0.6
2	0.4	0.50	0.95	1.000	0.6
2	0.4	0.50	0.95	1.000	0.6
2	0.4	0.50	0.95	1.000	0.6
2	0.4	0.50	0.95	1.000	0.8
2	0.4	0.50	0.95	1.000	0.8
2	0.4	0.50	0.95	1.000	0.8
2	0.4	0.50	0.95	1.000	0.8
2	0.4	0.50	0.95	1.000	0.8
3	0.3	0.01	0.05	0.500	0.6
3	0.3	0.01	0.05	0.500	0.6
3	0.3	0.01	0.05	0.500	0.6
3	0.3	0.01	0.05	0.500	0.6
3	0.3	0.01	0.05	0.500	0.6
3	0.3	0.01	0.05	0.500	0.8
3	0.3	0.01	0.05	0.500	0.8
3	0.3	0.01	0.05	0.500	0.8
3	0.3	0.01	0.05	0.500	0.8
3	0.3	0.01	0.05	0.500	0.8
3	0.3	0.01	0.05	0.625	0.6
3	0.3	0.01	0.05	0.625	0.6
3	0.3	0.01	0.05	0.625	0.6
3	0.3	0.01	0.05	0.625	0.6

3	0.3	0.01	0.05	0.625	0.6
3	0.3	0.01	0.05	0.625	0.8
3	0.3	0.01	0.05	0.625	0.8
3	0.3	0.01	0.05	0.625	0.8
3	0.3	0.01	0.05	0.625	0.8
3	0.3	0.01	0.05	0.625	0.8
3	0.3	0.01	0.05	0.750	0.6
3	0.3	0.01	0.05	0.750	0.6
3	0.3	0.01	0.05	0.750	0.6
3	0.3	0.01	0.05	0.750	0.6
3	0.3	0.01	0.05	0.750	0.6
3	0.3	0.01	0.05	0.750	0.8
3	0.3	0.01	0.05	0.750	0.8
3	0.3	0.01	0.05	0.750	0.8
3	0.3	0.01	0.05	0.750	0.8
3	0.3	0.01	0.05	0.750	0.8
3	0.3	0.01	0.05	0.875	0.6
3	0.3	0.01	0.05	0.875	0.6
3	0.3	0.01	0.05	0.875	0.6
3	0.3	0.01	0.05	0.875	0.6
3	0.3	0.01	0.05	0.875	0.6
3	0.3	0.01	0.05	0.875	0.8
3	0.3	0.01	0.05	0.875	0.8
3	0.3	0.01	0.05	0.875	0.8
3	0.3	0.01	0.05	0.875	0.8
3	0.3	0.01	0.05	0.875	0.8
3	0.3	0.01	0.05	1.000	0.6
3	0.3	0.01	0.05	1.000	0.6
3	0.3	0.01	0.05	1.000	0.6
3	0.3	0.01	0.05	1.000	0.6
3	0.3	0.01	0.05	1.000	0.6
3	0.3	0.01	0.05	1.000	0.8
3	0.3	0.01	0.05	1.000	0.8
3	0.3	0.01	0.05	1.000	0.8
3	0.3	0.01	0.05	1.000	0.8
3	0.3	0.01	0.05	1.000	0.8
3	0.3	0.01	0.95	0.500	0.6
3	0.3	0.01	0.95	0.500	0.6
3	0.3	0.01	0.95	0.500	0.6
3	0.3	0.01	0.95	0.500	0.6
3	0.3	0.01	0.95	0.500	0.6
3	0.3	0.01	0.95	0.500	0.8
3	0.3	0.01	0.95	0.500	0.8
3	0.3	0.01	0.95	0.500	0.8
3	0.3	0.01	0.95	0.500	0.8
3	0.3	0.01	0.95	0.500	0.8
3	0.3	0.01	0.95	0.625	0.6
3	0.3	0.01	0.95	0.625	0.6
3	0.3	0.01	0.95	0.625	0.6
3	0.3	0.01	0.95	0.625	0.6
3	0.3	0.01	0.95	0.625	0.6
3	0.3	0.01	0.95	0.625	0.8
3	0.3	0.01	0.95	0.625	0.8
3	0.3	0.01	0.95	0.625	0.8

3	0.3	0.01	0.95	0.625	0.8
3	0.3	0.01	0.95	0.625	0.8
3	0.3	0.01	0.95	0.750	0.6
3	0.3	0.01	0.95	0.750	0.6
3	0.3	0.01	0.95	0.750	0.6
3	0.3	0.01	0.95	0.750	0.6
3	0.3	0.01	0.95	0.750	0.6
3	0.3	0.01	0.95	0.750	0.8
3	0.3	0.01	0.95	0.750	0.8
3	0.3	0.01	0.95	0.750	0.8
3	0.3	0.01	0.95	0.750	0.8
3	0.3	0.01	0.95	0.750	0.8
3	0.3	0.01	0.95	0.875	0.6
3	0.3	0.01	0.95	0.875	0.6
3	0.3	0.01	0.95	0.875	0.6
3	0.3	0.01	0.95	0.875	0.6
3	0.3	0.01	0.95	0.875	0.6
3	0.3	0.01	0.95	0.875	0.8
3	0.3	0.01	0.95	0.875	0.8
3	0.3	0.01	0.95	0.875	0.8
3	0.3	0.01	0.95	0.875	0.8
3	0.3	0.01	0.95	0.875	0.8
3	0.3	0.01	0.95	1.000	0.6
3	0.3	0.01	0.95	1.000	0.6
3	0.3	0.01	0.95	1.000	0.6
3	0.3	0.01	0.95	1.000	0.6
3	0.3	0.01	0.95	1.000	0.6
3	0.3	0.01	0.95	1.000	0.8
3	0.3	0.01	0.95	1.000	0.8
3	0.3	0.01	0.95	1.000	0.8
3	0.3	0.01	0.95	1.000	0.8
3	0.3	0.01	0.95	1.000	0.8
3	0.3	0.50	0.05	0.500	0.6
3	0.3	0.50	0.05	0.500	0.6
3	0.3	0.50	0.05	0.500	0.6
3	0.3	0.50	0.05	0.500	0.6
3	0.3	0.50	0.05	0.500	0.6
3	0.3	0.50	0.05	0.500	0.8
3	0.3	0.50	0.05	0.500	0.8
3	0.3	0.50	0.05	0.500	0.8
3	0.3	0.50	0.05	0.500	0.8
3	0.3	0.50	0.05	0.500	0.8
3	0.3	0.50	0.05	0.625	0.6
3	0.3	0.50	0.05	0.625	0.6
3	0.3	0.50	0.05	0.625	0.6
3	0.3	0.50	0.05	0.625	0.6
3	0.3	0.50	0.05	0.625	0.6
3	0.3	0.50	0.05	0.625	0.8
3	0.3	0.50	0.05	0.625	0.8
3	0.3	0.50	0.05	0.625	0.8
3	0.3	0.50	0.05	0.625	0.8
3	0.3	0.50	0.05	0.625	0.8
3	0.3	0.50	0.05	0.750	0.6
3	0.3	0.50	0.05	0.750	0.6

3	0.3	0.50	0.05	0.750	0.6
3	0.3	0.50	0.05	0.750	0.6
3	0.3	0.50	0.05	0.750	0.6
3	0.3	0.50	0.05	0.750	0.8
3	0.3	0.50	0.05	0.750	0.8
3	0.3	0.50	0.05	0.750	0.8
3	0.3	0.50	0.05	0.750	0.8
3	0.3	0.50	0.05	0.750	0.8
3	0.3	0.50	0.05	0.875	0.6
3	0.3	0.50	0.05	0.875	0.6
3	0.3	0.50	0.05	0.875	0.6
3	0.3	0.50	0.05	0.875	0.6
3	0.3	0.50	0.05	0.875	0.6
3	0.3	0.50	0.05	0.875	0.8
3	0.3	0.50	0.05	0.875	0.8
3	0.3	0.50	0.05	0.875	0.8
3	0.3	0.50	0.05	0.875	0.8
3	0.3	0.50	0.05	0.875	0.8
3	0.3	0.50	0.05	1.000	0.6
3	0.3	0.50	0.05	1.000	0.6
3	0.3	0.50	0.05	1.000	0.6
3	0.3	0.50	0.05	1.000	0.6
3	0.3	0.50	0.05	1.000	0.6
3	0.3	0.50	0.05	1.000	0.8
3	0.3	0.50	0.05	1.000	0.8
3	0.3	0.50	0.05	1.000	0.8
3	0.3	0.50	0.05	1.000	0.8
3	0.3	0.50	0.05	1.000	0.8
3	0.3	0.50	0.95	0.500	0.6
3	0.3	0.50	0.95	0.500	0.6
3	0.3	0.50	0.95	0.500	0.6
3	0.3	0.50	0.95	0.500	0.6
3	0.3	0.50	0.95	0.500	0.6
3	0.3	0.50	0.95	0.500	0.8
3	0.3	0.50	0.95	0.500	0.8
3	0.3	0.50	0.95	0.500	0.8
3	0.3	0.50	0.95	0.500	0.8
3	0.3	0.50	0.95	0.500	0.8
3	0.3	0.50	0.95	0.625	0.6
3	0.3	0.50	0.95	0.625	0.6
3	0.3	0.50	0.95	0.625	0.6
3	0.3	0.50	0.95	0.625	0.6
3	0.3	0.50	0.95	0.625	0.6
3	0.3	0.50	0.95	0.625	0.8
3	0.3	0.50	0.95	0.625	0.8
3	0.3	0.50	0.95	0.625	0.8
3	0.3	0.50	0.95	0.625	0.8
3	0.3	0.50	0.95	0.625	0.8
3	0.3	0.50	0.95	0.750	0.6
3	0.3	0.50	0.95	0.750	0.6
3	0.3	0.50	0.95	0.750	0.6
3	0.3	0.50	0.95	0.750	0.6
3	0.3	0.50	0.95	0.750	0.6
3	0.3	0.50	0.95	0.750	0.8

3	0.3	0.50	0.95	0.750	0.8
3	0.3	0.50	0.95	0.750	0.8
3	0.3	0.50	0.95	0.750	0.8
3	0.3	0.50	0.95	0.750	0.8
3	0.3	0.50	0.95	0.875	0.6
3	0.3	0.50	0.95	0.875	0.6
3	0.3	0.50	0.95	0.875	0.6
3	0.3	0.50	0.95	0.875	0.6
3	0.3	0.50	0.95	0.875	0.6
3	0.3	0.50	0.95	0.875	0.8
3	0.3	0.50	0.95	0.875	0.8
3	0.3	0.50	0.95	0.875	0.8
3	0.3	0.50	0.95	0.875	0.8
3	0.3	0.50	0.95	0.875	0.8
3	0.3	0.50	0.95	1.000	0.6
3	0.3	0.50	0.95	1.000	0.6
3	0.3	0.50	0.95	1.000	0.6
3	0.3	0.50	0.95	1.000	0.6
3	0.3	0.50	0.95	1.000	0.6
3	0.3	0.50	0.95	1.000	0.8
3	0.3	0.50	0.95	1.000	0.8
3	0.3	0.50	0.95	1.000	0.8
3	0.3	0.50	0.95	1.000	0.8
3	0.3	0.50	0.95	1.000	0.8
3	0.4	0.01	0.05	0.500	0.6
3	0.4	0.01	0.05	0.500	0.6
3	0.4	0.01	0.05	0.500	0.6
3	0.4	0.01	0.05	0.500	0.6
3	0.4	0.01	0.05	0.500	0.6
3	0.4	0.01	0.05	0.500	0.8
3	0.4	0.01	0.05	0.500	0.8
3	0.4	0.01	0.05	0.500	0.8
3	0.4	0.01	0.05	0.500	0.8
3	0.4	0.01	0.05	0.500	0.8
3	0.4	0.01	0.05	0.625	0.6
3	0.4	0.01	0.05	0.625	0.6
3	0.4	0.01	0.05	0.625	0.6
3	0.4	0.01	0.05	0.625	0.6
3	0.4	0.01	0.05	0.625	0.6
3	0.4	0.01	0.05	0.625	0.8
3	0.4	0.01	0.05	0.625	0.8
3	0.4	0.01	0.05	0.625	0.8
3	0.4	0.01	0.05	0.625	0.8
3	0.4	0.01	0.05	0.625	0.8
3	0.4	0.01	0.05	0.750	0.6
3	0.4	0.01	0.05	0.750	0.6
3	0.4	0.01	0.05	0.750	0.6
3	0.4	0.01	0.05	0.750	0.6
3	0.4	0.01	0.05	0.750	0.6
3	0.4	0.01	0.05	0.750	0.8
3	0.4	0.01	0.05	0.750	0.8
3	0.4	0.01	0.05	0.750	0.8
3	0.4	0.01	0.05	0.750	0.8
3	0.4	0.01	0.05	0.750	0.8

3	0.4	0.01	0.05	0.875	0.6
3	0.4	0.01	0.05	0.875	0.6
3	0.4	0.01	0.05	0.875	0.6
3	0.4	0.01	0.05	0.875	0.6
3	0.4	0.01	0.05	0.875	0.6
3	0.4	0.01	0.05	0.875	0.8
3	0.4	0.01	0.05	0.875	0.8
3	0.4	0.01	0.05	0.875	0.8
3	0.4	0.01	0.05	0.875	0.8
3	0.4	0.01	0.05	0.875	0.8
3	0.4	0.01	0.05	1.000	0.6
3	0.4	0.01	0.05	1.000	0.6
3	0.4	0.01	0.05	1.000	0.6
3	0.4	0.01	0.05	1.000	0.6
3	0.4	0.01	0.05	1.000	0.6
3	0.4	0.01	0.05	1.000	0.8
3	0.4	0.01	0.05	1.000	0.8
3	0.4	0.01	0.05	1.000	0.8
3	0.4	0.01	0.05	1.000	0.8
3	0.4	0.01	0.05	1.000	0.8
3	0.4	0.01	0.95	0.500	0.6
3	0.4	0.01	0.95	0.500	0.6
3	0.4	0.01	0.95	0.500	0.6
3	0.4	0.01	0.95	0.500	0.6
3	0.4	0.01	0.95	0.500	0.6
3	0.4	0.01	0.95	0.500	0.8
3	0.4	0.01	0.95	0.500	0.8
3	0.4	0.01	0.95	0.500	0.8
3	0.4	0.01	0.95	0.500	0.8
3	0.4	0.01	0.95	0.500	0.8
3	0.4	0.01	0.95	0.625	0.6
3	0.4	0.01	0.95	0.625	0.6
3	0.4	0.01	0.95	0.625	0.6
3	0.4	0.01	0.95	0.625	0.6
3	0.4	0.01	0.95	0.625	0.6
3	0.4	0.01	0.95	0.625	0.8
3	0.4	0.01	0.95	0.625	0.8
3	0.4	0.01	0.95	0.625	0.8
3	0.4	0.01	0.95	0.625	0.8
3	0.4	0.01	0.95	0.625	0.8
3	0.4	0.01	0.95	0.750	0.6
3	0.4	0.01	0.95	0.750	0.6
3	0.4	0.01	0.95	0.750	0.6
3	0.4	0.01	0.95	0.750	0.6
3	0.4	0.01	0.95	0.750	0.6
3	0.4	0.01	0.95	0.750	0.8
3	0.4	0.01	0.95	0.750	0.8
3	0.4	0.01	0.95	0.750	0.8
3	0.4	0.01	0.95	0.750	0.8
3	0.4	0.01	0.95	0.750	0.8
3	0.4	0.01	0.95	0.875	0.6
3	0.4	0.01	0.95	0.875	0.6
3	0.4	0.01	0.95	0.875	0.6
3	0.4	0.01	0.95	0.875	0.6

3	0.4	0.01	0.95	0.875	0.6
3	0.4	0.01	0.95	0.875	0.8
3	0.4	0.01	0.95	0.875	0.8
3	0.4	0.01	0.95	0.875	0.8
3	0.4	0.01	0.95	0.875	0.8
3	0.4	0.01	0.95	0.875	0.8
3	0.4	0.01	0.95	1.000	0.6
3	0.4	0.01	0.95	1.000	0.6
3	0.4	0.01	0.95	1.000	0.6
3	0.4	0.01	0.95	1.000	0.6
3	0.4	0.01	0.95	1.000	0.6
3	0.4	0.01	0.95	1.000	0.8
3	0.4	0.01	0.95	1.000	0.8
3	0.4	0.01	0.95	1.000	0.8
3	0.4	0.01	0.95	1.000	0.8
3	0.4	0.01	0.95	1.000	0.8
3	0.4	0.50	0.05	0.500	0.6
3	0.4	0.50	0.05	0.500	0.6
3	0.4	0.50	0.05	0.500	0.6
3	0.4	0.50	0.05	0.500	0.6
3	0.4	0.50	0.05	0.500	0.6
3	0.4	0.50	0.05	0.500	0.8
3	0.4	0.50	0.05	0.500	0.8
3	0.4	0.50	0.05	0.500	0.8
3	0.4	0.50	0.05	0.500	0.8
3	0.4	0.50	0.05	0.500	0.8
3	0.4	0.50	0.05	0.625	0.6
3	0.4	0.50	0.05	0.625	0.6
3	0.4	0.50	0.05	0.625	0.6
3	0.4	0.50	0.05	0.625	0.6
3	0.4	0.50	0.05	0.625	0.6
3	0.4	0.50	0.05	0.625	0.8
3	0.4	0.50	0.05	0.625	0.8
3	0.4	0.50	0.05	0.625	0.8
3	0.4	0.50	0.05	0.625	0.8
3	0.4	0.50	0.05	0.625	0.8
3	0.4	0.50	0.05	0.750	0.6
3	0.4	0.50	0.05	0.750	0.6
3	0.4	0.50	0.05	0.750	0.6
3	0.4	0.50	0.05	0.750	0.6
3	0.4	0.50	0.05	0.750	0.6
3	0.4	0.50	0.05	0.750	0.8
3	0.4	0.50	0.05	0.750	0.8
3	0.4	0.50	0.05	0.750	0.8
3	0.4	0.50	0.05	0.750	0.8
3	0.4	0.50	0.05	0.750	0.8
3	0.4	0.50	0.05	0.875	0.6
3	0.4	0.50	0.05	0.875	0.6
3	0.4	0.50	0.05	0.875	0.6
3	0.4	0.50	0.05	0.875	0.6
3	0.4	0.50	0.05	0.875	0.6
3	0.4	0.50	0.05	0.875	0.8
3	0.4	0.50	0.05	0.875	0.8
3	0.4	0.50	0.05	0.875	0.8

3	0.4	0.50	0.05	0.875	0.8
3	0.4	0.50	0.05	0.875	0.8
3	0.4	0.50	0.05	1.000	0.6
3	0.4	0.50	0.05	1.000	0.6
3	0.4	0.50	0.05	1.000	0.6
3	0.4	0.50	0.05	1.000	0.6
3	0.4	0.50	0.05	1.000	0.6
3	0.4	0.50	0.05	1.000	0.8
3	0.4	0.50	0.05	1.000	0.8
3	0.4	0.50	0.05	1.000	0.8
3	0.4	0.50	0.05	1.000	0.8
3	0.4	0.50	0.05	1.000	0.8
3	0.4	0.50	0.95	0.500	0.6
3	0.4	0.50	0.95	0.500	0.6
3	0.4	0.50	0.95	0.500	0.6
3	0.4	0.50	0.95	0.500	0.6
3	0.4	0.50	0.95	0.500	0.6
3	0.4	0.50	0.95	0.500	0.8
3	0.4	0.50	0.95	0.500	0.8
3	0.4	0.50	0.95	0.500	0.8
3	0.4	0.50	0.95	0.500	0.8
3	0.4	0.50	0.95	0.500	0.8
3	0.4	0.50	0.95	0.625	0.6
3	0.4	0.50	0.95	0.625	0.6
3	0.4	0.50	0.95	0.625	0.6
3	0.4	0.50	0.95	0.625	0.6
3	0.4	0.50	0.95	0.625	0.6
3	0.4	0.50	0.95	0.625	0.8
3	0.4	0.50	0.95	0.625	0.8
3	0.4	0.50	0.95	0.625	0.8
3	0.4	0.50	0.95	0.625	0.8
3	0.4	0.50	0.95	0.625	0.8
3	0.4	0.50	0.95	0.750	0.6
3	0.4	0.50	0.95	0.750	0.6
3	0.4	0.50	0.95	0.750	0.6
3	0.4	0.50	0.95	0.750	0.6
3	0.4	0.50	0.95	0.750	0.6
3	0.4	0.50	0.95	0.750	0.8
3	0.4	0.50	0.95	0.750	0.8
3	0.4	0.50	0.95	0.750	0.8
3	0.4	0.50	0.95	0.750	0.8
3	0.4	0.50	0.95	0.750	0.8
3	0.4	0.50	0.95	0.875	0.6
3	0.4	0.50	0.95	0.875	0.6
3	0.4	0.50	0.95	0.875	0.6
3	0.4	0.50	0.95	0.875	0.6
3	0.4	0.50	0.95	0.875	0.6
3	0.4	0.50	0.95	0.875	0.8
3	0.4	0.50	0.95	0.875	0.8
3	0.4	0.50	0.95	0.875	0.8
3	0.4	0.50	0.95	0.875	0.8
3	0.4	0.50	0.95	0.875	0.8
3	0.4	0.50	0.95	1.000	0.6
3	0.4	0.50	0.95	1.000	0.6

3	0.4	0.50	0.95	1.000	0.6
3	0.4	0.50	0.95	1.000	0.6
3	0.4	0.50	0.95	1.000	0.6
3	0.4	0.50	0.95	1.000	0.8
3	0.4	0.50	0.95	1.000	0.8
3	0.4	0.50	0.95	1.000	0.8
3	0.4	0.50	0.95	1.000	0.8
3	0.4	0.50	0.95	1.000	0.8
4	0.3	0.01	0.05	0.500	0.6
4	0.3	0.01	0.05	0.500	0.6
4	0.3	0.01	0.05	0.500	0.6
4	0.3	0.01	0.05	0.500	0.6
4	0.3	0.01	0.05	0.500	0.6
4	0.3	0.01	0.05	0.500	0.8
4	0.3	0.01	0.05	0.500	0.8
4	0.3	0.01	0.05	0.500	0.8
4	0.3	0.01	0.05	0.500	0.8
4	0.3	0.01	0.05	0.500	0.8
4	0.3	0.01	0.05	0.625	0.6
4	0.3	0.01	0.05	0.625	0.6
4	0.3	0.01	0.05	0.625	0.6
4	0.3	0.01	0.05	0.625	0.6
4	0.3	0.01	0.05	0.625	0.6
4	0.3	0.01	0.05	0.625	0.8
4	0.3	0.01	0.05	0.625	0.8
4	0.3	0.01	0.05	0.625	0.8
4	0.3	0.01	0.05	0.625	0.8
4	0.3	0.01	0.05	0.625	0.8
4	0.3	0.01	0.05	0.750	0.6
4	0.3	0.01	0.05	0.750	0.6
4	0.3	0.01	0.05	0.750	0.6
4	0.3	0.01	0.05	0.750	0.6
4	0.3	0.01	0.05	0.750	0.6
4	0.3	0.01	0.05	0.750	0.8
4	0.3	0.01	0.05	0.750	0.8
4	0.3	0.01	0.05	0.750	0.8
4	0.3	0.01	0.05	0.750	0.8
4	0.3	0.01	0.05	0.750	0.8
4	0.3	0.01	0.05	0.875	0.6
4	0.3	0.01	0.05	0.875	0.6
4	0.3	0.01	0.05	0.875	0.6
4	0.3	0.01	0.05	0.875	0.6
4	0.3	0.01	0.05	0.875	0.6
4	0.3	0.01	0.05	0.875	0.8
4	0.3	0.01	0.05	0.875	0.8
4	0.3	0.01	0.05	0.875	0.8
4	0.3	0.01	0.05	0.875	0.8
4	0.3	0.01	0.05	0.875	0.8
4	0.3	0.01	0.05	1.000	0.6
4	0.3	0.01	0.05	1.000	0.6
4	0.3	0.01	0.05	1.000	0.6
4	0.3	0.01	0.05	1.000	0.6
4	0.3	0.01	0.05	1.000	0.6
4	0.3	0.01	0.05	1.000	0.8

4	0.3	0.01	0.05	1.000	0.8
4	0.3	0.01	0.05	1.000	0.8
4	0.3	0.01	0.05	1.000	0.8
4	0.3	0.01	0.05	1.000	0.8
4	0.3	0.01	0.95	0.500	0.6
4	0.3	0.01	0.95	0.500	0.6
4	0.3	0.01	0.95	0.500	0.6
4	0.3	0.01	0.95	0.500	0.6
4	0.3	0.01	0.95	0.500	0.6
4	0.3	0.01	0.95	0.500	0.8
4	0.3	0.01	0.95	0.500	0.8
4	0.3	0.01	0.95	0.500	0.8
4	0.3	0.01	0.95	0.500	0.8
4	0.3	0.01	0.95	0.500	0.8
4	0.3	0.01	0.95	0.625	0.6
4	0.3	0.01	0.95	0.625	0.6
4	0.3	0.01	0.95	0.625	0.6
4	0.3	0.01	0.95	0.625	0.6
4	0.3	0.01	0.95	0.625	0.6
4	0.3	0.01	0.95	0.625	0.8
4	0.3	0.01	0.95	0.625	0.8
4	0.3	0.01	0.95	0.625	0.8
4	0.3	0.01	0.95	0.625	0.8
4	0.3	0.01	0.95	0.625	0.8
4	0.3	0.01	0.95	0.750	0.6
4	0.3	0.01	0.95	0.750	0.6
4	0.3	0.01	0.95	0.750	0.6
4	0.3	0.01	0.95	0.750	0.6
4	0.3	0.01	0.95	0.750	0.6
4	0.3	0.01	0.95	0.750	0.8
4	0.3	0.01	0.95	0.750	0.8
4	0.3	0.01	0.95	0.750	0.8
4	0.3	0.01	0.95	0.750	0.8
4	0.3	0.01	0.95	0.750	0.8
4	0.3	0.01	0.95	0.875	0.6
4	0.3	0.01	0.95	0.875	0.6
4	0.3	0.01	0.95	0.875	0.6
4	0.3	0.01	0.95	0.875	0.6
4	0.3	0.01	0.95	0.875	0.6
4	0.3	0.01	0.95	0.875	0.8
4	0.3	0.01	0.95	0.875	0.8
4	0.3	0.01	0.95	0.875	0.8
4	0.3	0.01	0.95	0.875	0.8
4	0.3	0.01	0.95	0.875	0.8
4	0.3	0.01	0.95	1.000	0.6
4	0.3	0.01	0.95	1.000	0.6
4	0.3	0.01	0.95	1.000	0.6
4	0.3	0.01	0.95	1.000	0.6
4	0.3	0.01	0.95	1.000	0.6
4	0.3	0.01	0.95	1.000	0.8
4	0.3	0.01	0.95	1.000	0.8
4	0.3	0.01	0.95	1.000	0.8
4	0.3	0.01	0.95	1.000	0.8
4	0.3	0.01	0.95	1.000	0.8

4	0.3	0.50	0.05	0.500	0.6
4	0.3	0.50	0.05	0.500	0.6
4	0.3	0.50	0.05	0.500	0.6
4	0.3	0.50	0.05	0.500	0.6
4	0.3	0.50	0.05	0.500	0.6
4	0.3	0.50	0.05	0.500	0.8
4	0.3	0.50	0.05	0.500	0.8
4	0.3	0.50	0.05	0.500	0.8
4	0.3	0.50	0.05	0.500	0.8
4	0.3	0.50	0.05	0.500	0.8
4	0.3	0.50	0.05	0.625	0.6
4	0.3	0.50	0.05	0.625	0.6
4	0.3	0.50	0.05	0.625	0.6
4	0.3	0.50	0.05	0.625	0.6
4	0.3	0.50	0.05	0.625	0.6
4	0.3	0.50	0.05	0.625	0.8
4	0.3	0.50	0.05	0.625	0.8
4	0.3	0.50	0.05	0.625	0.8
4	0.3	0.50	0.05	0.625	0.8
4	0.3	0.50	0.05	0.625	0.8
4	0.3	0.50	0.05	0.750	0.6
4	0.3	0.50	0.05	0.750	0.6
4	0.3	0.50	0.05	0.750	0.6
4	0.3	0.50	0.05	0.750	0.6
4	0.3	0.50	0.05	0.750	0.6
4	0.3	0.50	0.05	0.750	0.8
4	0.3	0.50	0.05	0.750	0.8
4	0.3	0.50	0.05	0.750	0.8
4	0.3	0.50	0.05	0.750	0.8
4	0.3	0.50	0.05	0.750	0.8
4	0.3	0.50	0.05	0.875	0.6
4	0.3	0.50	0.05	0.875	0.6
4	0.3	0.50	0.05	0.875	0.6
4	0.3	0.50	0.05	0.875	0.6
4	0.3	0.50	0.05	0.875	0.6
4	0.3	0.50	0.05	0.875	0.8
4	0.3	0.50	0.05	0.875	0.8
4	0.3	0.50	0.05	0.875	0.8
4	0.3	0.50	0.05	0.875	0.8
4	0.3	0.50	0.05	0.875	0.8
4	0.3	0.50	0.05	1.000	0.6
4	0.3	0.50	0.05	1.000	0.6
4	0.3	0.50	0.05	1.000	0.6
4	0.3	0.50	0.05	1.000	0.6
4	0.3	0.50	0.05	1.000	0.6
4	0.3	0.50	0.05	1.000	0.8
4	0.3	0.50	0.05	1.000	0.8
4	0.3	0.50	0.05	1.000	0.8
4	0.3	0.50	0.05	1.000	0.8
4	0.3	0.50	0.05	1.000	0.8
4	0.3	0.50	0.95	0.500	0.6
4	0.3	0.50	0.95	0.500	0.6
4	0.3	0.50	0.95	0.500	0.6
4	0.3	0.50	0.95	0.500	0.6

4	0.3	0.50	0.95	0.500	0.6
4	0.3	0.50	0.95	0.500	0.8
4	0.3	0.50	0.95	0.500	0.8
4	0.3	0.50	0.95	0.500	0.8
4	0.3	0.50	0.95	0.500	0.8
4	0.3	0.50	0.95	0.500	0.8
4	0.3	0.50	0.95	0.625	0.6
4	0.3	0.50	0.95	0.625	0.6
4	0.3	0.50	0.95	0.625	0.6
4	0.3	0.50	0.95	0.625	0.6
4	0.3	0.50	0.95	0.625	0.6
4	0.3	0.50	0.95	0.625	0.8
4	0.3	0.50	0.95	0.625	0.8
4	0.3	0.50	0.95	0.625	0.8
4	0.3	0.50	0.95	0.625	0.8
4	0.3	0.50	0.95	0.625	0.8
4	0.3	0.50	0.95	0.750	0.6
4	0.3	0.50	0.95	0.750	0.6
4	0.3	0.50	0.95	0.750	0.6
4	0.3	0.50	0.95	0.750	0.6
4	0.3	0.50	0.95	0.750	0.6
4	0.3	0.50	0.95	0.750	0.8
4	0.3	0.50	0.95	0.750	0.8
4	0.3	0.50	0.95	0.750	0.8
4	0.3	0.50	0.95	0.750	0.8
4	0.3	0.50	0.95	0.750	0.8
4	0.3	0.50	0.95	0.875	0.6
4	0.3	0.50	0.95	0.875	0.6
4	0.3	0.50	0.95	0.875	0.6
4	0.3	0.50	0.95	0.875	0.6
4	0.3	0.50	0.95	0.875	0.6
4	0.3	0.50	0.95	0.875	0.8
4	0.3	0.50	0.95	0.875	0.8
4	0.3	0.50	0.95	0.875	0.8
4	0.3	0.50	0.95	0.875	0.8
4	0.3	0.50	0.95	0.875	0.8
4	0.3	0.50	0.95	1.000	0.6
4	0.3	0.50	0.95	1.000	0.6
4	0.3	0.50	0.95	1.000	0.6
4	0.3	0.50	0.95	1.000	0.6
4	0.3	0.50	0.95	1.000	0.6
4	0.3	0.50	0.95	1.000	0.8
4	0.3	0.50	0.95	1.000	0.8
4	0.3	0.50	0.95	1.000	0.8
4	0.3	0.50	0.95	1.000	0.8
4	0.3	0.50	0.95	1.000	0.8
4	0.4	0.01	0.05	0.500	0.6
4	0.4	0.01	0.05	0.500	0.6
4	0.4	0.01	0.05	0.500	0.6
4	0.4	0.01	0.05	0.500	0.6
4	0.4	0.01	0.05	0.500	0.6
4	0.4	0.01	0.05	0.500	0.8
4	0.4	0.01	0.05	0.500	0.8
4	0.4	0.01	0.05	0.500	0.8

4	0.4	0.01	0.05	0.500	0.8
4	0.4	0.01	0.05	0.500	0.8
4	0.4	0.01	0.05	0.625	0.6
4	0.4	0.01	0.05	0.625	0.6
4	0.4	0.01	0.05	0.625	0.6
4	0.4	0.01	0.05	0.625	0.6
4	0.4	0.01	0.05	0.625	0.6
4	0.4	0.01	0.05	0.625	0.8
4	0.4	0.01	0.05	0.625	0.8
4	0.4	0.01	0.05	0.625	0.8
4	0.4	0.01	0.05	0.625	0.8
4	0.4	0.01	0.05	0.625	0.8
4	0.4	0.01	0.05	0.750	0.6
4	0.4	0.01	0.05	0.750	0.6
4	0.4	0.01	0.05	0.750	0.6
4	0.4	0.01	0.05	0.750	0.6
4	0.4	0.01	0.05	0.750	0.6
4	0.4	0.01	0.05	0.750	0.8
4	0.4	0.01	0.05	0.750	0.8
4	0.4	0.01	0.05	0.750	0.8
4	0.4	0.01	0.05	0.750	0.8
4	0.4	0.01	0.05	0.750	0.8
4	0.4	0.01	0.05	0.875	0.6
4	0.4	0.01	0.05	0.875	0.6
4	0.4	0.01	0.05	0.875	0.6
4	0.4	0.01	0.05	0.875	0.6
4	0.4	0.01	0.05	0.875	0.6
4	0.4	0.01	0.05	0.875	0.8
4	0.4	0.01	0.05	0.875	0.8
4	0.4	0.01	0.05	0.875	0.8
4	0.4	0.01	0.05	0.875	0.8
4	0.4	0.01	0.05	0.875	0.8
4	0.4	0.01	0.05	1.000	0.6
4	0.4	0.01	0.05	1.000	0.6
4	0.4	0.01	0.05	1.000	0.6
4	0.4	0.01	0.05	1.000	0.6
4	0.4	0.01	0.05	1.000	0.6
4	0.4	0.01	0.05	1.000	0.8
4	0.4	0.01	0.05	1.000	0.8
4	0.4	0.01	0.05	1.000	0.8
4	0.4	0.01	0.05	1.000	0.8
4	0.4	0.01	0.05	1.000	0.8
4	0.4	0.01	0.95	0.500	0.6
4	0.4	0.01	0.95	0.500	0.6
4	0.4	0.01	0.95	0.500	0.6
4	0.4	0.01	0.95	0.500	0.6
4	0.4	0.01	0.95	0.500	0.6
4	0.4	0.01	0.95	0.500	0.8
4	0.4	0.01	0.95	0.500	0.8
4	0.4	0.01	0.95	0.500	0.8
4	0.4	0.01	0.95	0.500	0.8
4	0.4	0.01	0.95	0.500	0.8
4	0.4	0.01	0.95	0.625	0.6
4	0.4	0.01	0.95	0.625	0.6

4	0.4	0.01	0.95	0.625	0.6
4	0.4	0.01	0.95	0.625	0.6
4	0.4	0.01	0.95	0.625	0.6
4	0.4	0.01	0.95	0.625	0.8
4	0.4	0.01	0.95	0.625	0.8
4	0.4	0.01	0.95	0.625	0.8
4	0.4	0.01	0.95	0.625	0.8
4	0.4	0.01	0.95	0.625	0.8
4	0.4	0.01	0.95	0.750	0.6
4	0.4	0.01	0.95	0.750	0.6
4	0.4	0.01	0.95	0.750	0.6
4	0.4	0.01	0.95	0.750	0.6
4	0.4	0.01	0.95	0.750	0.6
4	0.4	0.01	0.95	0.750	0.8
4	0.4	0.01	0.95	0.750	0.8
4	0.4	0.01	0.95	0.750	0.8
4	0.4	0.01	0.95	0.750	0.8
4	0.4	0.01	0.95	0.750	0.8
4	0.4	0.01	0.95	0.875	0.6
4	0.4	0.01	0.95	0.875	0.6
4	0.4	0.01	0.95	0.875	0.6
4	0.4	0.01	0.95	0.875	0.6
4	0.4	0.01	0.95	0.875	0.6
4	0.4	0.01	0.95	0.875	0.8
4	0.4	0.01	0.95	0.875	0.8
4	0.4	0.01	0.95	0.875	0.8
4	0.4	0.01	0.95	0.875	0.8
4	0.4	0.01	0.95	0.875	0.8
4	0.4	0.01	0.95	1.000	0.6
4	0.4	0.01	0.95	1.000	0.6
4	0.4	0.01	0.95	1.000	0.6
4	0.4	0.01	0.95	1.000	0.6
4	0.4	0.01	0.95	1.000	0.6
4	0.4	0.01	0.95	1.000	0.8
4	0.4	0.01	0.95	1.000	0.8
4	0.4	0.01	0.95	1.000	0.8
4	0.4	0.01	0.95	1.000	0.8
4	0.4	0.01	0.95	1.000	0.8
4	0.4	0.50	0.05	0.500	0.6
4	0.4	0.50	0.05	0.500	0.6
4	0.4	0.50	0.05	0.500	0.6
4	0.4	0.50	0.05	0.500	0.6
4	0.4	0.50	0.05	0.500	0.6
4	0.4	0.50	0.05	0.500	0.8
4	0.4	0.50	0.05	0.500	0.8
4	0.4	0.50	0.05	0.500	0.8
4	0.4	0.50	0.05	0.500	0.8
4	0.4	0.50	0.05	0.500	0.8
4	0.4	0.50	0.05	0.625	0.6
4	0.4	0.50	0.05	0.625	0.6
4	0.4	0.50	0.05	0.625	0.6
4	0.4	0.50	0.05	0.625	0.6
4	0.4	0.50	0.05	0.625	0.8

4	0.4	0.50	0.05	0.625	0.8
4	0.4	0.50	0.05	0.625	0.8
4	0.4	0.50	0.05	0.625	0.8
4	0.4	0.50	0.05	0.625	0.8
4	0.4	0.50	0.05	0.750	0.6
4	0.4	0.50	0.05	0.750	0.6
4	0.4	0.50	0.05	0.750	0.6
4	0.4	0.50	0.05	0.750	0.6
4	0.4	0.50	0.05	0.750	0.6
4	0.4	0.50	0.05	0.750	0.8
4	0.4	0.50	0.05	0.750	0.8
4	0.4	0.50	0.05	0.750	0.8
4	0.4	0.50	0.05	0.750	0.8
4	0.4	0.50	0.05	0.750	0.8
4	0.4	0.50	0.05	0.875	0.6
4	0.4	0.50	0.05	0.875	0.6
4	0.4	0.50	0.05	0.875	0.6
4	0.4	0.50	0.05	0.875	0.6
4	0.4	0.50	0.05	0.875	0.6
4	0.4	0.50	0.05	0.875	0.8
4	0.4	0.50	0.05	0.875	0.8
4	0.4	0.50	0.05	0.875	0.8
4	0.4	0.50	0.05	0.875	0.8
4	0.4	0.50	0.05	0.875	0.8
4	0.4	0.50	0.05	1.000	0.6
4	0.4	0.50	0.05	1.000	0.6
4	0.4	0.50	0.05	1.000	0.6
4	0.4	0.50	0.05	1.000	0.6
4	0.4	0.50	0.05	1.000	0.6
4	0.4	0.50	0.05	1.000	0.8
4	0.4	0.50	0.05	1.000	0.8
4	0.4	0.50	0.05	1.000	0.8
4	0.4	0.50	0.05	1.000	0.8
4	0.4	0.50	0.05	1.000	0.8
4	0.4	0.50	0.95	0.500	0.6
4	0.4	0.50	0.95	0.500	0.6
4	0.4	0.50	0.95	0.500	0.6
4	0.4	0.50	0.95	0.500	0.6
4	0.4	0.50	0.95	0.500	0.6
4	0.4	0.50	0.95	0.500	0.8
4	0.4	0.50	0.95	0.500	0.8
4	0.4	0.50	0.95	0.500	0.8
4	0.4	0.50	0.95	0.500	0.8
4	0.4	0.50	0.95	0.500	0.8
4	0.4	0.50	0.95	0.625	0.6
4	0.4	0.50	0.95	0.625	0.6
4	0.4	0.50	0.95	0.625	0.6
4	0.4	0.50	0.95	0.625	0.6
4	0.4	0.50	0.95	0.625	0.6
4	0.4	0.50	0.95	0.625	0.8
4	0.4	0.50	0.95	0.625	0.8
4	0.4	0.50	0.95	0.625	0.8
4	0.4	0.50	0.95	0.625	0.8
4	0.4	0.50	0.95	0.625	0.8

4	0.4	0.50	0.95	0.750	0.6
4	0.4	0.50	0.95	0.750	0.6
4	0.4	0.50	0.95	0.750	0.6
4	0.4	0.50	0.95	0.750	0.6
4	0.4	0.50	0.95	0.750	0.6
4	0.4	0.50	0.95	0.750	0.8
4	0.4	0.50	0.95	0.750	0.8
4	0.4	0.50	0.95	0.750	0.8
4	0.4	0.50	0.95	0.750	0.8
4	0.4	0.50	0.95	0.750	0.8
4	0.4	0.50	0.95	0.875	0.6
4	0.4	0.50	0.95	0.875	0.6
4	0.4	0.50	0.95	0.875	0.6
4	0.4	0.50	0.95	0.875	0.6
4	0.4	0.50	0.95	0.875	0.6
4	0.4	0.50	0.95	0.875	0.8
4	0.4	0.50	0.95	0.875	0.8
4	0.4	0.50	0.95	0.875	0.8
4	0.4	0.50	0.95	0.875	0.8
4	0.4	0.50	0.95	0.875	0.8
4	0.4	0.50	0.95	1.000	0.6
4	0.4	0.50	0.95	1.000	0.6
4	0.4	0.50	0.95	1.000	0.6
4	0.4	0.50	0.95	1.000	0.6
4	0.4	0.50	0.95	1.000	0.6
4	0.4	0.50	0.95	1.000	0.8
4	0.4	0.50	0.95	1.000	0.8
4	0.4	0.50	0.95	1.000	0.8
4	0.4	0.50	0.95	1.000	0.8
4	0.4	0.50	0.95	1.000	0.8
5	0.3	0.01	0.05	0.500	0.6
5	0.3	0.01	0.05	0.500	0.6
5	0.3	0.01	0.05	0.500	0.6
5	0.3	0.01	0.05	0.500	0.6
5	0.3	0.01	0.05	0.500	0.6
5	0.3	0.01	0.05	0.500	0.8
5	0.3	0.01	0.05	0.500	0.8
5	0.3	0.01	0.05	0.500	0.8
5	0.3	0.01	0.05	0.500	0.8
5	0.3	0.01	0.05	0.500	0.8
5	0.3	0.01	0.05	0.625	0.6
5	0.3	0.01	0.05	0.625	0.6
5	0.3	0.01	0.05	0.625	0.6
5	0.3	0.01	0.05	0.625	0.6
5	0.3	0.01	0.05	0.625	0.6
5	0.3	0.01	0.05	0.625	0.8
5	0.3	0.01	0.05	0.625	0.8
5	0.3	0.01	0.05	0.625	0.8
5	0.3	0.01	0.05	0.625	0.8
5	0.3	0.01	0.05	0.625	0.8
5	0.3	0.01	0.05	0.750	0.6
5	0.3	0.01	0.05	0.750	0.6
5	0.3	0.01	0.05	0.750	0.6
5	0.3	0.01	0.05	0.750	0.6

5	0.3	0.01	0.05	0.750	0.6
5	0.3	0.01	0.05	0.750	0.8
5	0.3	0.01	0.05	0.750	0.8
5	0.3	0.01	0.05	0.750	0.8
5	0.3	0.01	0.05	0.750	0.8
5	0.3	0.01	0.05	0.750	0.8
5	0.3	0.01	0.05	0.875	0.6
5	0.3	0.01	0.05	0.875	0.6
5	0.3	0.01	0.05	0.875	0.6
5	0.3	0.01	0.05	0.875	0.6
5	0.3	0.01	0.05	0.875	0.6
5	0.3	0.01	0.05	0.875	0.8
5	0.3	0.01	0.05	0.875	0.8
5	0.3	0.01	0.05	0.875	0.8
5	0.3	0.01	0.05	0.875	0.8
5	0.3	0.01	0.05	0.875	0.8
5	0.3	0.01	0.05	1.000	0.6
5	0.3	0.01	0.05	1.000	0.6
5	0.3	0.01	0.05	1.000	0.6
5	0.3	0.01	0.05	1.000	0.6
5	0.3	0.01	0.05	1.000	0.6
5	0.3	0.01	0.05	1.000	0.8
5	0.3	0.01	0.05	1.000	0.8
5	0.3	0.01	0.05	1.000	0.8
5	0.3	0.01	0.05	1.000	0.8
5	0.3	0.01	0.05	1.000	0.8
5	0.3	0.01	0.95	0.500	0.6
5	0.3	0.01	0.95	0.500	0.6
5	0.3	0.01	0.95	0.500	0.6
5	0.3	0.01	0.95	0.500	0.6
5	0.3	0.01	0.95	0.500	0.6
5	0.3	0.01	0.95	0.500	0.8
5	0.3	0.01	0.95	0.500	0.8
5	0.3	0.01	0.95	0.500	0.8
5	0.3	0.01	0.95	0.500	0.8
5	0.3	0.01	0.95	0.500	0.8
5	0.3	0.01	0.95	0.625	0.6
5	0.3	0.01	0.95	0.625	0.6
5	0.3	0.01	0.95	0.625	0.6
5	0.3	0.01	0.95	0.625	0.6
5	0.3	0.01	0.95	0.625	0.6
5	0.3	0.01	0.95	0.625	0.8
5	0.3	0.01	0.95	0.625	0.8
5	0.3	0.01	0.95	0.625	0.8
5	0.3	0.01	0.95	0.625	0.8
5	0.3	0.01	0.95	0.625	0.8
5	0.3	0.01	0.95	0.750	0.6
5	0.3	0.01	0.95	0.750	0.6
5	0.3	0.01	0.95	0.750	0.6
5	0.3	0.01	0.95	0.750	0.6
5	0.3	0.01	0.95	0.750	0.6
5	0.3	0.01	0.95	0.750	0.8
5	0.3	0.01	0.95	0.750	0.8
5	0.3	0.01	0.95	0.750	0.8

5	0.3	0.01	0.95	0.750	0.8
5	0.3	0.01	0.95	0.750	0.8
5	0.3	0.01	0.95	0.875	0.6
5	0.3	0.01	0.95	0.875	0.6
5	0.3	0.01	0.95	0.875	0.6
5	0.3	0.01	0.95	0.875	0.6
5	0.3	0.01	0.95	0.875	0.6
5	0.3	0.01	0.95	0.875	0.8
5	0.3	0.01	0.95	0.875	0.8
5	0.3	0.01	0.95	0.875	0.8
5	0.3	0.01	0.95	0.875	0.8
5	0.3	0.01	0.95	0.875	0.8
5	0.3	0.01	0.95	1.000	0.6
5	0.3	0.01	0.95	1.000	0.6
5	0.3	0.01	0.95	1.000	0.6
5	0.3	0.01	0.95	1.000	0.6
5	0.3	0.01	0.95	1.000	0.6
5	0.3	0.01	0.95	1.000	0.8
5	0.3	0.01	0.95	1.000	0.8
5	0.3	0.01	0.95	1.000	0.8
5	0.3	0.01	0.95	1.000	0.8
5	0.3	0.01	0.95	1.000	0.8
5	0.3	0.50	0.05	0.500	0.6
5	0.3	0.50	0.05	0.500	0.6
5	0.3	0.50	0.05	0.500	0.6
5	0.3	0.50	0.05	0.500	0.6
5	0.3	0.50	0.05	0.500	0.6
5	0.3	0.50	0.05	0.500	0.8
5	0.3	0.50	0.05	0.500	0.8
5	0.3	0.50	0.05	0.500	0.8
5	0.3	0.50	0.05	0.500	0.8
5	0.3	0.50	0.05	0.500	0.8
5	0.3	0.50	0.05	0.625	0.6
5	0.3	0.50	0.05	0.625	0.6
5	0.3	0.50	0.05	0.625	0.6
5	0.3	0.50	0.05	0.625	0.6
5	0.3	0.50	0.05	0.625	0.6
5	0.3	0.50	0.05	0.625	0.8
5	0.3	0.50	0.05	0.625	0.8
5	0.3	0.50	0.05	0.625	0.8
5	0.3	0.50	0.05	0.625	0.8
5	0.3	0.50	0.05	0.625	0.8
5	0.3	0.50	0.05	0.750	0.6
5	0.3	0.50	0.05	0.750	0.6
5	0.3	0.50	0.05	0.750	0.6
5	0.3	0.50	0.05	0.750	0.6
5	0.3	0.50	0.05	0.750	0.6
5	0.3	0.50	0.05	0.750	0.8
5	0.3	0.50	0.05	0.750	0.8
5	0.3	0.50	0.05	0.750	0.8
5	0.3	0.50	0.05	0.750	0.8
5	0.3	0.50	0.05	0.750	0.8
5	0.3	0.50	0.05	0.875	0.6
5	0.3	0.50	0.05	0.875	0.6

5	0.3	0.50	0.05	0.875	0.6
5	0.3	0.50	0.05	0.875	0.6
5	0.3	0.50	0.05	0.875	0.6
5	0.3	0.50	0.05	0.875	0.8
5	0.3	0.50	0.05	0.875	0.8
5	0.3	0.50	0.05	0.875	0.8
5	0.3	0.50	0.05	0.875	0.8
5	0.3	0.50	0.05	0.875	0.8
5	0.3	0.50	0.05	1.000	0.6
5	0.3	0.50	0.05	1.000	0.6
5	0.3	0.50	0.05	1.000	0.6
5	0.3	0.50	0.05	1.000	0.6
5	0.3	0.50	0.05	1.000	0.6
5	0.3	0.50	0.05	1.000	0.8
5	0.3	0.50	0.05	1.000	0.8
5	0.3	0.50	0.05	1.000	0.8
5	0.3	0.50	0.05	1.000	0.8
5	0.3	0.50	0.05	1.000	0.8
5	0.3	0.50	0.95	0.500	0.6
5	0.3	0.50	0.95	0.500	0.6
5	0.3	0.50	0.95	0.500	0.6
5	0.3	0.50	0.95	0.500	0.6
5	0.3	0.50	0.95	0.500	0.6
5	0.3	0.50	0.95	0.500	0.8
5	0.3	0.50	0.95	0.500	0.8
5	0.3	0.50	0.95	0.500	0.8
5	0.3	0.50	0.95	0.500	0.8
5	0.3	0.50	0.95	0.500	0.8
5	0.3	0.50	0.95	0.625	0.6
5	0.3	0.50	0.95	0.625	0.6
5	0.3	0.50	0.95	0.625	0.6
5	0.3	0.50	0.95	0.625	0.6
5	0.3	0.50	0.95	0.625	0.6
5	0.3	0.50	0.95	0.625	0.8
5	0.3	0.50	0.95	0.625	0.8
5	0.3	0.50	0.95	0.625	0.8
5	0.3	0.50	0.95	0.625	0.8
5	0.3	0.50	0.95	0.625	0.8
5	0.3	0.50	0.95	0.750	0.6
5	0.3	0.50	0.95	0.750	0.6
5	0.3	0.50	0.95	0.750	0.6
5	0.3	0.50	0.95	0.750	0.6
5	0.3	0.50	0.95	0.750	0.6
5	0.3	0.50	0.95	0.750	0.8
5	0.3	0.50	0.95	0.750	0.8
5	0.3	0.50	0.95	0.750	0.8
5	0.3	0.50	0.95	0.750	0.8
5	0.3	0.50	0.95	0.750	0.8
5	0.3	0.50	0.95	0.875	0.6
5	0.3	0.50	0.95	0.875	0.6
5	0.3	0.50	0.95	0.875	0.6
5	0.3	0.50	0.95	0.875	0.6
5	0.3	0.50	0.95	0.875	0.6
5	0.3	0.50	0.95	0.875	0.8

5	0.3	0.50	0.95	0.875	0.8
5	0.3	0.50	0.95	0.875	0.8
5	0.3	0.50	0.95	0.875	0.8
5	0.3	0.50	0.95	0.875	0.8
5	0.3	0.50	0.95	1.000	0.6
5	0.3	0.50	0.95	1.000	0.6
5	0.3	0.50	0.95	1.000	0.6
5	0.3	0.50	0.95	1.000	0.6
5	0.3	0.50	0.95	1.000	0.6
5	0.3	0.50	0.95	1.000	0.8
5	0.3	0.50	0.95	1.000	0.8
5	0.3	0.50	0.95	1.000	0.8
5	0.3	0.50	0.95	1.000	0.8
5	0.3	0.50	0.95	1.000	0.8
5	0.4	0.01	0.05	0.500	0.6
5	0.4	0.01	0.05	0.500	0.6
5	0.4	0.01	0.05	0.500	0.6
5	0.4	0.01	0.05	0.500	0.6
5	0.4	0.01	0.05	0.500	0.6
5	0.4	0.01	0.05	0.500	0.8
5	0.4	0.01	0.05	0.500	0.8
5	0.4	0.01	0.05	0.500	0.8
5	0.4	0.01	0.05	0.500	0.8
5	0.4	0.01	0.05	0.500	0.8
5	0.4	0.01	0.05	0.625	0.6
5	0.4	0.01	0.05	0.625	0.6
5	0.4	0.01	0.05	0.625	0.6
5	0.4	0.01	0.05	0.625	0.6
5	0.4	0.01	0.05	0.625	0.6
5	0.4	0.01	0.05	0.625	0.8
5	0.4	0.01	0.05	0.625	0.8
5	0.4	0.01	0.05	0.625	0.8
5	0.4	0.01	0.05	0.625	0.8
5	0.4	0.01	0.05	0.625	0.8
5	0.4	0.01	0.05	0.750	0.6
5	0.4	0.01	0.05	0.750	0.6
5	0.4	0.01	0.05	0.750	0.6
5	0.4	0.01	0.05	0.750	0.6
5	0.4	0.01	0.05	0.750	0.6
5	0.4	0.01	0.05	0.750	0.8
5	0.4	0.01	0.05	0.750	0.8
5	0.4	0.01	0.05	0.750	0.8
5	0.4	0.01	0.05	0.750	0.8
5	0.4	0.01	0.05	0.750	0.8
5	0.4	0.01	0.05	0.875	0.6
5	0.4	0.01	0.05	0.875	0.6
5	0.4	0.01	0.05	0.875	0.6
5	0.4	0.01	0.05	0.875	0.6
5	0.4	0.01	0.05	0.875	0.6
5	0.4	0.01	0.05	0.875	0.8
5	0.4	0.01	0.05	0.875	0.8
5	0.4	0.01	0.05	0.875	0.8
5	0.4	0.01	0.05	0.875	0.8
5	0.4	0.01	0.05	0.875	0.8

5	0.4	0.01	0.05	1.000	0.6
5	0.4	0.01	0.05	1.000	0.6
5	0.4	0.01	0.05	1.000	0.6
5	0.4	0.01	0.05	1.000	0.6
5	0.4	0.01	0.05	1.000	0.6
5	0.4	0.01	0.05	1.000	0.8
5	0.4	0.01	0.05	1.000	0.8
5	0.4	0.01	0.05	1.000	0.8
5	0.4	0.01	0.05	1.000	0.8
5	0.4	0.01	0.05	1.000	0.8
5	0.4	0.01	0.95	0.500	0.6
5	0.4	0.01	0.95	0.500	0.6
5	0.4	0.01	0.95	0.500	0.6
5	0.4	0.01	0.95	0.500	0.6
5	0.4	0.01	0.95	0.500	0.6
5	0.4	0.01	0.95	0.500	0.8
5	0.4	0.01	0.95	0.500	0.8
5	0.4	0.01	0.95	0.500	0.8
5	0.4	0.01	0.95	0.500	0.8
5	0.4	0.01	0.95	0.500	0.8
5	0.4	0.01	0.95	0.625	0.6
5	0.4	0.01	0.95	0.625	0.6
5	0.4	0.01	0.95	0.625	0.6
5	0.4	0.01	0.95	0.625	0.6
5	0.4	0.01	0.95	0.625	0.6
5	0.4	0.01	0.95	0.625	0.8
5	0.4	0.01	0.95	0.625	0.8
5	0.4	0.01	0.95	0.625	0.8
5	0.4	0.01	0.95	0.625	0.8
5	0.4	0.01	0.95	0.625	0.8
5	0.4	0.01	0.95	0.750	0.6
5	0.4	0.01	0.95	0.750	0.6
5	0.4	0.01	0.95	0.750	0.6
5	0.4	0.01	0.95	0.750	0.6
5	0.4	0.01	0.95	0.750	0.6
5	0.4	0.01	0.95	0.750	0.8
5	0.4	0.01	0.95	0.750	0.8
5	0.4	0.01	0.95	0.750	0.8
5	0.4	0.01	0.95	0.750	0.8
5	0.4	0.01	0.95	0.750	0.8
5	0.4	0.01	0.95	0.875	0.6
5	0.4	0.01	0.95	0.875	0.6
5	0.4	0.01	0.95	0.875	0.6
5	0.4	0.01	0.95	0.875	0.6
5	0.4	0.01	0.95	0.875	0.6
5	0.4	0.01	0.95	0.875	0.8
5	0.4	0.01	0.95	0.875	0.8
5	0.4	0.01	0.95	0.875	0.8
5	0.4	0.01	0.95	0.875	0.8
5	0.4	0.01	0.95	0.875	0.8
5	0.4	0.01	0.95	1.000	0.6
5	0.4	0.01	0.95	1.000	0.6
5	0.4	0.01	0.95	1.000	0.6
5	0.4	0.01	0.95	1.000	0.6

5	0.4	0.01	0.95	1.000	0.6
5	0.4	0.01	0.95	1.000	0.8
5	0.4	0.01	0.95	1.000	0.8
5	0.4	0.01	0.95	1.000	0.8
5	0.4	0.01	0.95	1.000	0.8
5	0.4	0.01	0.95	1.000	0.8
5	0.4	0.50	0.05	0.500	0.6
5	0.4	0.50	0.05	0.500	0.6
5	0.4	0.50	0.05	0.500	0.6
5	0.4	0.50	0.05	0.500	0.6
5	0.4	0.50	0.05	0.500	0.6
5	0.4	0.50	0.05	0.500	0.8
5	0.4	0.50	0.05	0.500	0.8
5	0.4	0.50	0.05	0.500	0.8
5	0.4	0.50	0.05	0.500	0.8
5	0.4	0.50	0.05	0.500	0.8
5	0.4	0.50	0.05	0.625	0.6
5	0.4	0.50	0.05	0.625	0.6
5	0.4	0.50	0.05	0.625	0.6
5	0.4	0.50	0.05	0.625	0.6
5	0.4	0.50	0.05	0.625	0.6
5	0.4	0.50	0.05	0.625	0.8
5	0.4	0.50	0.05	0.625	0.8
5	0.4	0.50	0.05	0.625	0.8
5	0.4	0.50	0.05	0.625	0.8
5	0.4	0.50	0.05	0.625	0.8
5	0.4	0.50	0.05	0.750	0.6
5	0.4	0.50	0.05	0.750	0.6
5	0.4	0.50	0.05	0.750	0.6
5	0.4	0.50	0.05	0.750	0.6
5	0.4	0.50	0.05	0.750	0.6
5	0.4	0.50	0.05	0.750	0.8
5	0.4	0.50	0.05	0.750	0.8
5	0.4	0.50	0.05	0.750	0.8
5	0.4	0.50	0.05	0.750	0.8
5	0.4	0.50	0.05	0.750	0.8
5	0.4	0.50	0.05	0.875	0.6
5	0.4	0.50	0.05	0.875	0.6
5	0.4	0.50	0.05	0.875	0.6
5	0.4	0.50	0.05	0.875	0.6
5	0.4	0.50	0.05	0.875	0.6
5	0.4	0.50	0.05	0.875	0.8
5	0.4	0.50	0.05	0.875	0.8
5	0.4	0.50	0.05	0.875	0.8
5	0.4	0.50	0.05	0.875	0.8
5	0.4	0.50	0.05	0.875	0.8
5	0.4	0.50	0.05	1.000	0.6
5	0.4	0.50	0.05	1.000	0.6
5	0.4	0.50	0.05	1.000	0.6
5	0.4	0.50	0.05	1.000	0.6
5	0.4	0.50	0.05	1.000	0.6
5	0.4	0.50	0.05	1.000	0.8
5	0.4	0.50	0.05	1.000	0.8
5	0.4	0.50	0.05	1.000	0.8

```
5          0.4  0.50       0.05       1.000      0.8
5          0.4  0.50       0.05       1.000      0.8
5          0.4  0.50       0.95       0.500      0.6
5          0.4  0.50       0.95       0.500      0.6
5          0.4  0.50       0.95       0.500      0.6
5          0.4  0.50       0.95       0.500      0.6
5          0.4  0.50       0.95       0.500      0.6
5          0.4  0.50       0.95       0.500      0.8
5          0.4  0.50       0.95       0.500      0.8
5          0.4  0.50       0.95       0.500      0.8
5          0.4  0.50       0.95       0.500      0.8
5          0.4  0.50       0.95       0.500      0.8
5          0.4  0.50       0.95       0.625      0.6
5          0.4  0.50       0.95       0.625      0.6
5          0.4  0.50       0.95       0.625      0.6
5          0.4  0.50       0.95       0.625      0.6
5          0.4  0.50       0.95       0.625      0.6
5          0.4  0.50       0.95       0.625      0.8
5          0.4  0.50       0.95       0.625      0.8
5          0.4  0.50       0.95       0.625      0.8
5          0.4  0.50       0.95       0.625      0.8
5          0.4  0.50       0.95       0.625      0.8
5          0.4  0.50       0.95       0.750      0.6
5          0.4  0.50       0.95       0.750      0.6
5          0.4  0.50       0.95       0.750      0.6
5          0.4  0.50       0.95       0.750      0.6
5          0.4  0.50       0.95       0.750      0.6
5          0.4  0.50       0.95       0.750      0.8
5          0.4  0.50       0.95       0.750      0.8
5          0.4  0.50       0.95       0.750      0.8
5          0.4  0.50       0.95       0.750      0.8
5          0.4  0.50       0.95       0.750      0.8
5          0.4  0.50       0.95       0.875      0.6
5          0.4  0.50       0.95       0.875      0.6
5          0.4  0.50       0.95       0.875      0.6
5          0.4  0.50       0.95       0.875      0.6
5          0.4  0.50       0.95       0.875      0.6
5          0.4  0.50       0.95       0.875      0.8
5          0.4  0.50       0.95       0.875      0.8
5          0.4  0.50       0.95       0.875      0.8
5          0.4  0.50       0.95       0.875      0.8
5          0.4  0.50       0.95       0.875      0.8
5          0.4  0.50       0.95       1.000      0.6
5          0.4  0.50       0.95       1.000      0.6
5          0.4  0.50       0.95       1.000      0.6
5          0.4  0.50       0.95       1.000      0.6
5          0.4  0.50       0.95       1.000      0.6
5          0.4  0.50       0.95       1.000      0.8
5          0.4  0.50       0.95       1.000      0.8
5          0.4  0.50       0.95       1.000      0.8
5          0.4  0.50       0.95       1.000      0.8
5          0.4  0.50       0.95       1.000      0.8
nrounds  ROC        Sens       Spec
 50      0.8960186  0.8662454  0.7515152
```

100	0.8902421	0.8720147	0.7335595
150	0.8886492	0.8604945	0.7365445
200	0.8892948	0.8604945	0.7455450
250	0.8904725	0.8700366	0.7334690
50	0.8928439	0.8738095	0.7484848
100	0.8907852	0.8662271	0.7455450
150	0.8892761	0.8528388	0.7485753
200	0.8899604	0.8604762	0.7455902
250	0.8903281	0.8604762	0.7455902
50	0.8934964	0.8719597	0.7606513
100	0.8911244	0.8623993	0.7484848
150	0.8924696	0.8662271	0.7335142
200	0.8895248	0.8623810	0.7335142
250	0.8911664	0.8662454	0.7425147
50	0.8961702	0.8757692	0.7635459
100	0.8933179	0.8681319	0.7426504
150	0.8932278	0.8681319	0.7487110
200	0.8928067	0.8662088	0.7576662
250	0.8925213	0.8719414	0.7456807
50	0.8955663	0.8700183	0.7485301
100	0.8916898	0.8719414	0.7246043
150	0.8904588	0.8719414	0.7305292
200	0.8897593	0.8738645	0.7245138
250	0.8871142	0.8662271	0.7245590
50	0.8974020	0.8834066	0.7546359
100	0.8927987	0.8681136	0.7394392
150	0.8919847	0.8719597	0.7394392
200	0.8918237	0.8662088	0.7395296
250	0.8906771	0.8681136	0.7396201
50	0.8995255	0.8872344	0.7425147
100	0.8942583	0.8719414	0.7454998
150	0.8921111	0.8700549	0.7575758
200	0.8917898	0.8700366	0.7456355
250	0.8920570	0.8758059	0.7365898
50	0.8990379	0.8719414	0.7514699
100	0.8946386	0.8719597	0.7394844
150	0.8907695	0.8681319	0.7394844
200	0.8910188	0.8643040	0.7425147
250	0.8906181	0.8662271	0.7515152
50	0.8985021	0.8815018	0.7696065
100	0.8959032	0.8795971	0.7665762
150	0.8946196	0.8643223	0.7636364
200	0.8937872	0.8662271	0.7606513
250	0.8931570	0.8662088	0.7576662
50	0.8972702	0.8776374	0.7516056
100	0.8959721	0.8757326	0.7515152
150	0.8941081	0.8680952	0.7665310
200	0.8934957	0.8662088	0.7575758
250	0.8929250	0.8662088	0.7575758
50	0.8895502	0.8662088	0.7394392
100	0.8894556	0.8681319	0.7276346
150	0.8913948	0.8719048	0.7306196
200	0.8908883	0.8566667	0.7486658
250	0.8899983	0.8605128	0.7576662

```
 50      0.8940406  0.8605128  0.7634555
100      0.8917031  0.8623993  0.7425147
150      0.8886312  0.8662454  0.7545002
200      0.8900581  0.8700549  0.7425147
250      0.8902073  0.8623993  0.7635459
 50      0.8906868  0.8661538  0.7516056
100      0.8917738  0.8604945  0.7547264
150      0.8878802  0.8643223  0.7337404
200      0.8869544  0.8547619  0.7336499
250      0.8877100  0.8624176  0.7364993
 50      0.8913239  0.8585531  0.7456355
100      0.8883527  0.8528571  0.7275893
150      0.8854102  0.8528388  0.7546811
200      0.8879293  0.8566850  0.7516508
250      0.8819186  0.8528205  0.7395296
 50      0.8958286  0.8738828  0.7665762
100      0.8901497  0.8604762  0.7575758
150      0.8875836  0.8604579  0.7606513
200      0.8858177  0.8604762  0.7515152
250      0.8852822  0.8604762  0.7606513
 50      0.8942257  0.8738828  0.7604704
100      0.8916720  0.8738828  0.7456807
150      0.8900831  0.8643407  0.7335595
200      0.8881867  0.8642857  0.7426504
250      0.8843309  0.8585714  0.7486205
 50      0.8966043  0.8719963  0.7394844
100      0.8922880  0.8681502  0.7424695
150      0.8922361  0.8681136  0.7575758
200      0.8901907  0.8662454  0.7425599
250      0.8881721  0.8604762  0.7335142
 50      0.8963299  0.8815201  0.7604704
100      0.8930996  0.8681136  0.7515604
150      0.8906119  0.8662271  0.7454998
200      0.8889508  0.8643407  0.7394844
250      0.8870663  0.8605128  0.7364993
 50      0.8976866  0.8662271  0.7666667
100      0.8935327  0.8623993  0.7635459
150      0.8929162  0.8700549  0.7635459
200      0.8919539  0.8643223  0.7455450
250      0.8912568  0.8605128  0.7454998
 50      0.8978446  0.8795971  0.7636364
100      0.8944001  0.8719414  0.7636364
150      0.8928476  0.8681136  0.7576662
200      0.8927190  0.8643040  0.7485301
250      0.8914661  0.8604945  0.7485301
 50      0.8814187  0.8643773  0.7305744
100      0.8882077  0.8720513  0.7187698
150      0.8881343  0.8605861  0.7307101
200      0.8907243  0.8624542  0.7306649
250      0.8916138  0.8624176  0.7456355
 50      0.8813367  0.8643956  0.7185889
100      0.8828596  0.8586447  0.7305744
150      0.8849683  0.8586447  0.7456807
200      0.8853104  0.8605495  0.7427861
```

250	0.8901176	0.8682051	0.7186341
50	0.8807687	0.8682051	0.7245590
100	0.8834983	0.8757875	0.7126187
150	0.8848898	0.8662454	0.7215287
200	0.8916912	0.8738828	0.7246043
250	0.8932946	0.8682234	0.7246043
50	0.8826708	0.8586447	0.7156038
100	0.8847788	0.8624725	0.7245590
150	0.8880619	0.8586264	0.7364993
200	0.8886680	0.8605495	0.7245590
250	0.8904304	0.8643590	0.7215739
50	0.8803112	0.8701648	0.7095884
100	0.8855791	0.8624359	0.7456807
150	0.8892957	0.8681502	0.7487110
200	0.8897310	0.8796520	0.7094527
250	0.8905334	0.8738828	0.7397558
50	0.8789294	0.8509707	0.7487110
100	0.8866346	0.8643773	0.7335142
150	0.8871774	0.8682051	0.7424695
200	0.8909587	0.8681685	0.7335142
250	0.8916555	0.8681868	0.7364993
50	0.8776856	0.8643773	0.7155586
100	0.8862897	0.8586447	0.7334690
150	0.8863310	0.8701099	0.7335142
200	0.8910240	0.8662637	0.7246495
250	0.8908308	0.8662821	0.7216644
50	0.8843664	0.8586264	0.7245590
100	0.8870692	0.8682051	0.7215739
150	0.8881980	0.8643773	0.7215739
200	0.8898072	0.8643773	0.7275441
250	0.8905794	0.8643956	0.7514247
50	0.8771755	0.8643590	0.7126187
100	0.8854541	0.8719780	0.7336952
150	0.8869432	0.8624359	0.7396653
200	0.8906025	0.8681319	0.7516508
250	0.8929665	0.8643223	0.7336952
50	0.8706939	0.8567216	0.7245590
100	0.8814253	0.8586264	0.7305292
150	0.8834907	0.8605495	0.7305292
200	0.8889167	0.8624359	0.7215739
250	0.8926936	0.8586081	0.7275441
50	0.8939209	0.8681319	0.7726368
100	0.8923735	0.8643040	0.7636816
150	0.8896151	0.8642857	0.7426052
200	0.8848989	0.8681319	0.7395749
250	0.8843985	0.8528205	0.7545907
50	0.8942797	0.8776740	0.7454998
100	0.8895261	0.8662271	0.7515152
150	0.8885317	0.8566300	0.7364993
200	0.8865500	0.8508791	0.7604704
250	0.8851218	0.8547619	0.7575305
50	0.8966056	0.8776923	0.7575305
100	0.8929390	0.8662088	0.7485753
150	0.8890445	0.8662637	0.7454998

```
200       0.8873671  0.8643407  0.7335595
250       0.8873579  0.8605128  0.7545455
 50       0.8915880  0.8719597  0.7485301
100       0.8878300  0.8662088  0.7486205
150       0.8870560  0.8661905  0.7515604
200       0.8847473  0.8604579  0.7275441
250       0.8847297  0.8566850  0.7394844
 50       0.8922520  0.8700549  0.7365445
100       0.8908229  0.8642857  0.7485301
150       0.8890761  0.8719780  0.7425599
200       0.8885146  0.8604579  0.7244686
250       0.8893073  0.8681319  0.7365898
 50       0.8940635  0.8739011  0.7455450
100       0.8881689  0.8662271  0.7336499
150       0.8882711  0.8604945  0.7485301
200       0.8876211  0.8624176  0.7396201
250       0.8852502  0.8585897  0.7366350
 50       0.8989533  0.8738828  0.7515152
100       0.8943027  0.8700183  0.7455902
150       0.8914434  0.8623810  0.7395749
200       0.8892075  0.8662271  0.7365898
250       0.8880719  0.8623993  0.7365898
 50       0.8958333  0.8681136  0.7515152
100       0.8938780  0.8661905  0.7485301
150       0.8918562  0.8662088  0.7485753
200       0.8882081  0.8547253  0.7395749
250       0.8850397  0.8585348  0.7365445
 50       0.8982434  0.8796154  0.7546811
100       0.8955081  0.8662271  0.7665762
150       0.8935615  0.8623993  0.7575305
200       0.8925528  0.8643040  0.7515152
250       0.8915784  0.8604945  0.7424695
 50       0.8982506  0.8796520  0.7606513
100       0.8951454  0.8700183  0.7606061
150       0.8939735  0.8642857  0.7606513
200       0.8931625  0.8662088  0.7516508
250       0.8918505  0.8623993  0.7485301
 50       0.8915900  0.8605311  0.7724559
100       0.8877212  0.8585714  0.7575305
150       0.8896819  0.8682051  0.7454545
200       0.8863578  0.8604945  0.7364089
250       0.8876215  0.8681502  0.7395749
 50       0.8927244  0.8777106  0.7247399
100       0.8893058  0.8642857  0.7246495
150       0.8849874  0.8623993  0.7335142
200       0.8855643  0.8566850  0.7546359
250       0.8871618  0.8643223  0.7515152
 50       0.8946434  0.8586447  0.7786522
100       0.8894088  0.8662637  0.7546359
150       0.8868290  0.8509158  0.7456807
200       0.8920978  0.8662271  0.7576210
250       0.8912428  0.8585714  0.7425599
 50       0.8921332  0.8700366  0.7394844
100       0.8879189  0.8585714  0.7426052
```

150	0.8869561	0.8509158	0.7454545
200	0.8873148	0.8508791	0.7365445
250	0.8873887	0.8528022	0.7305292
50	0.8949434	0.8757509	0.7725916
100	0.8937414	0.8643407	0.7605156
150	0.8915793	0.8662454	0.7395749
200	0.8901568	0.8623993	0.7484848
250	0.8902786	0.8643223	0.7455450
50	0.8893264	0.8661722	0.7305744
100	0.8899208	0.8681136	0.7364541
150	0.8897547	0.8662271	0.7365898
200	0.8882821	0.8700366	0.7366350
250	0.8880560	0.8662454	0.7365898
50	0.8937727	0.8700366	0.7426052
100	0.8937521	0.8681136	0.7576210
150	0.8923178	0.8681136	0.7486205
200	0.8906615	0.8680952	0.7605608
250	0.8887715	0.8719414	0.7365445
50	0.8942624	0.8720147	0.7393939
100	0.8896887	0.8643223	0.7576210
150	0.8890452	0.8604945	0.7396201
200	0.8897712	0.8643407	0.7396653
250	0.8905601	0.8623993	0.7396653
50	0.8978452	0.8758791	0.7605608
100	0.8953020	0.8777656	0.7545002
150	0.8938747	0.8739194	0.7545002
200	0.8935512	0.8681319	0.7515152
250	0.8932604	0.8662088	0.7545002
50	0.8951263	0.8720330	0.7514699
100	0.8930260	0.8719780	0.7545455
150	0.8920356	0.8700549	0.7515152
200	0.8914544	0.8662271	0.7575758
250	0.8917726	0.8662088	0.7485301
50	0.8877858	0.8585531	0.7394844
100	0.8834739	0.8548168	0.7395749
150	0.8824606	0.8490476	0.7395749
200	0.8811762	0.8432784	0.7364541
250	0.8793631	0.8490110	0.7186341
50	0.8880160	0.8738828	0.7485753
100	0.8858115	0.8604762	0.7425147
150	0.8837223	0.8508974	0.7336047
200	0.8839874	0.8566850	0.7305292
250	0.8829593	0.8451648	0.7426956
50	0.8921796	0.8662637	0.7486658
100	0.8871346	0.8719963	0.7635007
150	0.8851497	0.8528205	0.7455450
200	0.8842705	0.8586081	0.7304839
250	0.8810212	0.8566850	0.7336047
50	0.8942068	0.8643223	0.7425147
100	0.8885623	0.8547619	0.7426052
150	0.8835344	0.8643040	0.7396201
200	0.8839662	0.8624542	0.7335142
250	0.8826506	0.8471245	0.7576210
50	0.8932944	0.8719048	0.7486205

```
100       0.8880039  0.8624176  0.7364541
150       0.8888306  0.8604762  0.7275441
200       0.8858432  0.8566667  0.7334690
250       0.8861773  0.8547619  0.7364993
 50       0.8963797  0.8719414  0.7546359
100       0.8922123  0.8681502  0.7455902
150       0.8896388  0.8642674  0.7426052
200       0.8866220  0.8547436  0.7425599
250       0.8861534  0.8547802  0.7605608
 50       0.8947184  0.8739194  0.7515604
100       0.8924027  0.8700549  0.7607417
150       0.8885858  0.8566850  0.7426956
200       0.8871351  0.8528205  0.7455450
250       0.8855018  0.8586081  0.7516508
 50       0.8957748  0.8700916  0.7455450
100       0.8884349  0.8586081  0.7395749
150       0.8890503  0.8604945  0.7365898
200       0.8861753  0.8528755  0.7515604
250       0.8826666  0.8471062  0.7365445
 50       0.8970515  0.8796520  0.7515152
100       0.8936889  0.8681502  0.7485301
150       0.8920148  0.8719780  0.7545002
200       0.8901260  0.8643223  0.7454998
250       0.8894079  0.8624176  0.7394844
 50       0.8957298  0.8719231  0.7575758
100       0.8929660  0.8700183  0.7605608
150       0.8924368  0.8623993  0.7515604
200       0.8904343  0.8604579  0.7425599
250       0.8889957  0.8643223  0.7395749
 50       0.8810548  0.8720330  0.7186793
100       0.8849732  0.8700733  0.7516961
150       0.8898137  0.8719780  0.7367707
200       0.8920272  0.8758242  0.7246947
250       0.8932201  0.8776923  0.7397558
 50       0.8818885  0.8490659  0.7305292
100       0.8835176  0.8586264  0.7334690
150       0.8842600  0.8644139  0.7156490
200       0.8864050  0.8586264  0.7365898
250       0.8924991  0.8739560  0.7365898
 50       0.8786546  0.8624908  0.7186341
100       0.8830827  0.8547985  0.7457711
150       0.8895784  0.8643223  0.7517413
200       0.8895114  0.8719780  0.7426504
250       0.8912141  0.8719963  0.7396201
 50       0.8787671  0.8605495  0.7215287
100       0.8824074  0.8586264  0.7155586
150       0.8851986  0.8605311  0.7364993
200       0.8899456  0.8681685  0.7305744
250       0.8898379  0.8643407  0.7364993
 50       0.8839096  0.8701099  0.7155586
100       0.8889718  0.8662821  0.7215739
150       0.8941889  0.8700733  0.7274536
200       0.8951795  0.8739194  0.7394844
250       0.8954174  0.8739011  0.7635007
```

50	0.8842480	0.8605678	0.7245590
100	0.8868812	0.8643956	0.7334690
150	0.8884471	0.8681685	0.7425599
200	0.8905271	0.8700733	0.7395749
250	0.8952627	0.8700366	0.7485301
50	0.8754735	0.8624542	0.7308458
100	0.8875762	0.8739377	0.7335142
150	0.8899247	0.8739194	0.7275441
200	0.8937959	0.8700733	0.7425147
250	0.8954346	0.8681319	0.7575758
50	0.8828878	0.8624725	0.7185889
100	0.8833703	0.8643956	0.7335142
150	0.8882478	0.8624542	0.7365445
200	0.8896124	0.8528571	0.7666214
250	0.8929078	0.8739011	0.7484396
50	0.8796651	0.8566850	0.7485301
100	0.8887883	0.8720147	0.7245590
150	0.8913975	0.8643407	0.7455902
200	0.8928466	0.8643407	0.7425599
250	0.8953825	0.8624542	0.7516508
50	0.8796171	0.8586264	0.7337856
100	0.8829549	0.8528571	0.7517413
150	0.8879400	0.8624725	0.7275893
200	0.8922665	0.8566850	0.7726368
250	0.8926609	0.8605311	0.7635007
50	0.8914152	0.8681502	0.7425599
100	0.8917110	0.8661538	0.7516961
150	0.8868219	0.8451099	0.7726368
200	0.8878267	0.8623260	0.7427408
250	0.8829176	0.8509341	0.7275441
50	0.8883079	0.8700549	0.7484396
100	0.8867841	0.8681319	0.7485753
150	0.8866701	0.8681502	0.7574853
200	0.8819591	0.8509341	0.7515604
250	0.8799618	0.8432234	0.7726820
50	0.8875800	0.8643040	0.7305744
100	0.8873730	0.8623993	0.7546359
150	0.8833609	0.8509524	0.7397105
200	0.8864373	0.8509341	0.7456355
250	0.8855091	0.8566850	0.7516508
50	0.8915242	0.8681136	0.7335595
100	0.8884584	0.8662088	0.7155586
150	0.8862508	0.8585897	0.7395749
200	0.8814699	0.8604945	0.7425147
250	0.8835586	0.8528938	0.7305292
50	0.8937068	0.8739011	0.7393939
100	0.8878414	0.8643040	0.7455902
150	0.8889541	0.8605128	0.7395296
200	0.8863178	0.8585531	0.7335595
250	0.8862261	0.8604945	0.7365898
50	0.8921223	0.8662454	0.7515152
100	0.8898465	0.8643040	0.7455902
150	0.8866901	0.8624359	0.7455450
200	0.8854159	0.8585897	0.7395296

```
250       0.8862791  0.8604945  0.7486658
 50       0.8929149  0.8777106  0.7514699
100       0.8885419  0.8700183  0.7366350
150       0.8894211  0.8738828  0.7335595
200       0.8857033  0.8623810  0.7335595
250       0.8863838  0.8623993  0.7335595
 50       0.8936962  0.8700549  0.7545907
100       0.8883653  0.8623626  0.7395749
150       0.8894888  0.8662271  0.7305744
200       0.8880530  0.8566667  0.7306196
250       0.8872601  0.8585714  0.7426052
 50       0.8965367  0.8834249  0.7726368
100       0.8939373  0.8681319  0.7696065
150       0.8929070  0.8681136  0.7545455
200       0.8916805  0.8643223  0.7454998
250       0.8899625  0.8585897  0.7545002
 50       0.8969907  0.8853480  0.7725916
100       0.8933976  0.8719597  0.7575305
150       0.8923237  0.8700366  0.7514699
200       0.8903984  0.8643223  0.7484848
250       0.8893560  0.8643407  0.7424695
 50       0.8870864  0.8566300  0.7545002
100       0.8819264  0.8585531  0.7483944
150       0.8840247  0.8624176  0.7424695
200       0.8851334  0.8490842  0.7665310
250       0.8866218  0.8509707  0.7603799
 50       0.8898196  0.8566300  0.7545002
100       0.8859041  0.8489744  0.7424695
150       0.8825826  0.8471062  0.7484396
200       0.8843048  0.8413187  0.7573496
250       0.8813454  0.8394139  0.7573496
 50       0.8884097  0.8681136  0.7485753
100       0.8845441  0.8470879  0.7545002
150       0.8829837  0.8451465  0.7605156
200       0.8819826  0.8489927  0.7574401
250       0.8787405  0.8394505  0.7574853
 50       0.8905789  0.8508974  0.7663953
100       0.8809967  0.8451832  0.7665762
150       0.8800792  0.8432418  0.7575305
200       0.8800815  0.8413553  0.7695161
250       0.8770165  0.8433150  0.7545002
 50       0.8893421  0.8566484  0.7544098
100       0.8872131  0.8451648  0.7604251
150       0.8845899  0.8547802  0.7634555
200       0.8852965  0.8509341  0.7543645
250       0.8860980  0.8471429  0.7483944
 50       0.8894545  0.8604762  0.7335595
100       0.8873298  0.8413370  0.7725464
150       0.8854458  0.8451832  0.7695613
200       0.8806062  0.8375275  0.7635911
250       0.8785520  0.8452015  0.7545455
 50       0.8949493  0.8623810  0.7425147
100       0.8891604  0.8547436  0.7544550
150       0.8859850  0.8547802  0.7454093
```

200	0.8835057	0.8528755	0.7573496
250	0.8822820	0.8451832	0.7544550
50	0.8914735	0.8509158	0.7545907
100	0.8905503	0.8490293	0.7455902
150	0.8869762	0.8452198	0.7395296
200	0.8869591	0.8490476	0.7664405
250	0.8827771	0.8547802	0.7605156
50	0.8897349	0.8528022	0.7484848
100	0.8876347	0.8528571	0.7484396
150	0.8842681	0.8509341	0.7574853
200	0.8825288	0.8471062	0.7515152
250	0.8811564	0.8394689	0.7515152
50	0.8894246	0.8471245	0.7515604
100	0.8875650	0.8490476	0.7635007
150	0.8839220	0.8547985	0.7605156
200	0.8816424	0.8509524	0.7574401
250	0.8823189	0.8509524	0.7484396
50	0.8956411	0.8604945	0.7545907
100	0.8888084	0.8566117	0.7484848
150	0.8807762	0.8432418	0.7364993
200	0.8764800	0.8413187	0.7484848
250	0.8729031	0.8356227	0.7663953
50	0.8833133	0.8623993	0.7514247
100	0.8793874	0.8585897	0.7306196
150	0.8744641	0.8604762	0.7336047
200	0.8738278	0.8662637	0.7216192
250	0.8706799	0.8585897	0.7395749
50	0.8882589	0.8604762	0.7245590
100	0.8821969	0.8470696	0.7604251
150	0.8807598	0.8547253	0.7634102
200	0.8756879	0.8509341	0.7394392
250	0.8736603	0.8490110	0.7483944
50	0.8851923	0.8662088	0.7306649
100	0.8778801	0.8394139	0.7395749
150	0.8753168	0.8336630	0.7394392
200	0.8746855	0.8413919	0.7395749
250	0.8700642	0.8395238	0.7275441
50	0.8896664	0.8585348	0.7425599
100	0.8867696	0.8585897	0.7515152
150	0.8820080	0.8566850	0.7544550
200	0.8780824	0.8490476	0.7604251
250	0.8780596	0.8528388	0.7485301
50	0.8886907	0.8585348	0.7394392
100	0.8795280	0.8490110	0.7515152
150	0.8761614	0.8356410	0.7695161
200	0.8729670	0.8490659	0.7515604
250	0.8731043	0.8586081	0.7635007
50	0.8882179	0.8509158	0.7544550
100	0.8820888	0.8471062	0.7394844
150	0.8803373	0.8490293	0.7514699
200	0.8799051	0.8528938	0.7514699
250	0.8729149	0.8471245	0.7574853
50	0.8891475	0.8528388	0.7635459
100	0.8874869	0.8547619	0.7635459

```
150       0.8825583  0.8471429  0.7515604
200       0.8793532  0.8490476  0.7604704
250       0.8767166  0.8528938  0.7514699
 50       0.8920368  0.8566484  0.7545907
100       0.8867237  0.8509158  0.7544550
150       0.8852221  0.8490476  0.7514699
200       0.8824523  0.8528938  0.7544550
250       0.8790420  0.8529121  0.7544550
 50       0.8928271  0.8585714  0.7545455
100       0.8871123  0.8490110  0.7575305
150       0.8821841  0.8471429  0.7485301
200       0.8793339  0.8433516  0.7515152
250       0.8761436  0.8490842  0.7454998
 50       0.8894493  0.8758242  0.7275441
100       0.8889917  0.8815385  0.7334690
150       0.8867871  0.8834249  0.7335142
200       0.8876455  0.8719597  0.7394844
250       0.8886562  0.8738828  0.7515604
 50       0.8929338  0.8567033  0.7696065
100       0.8934513  0.8624908  0.7665762
150       0.8926692  0.8700733  0.7515152
200       0.8931339  0.8700733  0.7515152
250       0.8929597  0.8662454  0.7574853
 50       0.8877562  0.8776923  0.7185436
100       0.8919126  0.8624176  0.7335595
150       0.8906440  0.8834249  0.7245590
200       0.8947498  0.8758059  0.7336499
250       0.8957928  0.8662454  0.7455902
 50       0.8935948  0.8643590  0.7395296
100       0.8932594  0.8662454  0.7484848
150       0.8960479  0.8815385  0.7574853
200       0.8949877  0.8719963  0.7634555
250       0.8945979  0.8739011  0.7606061
 50       0.8918818  0.8834982  0.7185436
100       0.8927459  0.8854029  0.7364541
150       0.8942742  0.8796703  0.7514247
200       0.8976232  0.8739011  0.7514247
250       0.8959767  0.8623810  0.7454093
 50       0.8942155  0.8643040  0.7395296
100       0.8947310  0.8758059  0.7455450
150       0.8953093  0.8757875  0.7455450
200       0.8951337  0.8719780  0.7485301
250       0.8969573  0.8795971  0.7515604
 50       0.8925760  0.8873260  0.7183627
100       0.8934774  0.8796520  0.7304387
150       0.8966857  0.8681502  0.7395296
200       0.8933126  0.8700733  0.7335142
250       0.8941509  0.8700366  0.7334238
 50       0.8902592  0.8624542  0.7545455
100       0.8947281  0.8624359  0.7515152
150       0.8952660  0.8720147  0.7574853
200       0.8962485  0.8700916  0.7544098
250       0.8943971  0.8739011  0.7573948
 50       0.8956945  0.8834066  0.7125735
```

100	0.8943701	0.8834799	0.7094980
150	0.8964299	0.8815934	0.7334238
200	0.8975647	0.8796520	0.7334238
250	0.8978712	0.8777289	0.7484848
50	0.8908819	0.8566850	0.7545002
100	0.8926490	0.8642857	0.7664405
150	0.8950922	0.8681319	0.7575305
200	0.8945372	0.8604945	0.7635007
250	0.8938782	0.8623993	0.7604704
50	0.8856098	0.8585714	0.7455450
100	0.8785268	0.8394505	0.7484848
150	0.8761538	0.8375641	0.7454998
200	0.8730808	0.8356777	0.7333333
250	0.8717934	0.8299451	0.7573948
50	0.8859714	0.8604762	0.7544550
100	0.8806480	0.8547436	0.7424695
150	0.8762264	0.8433150	0.7304387
200	0.8710631	0.8452015	0.7214383
250	0.8682954	0.8490659	0.7364541
50	0.8888091	0.8547436	0.7456355
100	0.8817186	0.8567033	0.7513795
150	0.8786946	0.8471062	0.7664405
200	0.8757249	0.8432967	0.7605156
250	0.8744538	0.8490476	0.7484848
50	0.8860284	0.8547070	0.7397105
100	0.8803712	0.8489377	0.7305744
150	0.8803688	0.8586081	0.7303935
200	0.8751201	0.8470879	0.7304839
250	0.8748120	0.8451648	0.7334690
50	0.8899618	0.8604762	0.7425599
100	0.8849160	0.8566667	0.7395749
150	0.8825756	0.8528388	0.7455450
200	0.8799761	0.8509341	0.7575758
250	0.8749545	0.8471429	0.7485753
50	0.8855705	0.8528388	0.7515152
100	0.8837721	0.8566484	0.7484848
150	0.8749524	0.8413736	0.7394844
200	0.8784577	0.8452015	0.7424242
250	0.8723507	0.8471612	0.7454998
50	0.8950504	0.8662088	0.7485753
100	0.8891133	0.8471062	0.7455450
150	0.8839638	0.8432967	0.7395749
200	0.8816139	0.8452198	0.7394844
250	0.8772813	0.8356593	0.7455450
50	0.8910644	0.8585531	0.7365898
100	0.8837329	0.8413919	0.7515152
150	0.8792708	0.8356410	0.7545002
200	0.8758740	0.8394872	0.7454998
250	0.8719113	0.8414103	0.7574401
50	0.8949855	0.8509158	0.7575305
100	0.8869039	0.8509524	0.7394392
150	0.8809980	0.8528571	0.7454093
200	0.8779712	0.8509707	0.7514699
250	0.8761236	0.8528938	0.7545002

```
 50      0.8916234  0.8566667  0.7485301
100      0.8866136  0.8490476  0.7485753
150      0.8835647  0.8490476  0.7485301
200      0.8802567  0.8509524  0.7515152
250      0.8765158  0.8452381  0.7425599
 50      0.8851564  0.8566300  0.7486205
100      0.8828399  0.8432784  0.7545455
150      0.8833347  0.8470696  0.7485301
200      0.8817081  0.8470879  0.7394392
250      0.8856167  0.8528571  0.7633650
 50      0.8827114  0.8509158  0.7545002
100      0.8804599  0.8604945  0.7336047
150      0.8797411  0.8432967  0.7454545
200      0.8807246  0.8528755  0.7453641
250      0.8808586  0.8509524  0.7484848
 50      0.8896835  0.8604212  0.7334238
100      0.8840985  0.8470147  0.7635911
150      0.8830361  0.8451099  0.7574853
200      0.8798687  0.8394139  0.7514699
250      0.8785469  0.8355861  0.7485301
 50      0.8822562  0.8489744  0.7514247
100      0.8823031  0.8452015  0.7395296
150      0.8783648  0.8413736  0.7424242
200      0.8747794  0.8413736  0.7395296
250      0.8781422  0.8394689  0.7394844
 50      0.8840467  0.8509158  0.7485301
100      0.8815501  0.8509341  0.7515152
150      0.8815141  0.8432784  0.7694708
200      0.8811239  0.8528571  0.7574401
250      0.8785740  0.8413553  0.7574853
 50      0.8823301  0.8508791  0.7335595
100      0.8772256  0.8375275  0.7574853
150      0.8780782  0.8451832  0.7634555
200      0.8786545  0.8318315  0.7664405
250      0.8782126  0.8452381  0.7545002
 50      0.8906812  0.8566117  0.7694256
100      0.8837983  0.8490659  0.7423338
150      0.8812710  0.8375458  0.7513795
200      0.8809829  0.8452198  0.7663953
250      0.8800076  0.8433333  0.7694256
 50      0.8904736  0.8547253  0.7426052
100      0.8856627  0.8509158  0.7395296
150      0.8838442  0.8471245  0.7544550
200      0.8816672  0.8413553  0.7514699
250      0.8799792  0.8413919  0.7544550
 50      0.8942465  0.8604945  0.7574853
100      0.8868910  0.8528571  0.7634102
150      0.8860251  0.8490293  0.7544550
200      0.8837979  0.8471245  0.7545002
250      0.8836610  0.8509524  0.7545002
 50      0.8883605  0.8471245  0.7485301
100      0.8837139  0.8490476  0.7516508
150      0.8807189  0.8452015  0.7545002
200      0.8817294  0.8509707  0.7544550
```

250	0.8819627	0.8490842	0.7484848
50	0.8810505	0.8623993	0.7365445
100	0.8751769	0.8432784	0.7393487
150	0.8732224	0.8298901	0.7634102
200	0.8728150	0.8375092	0.7545002
250	0.8692578	0.8374725	0.7304387
50	0.8815280	0.8470513	0.7515604
100	0.8746921	0.8470696	0.7484848
150	0.8711357	0.8374908	0.7604704
200	0.8634775	0.8126557	0.7633198
250	0.8580147	0.8164652	0.7455450
50	0.8852803	0.8471062	0.7304387
100	0.8746568	0.8413736	0.7545002
150	0.8734351	0.8508791	0.7423790
200	0.8754933	0.8413919	0.7394392
250	0.8687120	0.8471429	0.7394392
50	0.8823711	0.8471245	0.7486205
100	0.8767226	0.8433516	0.7513795
150	0.8715087	0.8356227	0.7393487
200	0.8697199	0.8337179	0.7603799
250	0.8646341	0.8279670	0.7333333
50	0.8858899	0.8547253	0.7514699
100	0.8799260	0.8452015	0.7544550
150	0.8755377	0.8432967	0.7635459
200	0.8774498	0.8375458	0.7574401
250	0.8754659	0.8356227	0.7605156
50	0.8856033	0.8566667	0.7455450
100	0.8799103	0.8452381	0.7425147
150	0.8726749	0.8356777	0.7274989
200	0.8682341	0.8337546	0.7483944
250	0.8650614	0.8279853	0.7424242
50	0.8909604	0.8623993	0.7394844
100	0.8825970	0.8489927	0.7304839
150	0.8772553	0.8547985	0.7454998
200	0.8710884	0.8509707	0.7365445
250	0.8708473	0.8509707	0.7335595
50	0.8861437	0.8508974	0.7454093
100	0.8798061	0.8509707	0.7454545
150	0.8756286	0.8413736	0.7604704
200	0.8710503	0.8394689	0.7544098
250	0.8681918	0.8471978	0.7454093
50	0.8884938	0.8566850	0.7485301
100	0.8844536	0.8566667	0.7424695
150	0.8786979	0.8433150	0.7364541
200	0.8772931	0.8490659	0.7424242
250	0.8729036	0.8452381	0.7453641
50	0.8901883	0.8585531	0.7454998
100	0.8841901	0.8489927	0.7604251
150	0.8805794	0.8547985	0.7604704
200	0.8777762	0.8567216	0.7484396
250	0.8737585	0.8490659	0.7394392
50	0.8925985	0.8815751	0.7304387
100	0.8947393	0.8700549	0.7393487
150	0.8939503	0.8719597	0.7542741

```
200       0.8931661  0.8700183  0.7635459
250       0.8946338  0.8757692  0.7545907
 50       0.8929779  0.8623993  0.7574401
100       0.8953734  0.8681136  0.7424242
150       0.8962380  0.8719597  0.7606513
200       0.8954944  0.8681685  0.7636816
250       0.8934958  0.8681502  0.7395749
 50       0.8948839  0.8681502  0.7545002
100       0.8943179  0.8777289  0.7304387
150       0.8936824  0.8700549  0.7514247
200       0.8930129  0.8700366  0.7333786
250       0.8932773  0.8719414  0.7605156
 50       0.8927823  0.8739011  0.7395296
100       0.8924807  0.8662454  0.7603799
150       0.8919129  0.8776923  0.7574401
200       0.8942409  0.8739011  0.7573948
250       0.8958499  0.8777473  0.7635007
 50       0.8954339  0.8853663  0.7305292
100       0.8926795  0.8815751  0.7335142
150       0.8900592  0.8777106  0.7365898
200       0.8961953  0.8719414  0.7516056
250       0.8961223  0.8700549  0.7425599
 50       0.8902341  0.8661905  0.7546359
100       0.8899944  0.8700549  0.7576210
150       0.8939811  0.8700366  0.7635007
200       0.8915652  0.8661722  0.7576210
250       0.8924724  0.8700183  0.7515604
 50       0.8935585  0.8911172  0.7034374
100       0.8995839  0.8738828  0.7304387
150       0.8976970  0.8796337  0.7364541
200       0.8957017  0.8777289  0.7423790
250       0.8960456  0.8700733  0.7603799
 50       0.8915445  0.8662454  0.7484848
100       0.8953889  0.8624176  0.7425147
150       0.8943087  0.8643590  0.7395296
200       0.8937923  0.8681685  0.7454545
250       0.8943966  0.8757875  0.7484396
 50       0.8938828  0.8815568  0.7335142
100       0.8947836  0.8643223  0.7336499
150       0.8950215  0.8662271  0.7425147
200       0.8969815  0.8738645  0.7335142
250       0.8957846  0.8719414  0.7395749
 50       0.8924357  0.8681502  0.7606061
100       0.8925066  0.8719780  0.7426052
150       0.8950153  0.8738828  0.7575305
200       0.8935854  0.8739194  0.7635007
250       0.8927836  0.8662454  0.7545002
 50       0.8915208  0.8624176  0.7515604
100       0.8837116  0.8509524  0.7602895
150       0.8763063  0.8393956  0.7394844
200       0.8756314  0.8394322  0.7514247
250       0.8699025  0.8336813  0.7453641
 50       0.8870774  0.8509341  0.7664405
100       0.8727951  0.8566850  0.7305292
```

150	0.8728078	0.8394322	0.7424695
200	0.8718250	0.8375458	0.7274536
250	0.8624277	0.8432967	0.7246495
50	0.8871698	0.8585531	0.7365445
100	0.8812557	0.8432051	0.7575758
150	0.8794228	0.8451832	0.7575305
200	0.8733762	0.8337363	0.7663501
250	0.8768085	0.8356410	0.7485753
50	0.8827243	0.8546703	0.7305744
100	0.8774975	0.8356227	0.7425147
150	0.8740889	0.8241209	0.7454545
200	0.8718227	0.8222161	0.7335595
250	0.8624738	0.8203480	0.7455450
50	0.8896144	0.8623810	0.7425147
100	0.8819685	0.8566667	0.7514247
150	0.8758559	0.8585531	0.7364541
200	0.8743769	0.8509341	0.7545002
250	0.8721840	0.8356227	0.7544550
50	0.8831902	0.8528938	0.7245590
100	0.8775332	0.8395055	0.7423790
150	0.8688252	0.8432967	0.7303483
200	0.8706164	0.8318132	0.7334238
250	0.8658945	0.8298901	0.7514247
50	0.8891116	0.8547619	0.7574853
100	0.8818109	0.8509524	0.7515604
150	0.8782237	0.8471429	0.7545002
200	0.8746879	0.8509890	0.7455450
250	0.8714711	0.8548168	0.7484848
50	0.8903126	0.8623993	0.7664858
100	0.8856305	0.8586264	0.7454545
150	0.8813601	0.8490476	0.7484848
200	0.8773358	0.8490659	0.7394844
250	0.8751844	0.8529121	0.7544098
50	0.8905429	0.8604762	0.7454545
100	0.8866799	0.8528755	0.7454093
150	0.8822193	0.8490842	0.7514699
200	0.8789403	0.8395055	0.7544550
250	0.8744031	0.8452381	0.7514699
50	0.8925270	0.8547436	0.7514699
100	0.8861057	0.8471612	0.7454998
150	0.8817989	0.8490842	0.7544550
200	0.8772499	0.8548718	0.7514247
250	0.8746761	0.8510440	0.7454093
50	0.8773612	0.8414286	0.7425599
100	0.8774319	0.8452198	0.7484396
150	0.8781112	0.8432967	0.7425599
200	0.8759741	0.8490110	0.7455902
250	0.8738702	0.8432601	0.7603799
50	0.8869939	0.8508974	0.7424695
100	0.8789366	0.8394689	0.7514247
150	0.8794822	0.8528571	0.7574853
200	0.8756898	0.8414469	0.7634102
250	0.8730946	0.8375824	0.7484848
50	0.8816351	0.8489744	0.7514247

```
100       0.8755126  0.8394689  0.7454093
150       0.8712943  0.8375641  0.7423790
200       0.8725967  0.8471062  0.7423790
250       0.8733532  0.8433150  0.7484396
 50       0.8836192  0.8548352  0.7514699
100       0.8800251  0.8413919  0.7484396
150       0.8804442  0.8375824  0.7485301
200       0.8742086  0.8375641  0.7425147
250       0.8719754  0.8375641  0.7544550
 50       0.8807553  0.8528571  0.7396201
100       0.8761915  0.8413736  0.7426052
150       0.8744764  0.8298901  0.7455902
200       0.8744020  0.8356410  0.7484848
250       0.8692816  0.8375458  0.7485301
 50       0.8881806  0.8624176  0.7514247
100       0.8813797  0.8452564  0.7544550
150       0.8785106  0.8375824  0.7574853
200       0.8765233  0.8433333  0.7485753
250       0.8750440  0.8356593  0.7365898
 50       0.8856511  0.8509707  0.7455450
100       0.8797050  0.8471245  0.7515152
150       0.8754249  0.8509890  0.7544098
200       0.8752269  0.8433333  0.7483944
250       0.8737303  0.8471245  0.7454545
 50       0.8832421  0.8604945  0.7425147
100       0.8790793  0.8547802  0.7424695
150       0.8754271  0.8471245  0.7514247
200       0.8745218  0.8452381  0.7544098
250       0.8725992  0.8433333  0.7544098
 50       0.8900573  0.8623810  0.7604704
100       0.8844989  0.8547985  0.7574401
150       0.8784854  0.8471612  0.7663953
200       0.8772062  0.8433516  0.7574401
250       0.8767642  0.8452747  0.7544550
 50       0.8845411  0.8490659  0.7454093
100       0.8798849  0.8452564  0.7424695
150       0.8782216  0.8471795  0.7544098
200       0.8748938  0.8452747  0.7514247
250       0.8737741  0.8452747  0.7544098
 50       0.8816972  0.8547070  0.7365445
100       0.8728406  0.8432967  0.7454093
150       0.8641384  0.8356593  0.7334238
200       0.8589828  0.8356593  0.7364993
250       0.8620717  0.8394505  0.7425147
 50       0.8813518  0.8490293  0.7544550
100       0.8752863  0.8356227  0.7573948
150       0.8686373  0.8433516  0.7334238
200       0.8682788  0.8509524  0.7335595
250       0.8624169  0.8337729  0.7514699
 50       0.8805253  0.8547619  0.7364089
100       0.8769432  0.8452198  0.7514699
150       0.8713705  0.8394872  0.7364993
200       0.8686567  0.8452015  0.7305744
250       0.8645485  0.8299451  0.7395749
```

50	0.8786678	0.8414469	0.7334690
100	0.8748263	0.8432784	0.7484848
150	0.8673568	0.8432784	0.7394392
200	0.8686194	0.8394322	0.7514247
250	0.8615217	0.8279487	0.7545002
50	0.8827454	0.8528571	0.7453641
100	0.8757905	0.8490110	0.7454545
150	0.8693024	0.8471062	0.7544550
200	0.8642383	0.8356593	0.7425147
250	0.8634975	0.8261172	0.7364993
50	0.8831081	0.8624176	0.7394844
100	0.8729177	0.8509341	0.7303935
150	0.8695795	0.8356777	0.7454545
200	0.8672193	0.8376007	0.7335142
250	0.8643490	0.8337729	0.7335142
50	0.8861210	0.8509158	0.7574853
100	0.8782597	0.8529121	0.7393939
150	0.8744310	0.8356960	0.7425599
200	0.8712003	0.8375824	0.7305292
250	0.8680044	0.8356777	0.7245138
50	0.8840662	0.8605128	0.7424695
100	0.8764894	0.8529304	0.7364993
150	0.8708803	0.8376374	0.7454545
200	0.8663417	0.8262088	0.7425147
250	0.8654906	0.8280952	0.7335142
50	0.8874762	0.8490110	0.7635007
100	0.8817674	0.8567216	0.7665310
150	0.8748292	0.8491209	0.7424695
200	0.8714708	0.8376374	0.7364993
250	0.8697157	0.8357326	0.7274989
50	0.8846131	0.8452381	0.7453641
100	0.8783642	0.8433516	0.7303935
150	0.8739162	0.8510256	0.7334238
200	0.8731810	0.8414835	0.7393939
250	0.8692218	0.8414652	0.7244686
50	0.8913740	0.8700366	0.7514699
100	0.8954000	0.8681685	0.7513795
150	0.8938964	0.8719780	0.7274084
200	0.8911061	0.8719780	0.7394844
250	0.8910153	0.8700549	0.7454545
50	0.8931652	0.8700549	0.7515152
100	0.8908707	0.8585897	0.7545002
150	0.8921374	0.8662271	0.7425147
200	0.8916195	0.8662088	0.7425599
250	0.8928591	0.8642857	0.7485301
50	0.8928134	0.8777289	0.7275441
100	0.8915080	0.8777106	0.7365445
150	0.8909438	0.8585714	0.7365445
200	0.8908523	0.8604762	0.7365898
250	0.8898307	0.8605311	0.7364541
50	0.8914841	0.8700549	0.7514699
100	0.8925563	0.8739011	0.7394392
150	0.8926600	0.8604762	0.7545002
200	0.8930831	0.8643407	0.7455902

```
250       0.8940312  0.8643040  0.7425147
 50       0.8923161  0.8738828  0.7424695
100       0.8953911  0.8739011  0.7424695
150       0.8935505  0.8662637  0.7424695
200       0.8931446  0.8604396  0.7364993
250       0.8946466  0.8566484  0.7425147
 50       0.8941792  0.8719414  0.7424242
100       0.8971141  0.8700549  0.7393939
150       0.8945278  0.8623993  0.7454093
200       0.8950720  0.8623993  0.7424242
250       0.8962874  0.8624176  0.7514247
 50       0.8929490  0.8758059  0.7424242
100       0.8965943  0.8681502  0.7333786
150       0.8961429  0.8662271  0.7363636
200       0.8945422  0.8643040  0.7454545
250       0.8932773  0.8681136  0.7454545
 50       0.8936791  0.8585897  0.7574853
100       0.8961012  0.8623993  0.7544098
150       0.8947107  0.8643407  0.7543645
200       0.8936316  0.8643223  0.7573948
250       0.8923041  0.8623993  0.7574401
 50       0.8888519  0.8834249  0.7334690
100       0.8949587  0.8777106  0.7573948
150       0.8936526  0.8643040  0.7603799
200       0.8956895  0.8623993  0.7603347
250       0.8958975  0.8585897  0.7603347
 50       0.8944346  0.8739011  0.7424695
100       0.8939118  0.8758059  0.7454998
150       0.8941867  0.8642857  0.7575305
200       0.8940047  0.8643040  0.7454545
250       0.8935580  0.8662271  0.7424242
 50       0.8850146  0.8527656  0.7395296
100       0.8782633  0.8527839  0.7336047
150       0.8747305  0.8375092  0.7304839
200       0.8638325  0.8355861  0.7244686
250       0.8643353  0.8452381  0.7334690
 50       0.8805021  0.8490476  0.7514247
100       0.8759015  0.8413736  0.7215739
150       0.8755416  0.8528755  0.7425147
200       0.8688184  0.8490659  0.7424695
250       0.8678156  0.8452747  0.7364993
 50       0.8881949  0.8470879  0.7573948
100       0.8793114  0.8471245  0.7393939
150       0.8711676  0.8452015  0.7393939
200       0.8679362  0.8394139  0.7334690
250       0.8659592  0.8355678  0.7424695
 50       0.8808255  0.8490110  0.7483039
100       0.8740959  0.8528938  0.7543645
150       0.8707342  0.8337912  0.7514247
200       0.8645470  0.8319048  0.7305292
250       0.8629456  0.8280952  0.7425147
 50       0.8860815  0.8529304  0.7394844
100       0.8822826  0.8490842  0.7424242
150       0.8740383  0.8433516  0.7395749
```

200	0.8715296	0.8433333	0.7365445
250	0.8670994	0.8395055	0.7365445
50	0.8844637	0.8490476	0.7724107
100	0.8728620	0.8414469	0.7574853
150	0.8685493	0.8356777	0.7454545
200	0.8673158	0.8242491	0.7365445
250	0.8624353	0.8223443	0.7275893
50	0.8882813	0.8509158	0.7633198
100	0.8818852	0.8452198	0.7664405
150	0.8766524	0.8509524	0.7544550
200	0.8707894	0.8413736	0.7424242
250	0.8684345	0.8336996	0.7454093
50	0.8841683	0.8433150	0.7545002
100	0.8791919	0.8395238	0.7514699
150	0.8736929	0.8433516	0.7365898
200	0.8714521	0.8356960	0.7485301
250	0.8671170	0.8318315	0.7395296
50	0.8857903	0.8490476	0.7424695
100	0.8806222	0.8433333	0.7514699
150	0.8769983	0.8414652	0.7514247
200	0.8731708	0.8433516	0.7454545
250	0.8703350	0.8299817	0.7394844
50	0.8848441	0.8490110	0.7484848
100	0.8814950	0.8490659	0.7394392
150	0.8749868	0.8471429	0.7364541
200	0.8721568	0.8376374	0.7364089
250	0.8714881	0.8319048	0.7274989
50	0.8786324	0.8547619	0.7395749
100	0.8718142	0.8432784	0.7484396
150	0.8723101	0.8509158	0.7545002
200	0.8705041	0.8356593	0.7605156
250	0.8719681	0.8452198	0.7574853
50	0.8787421	0.8433150	0.7513795
100	0.8710246	0.8566300	0.7484848
150	0.8697552	0.8509341	0.7514699
200	0.8652094	0.8490293	0.7305292
250	0.8612317	0.8433150	0.7424242
50	0.8779580	0.8509158	0.7454545
100	0.8735530	0.8509524	0.7305292
150	0.8714992	0.8567033	0.7304839
200	0.8691593	0.8452198	0.7275441
250	0.8679840	0.8356410	0.7424695
50	0.8672154	0.8374908	0.7215739
100	0.8748092	0.8317766	0.7515604
150	0.8731220	0.8356777	0.7364541
200	0.8683297	0.8318498	0.7365445
250	0.8670207	0.8318498	0.7274989
50	0.8847634	0.8528388	0.7484848
100	0.8740490	0.8432967	0.7514247
150	0.8716119	0.8355495	0.7424242
200	0.8695564	0.8355861	0.7335595
250	0.8647883	0.8374908	0.7365445
50	0.8779954	0.8452015	0.7514699
100	0.8752859	0.8433150	0.7485301

```
150       0.8733384  0.8356777  0.7725011
200       0.8678160  0.8337912  0.7605608
250       0.8713657  0.8414469  0.7395749
 50       0.8811202  0.8432784  0.7335595
100       0.8739783  0.8432967  0.7545002
150       0.8759091  0.8509890  0.7514247
200       0.8714645  0.8432967  0.7633650
250       0.8694069  0.8433333  0.7513342
 50       0.8774225  0.8432967  0.7515604
100       0.8720158  0.8338095  0.7485301
150       0.8699173  0.8414469  0.7514699
200       0.8720337  0.8433700  0.7424695
250       0.8696724  0.8337729  0.7395296
 50       0.8840877  0.8432784  0.7454545
100       0.8801273  0.8509890  0.7603347
150       0.8796048  0.8452381  0.7544098
200       0.8747551  0.8452564  0.7574401
250       0.8738451  0.8414469  0.7483944
 50       0.8823978  0.8491026  0.7424695
100       0.8757048  0.8414469  0.7394844
150       0.8750614  0.8376190  0.7304839
200       0.8745863  0.8395421  0.7424242
250       0.8717478  0.8357509  0.7483944
 50       0.8760825  0.8451832  0.7455450
100       0.8716872  0.8490659  0.7186341
150       0.8618036  0.8146337  0.7275893
200       0.8608937  0.8356044  0.7305744
250       0.8517881  0.8280037  0.7215739
 50       0.8735015  0.8394505  0.7364541
100       0.8625099  0.8471245  0.7216192
150       0.8578344  0.8394689  0.6976029
200       0.8568027  0.8356777  0.7125283
250       0.8501916  0.8280220  0.7185889
 50       0.8804410  0.8490110  0.7424695
100       0.8744294  0.8337363  0.7632293
150       0.8720610  0.8356777  0.7394844
200       0.8660416  0.8318315  0.7305292
250       0.8632583  0.8260623  0.7336499
 50       0.8744292  0.8546886  0.7364993
100       0.8661804  0.8566667  0.7304839
150       0.8646532  0.8432601  0.7184984
200       0.8584052  0.8375641  0.7215739
250       0.8570249  0.8337179  0.7245590
 50       0.8843683  0.8547619  0.7425599
100       0.8780384  0.8452381  0.7485301
150       0.8706538  0.8432784  0.7185436
200       0.8678562  0.8356777  0.7185436
250       0.8647346  0.8356777  0.7335142
 50       0.8760349  0.8528571  0.7393939
100       0.8670265  0.8394505  0.7454545
150       0.8629361  0.8356593  0.7334238
200       0.8627210  0.8299451  0.7424695
250       0.8623080  0.8203846  0.7544098
 50       0.8808266  0.8452381  0.7664405
```

100	0.8728366	0.8413919	0.7484848
150	0.8680174	0.8375641	0.7514247
200	0.8647835	0.8356777	0.7484396
250	0.8612509	0.8395238	0.7335595
50	0.8864849	0.8452381	0.7513795
100	0.8749460	0.8299451	0.7245590
150	0.8715506	0.8299451	0.7394392
200	0.8653466	0.8280403	0.7394392
250	0.8631483	0.8318681	0.7395296
50	0.8804682	0.8432967	0.7633198
100	0.8768433	0.8490659	0.7573496
150	0.8736393	0.8395788	0.7454545
200	0.8700163	0.8395421	0.7454545
250	0.8650016	0.8357143	0.7274989
50	0.8826793	0.8490659	0.7483944
100	0.8764240	0.8529304	0.7573948
150	0.8720215	0.8414469	0.7454545
200	0.8706754	0.8376374	0.7274536
250	0.8661734	0.8338095	0.7334690
50	0.8884827	0.8681685	0.7274536
100	0.8900922	0.8777106	0.7394392
150	0.8895041	0.8815385	0.7454093
200	0.8921389	0.8700916	0.7514247
250	0.8888171	0.8624542	0.7453641
50	0.8876116	0.8662454	0.7423338
100	0.8903244	0.8547436	0.7395749
150	0.8925607	0.8662637	0.7544550
200	0.8933006	0.8566667	0.7695161
250	0.8855004	0.8662454	0.7576210
50	0.8925102	0.8815201	0.7064677
100	0.8923309	0.8757875	0.7335142
150	0.8888678	0.8643223	0.7394844
200	0.8881819	0.8662637	0.7304839
250	0.8916617	0.8681685	0.7364541
50	0.8917856	0.8777106	0.7394392
100	0.8915990	0.8643223	0.7483944
150	0.8922233	0.8700549	0.7484396
200	0.8939153	0.8642857	0.7573948
250	0.8936697	0.8643040	0.7364541
50	0.8898469	0.8623626	0.7515152
100	0.8950708	0.8623993	0.7603799
150	0.8976444	0.8643223	0.7694708
200	0.8975997	0.8662454	0.7574401
250	0.8952430	0.8624176	0.7634555
50	0.8933948	0.8624359	0.7573948
100	0.8929671	0.8643223	0.7483944
150	0.8951388	0.8662271	0.7603347
200	0.8916059	0.8643407	0.7574401
250	0.8926522	0.8604945	0.7544098
50	0.8927217	0.8777289	0.7394392
100	0.8955979	0.8662454	0.7574853
150	0.8947155	0.8681685	0.7484396
200	0.8938583	0.8662271	0.7483492
250	0.8929745	0.8586081	0.7574401

```
 50      0.8932811  0.8681319  0.7545455
100      0.8949344  0.8700549  0.7514247
150      0.8926841  0.8624176  0.7424242
200      0.8921751  0.8662088  0.7364541
250      0.8895024  0.8643040  0.7334690
 50      0.8938906  0.8777656  0.7424242
100      0.8952897  0.8758608  0.7573496
150      0.8956842  0.8643407  0.7483492
200      0.8947036  0.8643590  0.7513342
250      0.8942151  0.8585897  0.7483039
 50      0.8926348  0.8758242  0.7573948
100      0.8923460  0.8681319  0.7604704
150      0.8913920  0.8681502  0.7604704
200      0.8919545  0.8623993  0.7545002
250      0.8910861  0.8623993  0.7454093
 50      0.8866092  0.8413553  0.7544098
100      0.8773049  0.8452015  0.7753053
150      0.8708346  0.8470879  0.7514247
200      0.8713639  0.8356410  0.7514699
250      0.8698207  0.8299084  0.7364541
 50      0.8775733  0.8375641  0.7574853
100      0.8666954  0.8413919  0.7364541
150      0.8633668  0.8413919  0.7334690
200      0.8629879  0.8356777  0.7365445
250      0.8623917  0.8414103  0.7394844
 50      0.8744470  0.8394505  0.7394844
100      0.8679836  0.8585348  0.7455450
150      0.8620238  0.8566117  0.7425147
200      0.8629891  0.8432418  0.7365445
250      0.8623909  0.8375275  0.7335595
 50      0.8769377  0.8451832  0.7484848
100      0.8735785  0.8414103  0.7454545
150      0.8671140  0.8375824  0.7483944
200      0.8596719  0.8222711  0.7394844
250      0.8592878  0.8241392  0.7364993
 50      0.8839135  0.8547802  0.7274989
100      0.8740698  0.8471795  0.7365445
150      0.8701289  0.8471429  0.7335595
200      0.8641629  0.8299634  0.7336047
250      0.8623220  0.8280220  0.7275893
 50      0.8794791  0.8471245  0.7575758
100      0.8673144  0.8413919  0.7336047
150      0.8652514  0.8375824  0.7246495
200      0.8631931  0.8470879  0.7485301
250      0.8585148  0.8299084  0.7365445
 50      0.8836430  0.8395238  0.7545002
100      0.8709400  0.8395604  0.7424695
150      0.8644987  0.8376374  0.7394392
200      0.8638938  0.8337729  0.7394844
250      0.8623973  0.8222711  0.7304839
 50      0.8852866  0.8432784  0.7635007
100      0.8783481  0.8433150  0.7544550
150      0.8717465  0.8471429  0.7514699
200      0.8707459  0.8471795  0.7514699
```

250	0.8629286	0.8280403	0.7394844
50	0.8840086	0.8490293	0.7604704
100	0.8799281	0.8471429	0.7694256
150	0.8751276	0.8413736	0.7514699
200	0.8685827	0.8280220	0.7364993
250	0.8665643	0.8318315	0.7304839
50	0.8855252	0.8490659	0.7634555
100	0.8769001	0.8414286	0.7574853
150	0.8722382	0.8318681	0.7395296
200	0.8692580	0.8280586	0.7275441
250	0.8673183	0.8280769	0.7484848
50	0.8754668	0.8508791	0.7394844
100	0.8695753	0.8451465	0.7364993
150	0.8675713	0.8432967	0.7514247
200	0.8694054	0.8356044	0.7485301
250	0.8662364	0.8279670	0.7395749
50	0.8751839	0.8490476	0.7334690
100	0.8686115	0.8375275	0.7245138
150	0.8693695	0.8433516	0.7335142
200	0.8628746	0.8318132	0.7365445
250	0.8652701	0.8394689	0.7365445
50	0.8803984	0.8451832	0.7545907
100	0.8748962	0.8375824	0.7246043
150	0.8737689	0.8394872	0.7395749
200	0.8698974	0.8432967	0.7245590
250	0.8703854	0.8299267	0.7216192
50	0.8743557	0.8433150	0.7545455
100	0.8711812	0.8432601	0.7394844
150	0.8651284	0.8432601	0.7393939
200	0.8637852	0.8394689	0.7394844
250	0.8630316	0.8356044	0.7334690
50	0.8765492	0.8452381	0.7393939
100	0.8773339	0.8375458	0.7544550
150	0.8742715	0.8394505	0.7364993
200	0.8693044	0.8336813	0.7425147
250	0.8674291	0.8394505	0.7336499
50	0.8802328	0.8547985	0.7484396
100	0.8718842	0.8414286	0.7514247
150	0.8686178	0.8319048	0.7424695
200	0.8659377	0.8356960	0.7364993
250	0.8641159	0.8299634	0.7335142
50	0.8814352	0.8452381	0.7544550
100	0.8761650	0.8395604	0.7454998
150	0.8708513	0.8433516	0.7275893
200	0.8675343	0.8394872	0.7156038
250	0.8653396	0.8376190	0.7156490
50	0.8800908	0.8395055	0.7603799
100	0.8742996	0.8261355	0.7514247
150	0.8719062	0.8242308	0.7454545
200	0.8695955	0.8280220	0.7364993
250	0.8691428	0.8261172	0.7335142
50	0.8812227	0.8452747	0.7604251
100	0.8779345	0.8357326	0.7544550
150	0.8732437	0.8357509	0.7394844

```
200       0.8704979  0.8357326  0.7425147
250       0.8692787  0.8338278  0.7425147
 50       0.8824874  0.8490659  0.7513795
100       0.8766893  0.8452564  0.7573948
150       0.8728448  0.8433333  0.7484396
200       0.8700345  0.8395238  0.7394844
250       0.8698170  0.8376190  0.7454545
 50       0.8819219  0.8394689  0.7544098
100       0.8717419  0.8356410  0.7544098
150       0.8662438  0.8279304  0.7513795
200       0.8580683  0.8222344  0.7364089
250       0.8594296  0.8107875  0.7394844
 50       0.8734122  0.8509524  0.7424695
100       0.8670904  0.8298718  0.7364541
150       0.8636198  0.8413553  0.7394392
200       0.8582029  0.8318681  0.7425599
250       0.8599388  0.8394689  0.7275441
 50       0.8769726  0.8432967  0.7573496
100       0.8689235  0.8222161  0.7604704
150       0.8664713  0.8299451  0.7515152
200       0.8603594  0.8241758  0.7365445
250       0.8620024  0.8146154  0.7485301
 50       0.8765069  0.8547985  0.7365445
100       0.8669063  0.8471978  0.7304387
150       0.8636690  0.8433516  0.7215287
200       0.8590424  0.8318864  0.7275441
250       0.8581681  0.8356777  0.7245590
 50       0.8768048  0.8433516  0.7394844
100       0.8722546  0.8376190  0.7514699
150       0.8648202  0.8280586  0.7275893
200       0.8601924  0.8395421  0.7245138
250       0.8565645  0.8318498  0.7275441
 50       0.8792900  0.8376374  0.7334690
100       0.8712564  0.8395055  0.7335142
150       0.8671929  0.8375824  0.7425147
200       0.8655490  0.8260989  0.7483944
250       0.8632939  0.8394872  0.7485301
 50       0.8783079  0.8452198  0.7483944
100       0.8699176  0.8337729  0.7275441
150       0.8664902  0.8395238  0.7215287
200       0.8624936  0.8413919  0.7364993
250       0.8626544  0.8375641  0.7245590
 50       0.8835832  0.8490110  0.7574401
100       0.8735557  0.8433516  0.7454545
150       0.8683292  0.8395421  0.7454545
200       0.8656545  0.8376007  0.7335142
250       0.8644601  0.8280403  0.7305292
 50       0.8794897  0.8394689  0.7393487
100       0.8750312  0.8337546  0.7305292
150       0.8690831  0.8376007  0.7215739
200       0.8676470  0.8318681  0.7185436
250       0.8651973  0.8241941  0.7185436
 50       0.8837854  0.8490659  0.7544098
100       0.8756896  0.8490842  0.7484848
```

150	0.8715958	0.8433516	0.7364993
200	0.8710424	0.8471612	0.7364993
250	0.8699448	0.8338095	0.7394844
50	0.8922908	0.8796703	0.7393939
100	0.8915264	0.8700549	0.7244233
150	0.8911781	0.8719780	0.7274989
200	0.8923261	0.8585897	0.7425147
250	0.8936699	0.8566667	0.7485301
50	0.8928337	0.8661905	0.7453189
100	0.8915100	0.8604945	0.7634102
150	0.8906154	0.8585897	0.7694256
200	0.8880459	0.8681685	0.7694256
250	0.8872378	0.8662271	0.7573948
50	0.8917524	0.8776923	0.7303483
100	0.8922301	0.8680952	0.7363184
150	0.8901427	0.8681136	0.7393487
200	0.8911232	0.8642674	0.7424242
250	0.8888809	0.8566117	0.7424695
50	0.8914084	0.8585714	0.7545002
100	0.8905891	0.8623626	0.7574401
150	0.8911000	0.8528571	0.7574853
200	0.8923282	0.8452015	0.7574401
250	0.8909462	0.8509707	0.7514247
50	0.8950477	0.8776740	0.7363184
100	0.8976084	0.8681319	0.7573948
150	0.8963291	0.8604945	0.7574401
200	0.8950734	0.8547619	0.7663953
250	0.8959266	0.8566667	0.7634102
50	0.8908533	0.8604212	0.7544550
100	0.8914956	0.8642857	0.7514247
150	0.8912143	0.8566484	0.7544098
200	0.8937632	0.8528205	0.7483944
250	0.8923564	0.8547436	0.7483944
50	0.8931372	0.8700916	0.7333333
100	0.8924161	0.8681502	0.7363636
150	0.8925712	0.8624176	0.7333786
200	0.8918126	0.8566850	0.7393035
250	0.8918037	0.8586081	0.7513342
50	0.8920348	0.8604762	0.7363184
100	0.8927560	0.8623993	0.7393487
150	0.8923436	0.8547619	0.7454093
200	0.8932318	0.8605128	0.7543645
250	0.8920227	0.8528755	0.7543645
50	0.8895682	0.8739011	0.7364089
100	0.8938126	0.8567033	0.7572592
150	0.8926049	0.8605128	0.7573496
200	0.8907511	0.8528755	0.7513342
250	0.8915413	0.8566850	0.7453641
50	0.8947993	0.8719414	0.7604251
100	0.8949801	0.8680952	0.7604704
150	0.8934018	0.8643040	0.7634555
200	0.8950510	0.8604945	0.7694256
250	0.8936950	0.8566850	0.7634102
50	0.8778035	0.8433883	0.7334238

```
100       0.8698001  0.8395055  0.7394844
150       0.8632057  0.8318315  0.7513795
200       0.8617197  0.8299634  0.7454545
250       0.8589009  0.8338095  0.7184984
 50       0.8804314  0.8490110  0.7483944
100       0.8684648  0.8432234  0.7215739
150       0.8646357  0.8337179  0.7365445
200       0.8646902  0.8260989  0.7275893
250       0.8648444  0.8184432  0.7364993
 50       0.8848274  0.8586447  0.7575758
100       0.8719709  0.8433150  0.7426052
150       0.8683186  0.8337546  0.7455902
200       0.8653618  0.8337729  0.7395749
250       0.8631350  0.8356777  0.7275893
 50       0.8716342  0.8337363  0.7305744
100       0.8666351  0.8433150  0.7187698
150       0.8637687  0.8376007  0.7216644
200       0.8604389  0.8242125  0.7365898
250       0.8610216  0.8337546  0.7336047
 50       0.8746506  0.8452381  0.7425147
100       0.8763765  0.8318681  0.7364541
150       0.8698348  0.8300183  0.7515152
200       0.8682532  0.8280586  0.7425147
250       0.8648689  0.8299451  0.7395296
 50       0.8791431  0.8509341  0.7454093
100       0.8736645  0.8452747  0.7306196
150       0.8669613  0.8280769  0.7365445
200       0.8658602  0.8280586  0.7365898
250       0.8643956  0.8318864  0.7365898
 50       0.8792321  0.8375824  0.7335595
100       0.8700843  0.8413919  0.7424242
150       0.8654637  0.8317766  0.7514699
200       0.8643437  0.8356410  0.7424242
250       0.8599343  0.8337179  0.7275893
 50       0.8837901  0.8490110  0.7483944
100       0.8767158  0.8376374  0.7545907
150       0.8728313  0.8357143  0.7545002
200       0.8679907  0.8337729  0.7334690
250       0.8661643  0.8299817  0.7394844
 50       0.8824529  0.8490842  0.7483944
100       0.8735733  0.8452747  0.7514247
150       0.8688869  0.8376557  0.7334690
200       0.8659076  0.8318681  0.7394844
250       0.8643495  0.8395238  0.7335142
 50       0.8846959  0.8356593  0.7393939
100       0.8763534  0.8452930  0.7334238
150       0.8723052  0.8415018  0.7274989
200       0.8714350  0.8319048  0.7184984
250       0.8689076  0.8452930  0.7245138
 50       0.8724575  0.8604579  0.7274989
100       0.8671857  0.8413553  0.7304839
150       0.8627209  0.8490659  0.7275441
200       0.8611521  0.8433333  0.7274989
250       0.8664165  0.8260989  0.7483944
```

50	0.8646594	0.8355861	0.7544098
100	0.8630728	0.8413919	0.7424242
150	0.8648167	0.8280037	0.7364993
200	0.8637042	0.8432601	0.7513795
250	0.8635405	0.8490293	0.7304839
50	0.8746713	0.8452015	0.7394844
100	0.8652904	0.8509890	0.7304839
150	0.8650608	0.8528755	0.7304387
200	0.8646591	0.8433333	0.7304387
250	0.8689805	0.8337546	0.7304387
50	0.8688179	0.8490842	0.7275441
100	0.8613536	0.8357143	0.7365898
150	0.8613730	0.8356777	0.7125283
200	0.8583205	0.8394689	0.7215287
250	0.8620000	0.8452015	0.7334690
50	0.8838780	0.8490293	0.7453641
100	0.8714218	0.8433150	0.7305744
150	0.8686238	0.8432967	0.7275441
200	0.8637598	0.8413553	0.7185436
250	0.8641717	0.8471429	0.7274989
50	0.8682921	0.8414286	0.7395296
100	0.8692590	0.8433516	0.7304387
150	0.8623710	0.8395238	0.7125735
200	0.8603620	0.8414652	0.7125735
250	0.8601470	0.8395421	0.7156038
50	0.8739738	0.8529487	0.7634555
100	0.8655927	0.8414469	0.7454998
150	0.8668856	0.8375641	0.7425147
200	0.8640161	0.8376007	0.7304387
250	0.8647582	0.8356593	0.7155133
50	0.8794621	0.8471429	0.7574853
100	0.8718664	0.8471429	0.7334690
150	0.8701471	0.8394689	0.7395296
200	0.8687550	0.8318864	0.7455450
250	0.8671930	0.8395055	0.7395749
50	0.8852038	0.8510440	0.7514699
100	0.8803856	0.8567582	0.7605156
150	0.8735232	0.8471795	0.7425147
200	0.8714364	0.8433333	0.7395296
250	0.8701395	0.8433333	0.7364993
50	0.8807913	0.8490842	0.7454998
100	0.8741169	0.8414469	0.7483944
150	0.8711078	0.8452381	0.7394844
200	0.8682003	0.8375824	0.7304387
250	0.8672883	0.8376007	0.7364541
50	0.8734052	0.8509158	0.7215287
100	0.8645536	0.8279670	0.7424242
150	0.8576572	0.8202930	0.7364541
200	0.8558360	0.8183883	0.7246043
250	0.8544761	0.8241392	0.7095884
50	0.8631138	0.8356593	0.7395749
100	0.8569794	0.8261538	0.7394392
150	0.8547356	0.8356410	0.7365898
200	0.8518920	0.8318498	0.7335595

```
250       0.8482767  0.8299084  0.7244686
 50       0.8705495  0.8452198  0.7425599
100       0.8622409  0.8356593  0.7245590
150       0.8587324  0.8509707  0.7244686
200       0.8585570  0.8356777  0.7334690
250       0.8570321  0.8376190  0.7244233
 50       0.8717817  0.8471612  0.7486205
100       0.8641424  0.8567216  0.7335142
150       0.8579043  0.8433150  0.7216644
200       0.8567285  0.8280952  0.7156490
250       0.8538829  0.8338462  0.7126187
 50       0.8712639  0.8471245  0.7305292
100       0.8621640  0.8336996  0.7305292
150       0.8605754  0.8356593  0.7275441
200       0.8588701  0.8222711  0.7275893
250       0.8547879  0.8184432  0.7185889
 50       0.8796985  0.8395055  0.7514247
100       0.8685785  0.8375824  0.7305744
150       0.8599932  0.8222894  0.7364993
200       0.8587404  0.8280403  0.7215739
250       0.8567784  0.8204212  0.7274989
 50       0.8764763  0.8433150  0.7244686
100       0.8685092  0.8375824  0.7455450
150       0.8630237  0.8357143  0.7215739
200       0.8608829  0.8280586  0.7245590
250       0.8575431  0.8241758  0.7275441
 50       0.8758790  0.8509707  0.7483944
100       0.8676105  0.8433516  0.7335142
150       0.8625018  0.8356593  0.7364993
200       0.8595373  0.8318315  0.7275441
250       0.8593582  0.8318315  0.7394844
 50       0.8795114  0.8471795  0.7515152
100       0.8727497  0.8357326  0.7364993
150       0.8691327  0.8280952  0.7275441
200       0.8661861  0.8261538  0.7185889
250       0.8640209  0.8261538  0.7275441
 50       0.8789163  0.8414835  0.7454545
100       0.8687999  0.8280769  0.7334238
150       0.8662551  0.8300000  0.7156038
200       0.8628152  0.8318681  0.7215739
250       0.8621628  0.8280220  0.7126187
 50       0.8908888  0.8738278  0.7574853
100       0.8853013  0.8776557  0.7545002
150       0.8841661  0.8719231  0.7245138
200       0.8834580  0.8700183  0.7274989
250       0.8846847  0.8661722  0.7274989
 50       0.8927438  0.8643407  0.7544550
100       0.8941758  0.8643590  0.7603799
150       0.8907812  0.8528571  0.7573948
200       0.8866979  0.8643223  0.7274084
250       0.8829374  0.8566484  0.7514699
 50       0.8897381  0.8719780  0.7453641
100       0.8906130  0.8662271  0.7513795
150       0.8922144  0.8662271  0.7243329
```

200	0.8878407	0.8642857	0.7124378
250	0.8898359	0.8681319	0.7394392
50	0.8923151	0.8567033	0.7425147
100	0.8863755	0.8471245	0.7274989
150	0.8906592	0.8471429	0.7514247
200	0.8868280	0.8452198	0.7454545
250	0.8875013	0.8490659	0.7484848
50	0.8894751	0.8643040	0.7393939
100	0.8898568	0.8585897	0.7453641
150	0.8933509	0.8509341	0.7454093
200	0.8919864	0.8624176	0.7604704
250	0.8926226	0.8509341	0.7604251
50	0.8952046	0.8700549	0.7363636
100	0.8916590	0.8643040	0.7423790
150	0.8929689	0.8623993	0.7424242
200	0.8932140	0.8623993	0.7484848
250	0.8919398	0.8605128	0.7484848
50	0.8873336	0.8700916	0.7303935
100	0.8934121	0.8585897	0.7454545
150	0.8902751	0.8567033	0.7483492
200	0.8922348	0.8547985	0.7602895
250	0.8896643	0.8528571	0.7723654
50	0.8915674	0.8547436	0.7694708
100	0.8906929	0.8490293	0.7663953
150	0.8903501	0.8509341	0.7694708
200	0.8903994	0.8509524	0.7454998
250	0.8911968	0.8490293	0.7545002
50	0.8858535	0.8719780	0.7214383
100	0.8937277	0.8643040	0.7574401
150	0.8968683	0.8604762	0.7663953
200	0.8967193	0.8604762	0.7693351
250	0.8937695	0.8566667	0.7542741
50	0.8938694	0.8681319	0.7604704
100	0.8960009	0.8547802	0.7694256
150	0.8937206	0.8567033	0.7634555
200	0.8916980	0.8528388	0.7604251
250	0.8893261	0.8509341	0.7514247
50	0.8786145	0.8508974	0.7485301
100	0.8641868	0.8510073	0.7274989
150	0.8600687	0.8356227	0.7275441
200	0.8590847	0.8509158	0.7335142
250	0.8591797	0.8432418	0.7396201
50	0.8678726	0.8394505	0.7454998
100	0.8669330	0.8433333	0.7333786
150	0.8626191	0.8394872	0.7185436
200	0.8619573	0.8433333	0.7244686
250	0.8609994	0.8376190	0.7274536
50	0.8776234	0.8356593	0.7364541
100	0.8710216	0.8356410	0.7393939
150	0.8662758	0.8317949	0.7454545
200	0.8634910	0.8374908	0.7573496
250	0.8593445	0.8164835	0.7454545
50	0.8761737	0.8566850	0.7545455
100	0.8701819	0.8451832	0.7336047

```
150       0.8646912  0.8318132  0.7485301
200       0.8620066  0.8375641  0.7336047
250       0.8579163  0.8356044  0.7245138
 50       0.8859613  0.8509158  0.7693804
100       0.8735734  0.8414103  0.7393939
150       0.8645789  0.8318315  0.7335142
200       0.8624408  0.8356410  0.7336047
250       0.8603878  0.8337729  0.7304839
 50       0.8710812  0.8318681  0.7574853
100       0.8645898  0.8337546  0.7365898
150       0.8652094  0.8375824  0.7335142
200       0.8615821  0.8318681  0.7186341
250       0.8585315  0.8356227  0.7246495
 50       0.8744053  0.8528938  0.7633198
100       0.8697292  0.8413919  0.7394392
150       0.8652509  0.8433516  0.7305292
200       0.8609636  0.8356960  0.7275441
250       0.8586055  0.8337363  0.7185436
 50       0.8737096  0.8395421  0.7603799
100       0.8636489  0.8184982  0.7335142
150       0.8609657  0.8300183  0.7274989
200       0.8594631  0.8204212  0.7245138
250       0.8585483  0.8261172  0.7394844
 50       0.8764613  0.8433700  0.7245138
100       0.8694829  0.8338278  0.7244686
150       0.8655955  0.8318681  0.7125283
200       0.8630756  0.8356960  0.7185889
250       0.8614160  0.8299267  0.7125735
 50       0.8781926  0.8395421  0.7393939
100       0.8722143  0.8414652  0.7484396
150       0.8679772  0.8376007  0.7274084
200       0.8645610  0.8356960  0.7184984
250       0.8630198  0.8261355  0.7155133
 50       0.8774443  0.8471795  0.7304839
100       0.8705586  0.8491026  0.7305292
150       0.8685462  0.8357143  0.7156038
200       0.8658272  0.8414469  0.7155133
250       0.8627209  0.8299634  0.7304387
 50       0.8713608  0.8433150  0.7424242
100       0.8639601  0.8414103  0.7365445
150       0.8671996  0.8356410  0.7426052
200       0.8642775  0.8375641  0.7425599
250       0.8626741  0.8375824  0.7365445
 50       0.8760872  0.8509707  0.7604251
100       0.8675522  0.8586081  0.7425147
150       0.8630700  0.8509524  0.7454998
200       0.8666347  0.8452381  0.7425599
250       0.8655137  0.8413919  0.7365445
 50       0.8729192  0.8547619  0.7425147
100       0.8689065  0.8414469  0.7425147
150       0.8644167  0.8356960  0.7365898
200       0.8680749  0.8452564  0.7455450
250       0.8660586  0.8395238  0.7185889
 50       0.8783323  0.8375641  0.7574853
```

100	0.8728494	0.8376374	0.7364993
150	0.8684511	0.8242308	0.7425599
200	0.8667504	0.8261905	0.7335595
250	0.8652319	0.8300000	0.7365445
50	0.8791134	0.8414103	0.7485301
100	0.8732475	0.8356960	0.7305292
150	0.8709421	0.8337912	0.7364993
200	0.8666742	0.8357143	0.7394844
250	0.8664137	0.8337912	0.7364993
50	0.8732114	0.8452198	0.7544550
100	0.8686704	0.8356777	0.7424695
150	0.8678936	0.8222711	0.7335595
200	0.8659893	0.8184249	0.7425147
250	0.8645884	0.8280220	0.7304839
50	0.8758658	0.8414286	0.7425599
100	0.8681207	0.8376007	0.7364993
150	0.8658094	0.8356777	0.7125735
200	0.8662774	0.8337912	0.7335595
250	0.8648304	0.8376007	0.7245590
50	0.8770141	0.8452564	0.7573948
100	0.8696163	0.8433333	0.7305292
150	0.8685882	0.8394872	0.7184984
200	0.8667282	0.8395055	0.7215287
250	0.8651623	0.8395055	0.7155586
50	0.8790017	0.8318681	0.7363636
100	0.8739991	0.8376190	0.7244686
150	0.8731045	0.8376374	0.7274989
200	0.8713456	0.8357326	0.7155586
250	0.8699330	0.8337912	0.7185436
50	0.8661302	0.8509341	0.7005880
100	0.8628389	0.8395421	0.7185889
150	0.8616857	0.8376007	0.7245590
200	0.8591570	0.8395421	0.7245138
250	0.8566960	0.8299817	0.7304839
50	0.8686134	0.8432418	0.7425599
100	0.8661724	0.8260623	0.7605156
150	0.8621175	0.8261172	0.7454998
200	0.8629929	0.8280403	0.7454545
250	0.8577852	0.8260623	0.7484848
50	0.8774289	0.8451648	0.7514699
100	0.8676155	0.8509158	0.7394844
150	0.8642274	0.8337363	0.7364993
200	0.8621109	0.8394505	0.7305744
250	0.8598153	0.8375458	0.7185436
50	0.8748293	0.8413919	0.7365445
100	0.8675573	0.8299451	0.7335142
150	0.8634108	0.8184432	0.7335142
200	0.8614353	0.8203846	0.7335142
250	0.8621470	0.8146520	0.7215739
50	0.8784559	0.8529304	0.7304839
100	0.8684330	0.8433333	0.7213930
150	0.8627458	0.8376190	0.7155133
200	0.8621828	0.8414652	0.7214835
250	0.8593868	0.8395421	0.7244686

```
 50      0.8690078  0.8394872  0.7485301
100      0.8660291  0.8260806  0.7395296
150      0.8629505  0.8222711  0.7425599
200      0.8601105  0.8280037  0.7395749
250      0.8579807  0.8280220  0.7425599
 50      0.8768187  0.8433700  0.7423790
100      0.8685422  0.8375641  0.7275441
150      0.8634059  0.8337729  0.7274989
200      0.8584116  0.8356593  0.7305292
250      0.8577010  0.8318681  0.7245138
 50      0.8687626  0.8395055  0.7424695
100      0.8640344  0.8394689  0.7305292
150      0.8622272  0.8356777  0.7275441
200      0.8604384  0.8241941  0.7215287
250      0.8612317  0.8318315  0.7155586
 50      0.8781185  0.8510073  0.7485753
100      0.8725636  0.8510073  0.7394844
150      0.8680547  0.8356960  0.7215287
200      0.8658209  0.8376007  0.7274989
250      0.8625425  0.8375824  0.7245138
 50      0.8761670  0.8452381  0.7514247
100      0.8711687  0.8356960  0.7244686
150      0.8665241  0.8376190  0.7304839
200      0.8641546  0.8299267  0.7274989
250      0.8614860  0.8318498  0.7244686
 50      0.8895207  0.8700733  0.7393939
100      0.8870218  0.8605311  0.7304387
150      0.8831429  0.8662637  0.7364993
200      0.8837276  0.8624359  0.7304387
250      0.8865297  0.8624359  0.7305744
 50      0.8909579  0.8642857  0.7454545
100      0.8913533  0.8662088  0.7454545
150      0.8901275  0.8566850  0.7484848
200      0.8875826  0.8490842  0.7545002
250      0.8860873  0.8509341  0.7484396
 50      0.8877194  0.8624176  0.7393487
100      0.8860739  0.8624176  0.7393939
150      0.8867688  0.8700733  0.7423790
200      0.8853933  0.8604945  0.7425599
250      0.8859727  0.8566850  0.7335142
 50      0.8899911  0.8623993  0.7453641
100      0.8919417  0.8604579  0.7513795
150      0.8914396  0.8662454  0.7573044
200      0.8911764  0.8586081  0.7694708
250      0.8894919  0.8547802  0.7574401
 50      0.8871858  0.8719597  0.7274084
100      0.8902973  0.8700733  0.7304387
150      0.8904721  0.8413370  0.7394392
200      0.8891343  0.8489927  0.7484848
250      0.8875884  0.8509158  0.7454093
 50      0.8948051  0.8528571  0.7604251
100      0.8928638  0.8566667  0.7573948
150      0.8930362  0.8509341  0.7604251
200      0.8922150  0.8547619  0.7573948
```

250	0.8896097	0.8528755	0.7573948
50	0.8904203	0.8624176	0.7543645
100	0.8897477	0.8490476	0.7573948
150	0.8888398	0.8547436	0.7484396
200	0.8891596	0.8490476	0.7663501
250	0.8889250	0.8528755	0.7633198
50	0.8892078	0.8547802	0.7604251
100	0.8865172	0.8566300	0.7544098
150	0.8883495	0.8623810	0.7664405
200	0.8858961	0.8566850	0.7574853
250	0.8865684	0.8567033	0.7515152
50	0.8860398	0.8585531	0.7453641
100	0.8886152	0.8566484	0.7633650
150	0.8898043	0.8585897	0.7633650
200	0.8882322	0.8547436	0.7633198
250	0.8882415	0.8451832	0.7632745
50	0.8916261	0.8471062	0.7633198
100	0.8894058	0.8509158	0.7723202
150	0.8875746	0.8490293	0.7663048
200	0.8877057	0.8452198	0.7663501
250	0.8872300	0.8471245	0.7693351
50	0.8746448	0.8566667	0.7216644
100	0.8633606	0.8509341	0.7395749
150	0.8584792	0.8451832	0.7215739
200	0.8543046	0.8337546	0.7365898
250	0.8517285	0.8413919	0.7365445
50	0.8705606	0.8376007	0.7455902
100	0.8624656	0.8394505	0.7485753
150	0.8595086	0.8433333	0.7335595
200	0.8561682	0.8337729	0.7454998
250	0.8551093	0.8356227	0.7425599
50	0.8699312	0.8470879	0.7454998
100	0.8558111	0.8433150	0.7275893
150	0.8568717	0.8471245	0.7276346
200	0.8560625	0.8432967	0.7335595
250	0.8552970	0.8451832	0.7275893
50	0.8774914	0.8490659	0.7365445
100	0.8688057	0.8356777	0.7395296
150	0.8692051	0.8260989	0.7335142
200	0.8643557	0.8318315	0.7395296
250	0.8630671	0.8241758	0.7335595
50	0.8766326	0.8471429	0.7485753
100	0.8702351	0.8356593	0.7515152
150	0.8655388	0.8260623	0.7246495
200	0.8646112	0.8337179	0.7275441
250	0.8592027	0.8280037	0.7186341
50	0.8777985	0.8567399	0.7483492
100	0.8688550	0.8509707	0.7364089
150	0.8636856	0.8338278	0.7364993
200	0.8590142	0.8223260	0.7305292
250	0.8560568	0.8242491	0.7335142
50	0.8775481	0.8509524	0.7516056
100	0.8712120	0.8452564	0.7306196
150	0.8670986	0.8318315	0.7395749

```
200       0.8640921  0.8299267  0.7336047
250       0.8622131  0.8318498  0.7395749
 50       0.8771846  0.8548168  0.7484396
100       0.8682329  0.8433333  0.7424695
150       0.8628471  0.8376007  0.7454545
200       0.8620356  0.8299634  0.7394844
250       0.8589761  0.8242125  0.7394392
 50       0.8801667  0.8434249  0.7543645
100       0.8705129  0.8319048  0.7364993
150       0.8661562  0.8318864  0.7245590
200       0.8646367  0.8337729  0.7275441
250       0.8630275  0.8242125  0.7215739
 50       0.8818305  0.8471429  0.7543645
100       0.8751590  0.8433883  0.7454998
150       0.8703310  0.8337912  0.7365898
200       0.8693205  0.8223626  0.7305292
250       0.8673252  0.8223443  0.7305292
 50       0.8673538  0.8395055  0.7574853
100       0.8612184  0.8280037  0.7246495
150       0.8595141  0.8395238  0.7365445
200       0.8596461  0.8338095  0.7336499
250       0.8547695  0.8376374  0.7276346
 50       0.8628369  0.8318132  0.7153324
100       0.8568422  0.8337179  0.7155133
150       0.8558575  0.8471612  0.7274989
200       0.8558528  0.8356593  0.7155133
250       0.8536827  0.8280220  0.7184984
 50       0.8753068  0.8433516  0.7514699
100       0.8667217  0.8432784  0.7425147
150       0.8629627  0.8279670  0.7515152
200       0.8602513  0.8299267  0.7335142
250       0.8579470  0.8299084  0.7184984
 50       0.8652456  0.8393956  0.7364541
100       0.8597843  0.8394322  0.7244686
150       0.8551270  0.8432784  0.7394844
200       0.8606370  0.8279670  0.7395296
250       0.8578279  0.8394689  0.7214835
 50       0.8606725  0.8375824  0.7245138
100       0.8592414  0.8395238  0.7216192
150       0.8587227  0.8471429  0.7244686
200       0.8562155  0.8375641  0.7245138
250       0.8548586  0.8356777  0.7334690
 50       0.8694231  0.8414286  0.7275441
100       0.8667236  0.8299084  0.7424695
150       0.8654376  0.8356227  0.7364541
200       0.8637922  0.8413919  0.7394844
250       0.8636095  0.8337363  0.7395296
 50       0.8732507  0.8318498  0.7244233
100       0.8663719  0.8375824  0.7243781
150       0.8655072  0.8241941  0.7304839
200       0.8642904  0.8299267  0.7275441
250       0.8628440  0.8337729  0.7214835
 50       0.8732180  0.8357143  0.7425599
100       0.8700518  0.8337912  0.7365445
```

150	0.8683801	0.8318681	0.7215287
200	0.8673448	0.8318498	0.7455450
250	0.8673050	0.8356593	0.7455450
50	0.8758305	0.8414469	0.7544098
100	0.8679932	0.8337546	0.7455450
150	0.8648954	0.8318498	0.7485301
200	0.8663573	0.8241941	0.7394844
250	0.8649011	0.8337729	0.7365445
50	0.8765266	0.8510256	0.7513795
100	0.8699535	0.8337729	0.7394844
150	0.8677131	0.8318498	0.7335142
200	0.8668771	0.8318315	0.7365445
250	0.8667878	0.8261172	0.7245138
50	0.8706189	0.8489744	0.7246947
100	0.8630999	0.8471978	0.7216192
150	0.8586042	0.8414286	0.7215287
200	0.8549332	0.8414103	0.7365445
250	0.8539145	0.8376007	0.7306196
50	0.8649371	0.8413553	0.7304839
100	0.8539137	0.8280403	0.7215287
150	0.8539147	0.8261355	0.7305744
200	0.8474083	0.8280220	0.7304839
250	0.8449044	0.8223077	0.7394844
50	0.8699457	0.8432234	0.7455902
100	0.8659514	0.8490110	0.7425147
150	0.8596327	0.8452198	0.7364993
200	0.8533583	0.8413919	0.7365445
250	0.8567973	0.8433150	0.7365445
50	0.8704410	0.8337912	0.7395749
100	0.8585982	0.8280037	0.7365898
150	0.8557631	0.8318864	0.7306196
200	0.8535902	0.8261172	0.7336047
250	0.8526361	0.8280586	0.7336047
50	0.8707864	0.8490476	0.7544550
100	0.8609674	0.8394872	0.7155586
150	0.8573464	0.8280037	0.7275441
200	0.8574210	0.8260806	0.7216644
250	0.8545077	0.8184799	0.7276346
50	0.8649654	0.8414103	0.7425599
100	0.8572514	0.8299817	0.7335595
150	0.8559922	0.8242674	0.7274084
200	0.8531080	0.8242308	0.7305292
250	0.8537534	0.8261172	0.7304839
50	0.8721748	0.8261172	0.7365445
100	0.8651696	0.8318498	0.7185436
150	0.8592143	0.8203480	0.7155133
200	0.8566429	0.8203663	0.7155586
250	0.8546949	0.8279853	0.7185436
50	0.8708461	0.8356777	0.7484396
100	0.8678161	0.8356593	0.7424242
150	0.8631401	0.8280220	0.7364541
200	0.8583061	0.8280403	0.7274536
250	0.8562568	0.8299634	0.7304839
50	0.8762873	0.8433333	0.7364541

```
100       0.8699656  0.8318498  0.7364541
150       0.8655433  0.8299451  0.7334690
200       0.8616847  0.8299634  0.7334690
250       0.8610589  0.8280037  0.7215287
 50       0.8747583  0.8280769  0.7364993
100       0.8719871  0.8356777  0.7394844
150       0.8693228  0.8356777  0.7245590
200       0.8663461  0.8318864  0.7215287
250       0.8655713  0.8261355  0.7245138
 50       0.8890298  0.8623993  0.7426052
100       0.8905668  0.8643040  0.7425599
150       0.8839768  0.8566484  0.7185436
200       0.8797903  0.8566484  0.7394844
250       0.8827349  0.8623993  0.7453641
 50       0.8903368  0.8547802  0.7514699
100       0.8866076  0.8432418  0.7544550
150       0.8856900  0.8547253  0.7424242
200       0.8837851  0.8527839  0.7334238
250       0.8832460  0.8566117  0.7393939
 50       0.8892736  0.8663004  0.7303483
100       0.8897415  0.8624359  0.7244686
150       0.8846549  0.8509524  0.7364089
200       0.8823583  0.8452381  0.7423338
250       0.8817523  0.8567033  0.7454093
 50       0.8948866  0.8624359  0.7454093
100       0.8891008  0.8529304  0.7514699
150       0.8875395  0.8414469  0.7424695
200       0.8884832  0.8567033  0.7363636
250       0.8865324  0.8586081  0.7453641
 50       0.8929028  0.8604762  0.7453641
100       0.8874532  0.8567216  0.7364541
150       0.8865052  0.8471429  0.7484396
200       0.8876548  0.8547802  0.7514699
250       0.8863272  0.8567033  0.7544550
 50       0.8896928  0.8585897  0.7454545
100       0.8884119  0.8528571  0.7514247
150       0.8863766  0.8471245  0.7485301
200       0.8880284  0.8471062  0.7483944
250       0.8857773  0.8471062  0.7544098
 50       0.8912134  0.8623993  0.7394392
100       0.8888059  0.8623993  0.7454545
150       0.8896700  0.8604945  0.7484396
200       0.8876484  0.8547619  0.7514699
250       0.8875058  0.8585897  0.7484848
 50       0.8936409  0.8509524  0.7603799
100       0.8916108  0.8490476  0.7664405
150       0.8926699  0.8490293  0.7633198
200       0.8905366  0.8471429  0.7543645
250       0.8846088  0.8490476  0.7513795
 50       0.8901608  0.8547436  0.7424242
100       0.8884215  0.8509341  0.7543193
150       0.8882353  0.8547802  0.7603347
200       0.8860334  0.8490659  0.7513342
250       0.8865944  0.8471429  0.7543645
```

50	0.8907418	0.8547070	0.7633650
100	0.8886095	0.8471062	0.7723654
150	0.8861470	0.8471062	0.7664405
200	0.8873350	0.8452198	0.7784261
250	0.8863502	0.8414286	0.7604251
50	0.8701412	0.8528388	0.7454998
100	0.8625718	0.8395055	0.7484848
150	0.8629900	0.8375641	0.7395749
200	0.8579777	0.8261172	0.7394844
250	0.8594672	0.8318498	0.7395749
50	0.8661575	0.8356593	0.7246043
100	0.8628503	0.8280586	0.7395296
150	0.8570431	0.8376007	0.7394392
200	0.8522007	0.8223077	0.7365445
250	0.8503596	0.8127656	0.7424695
50	0.8667244	0.8433516	0.7514699
100	0.8609705	0.8260989	0.7395749
150	0.8563640	0.8241941	0.7515604
200	0.8540687	0.8299451	0.7515152
250	0.8532536	0.8203663	0.7455450
50	0.8753378	0.8471245	0.7485301
100	0.8606371	0.8394322	0.7155586
150	0.8556438	0.8298901	0.7156038
200	0.8514963	0.8356410	0.7125735
250	0.8503770	0.8356044	0.7095884
50	0.8724264	0.8395055	0.7514699
100	0.8640699	0.8318315	0.7304387
150	0.8581699	0.8337729	0.7304839
200	0.8549944	0.8261355	0.7395296
250	0.8536701	0.8260989	0.7335142
50	0.8708739	0.8299451	0.7215287
100	0.8653448	0.8242308	0.7275441
150	0.8624148	0.8223260	0.7364993
200	0.8583785	0.8184615	0.7305292
250	0.8582434	0.8165934	0.7364993
50	0.8698650	0.8548168	0.7485301
100	0.8641207	0.8242125	0.7335595
150	0.8612409	0.8204029	0.7335595
200	0.8585489	0.8242308	0.7246043
250	0.8554541	0.8242125	0.7365445
50	0.8724089	0.8548718	0.7455902
100	0.8642356	0.8261172	0.7185889
150	0.8601842	0.8375824	0.7214835
200	0.8583151	0.8413919	0.7274989
250	0.8554467	0.8299451	0.7214835
50	0.8770816	0.8376190	0.7453641
100	0.8681332	0.8261172	0.7244686
150	0.8658084	0.8280403	0.7155133
200	0.8628919	0.8242308	0.7095432
250	0.8590119	0.8184799	0.7065129
50	0.8765175	0.8375824	0.7424695
100	0.8704513	0.8414469	0.7214835
150	0.8648721	0.8299817	0.7215287
200	0.8634965	0.8299634	0.7155586

```
  250      0.8629786  0.8299634  0.7184984

Tuning parameter 'gamma' was held constant at a value of 0

Tuning parameter 'min_child_weight' was held constant at a value of 1
ROC was used to select the optimal model using the largest value.
The final values used for the model were nrounds = 100, max_depth = 2,
 eta = 0.4, gamma = 0, subsample = 0.875, colsample_bytree =
 0.6, rate_drop = 0.5, skip_drop = 0.05 and min_child_weight = 1.
```

17.8.4 8.4. Training SVM

```r
set.seed(100)

# Train the model using MARS
model_svmRadial = train(Purchase ~ .,
                        data = trainData,
                        method = 'svmRadial',
                        tuneLength = 15,
                        trControl = fitControl)
model_svmRadial
```

```
Support Vector Machines with Radial Basis Function Kernel

857 samples
 18 predictor
  2 classes: 'CH', 'MM'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 685, 685, 687, 686, 685
Resampling results across tuning parameters:

  C        ROC        Sens       Spec
     0.25  0.8858137  0.8775824  0.7213930
     0.50  0.8907736  0.8814469  0.7363636
     1.00  0.8888077  0.8852930  0.7333786
     2.00  0.8864533  0.8680952  0.7305292
     4.00  0.8832598  0.8719048  0.7335142
     8.00  0.8773941  0.8681502  0.7275441
    16.00  0.8719099  0.8623626  0.7245138
    32.00  0.8684735  0.8661905  0.6886929
    64.00  0.8583358  0.8700366  0.6946630
   128.00  0.8481861  0.8681136  0.6856174
   256.00  0.8365898  0.8719414  0.6647218
   512.00  0.8246331  0.8719414  0.6436906
  1024.00  0.8174809  0.8757326  0.6228856
  2048.00  0.8147110  0.8949084  0.5629579
  4096.00  0.8113213  0.9043956  0.5418363

Tuning parameter 'sigma' was held constant at a value of 0.06414448
ROC was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.06414448 and C = 0.5.
```

17.8.5 8.5. Run `resamples()` to compare the models

```
# Compare model performances using resample()
models_compare <- resamples(list(
                    ADABOOST = model_adaboost,
                    RF = model_rf,
                    XGBDART = model_xgbDART,
                    MARS = model_mars3,
                    SVM = model_svmRadial)
    )

# Summary of the models performances
summary(models_compare)
```

```
Call:
summary.resamples(object = models_compare)

Models: ADABOOST, RF, XGBDART, MARS, SVM
Number of resamples: 5

ROC
              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
ADABOOST 0.8126510 0.8462687 0.8682549 0.8657598 0.8868515 0.9147727    0
RF       0.8253021 0.8568586 0.8877727 0.8775297 0.9017768 0.9159382    0
XGBDART  0.8589197 0.8775409 0.9114524 0.8995839 0.9198718 0.9301350    0
MARS     0.8433547 0.8839375 0.9073622 0.8947444 0.9084353 0.9306326    0
SVM      0.8537313 0.8728500 0.8903559 0.8907736 0.9053030 0.9316276    0

Sens
              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
ADABOOST 0.7619048 0.7904762 0.7904762 0.8070330 0.8076923 0.8846154    0
RF       0.8000000 0.8190476 0.8365385 0.8490476 0.8761905 0.9134615    0
XGBDART  0.8285714 0.8666667 0.8846154 0.8738828 0.8857143 0.9038462    0
MARS     0.8190476 0.8476190 0.8653846 0.8604579 0.8750000 0.8952381    0
SVM      0.8653846 0.8761905 0.8857143 0.8814469 0.8857143 0.8942308    0

Spec
              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
ADABOOST 0.7014925 0.7462687 0.7727273 0.7635007 0.7761194 0.8208955    0
RF       0.6417910 0.7313433 0.7313433 0.7365445 0.7424242 0.8358209    0
XGBDART  0.6567164 0.6969697 0.7014925 0.7304387 0.7462687 0.8507463    0
MARS     0.6865672 0.7727273 0.7761194 0.7784261 0.7910448 0.8656716    0
SVM      0.6417910 0.6818182 0.7313433 0.7363636 0.7462687 0.8805970    0
```

Let's plot the resamples summary output.

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

In the above output you can see clearly how the algorithms performed in terms of ROC, Specificity and Sensitivity and how consistent has it been.

The `xgbDART` model appears to be the be best performing model overall because of the high ROC. But if you need a model that predicts the positives better, you might want to consider `MARS`, given its high sensitivity.

Either way, you can now make an informed decision on which model to pick.

17.9 9. Ensembling the predictions

17.9.1 9.1. How to ensemble predictions from multiple models using `caretEnsemble`?

So we have predictions from multiple individual models. To do this we had to run the `train()` function once for each model, store the models and pass it to the res

The `caretEnsemble` package lets you do just that.

All you have to do is put the names of all the algorithms you want to run in a vector and pass it to `caretEnsemble::caretList()` instead of `caret::train()`.

```
library(caretEnsemble)
tic()
# Stacking Algorithms - Run multiple algos in one call.
trainControl <- trainControl(method = "repeatedcv",
                             number = 10,
                             repeats = 3,
                             savePredictions = TRUE,
                             classProbs = TRUE)
```

```r
algorithmList <- c('rf', 'adaboost', 'earth', 'xgbDART', 'svmRadial')

set.seed(100)
models <- caretList(Purchase ~ .,
                    data = trainData,
                    trControl = trainControl,
                    methodList = algorithmList)

results <- resamples(models)
toc()
```

```
1315.176 sec elapsed
```

```r
summary(results)
```

```
Call:
summary.resamples(object = results)

Models: rf, adaboost, earth, xgbDART, svmRadial
Number of resamples: 30

Accuracy
               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
rf        0.7011494 0.7764706 0.7965116 0.8033148 0.8250684 0.9058824    0
adaboost  0.7176471 0.7717039 0.8023256 0.7993576 0.8235294 0.8705882    0
earth     0.7209302 0.7906977 0.8245554 0.8175940 0.8445964 0.8604651    0
xgbDART   0.7325581 0.8117647 0.8381449 0.8311919 0.8488372 0.8953488    0
svmRadial 0.7674419 0.8005814 0.8362517 0.8300693 0.8604651 0.9058824    0

Kappa
               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
rf        0.3518625 0.5184810 0.5504351 0.5737290 0.6253768 0.8040346    0
adaboost  0.4055944 0.5166378 0.5887759 0.5772070 0.6344328 0.7320149    0
earth     0.4102857 0.5609657 0.6232414 0.6121681 0.6757779 0.7147595    0
xgbDART   0.4434440 0.5912748 0.6641687 0.6410231 0.6816051 0.7848805    0
svmRadial 0.5034642 0.5763848 0.6464787 0.6366961 0.7043768 0.7996464    0
```

Plot the resamples output to compare the models.

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

Excellent! It is possible for further tune the model within `caretList` in a customised way.

17.9.2 9.2. How to combine the predictions of multiple models to form a final prediction

That one function simplified a whole lot of work in one line of code.

Here is another thought:

Is it possible to combine these predicted values from multiple models somehow and make a new ensemble that predicts better?

Turns out this can be done too, using the `caretStack()`. You just need to make sure you don't use the same `trainControl` you used to build the models.

```
# Create the trainControl
set.seed(101)
stackControl <- trainControl(method = "repeatedcv",
                             number = 10,
                             repeats = 3,
                             savePredictions = TRUE,
                             classProbs = TRUE)

# Ensemble the predictions of `models` to form a new combined prediction based on glm
stack.glm <- caretStack(models,
                        method = "glm",
                        metric="Accuracy",
                        trControl=stackControl)
print(stack.glm)
```

```
A glm ensemble of 2 base models: rf, adaboost, earth, xgbDART, svmRadial
```

```
Ensemble results:
Generalized Linear Model

2571 samples
   5 predictor
   2 classes: 'CH', 'MM'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 2314, 2314, 2314, 2314, 2313, 2313, ...
Resampling results:

  Accuracy   Kappa
  0.8331519  0.6444917
```

A point to consider: The ensembles tend to perform better if the predictions are less correlated with each other.

So you may want to try passing different types of models, both high and low performing rather than just stick to passing high accuracy models to the `caretStack`.

```r
print(stack.glm)
```

```
A glm ensemble of 2 base models: rf, adaboost, earth, xgbDART, svmRadial

Ensemble results:
Generalized Linear Model

2571 samples
   5 predictor
   2 classes: 'CH', 'MM'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 2314, 2314, 2314, 2314, 2313, 2313, ...
Resampling results:

  Accuracy   Kappa
  0.8331519  0.6444917
```

```r
# Predict on testData
stack_predicteds <- predict(stack.glm, newdata=testData4)
head(stack_predicteds)
```

```
[1] CH CH CH CH CH CH
Levels: CH MM
```

17.10 10. Conclusion

The purpose of this post was to cover the core pieces of the `caret` package and how you can effectively use it to build machine learning models.

This information should serve as a reference and also as a template you can use to build a standardised machine learning workflow, so you can develop it further from there.

```r
sessionInfo()
```

```
R version 3.5.3 (2019-03-11)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.2 LTS

Matrix products: default
BLAS: /opt/R/3.5.3/lib/R/lib/libRblas.so
LAPACK: /opt/R/3.5.3/lib/R/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] caretEnsemble_2.0.0 earth_5.1.1         plotmo_3.5.4
 [4] TeachingDemos_2.10  plotrix_3.7-5       Formula_1.2-3
 [7] RANN_2.6.1          skimr_1.0.5         tictoc_1.0
[10] readr_1.3.1         caret_6.0-84        ggplot2_3.1.1
[13] lattice_0.20-38

loaded via a namespace (and not attached):
 [1] tidyr_0.8.3         splines_3.5.3       foreach_1.4.4
 [4] prodlim_2018.04.18  assertthat_0.2.1    stats4_3.5.3
 [7] yaml_2.2.0          ipred_0.9-9         pillar_1.4.0
[10] backports_1.1.4     glue_1.3.1          digest_0.6.18
[13] randomForest_4.6-14 colorspace_1.4-1    recipes_0.1.5
[16] htmltools_0.3.6     Matrix_1.2-17       plyr_1.8.4
[19] timeDate_3043.102   pkgconfig_2.0.2     bookdown_0.10
[22] purrr_0.3.2         scales_1.0.0        gower_0.2.0
[25] lava_1.6.5          tibble_2.1.1        generics_0.0.2
[28] xgboost_0.82.1      withr_2.1.2         pbapply_1.4-0
[31] nnet_7.3-12         lazyeval_0.2.2      cli_1.1.0
[34] survival_2.44-1.1   magrittr_1.5        crayon_1.3.4
[37] evaluate_0.13       fansi_0.4.0         nlme_3.1-139
[40] MASS_7.3-51.4       class_7.3-15        tools_3.5.3
[43] data.table_1.12.2   hms_0.4.2           stringr_1.4.0
[46] kernlab_0.9-27      munsell_0.5.0       compiler_3.5.3
[49] e1071_1.7-1         fastAdaboost_1.0.0  rlang_0.3.4
[52] grid_3.5.3          iterators_1.0.10    rmarkdown_1.12
[55] gtable_0.3.0        ModelMetrics_1.2.2  codetools_0.2-16
[58] reshape2_1.4.3      R6_2.4.0            gridExtra_2.3
[61] lubridate_1.7.4     knitr_1.22          dplyr_0.8.0.1
[64] utf8_1.1.4          zeallot_0.1.0       rprojroot_1.3-2
[67] stringi_1.4.3       parallel_3.5.3      Rcpp_1.0.1
[70] vctrs_0.1.0         rpart_4.1-15        tidyselect_0.2.5
[73] xfun_0.6
```

Chapter 18

Feature selection, Variable Importance

Source: https://www.machinelearningplus.com/machine-learning/feature-selection/

18.1 Introduction

In real-world datasets, it is fairly common to have columns that are nothing but noise.

You are better off getting rid of such variables because of the memory space they occupy, the time and the computational esources it is going to cost, especially in large datasets.

Sometimes, you have a variable that makes business sense, but you are not sure if it actually helps in predicting the Y. You also need to consider the fact that, a feature that could be useful in one ML algorithm (say a decision tree) may go underrepresented or unused by another (like a regression model).

Having said that, it is still possible that a variable that shows poor signs of helping to explain the response variable (Y), can turn out to be significantly useful in the presence of (or combination with) other predictors. What I mean by that is, a variable might have a low correlation value of (~0.2) with Y. But in the presence of other variables, it can help to explain certain patterns/phenomenon that other variables can't explain.

In such cases, it can be hard to make a call whether to include or exclude such variables.

The strategies we are about to discuss can help fix such problems. Not only that, it will also help understand if a particular variable is important or not and how much it is contributing to the model

An important caveat. It is always best to have variables that have sound business logic backing the inclusion of a variable and rely solely on variable importance metrics.

Alright. Let's load up the 'Glaucoma' dataset where the goal is to predict if a patient has Glaucoma or not based on 63 different physiological measurements. You can directly run the codes or download the dataset here.

A lot of interesting examples ahead. Let's get started.

```r
# Load Packages and prepare dataset
library(TH.data)
library(caret)
library(tictoc)

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

```r
trainData <- GlaucomaM
head(trainData)
```

```
      ag    at    as    an    ai   eag   eat   eas   ean   eai  abrg  abrt
2  2.220 0.354 0.580 0.686 0.601 1.267 0.336 0.346 0.255 0.331 0.479 0.260
43 2.681 0.475 0.672 0.868 0.667 2.053 0.440 0.520 0.639 0.454 1.090 0.377
25 1.979 0.343 0.508 0.624 0.504 1.200 0.299 0.396 0.259 0.246 0.465 0.209
65 1.747 0.269 0.476 0.525 0.476 0.612 0.147 0.017 0.044 0.405 0.170 0.062
70 2.990 0.599 0.686 1.039 0.667 2.513 0.543 0.607 0.871 0.492 1.800 0.431
16 2.917 0.483 0.763 0.901 0.770 2.200 0.462 0.637 0.504 0.597 1.311 0.394
    abrs  abrn  abri    hic   mhcg  mhct   mhcs   mhcn   mhci   phcg
2  0.107 0.014 0.098  0.214  0.111 0.412  0.036  0.105 -0.022 -0.139
43 0.257 0.212 0.245  0.382  0.140 0.338  0.104  0.080  0.109 -0.015
25 0.112 0.041 0.103  0.195  0.062 0.356  0.045 -0.009 -0.048 -0.149
65 0.000 0.000 0.108 -0.030 -0.015 0.074 -0.084 -0.050  0.035 -0.182
70 0.494 0.601 0.274  0.383  0.089 0.233  0.145  0.023  0.007 -0.131
16 0.365 0.251 0.301  0.442  0.128 0.375  0.049  0.111  0.052 -0.088
     phct   phcs   phcn   phci   hvc  vbsg  vbst  vbss  vbsn  vbsi  vasg
2   0.242 -0.053  0.010 -0.139 0.613 0.303 0.103 0.088 0.022 0.090 0.062
43  0.296 -0.015 -0.015  0.036 0.382 0.676 0.181 0.186 0.141 0.169 0.029
25  0.206 -0.092 -0.081 -0.149 0.557 0.300 0.084 0.088 0.046 0.082 0.036
65 -0.097 -0.125 -0.138 -0.182 0.373 0.048 0.011 0.000 0.000 0.036 0.070
70  0.163  0.055 -0.131 -0.115 0.405 0.889 0.151 0.253 0.330 0.155 0.020
16  0.281 -0.067 -0.062 -0.088 0.507 0.972 0.213 0.316 0.197 0.246 0.043
    vast  vass  vasn  vasi  vbrg  vbrt  vbrs  vbrn  vbri  varg  vart  vars
2  0.000 0.011 0.032 0.018 0.075 0.039 0.021 0.002 0.014 0.756 0.009 0.209
43 0.001 0.007 0.011 0.010 0.370 0.127 0.099 0.050 0.093 0.410 0.006 0.105
25 0.002 0.004 0.016 0.013 0.081 0.034 0.019 0.007 0.021 0.565 0.014 0.132
65 0.005 0.030 0.033 0.002 0.005 0.001 0.000 0.000 0.004 0.380 0.032 0.147
70 0.001 0.004 0.008 0.007 0.532 0.103 0.173 0.181 0.075 0.228 0.011 0.026
16 0.001 0.005 0.028 0.009 0.467 0.136 0.148 0.078 0.104 0.540 0.008 0.133
    varn  vari   mdg   mdt   mds   mdn   mdi    tmg    tmt    tms    tmn
2  0.298 0.240 0.705 0.637 0.738 0.596 0.691 -0.236 -0.018 -0.230 -0.510
43 0.181 0.117 0.898 0.850 0.907 0.771 0.940 -0.211 -0.014 -0.165 -0.317
25 0.243 0.177 0.687 0.643 0.689 0.684 0.700 -0.185 -0.097 -0.235 -0.337
65 0.151 0.050 0.207 0.171 0.022 0.046 0.221 -0.148 -0.035 -0.449 -0.217
70 0.105 0.087 0.721 0.638 0.730 0.730 0.640 -0.052 -0.105  0.084 -0.012
16 0.232 0.167 0.927 0.842 0.953 0.906 0.898 -0.040  0.087  0.018 -0.094
      tmi    mr   rnf  mdic   emd    mv  Class
2  -0.158 0.841 0.410 0.137 0.239 0.035 normal
43 -0.192 0.924 0.256 0.252 0.329 0.022 normal
25 -0.020 0.795 0.378 0.152 0.250 0.029 normal
65 -0.091 0.746 0.200 0.027 0.078 0.023 normal
70 -0.054 0.977 0.193 0.297 0.354 0.034 normal
16 -0.051 0.965 0.339 0.333 0.442 0.028 normal
```

18.2 1. Boruta

Boruta is a feature ranking and selection algorithm based on random forests algorithm.

The advantage with Boruta is that it clearly decides if a variable is important or not and helps to select variables that are statistically significant. Besides, you can adjust the strictness of the algorithm by adjusting the p values that defaults to 0.01 and the `maxRuns`.

`maxRuns` is the number of times the algorithm is run. The higher the `maxRuns` the more selective you get in picking the variables. The default value is 100.

In the process of deciding if a feature is important or not, some features may be marked by Boruta as 'Tentative'. Sometimes increasing the maxRuns can help resolve the 'Tentativeness' of the feature.

Lets see an example based on the Glaucoma dataset from `TH.data` package that I created earlier.

```
# install.packages('Boruta')
library(Boruta)
```

The `boruta` function uses a formula interface just like most predictive modeling functions. So the first argument to `boruta()` is the formula with the response variable on the left and all the predictors on the right.

By placing a dot, all the variables in `trainData` other than Class will be included in the model.

The `doTrace` argument controls the amount of output printed to the console. Higher the value, more the log details you get. So save space I have set it to 0, but try setting it to 1 and 2 if you are running the code.

Finally the output is stored in `boruta_output`.

```
# Perform Boruta search
boruta_output <- Boruta(Class ~ ., data=na.omit(trainData), doTrace=0)
```

Let's see what the boruta_output contains.

```
names(boruta_output)
```

```
 [1] "finalDecision" "ImpHistory"    "pValue"        "maxRuns"
 [5] "light"         "mcAdj"         "timeTaken"     "roughfixed"
 [9] "call"          "impSource"
```

```
# Get significant variables including tentatives
boruta_signif <- getSelectedAttributes(boruta_output, withTentative = TRUE)
print(boruta_signif)
```

```
 [1] "ag"   "as"   "eag"  "eas"  "ean"  "abrg" "abrs" "abrn" "abri" "hic"
[11] "mhcg" "mhcs" "mhcn" "mhci" "phcg" "phcn" "phci" "hvc"  "vbsg" "vbss"
[21] "vbsn" "vbsi" "vasg" "vass" "vasi" "vbrg" "vbrs" "vbrn" "vbri" "varg"
[31] "vart" "vars" "varn" "vari" "mdn"  "tmg"  "tmt"  "tms"  "tmi"  "mr"
[41] "rnf"  "mdic" "emd"
```

If you are not sure about the tentative variables being selected for granted, you can choose a `TentativeRoughFix` on `boruta_output`.

```
# Do a tentative rough fix
roughFixMod <- TentativeRoughFix(boruta_output)
boruta_signif <- getSelectedAttributes(roughFixMod)
print(boruta_signif)
```

```
 [1] "as"   "ean"  "abrg" "abrs" "abrn" "abri" "hic"  "mhcg" "mhcn" "mhci"
[11] "phcg" "phcn" "phci" "hvc"  "vbss" "vbsn" "vbsi" "vasg" "vass" "vbrg"
[21] "vbrs" "vbrn" "vbri" "varg" "vart" "vars" "varn" "vari" "tmg"  "tms"
[31] "tmi"  "rnf"  "mdic"
```

There you go. Boruta has decided on the 'Tentative' variables on our behalf. Let's find out the importance scores of these variables.

```
# Variable Importance Scores
imps <- attStats(roughFixMod)
```

```
imps2 = imps[imps$decision != 'Rejected', c('meanImp', 'decision')]
head(imps2[order(-imps2$meanImp), ])  # descending sort
```

```
       meanImp  decision
vari 12.173136 Confirmed
varg 11.648743 Confirmed
vars 10.662532 Confirmed
phci  8.275650 Confirmed
hic   8.117015 Confirmed
varn  7.810182 Confirmed
```

Let's plot it to see the importances of these variables.

```
# Plot variable importance
plot(boruta_output, cex.axis=.7, las=2, xlab="", main="Variable Importance")
```

This plot reveals the importance of each of the features.

The columns in green are 'confirmed' and the ones in red are not. There are couple of blue bars representing `ShadowMax` and `ShadowMin`. They are not actual features, but are used by the boruta algorithm to decide if a variable is important or not.

18.3 2. Variable Importance from Machine Learning Algorithms

Another way to look at feature selection is to consider variables most used by various ML algorithms the most to be important.

Depending on how the machine learning algorithm learns the relationship between X's and Y, different machine learning algorithms may possibly end up using different variables (but mostly common vars) to various degrees.

What I mean by that is, the variables that proved useful in a tree-based algorithm like `rpart`, can turn out to be less useful in a regression-based model. So all variables need not be equally useful to all algorithms.

So how do we find the variable importance for a given ML algo?

`train()` the desired model using the caret package. Then, use `varImp()` to determine the feature importances.

You may want to try out multiple algorithms, to get a feel of the usefulness of the features across algos.

18.3.1 rpart

```
# Train an rpart model and compute variable importance.
library(caret)
set.seed(100)
rPartMod <- train(Class ~ .,
                  data=trainData,
                  method="rpart")

rpartImp <- varImp(rPartMod)
print(rpartImp)
```

```
rpart variable importance

  only 20 most important variables shown (out of 62)

     Overall
varg  100.00
vari   93.19
vars   85.20
varn   76.86
tmi    72.31
vbss    0.00
eai     0.00
tmg     0.00
tmt     0.00
vbst    0.00
vasg    0.00
at      0.00
abrg    0.00
vbsg    0.00
eag     0.00
phcs    0.00
abrs    0.00
mdic    0.00
abrt    0.00
ean     0.00
```

Only 5 of the 63 features was used by rpart and if you look closely, the 5 variables used here are in the top 6 that boruta selected.

Let's do one more: the variable importances from Regularized Random Forest (RRF) algorithm.

18.3.2 Regularized Random Forest (RRF)

```
tic()
# Train an RRF model and compute variable importance.
set.seed(100)
rrfMod <- train(Class ~ .,
                data = trainData,
                method = "RRF")

rrfImp <- varImp(rrfMod, scale=F)
toc()
```

```
419.836 sec elapsed
```

```
rrfImp
```

```
RRF variable importance

  only 20 most important variables shown (out of 62)

     Overall
varg 24.6101
vari 19.2878
vars  5.4366
tmi   4.1948
mhcg  3.9126
mhci  2.8946
hic   2.5906
hvc   2.2555
mv    2.0451
vasg  2.0381
phci  1.8706
tms   1.4188
mdic  1.3293
tmt   1.3080
phcg  1.2273
abrs  1.2221
vass  1.1631
phct  1.1460
tmg   1.1169
tmn   0.9702
```

```
plot(rrfImp, top = 20, main='Variable Importance')
```

The topmost important variables are pretty much from the top tier of Boruta's selections.

Some of the other algorithms available in `train()` that you can use to compute varImp are the following:

`ada, AdaBag, AdaBoost.M1, adaboost, bagEarth, bagEarthGCV, bagFDA, bagFDAGCV, bartMachine, blasso, BstLm`

18.4 3. Lasso Regression

Least Absolute Shrinkage and Selection Operator (LASSO) regression is a type of regularization method that penalizes with L1-norm.

It basically imposes a cost to having large weights (value of coefficients). And its called L1 regularization, because the cost added, is proportional to the absolute value of weight coefficients.

As a result, in the process of shrinking the coefficients, it eventually reduces the coefficients of certain unwanted features all the to zero. That is, it removes the unneeded variables altogether.

So effectively, LASSO regression can be considered as a variable selection technique as well.

```
library(glmnet)

# online data
# trainData <- read.csv('https://raw.githubusercontent.com/selva86/datasets/master/GlaucomaM.csv')

trainData <- read.csv(file.path(data_raw_dir, "glaucoma.csv"))

x <- as.matrix(trainData[,-63]) # all X vars
y <- as.double(as.matrix(ifelse(trainData[, 63]=='normal', 0, 1))) # Only Class
```

```
# Fit the LASSO model (Lasso: Alpha = 1)
set.seed(100)
cv.lasso <- cv.glmnet(x, y, family='binomial', alpha=1, parallel=TRUE, standardize=TRUE, type.measure='a

# Results
plot(cv.lasso)
```

Let's see how to interpret this plot.

The X axis of the plot is the log of `lambda`. That means when it is 2 here, the lambda value is actually 100.

The numbers at the top of the plot show how many predictors were included in the model. The position of red dots along the Y-axis tells what `AUC` we got when you include as many variables shown on the top x-axis.

You can also see two dashed vertical lines.

The first one on the left points to the lambda with the lowest mean squared error. The one on the right point to the number of variables with the highest deviance within 1 standard deviation.

The best lambda value is stored inside 'cv.lasso$lambda.min'.

```
# plot(cv.lasso$glmnet.fit, xvar="lambda", label=TRUE)
cat('Min Lambda: ', cv.lasso$lambda.min, '\n 1Sd Lambda: ', cv.lasso$lambda.1se)
```

```
Min Lambda:  0.01166507
 1Sd Lambda:  0.2513163
```

```
df_coef <- round(as.matrix(coef(cv.lasso, s=cv.lasso$lambda.min)), 2)

# See all contributing variables
df_coef[df_coef[, 1] != 0, ]
```

```
(Intercept)          at          as         eat        mhci        phcs
       3.65       -0.17       -2.05       -0.53        6.22       -0.83
```

```
      phci         hvc        vass        vbrn        vars         mdt
      6.03       -4.15      -23.72       -0.26      -25.86       -2.34
       mds         mdn         mdi         tmg         tms         tmi
      0.50        0.83        3.00        0.01        3.02        2.65
        mv
      4.94
```

The above output shows what variables LASSO considered important. A high positive or low negative implies more important is that variable.

18.5 4. Step wise Forward and Backward Selection

Stepwise regression can be used to select features if the Y variable is a numeric variable. It is particularly used in selecting best linear regression models.

It searches for the best possible regression model by iteratively selecting and dropping variables to arrive at a model with the lowest possible AIC.

It can be implemented using the `step()` function and you need to provide it with a lower model, which is the base model from which it won't remove any features and an upper model, which is a full model that has all possible features you want to have.

Our case is not so complicated (< 20 vars), so lets just do a simple stepwise in 'both' directions.

I will use the `ozone` dataset for this where the objective is to predict the `ozone_reading` based on other weather related observations.

```r
# Load data
# online
# trainData <- read.csv("http://rstatistics.net/wp-content/uploads/2015/09/ozone1.csv",
#                       stringsAsFactors=F)
trainData <- read.csv(file.path(data_raw_dir, "ozone1.csv"))
print(head(trainData))
```

```
  Month Day_of_month Day_of_week ozone_reading pressure_height Wind_speed
1     1            1           4             3            5480          8
2     1            2           5             3            5660          6
3     1            3           6             3            5710          4
4     1            4           7             5            5700          3
5     1            5           1             5            5760          3
6     1            6           2             6            5720          4
  Humidity Temperature_Sandburg Temperature_ElMonte Inversion_base_height
1 20.00000             40.53473            39.77461              5000.000
2 40.96306             38.00000            46.74935              4108.904
3 28.00000             40.00000            49.49278              2693.000
4 37.00000             45.00000            52.29403               590.000
5 51.00000             54.00000            45.32000              1450.000
6 69.00000             35.00000            49.64000              1568.000
  Pressure_gradient Inversion_temperature Visibility
1               -15              30.56000        200
2               -14              48.02557        300
3               -25              47.66000        250
4               -24              55.04000        100
5                25              57.02000         60
6                15              53.78000         60
```

The data is ready. Let's perform the stepwise.

```
# Step 1: Define base intercept only model
base.mod <- lm(ozone_reading ~ 1 , data=trainData)

# Step 2: Full model with all predictors
all.mod <- lm(ozone_reading ~ . , data= trainData)

# Step 3: Perform step-wise algorithm. direction='both' implies both forward and backward stepwise
stepMod <- step(base.mod, scope = list(lower = base.mod, upper = all.mod), direction = "both", trace = 

# Step 4: Get the shortlisted variable.
shortlistedVars <- names(unlist(stepMod[[1]]))
shortlistedVars <- shortlistedVars[!shortlistedVars %in% "(Intercept)"] # remove intercept

# Show
print(shortlistedVars)
```

```
[1] "Temperature_Sandburg"  "Humidity"              "Temperature_ElMonte"
[4] "Month"                 "pressure_height"       "Inversion_base_height"
```

The selected model has the above 6 features in it.

But if you have too many features (> 100) in training data, then it might be a good idea to split the dataset into chunks of 10 variables each with Y as mandatory in each dataset. Loop through all the chunks and collect the best features.

We are doing it this way because some variables that came as important in a training data with fewer features may not show up in a linear reg model built on lots of features.

Finally, from a pool of shortlisted features (from small chunk models), run a full stepwise model to get the final set of selected features.

You can take this as a learning assignment to be solved within 20 minutes.

18.6 5. Relative Importance from Linear Regression

This technique is specific to linear regression models.

Relative importance can be used to assess which variables contributed how much in explaining the linear model's R-squared value. So, if you sum up the produced importances, it will add up to the model's R-sq value.

In essence, it is not directly a feature selection method, because you have already provided the features that go in the model. But after building the model, the `relaimpo` can provide a sense of how important each feature is in contributing to the R-sq, or in other words, in 'explaining the Y variable'.

So, how to calculate relative importance?

It is implemented in the `relaimpo` package. Basically, you build a linear regression model and pass that as the main argument to `calc.relimp()`. relaimpo has multiple options to compute the relative importance, but the recommended method is to use `type='lmg'`, as I have done below.

```
# install.packages('relaimpo')
library(relaimpo)

# Build linear regression model
model_formula = ozone_reading ~ Temperature_Sandburg + Humidity + Temperature_ElMonte + Month + pressure
```

```
lmMod <- lm(model_formula, data=trainData)

# calculate relative importance
relImportance <- calc.relimp(lmMod, type = "lmg", rela = F)

# Sort
cat('Relative Importances: \n')
```

```
Relative Importances:
```

```
sort(round(relImportance$lmg, 3), decreasing=TRUE)
```

```
  Temperature_ElMonte  Temperature_Sandburg       pressure_height
                0.214                 0.203                 0.104
Inversion_base_height              Humidity                 Month
                0.096                 0.086                 0.012
```

Additionally, you can use bootstrapping (using `boot.relimp`) to compute the confidence intervals of the produced relative importances.

```
bootsub <- boot.relimp(ozone_reading ~ Temperature_Sandburg + Humidity + Temperature_ElMonte + Month + p
                       b = 1000, type = 'lmg', rank = TRUE, diff = TRUE)

plot(booteval.relimp(bootsub, level=.95))
```

18.7 6. Recursive Feature Elimination (RFE)

Recursive feature elimnation (rfe) offers a rigorous way to determine the important variables before you even feed them into a ML algo.

It can be implemented using the `rfe()` from caret package.

The rfe() also takes two important parameters.

- `sizes`
- `rfeControl`

So, what does `sizes` and `rfeControl` represent?

The sizes determines the number of most important features the rfe should iterate. Below, I have set the size as 1 to 5, 10, 15 and 18.

Secondly, the `rfeControl` parameter receives the output of the `rfeControl()`. You can set what type of variable evaluation algorithm must be used. Here, I have used random forests based rfFuncs. The `method='repeatedCV'` means it will do a repeated k-Fold cross validation with `repeats=5`.

Once complete, you get the accuracy and kappa for each model size you provided. The final selected model subset size is marked with a * in the rightmost selected column.

```
str(trainData)
```

```
'data.frame':   366 obs. of  13 variables:
 $ Month                : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Day_of_month         : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Day_of_week          : int  4 5 6 7 1 2 3 4 5 6 ...
 $ ozone_reading        : num  3 3 3 5 5 6 4 4 6 7 ...
 $ pressure_height      : num  5480 5660 5710 5700 5760 5720 5790 5790 5700 5700 ...
 $ Wind_speed           : int  8 6 4 3 3 4 6 3 3 3 ...
 $ Humidity             : num  20 41 28 37 51 ...
 $ Temperature_Sandburg : num  40.5 38 40 45 54 ...
 $ Temperature_ElMonte  : num  39.8 46.7 49.5 52.3 45.3 ...
 $ Inversion_base_height: num  5000 4109 2693 590 1450 ...
 $ Pressure_gradient    : num  -15 -14 -25 -24 25 15 -33 -28 23 -2 ...
 $ Inversion_temperature: num  30.6 48 47.7 55 57 ...
 $ Visibility           : int  200 300 250 100 60 60 100 250 120 120 ...
```

```
tic()
set.seed(100)
options(warn=-1)

subsets <- c(1:5, 10, 15, 18)

ctrl <- rfeControl(functions = rfFuncs,
                   method = "repeatedcv",
                   repeats = 5,
                   verbose = FALSE)

lmProfile <- rfe(x=trainData[, c(1:3, 5:13)], y=trainData$ozone_reading,
                 sizes = subsets,
                 rfeControl = ctrl)
toc()
```

```
125.789 sec elapsed
```

```
lmProfile
```

```
Recursive feature selection

Outer resampling method: Cross-Validated (10 fold, repeated 5 times)

Resampling performance over subset size:

 Variables  RMSE Rsquared   MAE RMSESD RsquaredSD  MAESD Selected
         1 5.267   0.5714 4.045 1.0495    0.17150 0.8594
         2 4.005   0.7480 3.086 0.5267    0.07373 0.3766
         3 3.950   0.7545 3.056 0.4757    0.06830 0.3736
         4 3.928   0.7575 3.016 0.5101    0.06825 0.4125
         5 3.844   0.7684 2.945 0.5401    0.07007 0.4199
        10 3.754   0.7786 2.867 0.5854    0.06988 0.4624        *
        12 3.782   0.7759 2.880 0.5644    0.06901 0.4496

The top 5 variables (out of 10):
   Temperature_ElMonte, Pressure_gradient, Temperature_Sandburg, Inversion_temperature, Humidity
```

So, it says, Temperature_ElMonte, Pressure_gradient, Temperature_Sandburg, Inversion_temperature, Humidity are the top 5 variables in that order.

And the best model size out of the provided models sizes (in subsets) is 10.

You can see all of the top 10 variables from '`lmProfile$optVariables`' that was created using `rfe` function above.

18.8 7. Genetic Algorithm

You can perform a supervised feature selection with genetic algorithms using the `gafs()`. This is **quite resource expensive** so consider that before choosing the number of iterations (iters) and the number of repeats in `gafsControl()`.

```
tic()
# Define control function
ga_ctrl <- gafsControl(functions = rfGA,  # another option is `caretGA`.
                       method = "cv",
                       repeats = 3)

# Genetic Algorithm feature selection
set.seed(100)
ga_obj <- gafs(x=trainData[, c(1:3, 5:13)],
               y=trainData[, 4],
               iters = 3,   # normally much higher (100+)
               gafsControl = ga_ctrl)
toc()
```

```
1209.598 sec elapsed
```

```
ga_obj
```

```
Genetic Algorithm Feature Selection
```

```
366 samples
12 predictors

Maximum generations: 3
Population per generation: 50
Crossover probability: 0.8
Mutation probability: 0.1
Elitism: 0

Internal performance values: RMSE, Rsquared
Subset selection driven to minimize internal RMSE

External performance values: RMSE, Rsquared, MAE
Best iteration chose by minimizing external RMSE
External resampling method: Cross-Validated (10 fold)

During resampling:
  * the top 5 selected variables (out of a possible 12):
    Inversion_base_height (100%), Month (100%), Pressure_gradient (100%), Temperature_ElMonte (100%), Vi
  * on average, 7.9 variables were selected (min = 6, max = 10)

In the final search using the entire training set:
   * 9 features selected at iteration 2 including:
     Month, Day_of_month, Day_of_week, pressure_height, Wind_speed ...
   * external performance at this iteration is

       RMSE    Rsquared         MAE
     3.6944      0.7878      2.8173
```

```r
# Optimal variables
ga_obj$optVariables
```

```
[1] "Month"                 "Day_of_month"          "Day_of_week"
[4] "pressure_height"       "Wind_speed"            "Humidity"
[7] "Temperature_ElMonte"   "Inversion_base_height" "Pressure_gradient"
```

'Month' 'Day_of_month' 'Wind_speed' 'Temperature_ElMonte' 'Pressure_gradient' 'Visibility'

So the optimal variables according to the genetic algorithms are listed above. But, I wouldn't use it just yet because, the above variant was tuned for only 3 iterations, which is quite low. I had to set it so low to save computing time.

18.9 8. Simulated Annealing

Simulated annealing is a global search algorithm that allows a suboptimal solution to be accepted in hope that a better solution will show up eventually.

It works by making small random changes to an initial solution and sees if the performance improved. The change is accepted if it improves, else it can still be accepted if the difference of performances meet an acceptance criteria.

In caret it has been implemented in the `safs()` which accepts a control parameter that can be set using the `safsControl()` function.

`safsControl` is similar to other control functions in caret (like you saw in rfe and ga), and additionally it accepts an improve parameter which is the number of iterations it should wait without improvement until the values are reset to previous iteration.

```
tic()
# Define control function
sa_ctrl <- safsControl(functions = rfSA,
                        method = "repeatedcv",
                        repeats = 3,
                        improve = 5) # n iterations without improvement before a reset

# Genetic Algorithm feature selection
set.seed(100)
sa_obj <- safs(x=trainData[, c(1:3, 5:13)],
               y=trainData[, 4],
               safsControl = sa_ctrl)
toc()
```

```
201.833 sec elapsed
```

```
sa_obj
```

```
Simulated Annealing Feature Selection

366 samples
12 predictors

Maximum search iterations: 10
Restart after 5 iterations without improvement (0.3 restarts on average)

Internal performance values: RMSE, Rsquared
Subset selection driven to minimize internal RMSE

External performance values: RMSE, Rsquared, MAE
Best iteration chose by minimizing external RMSE
External resampling method: Cross-Validated (10 fold, repeated 3 times)

During resampling:
  * the top 5 selected variables (out of a possible 12):
    Month (63.3%), Temperature_Sandburg (60%), Pressure_gradient (53.3%), Temperature_ElMonte (53.3%), 
  * on average, 5.4 variables were selected (min = 3, max = 7)

In the final search using the entire training set:
   * 5 features selected at iteration 9 including:
     Month, Day_of_month, Humidity, Temperature_ElMonte, Visibility
   * external performance at this iteration is

       RMSE    Rsquared         MAE
     4.1921      0.7241      3.2183
```

```
# Optimal variables
print(sa_obj$optVariables)
```

```
[1] "Month"               "Day_of_month"        "Humidity"
[4] "Temperature_ElMonte" "Visibility"
```

18.10 9. Information Value and Weights of Evidence

The Information Value can be used to judge how important a given categorical variable is in explaining the binary Y variable. It goes well with logistic regression and other classification models that can model binary variables.

Let's try to find out how important the categorical variables are in predicting if an individual will earn > 50k from the `adult.csv` dataset. Just run the code below to import the dataset.

```
library(InformationValue)

# online data
# inputData <- read.csv("http://rstatistics.net/wp-content/uploads/2015/09/adult.csv")

inputData <- read.csv(file.path(data_raw_dir, "adult.csv"))
print(head(inputData))
```

```
  AGE        WORKCLASS FNLWGT EDUCATION EDUCATIONNUM      MARITALSTATUS
1  39        State-gov  77516 Bachelors           13      Never-married
2  50 Self-emp-not-inc  83311 Bachelors           13 Married-civ-spouse
3  38          Private 215646   HS-grad            9           Divorced
4  53          Private 234721      11th            7 Married-civ-spouse
5  28          Private 338409 Bachelors           13 Married-civ-spouse
6  37          Private 284582   Masters           14 Married-civ-spouse
         OCCUPATION  RELATIONSHIP  RACE    SEX CAPITALGAIN CAPITALLOSS
1      Adm-clerical Not-in-family White   Male        2174           0
2   Exec-managerial       Husband White   Male           0           0
3 Handlers-cleaners Not-in-family White   Male           0           0
4 Handlers-cleaners       Husband Black   Male           0           0
5    Prof-specialty          Wife Black Female           0           0
6   Exec-managerial          Wife White Female           0           0
  HOURSPERWEEK NATIVECOUNTRY ABOVE50K
1           40 United-States        0
2           13 United-States        0
3           40 United-States        0
4           40 United-States        0
5           40          Cuba        0
6           40 United-States        0
```

```
# Choose Categorical Variables to compute Info Value.
cat_vars <- c ("WORKCLASS", "EDUCATION", "MARITALSTATUS", "OCCUPATION", "RELATIONSHIP", "RACE", "SEX",

factor_vars <- cat_vars



# Init Output
df_iv <- data.frame(VARS=cat_vars, IV=numeric(length(cat_vars)), STRENGTH=character(length(cat_vars)),

# Get Information Value for each variable
for (factor_var in factor_vars){
  df_iv[df_iv$VARS == factor_var, "IV"] <- InformationValue::IV(X=inputData[, factor_var], Y=inputData$
  df_iv[df_iv$VARS == factor_var, "STRENGTH"] <- attr(InformationValue::IV(X=inputData[, factor_var], Y=
}

# Sort
```

```
df_iv <- df_iv[order(-df_iv$IV), ]

df_iv
```

```
           VARS         IV            STRENGTH
5  RELATIONSHIP 1.53560810   Highly Predictive
3 MARITALSTATUS 1.33882907   Highly Predictive
4    OCCUPATION 0.77622839   Highly Predictive
2     EDUCATION 0.74105372   Highly Predictive
7           SEX 0.30328938   Highly Predictive
1     WORKCLASS 0.16338802   Highly Predictive
8 NATIVECOUNTRY 0.07939344 Somewhat Predictive
6          RACE 0.06929987 Somewhat Predictive
```

Here is what the quantum of Information Value means:

Less than 0.02, then the predictor is not useful for modeling (separating the Goods from the Bads)

0.02 to 0.1, then the predictor has only a weak relationship. 0.1 to 0.3, then the predictor has a medium strength relationship. 0.3 or higher, then the predictor has a strong relationship. That was about IV. Then what is Weight of Evidence?

Weights of evidence can be useful to find out how important a given categorical variable is in explaining the 'events' (called 'Goods' in below table.)

The 'Information Value' of the categorical variable can then be derived from the respective WOE values.

IV=(perc good of all goods−perc bad of all bads) *WOE

The 'WOETable' below given the computation in more detail.

```
WOETable(X=inputData[, 'WORKCLASS'], Y=inputData$ABOVE50K)
```

```
               CAT GOODS  BADS TOTAL        PCT_G        PCT_B        WOE
1                ?   191  1645  1836 0.0242940728 0.0665453074 -1.0076506
2      Federal-gov   371   589   960 0.0471890104 0.0238268608  0.6833475
3        Local-gov   617  1476  2093 0.0784787586 0.0597087379  0.2733496
4     Never-worked     7     7     7 0.0008903587 0.0002831715  1.1455716
5          Private  4963 17733 22696 0.6312643093 0.7173543689 -0.1278453
6     Self-emp-inc   622   494  1116 0.0791147291 0.0199838188  1.3759762
7 Self-emp-not-inc   724  1817  2541 0.0920885271 0.0735032362  0.2254209
8        State-gov   353   945  1298 0.0448995167 0.0382281553  0.1608547
9      Without-pay    14    14    14 0.0017807174 0.0005663430  1.1455716
            IV
1 0.0425744832
2 0.0159644662
3 0.0051307781
4 0.0006955764
5 0.0110062102
6 0.0813627242
7 0.0041895135
8 0.0010731201
9 0.0013911528
```

The total IV of a variable is the sum of IV's of its categories.

18.11 10. DALEX Package

The `DALEX` is a powerful package that explains various things about the variables used in an ML model.

For example, using the `variable_dropout()` function you can find out how important a variable is based on a dropout loss, that is how much loss is incurred by removing a variable from the model.

Apart from this, it also has the `single_variable()` function that gives you an idea of how the model's output will change by changing the values of one of the X's in the model.

It also has the `single_prediction()` that can decompose a single model prediction so as to understand which variable caused what effect in predicting the value of Y.

```
library(randomForest)
library(DALEX)

# Load data
# inputData <- read.csv("http://rstatistics.net/wp-content/uploads/2015/09/adult.csv")

inputData <- read.csv(file.path(data_raw_dir, "adult.csv"))

# Train random forest model
rf_mod <- randomForest(factor(ABOVE50K) ~ ., data=inputData, ntree=100)
rf_mod
```

```
Call:
 randomForest(formula = factor(ABOVE50K) ~ ., data = inputData,      ntree = 100)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 3

        OOB estimate of  error rate: 13.63%
Confusion matrix:
      0    1 class.error
0 23053 1667  0.06743528
1  2772 5069  0.35352634
```

```
# Variable importance with DALEX
explained_rf <- explain(rf_mod, data=inputData, y=inputData$ABOVE50K)

# Get the variable importances
varimps = variable_dropout(explained_rf, type='raw')

print(varimps)
```

```
        variable dropout_loss        label
1   _full_model_      39.0698 randomForest
2       ABOVE50K      39.0698 randomForest
3           RACE      42.1631 randomForest
4  NATIVECOUNTRY      46.6201 randomForest
5            SEX      47.2284 randomForest
6    CAPITALLOSS      47.8775 randomForest
7      WORKCLASS      51.2492 randomForest
8      EDUCATION      64.7537 randomForest
9   EDUCATIONNUM      66.2575 randomForest
10        FNLWGT      68.2419 randomForest
```

```
11  HOURSPERWEEK      68.6068 randomForest
12 MARITALSTATUS      71.1565 randomForest
13  RELATIONSHIP      72.3072 randomForest
14   CAPITALGAIN      84.4786 randomForest
15    OCCUPATION      86.8011 randomForest
16           AGE      87.7040 randomForest
17    _baseline_     330.8898 randomForest
```

```
plot(varimps)
```

18.12 Conclusion

Hope you find these methods useful. As it turns out different methods showed different variables as important, or at least the degree of importance changed. This need not be a conflict, because each method gives a different perspective of how the variable can be useful depending on how the algorithms learn `Y ~ x`. So its cool.

If you find any code breaks or bugs, report the issue here or just write it below.

Chapter 19

Linear Regression. World Happiness

19.1 Introduction

Source: http://enhancedatascience.com/2017/04/25/r-basics-linear-regression-with-r/ Data: https://www.kaggle.com/unsdsn/world-happiness

Linear regression is one of the basics of statistics and machine learning. Hence, it is a must-have to know how to perform a linear regression with R and how to interpret the results.

Linear regression algorithm will fit the best straight line that fits the data? To do so, it will minimise the squared distance between the points of the dataset and the fitted line.

For this tutorial, we will use the World Happiness report dataset from Kaggle. This report analyses the Happiness of each country according to several factors such as wealth, health, family life, … Our goal will be to find the most important factors of happiness. What a noble goal!

19.2 A quick exploration of the data

Before fitting any model, we need to know our data better. First, let's import the data into R. Please download the dataset from Kaggle and put it in your working directory.

The code below imports the data as data.table and clean the column names (a lot of . were appearing in the original ones)

```
require(data.table)
data_happiness_dir <- file.path(data_raw_dir, "happiness")

Happiness_Data = data.table(read.csv(file.path(data_happiness_dir, '2016.csv')))
colnames(Happiness_Data) <- gsub('.','',colnames(Happiness_Data), fixed=T)
```

Now, let's plot a Scatter Plot Matrix to get a grasp of how our variables are related one to another. To do so, the GGally package is great.

```
require(ggplot2)
require(GGally)
ggpairs(Happiness_Data[,c(4,7:13), with=F], lower = list( continuous = "smooth"))
```

All the variables are positively correlated with the Happiness score. We can expect that most of the coefficients in the linear regression will be positive. However, the correlation between the variable is often more than 0.5, so we can expect that multicollinearity will appear in the regression.

In the data, we also have access to the Country where the score was computed. Even if it's not useful for the regression, let's plot the data on a map!

```
require('rworldmap')
library(reshape2)

map.world <- map_data(map="world")

dataPlot<- melt(Happiness_Data, id.vars ='Country',
                measure.vars = colnames(Happiness_Data)[c(4,7:13)])

#Correcting names that are different
```

```
dataPlot[Country == 'United States', Country:='USA']
dataPlot[Country == 'United Kingdoms', Country:='UK']

##Rescaling each variable to have nice gradient
dataPlot[,value:=value/max(value), by=variable]
dataMap = data.table(merge(map.world, dataPlot,
                           by.x='region',
                           by.y='Country',
                           all.x=T))
dataMap = dataMap[order(order)]
dataMap = dataMap[order(order)][!is.na(variable)]
gg <- ggplot()
gg <- gg +
    geom_map(data=dataMap, map=dataMap,
             aes(map_id = region, x=long, y=lat, fill=value)) +
    # facet_wrap(~variable, scale='free')
    facet_wrap(~variable)
gg <- gg + scale_fill_gradient(low = "navy", high = "lightblue")
gg <- gg + coord_equal()
```

The code above is a classic code for a map. A few important points:

We reordered the point before plotting to avoid some artefacts. The merge is a right outer join, all the points of the map need to be kept. Otherwise, points will be missing which will mess up the map. Each variable is rescaled so that a facet_wrap can be used. Here, the absolute level of a variable is not of primary interest. This is the relative level of a variable between countries that we want to visualise.

```
gg
```

The distinction between North and South is quite visible. In addition to this, countries that have suffered from the crisis are also really visible.

19.3 Linear regression with R

Now that we have taken a look at our data, a first model can be fitted. The explanatory variables are the DGP per capita, the life expectancy, the level of freedom and the trust in the government.

```
##First model
model1 <- lm(HappinessScore ~ EconomyGDPperCapita + Family +
                 HealthLifeExpectancy + Freedom + TrustGovernmentCorruption,
             data=Happiness_Data)
```

19.4 Regression summary

The summary function provides a very easy way to assess a linear regression in R.

```
require(stargazer)

##Quick summary
sum1=summary(model1)
sum1
```

```
Call:
lm(formula = HappinessScore ~ EconomyGDPperCapita + Family +
    HealthLifeExpectancy + Freedom + TrustGovernmentCorruption,
    data = Happiness_Data)

Residuals:
     Min       1Q   Median       3Q      Max
-1.48327 -0.28172 -0.02771  0.32803  1.46147

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                 2.2119     0.1502  14.731  < 2e-16 ***
EconomyGDPperCapita         0.6971     0.2094   3.329   0.0011 **
Family                      1.2344     0.2289   5.393 2.62e-07 ***
HealthLifeExpectancy        1.4623     0.3430   4.263 3.53e-05 ***
Freedom                     1.5588     0.3733   4.175 5.01e-05 ***
TrustGovernmentCorruption   0.9590     0.4546   2.110   0.0365 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5353 on 151 degrees of freedom
Multiple R-squared:  0.7872,    Adjusted R-squared:  0.7802
F-statistic: 111.7 on 5 and 151 DF,  p-value: < 2.2e-16
```

```
stargazer(model1,type='text')
```

```
=====================================================
                              Dependent variable:
                          ---------------------------
                                HappinessScore
-----------------------------------------------------
EconomyGDPperCapita                0.697***
```

```
                                  (0.209)

Family                           1.234***
                                  (0.229)

HealthLifeExpectancy             1.462***
                                  (0.343)

Freedom                          1.559***
                                  (0.373)

TrustGovernmentCorruption         0.959**
                                  (0.455)

Constant                         2.212***
                                  (0.150)

-----------------------------------------------------
Observations                        157
R2                                 0.787
Adjusted R2                        0.780
Residual Std. Error          0.535 (df = 151)
F Statistic               111.742*** (df = 5; 151)
=====================================================
Note:                     *p<0.1; **p<0.05; ***p<0.01
```

A quick interpretation:

- All the coefficient are significative at a .05 threshold
- The overall model is also significative
- It explains 78.7% of Happiness in the dataset
- As expected all the relationship between the explanatory variables and the output variable are positives.

The model is doing well!

You can also easily get a given indicator of the model performance, such as R^2, the different coefficients or the p-value of the overall model.

```
##R²
sum1$r.squared*100
```

```
[1] 78.72371
```

```
##Coefficients
sum1$coefficients
```

```
                          Estimate Std. Error   t value     Pr(>|t|)
(Intercept)               2.2118706  0.1501522 14.730861 5.198486e-31
EconomyGDPperCapita       0.6971199  0.2094382  3.328524 1.096943e-03
Family                    1.2343812  0.2288960  5.392760 2.622784e-07
HealthLifeExpectancy      1.4623304  0.3429916  4.263458 3.532468e-05
Freedom                   1.5588161  0.3733455  4.175264 5.011385e-05
TrustGovernmentCorruption 0.9589925  0.4545551  2.109739 3.652997e-02
```

```
##p-value
df(sum1$fstatistic[1],sum1$fstatistic[2],sum1$fstatistic[3])
```

```
       value
```

```
3.389338e-49
```

```r
##Confidence interval of the coefficient
confint(model1,level = 0.95)
```

```
                              2.5 %   97.5 %
(Intercept)               1.91520015 2.508541
EconomyGDPperCapita       0.28331217 1.110928
Family                    0.78212871 1.686634
HealthLifeExpectancy      0.78464798 2.140013
Freedom                   0.82116048 2.296472
TrustGovernmentCorruption 0.06088303 1.857102
```

```r
confint(model1,level = 0.99)
```

```
                              0.5 %   99.5 %
(Intercept)                1.8201566 2.603585
EconomyGDPperCapita        0.1507417 1.243498
Family                     0.6372418 1.831521
HealthLifeExpectancy       0.5675407 2.357120
Freedom                    0.5848398 2.532792
TrustGovernmentCorruption -0.2268418 2.144827
```

```r
confint(model1,level = 0.90)
```

```
                               5 %     95 %
(Intercept)               1.9633677 2.460374
EconomyGDPperCapita       0.3504982 1.043742
Family                    0.8555566 1.613206
HealthLifeExpectancy      0.8946768 2.029984
Freedom                   0.9409265 2.176706
TrustGovernmentCorruption 0.2067005 1.711284
```

19.5 Regression analysis

19.5.1 Residual analysis

Now that the regression has been done, the analysis and validity of the result can be analysed. Let's begin with residuals and the assumption of normality and homoscedasticity.

```r
# Visualisation of residuals
ggplot(model1, aes(model1$residuals)) +
    geom_histogram(bins=20, aes(y = ..density..)) +
    geom_density(color='blue', fill = 'blue', alpha = 0.2) +
    geom_vline(xintercept = mean(model1$residuals), color='red') +
    stat_function(fun=dnorm, color="red", size=1,
                  args = list(mean = mean(model1$residuals),
                              sd = sd(model1$residuals))) +
    xlab('residuals values')
```

The residual versus fitted plot is used to see if the residuals behave the same for the different value of the output (i.e, they have the same variance and mean). The plot shows no strong evidence of heteroscedasticity.

```
ggplot(model1, aes(model1$fitted.values, model1$residuals)) +
    geom_point() +
    geom_hline(yintercept = c(1.96 * sd(model1$residuals),
                              - 1.96 * sd(model1$residuals)), color='red') +
    xlab('fitted value') +
    ylab('residuals values')
```

19.6 Analysis of colinearity

The colinearity can be assessed using VIF, the car package provides a function to compute it directly.

```
require('car')
vif(model1)
```

```
      EconomyGDPperCapita                    Family
                 4.065908                  2.029263
     HealthLifeExpectancy                   Freedom
                 3.369446                  1.606887
TrustGovernmentCorruption
                 1.387116
```

All the VIF are less than 5, and hence there is no sign of colinearity.

19.7 What drives happiness

Now let's compute standardised betas to see what really drives happiness.

```
##Standardized betas
std_betas = sum1$coefficients[-1,1] *
    data.table(model1$model)[, lapply(.SD, sd), .SDcols=2:6] /
    sd(model1$model$HappinessScore)
```

```
std_betas
```

```
   EconomyGDPperCapita    Family HealthLifeExpectancy   Freedom
1:           0.2519358 0.2883631            0.2937655 0.1986718
   TrustGovernmentCorruption
1:                0.09327055
```

Though the code above may seem complicated, it is just computing the standardised betas for all variables `std_beta=beta*sd(x)/sd(y)`.

The top three coefficients are **Health and Life expectancy**, **Family** and **GDP per Capita**. Though money does not make happiness it is among the top three factors of Happiness!

Now you know how to perform a linear regression with R!

Chapter 20

Logistic Regression. Diabetes

20.1 Introduction

Source: https://github.com/AntoineGuillot2/Logistic-Regression-R/blob/master/script.R Source: http://enhancedatascience.com/2017/04/26/r-basics-logistic-regression-with-r/ Data: https://www.kaggle.com/uciml/pima-indians-diabetes-database

The goal of logistic regression is to predict whether an outcome will be positive (aka 1) or negative (i.e: 0). Some real life example could be:

- Will Emmanuel Macron win the French Presidential election or will he lose?
- Does Mr.X has this illness or not?
- Will this visitor click on my link or not?

So, logistic regression can be used in a lot of binary classification cases and will often be run before more advanced methods. For this tutorial, we will use the diabetes detection dataset from Kaggle.

This dataset contains data from Pima Indians Women such as the number of pregnancies, the blood pressure, the skin thickness, … the goal of the tutorial is to be able to detect diabetes using only these measures.

20.2 Exploring the data

As usual, first, let's take a look at our data. You can download the data here then please put the file diabetes.csv in your working directory. With the summary function, we can easily summarise the different variables:

```
library(ggplot2)
library(data.table)

DiabetesData <- data.table(read.csv(file.path(data_raw_dir, 'diabetes.csv')))

# Quick data summary
summary(DiabetesData)
```

```
  Pregnancies        Glucose      BloodPressure    SkinThickness
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
 Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
 Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54
```

```
 3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
    Insulin           BMI        DiabetesPedigreeFunction      Age
 Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
 1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00
 Median : 30.5   Median :32.00   Median :0.3725           Median :29.00
 Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24
 3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
 Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
    Outcome
 Min.   :0.000
 1st Qu.:0.000
 Median :0.000
 Mean   :0.349
 3rd Qu.:1.000
 Max.   :1.000
```

The mean of the outcome is 0.35 which shows an imbalance between the classes. However, the imbalance should not be too strong to be a problem.

To understand the relationship between variables, a Scatter Plot Matrix will be used. To plot it, the GGally package was used.

```r
# Scatter plot matrix
library(GGally)
ggpairs(DiabetesData, lower = list(continuous='smooth'))
```

The correlations between explanatory variables do not seem too strong. Hence the model is not likely to suffer from multicollinearity. All explanatory variable are correlated with the Outcome. At first sight, glucose rate is the most important factor to detect the outcome.

20.3 Logistic regression with R

After variable exploration, a first model can be fitted using the glm function. With stargazer, it is easy to get nice output in ASCII or even Latex.

```
# first model: all features
glm1 = glm(Outcome~.,
           DiabetesData,
           family = binomial(link="logit"))
```

```r
summary(glm1)
```

```

Call:
glm(formula = Outcome ~ ., family = binomial(link = "logit"),
    data = DiabetesData)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5566  -0.7274  -0.4159   0.7267   2.9297

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)              -8.4046964  0.7166359 -11.728  < 2e-16 ***
Pregnancies               0.1231823  0.0320776   3.840 0.000123 ***
Glucose                   0.0351637  0.0037087   9.481  < 2e-16 ***
BloodPressure            -0.0132955  0.0052336  -2.540 0.011072 *
SkinThickness             0.0006190  0.0068994   0.090 0.928515
Insulin                  -0.0011917  0.0009012  -1.322 0.186065
BMI                       0.0897010  0.0150876   5.945 2.76e-09 ***
DiabetesPedigreeFunction  0.9451797  0.2991475   3.160 0.001580 **
Age                       0.0148690  0.0093348   1.593 0.111192
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 993.48  on 767  degrees of freedom
Residual deviance: 723.45  on 759  degrees of freedom
AIC: 741.45

Number of Fisher Scoring iterations: 5
```

```r
require(stargazer)
stargazer(glm1,type='text')
```

```
=====================================================
                            Dependent variable:
                        ----------------------------
                                  Outcome
-----------------------------------------------------
Pregnancies                      0.123***
                                 (0.032)

Glucose                          0.035***
                                 (0.004)

BloodPressure                    -0.013**
                                 (0.005)

SkinThickness                     0.001
                                 (0.007)
```

```
Insulin                           -0.001
                                  (0.001)

BMI                              0.090***
                                  (0.015)

DiabetesPedigreeFunction         0.945***
                                  (0.299)

Age                                0.015
                                  (0.009)

Constant                         -8.405***
                                  (0.717)

---------------------------------------------------
Observations                        768
Log Likelihood                   -361.723
Akaike Inf. Crit.                 741.445
===================================================
Note:                     *p<0.1; **p<0.05; ***p<0.01
```

The overall model is significant. As expected the glucose rate has the lowest p-value of all the variables. However, Age, Insulin and Skin Thickness are not good predictors of Diabetes.

20.4 A second model

Since some variables are not significant, removing them is a good way to improve model robustness. In the second model, SkinThickness, Insulin, and Age are removed.

```r
# second model: selected features
glm2 = glm(Outcome~.,
         data = DiabetesData[,c(1:3,6:7,9), with=F],
         family = binomial(link="logit"))

summary(glm2)
```

```
Call:
glm(formula = Outcome ~ ., family = binomial(link = "logit"),
    data = DiabetesData[, c(1:3, 6:7, 9), with = F])

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.7931  -0.7362  -0.4188   0.7251   2.9555

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)              -7.954952   0.675823 -11.771  < 2e-16 ***
Pregnancies               0.153492   0.027835   5.514  3.5e-08 ***
Glucose                   0.034658   0.003394  10.213  < 2e-16 ***
BloodPressure            -0.012007   0.005031  -2.387  0.01700 *
BMI                       0.084832   0.014125   6.006  1.9e-09 ***
DiabetesPedigreeFunction  0.910628   0.294027   3.097  0.00195 **
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 993.48  on 767  degrees of freedom
Residual deviance: 728.56  on 762  degrees of freedom
AIC: 740.56

Number of Fisher Scoring iterations: 5
```

20.5 Classification rate and confusion matrix

Now that we have our model, let's access its performance.

```r
# Correctly classified observations
mean((glm2$fitted.values>0.5)==DiabetesData$Outcome)
```

```
[1] 0.7747396
```

Around 77.4% of all observations are correctly classified. Due to class imbalance, we need to go further with a confusion matrix.

```r
###Confusion matrix count
RP=sum((glm2$fitted.values>=0.5)==DiabetesData$Outcome & DiabetesData$Outcome==1)
FP=sum((glm2$fitted.values>=0.5)!=DiabetesData$Outcome & DiabetesData$Outcome==0)
RN=sum((glm2$fitted.values>=0.5)==DiabetesData$Outcome & DiabetesData$Outcome==0)
FN=sum((glm2$fitted.values>=0.5)!=DiabetesData$Outcome & DiabetesData$Outcome==1)
confMat<-matrix(c(RP,FP,FN,RN),ncol = 2)
colnames(confMat)<-c("Pred Diabetes",'Pred no diabetes')
rownames(confMat)<-c("Real Diabetes",'Real no diabetes')
confMat
```

```
                 Pred Diabetes Pred no diabetes
Real Diabetes              154              114
Real no diabetes            59              441
```

The model is good to detect people who do not have diabetes. However, its performance on ill people is not great (only 154 out of 268 have been correctly classified).

You can also get the percentage of Real/False Positive/Negative:

```r
# Confusion matrix proportion
RPR=RP/sum(DiabetesData$Outcome==1)*100
FNR=FN/sum(DiabetesData$Outcome==1)*100
FPR=FP/sum(DiabetesData$Outcome==0)*100
RNR=RN/sum(DiabetesData$Outcome==0)*100
confMat<-matrix(c(RPR,FPR,FNR,RNR),ncol = 2)
colnames(confMat)<-c("Pred Diabetes",'Pred no diabetes')
rownames(confMat)<-c("Real Diabetes",'Real no diabetes')
confMat
```

```
                 Pred Diabetes Pred no diabetes
Real Diabetes         57.46269         42.53731
Real no diabetes      11.80000         88.20000
```

And here is the matrix, 57.5% of people with diabetes are correctly classified. A way to improve the false negative rate would lower the detection threshold. However, as a consequence, the false positive rate would increase.

20.6 Plots and decision boundaries

The two strongest predictors of the outcome are Glucose rate and BMI. High glucose rate and high BMI are strong indicators of Diabetes.

```
# Plot and decision boundaries
DiabetesData$Predicted <- glm2$fitted.values

ggplot(DiabetesData, aes(x = BMI, y = Glucose, color = Predicted > 0.5)) +
    geom_point(size=2, alpha=0.5)
```

```
ggplot(DiabetesData, aes(x=BMI, y = Glucose, color=Outcome == (Predicted > 0.5))) +
    geom_point(size=2, alpha=0.5)
```

We can also plot both BMI and glucose against the outcomes, the other variables are taken at their mean level.

```
range(DiabetesData$BMI)
```

```
[1]  0.0 67.1
```

```
# BMI vs predicted
BMI_plot = data.frame(BMI = ((min(DiabetesData$BMI-2)*100):
                              (max(DiabetesData$BMI+2)*100))/100,
                      Glucose = mean(DiabetesData$Glucose),
                      Pregnancies = mean(DiabetesData$Pregnancies),
                      BloodPressure = mean(DiabetesData$BloodPressure),
                      DiabetesPedigreeFunction = mean(DiabetesData$DiabetesPedigreeFunction))

BMI_plot$Predicted = predict(glm2, newdata = BMI_plot, type = 'response')
ggplot(BMI_plot, aes(x = BMI, y = Predicted)) +
    geom_line()
```

```
range(BMI_plot$BMI)
```

```
[1] -2.0 69.1
```

```
range(DiabetesData$Glucose)
```

```
[1]   0 199
```

```
# Glucose vs predicted
Glucose_plot=data.frame(Glucose =
                        ((min(DiabetesData$Glucose-2)*100):
                             (max(DiabetesData$Glucose+2)*100))/100,
                    BMI=mean(DiabetesData$BMI),
                    Pregnancies=mean(DiabetesData$Pregnancies),
                    BloodPressure=mean(DiabetesData$BloodPressure),
                    DiabetesPedigreeFunction=mean(DiabetesData$DiabetesPedigreeFunction))

Glucose_plot$Predicted = predict(glm2, newdata = Glucose_plot, type = 'response')
ggplot(Glucose_plot, aes(x = Glucose, y = Predicted)) +
    geom_line()
```

```
range(Glucose_plot$Glucose)
```

```
[1]  -2 201
```

Chapter 21

Linear Regression on Advertising

Videos, slides:

- https://www.r-bloggers.com/in-depth-introduction-to-machine-learning-in-15-hours-of-expert-videos/

Data:

- http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv

code:

- http://subasish.github.io/pages/ISLwithR/
- http://math480-s15-zarringhalam.wikispaces.umb.edu/R+Code
- https://github.com/yahwes/ISLR
- https://www.tau.ac.il/~saharon/IntroStatLearn.html
- https://www.waxworksmath.com/Authors/G_M/James/WWW/chapter_3.html
- https://github.com/asadoughi/stat-learning

plots:

- https://onlinecourses.science.psu.edu/stat857/node/28/

```r
library(readr)

advertising <- read_csv(file.path(data_raw_dir, "Advertising.csv"))
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   TV = col_double(),
##   radio = col_double(),
##   newspaper = col_double(),
##   sales = col_double()
## )
```

```r
advertising
```

```
## # A tibble: 200 x 5
##       X1    TV radio newspaper sales
##    <dbl> <dbl> <dbl>     <dbl> <dbl>
##  1     1 230.   37.8      69.2  22.1
##  2     2  44.5  39.3      45.1  10.4
```

```
## 3      3  17.2  45.9      69.3   9.3
## 4      4 152.   41.3      58.5  18.5
## 5      5 181.   10.8      58.4  12.9
## 6      6   8.7  48.9      75     7.2
## 7      7  57.5  32.8      23.5  11.8
## 8      8 120.   19.6      11.6  13.2
## 9      9   8.6   2.1       1     4.8
## 10    10 200.    2.6      21.2  10.6
## # ... with 190 more rows
```

The Advertising data set. The plot displays sales, in thousands of units, as a function of TV, radio, and newspaper budgets, in thousands of dollars, for 200 different markets.

```
par(mfrow=c(1,3))
plot(advertising$TV, advertising$sales, xlab = "TV", ylab = "Sales", col = "red")
plot(advertising$radio, advertising$sales, xlab="Radio", ylab="Sales", col="red")
plot(advertising$radio, advertising$newspaper, xlab="Newspaper",
     ylab="Sales", col="red")
```

In each plot we show the simple least squares fit of sales to that variable, as described in Chapter 3. In other words, each blue line represents a simple model that can be used to predict sales using TV, radio, and newspaper, respectively.

```
par(mfrow=c(1,3))
tv_model <- lm(sales ~ TV, data = advertising)
radio_model <- lm(sales ~ radio, data = advertising)
newspaper_model <- lm(sales ~ newspaper, data = advertising)

plot(advertising$TV, advertising$sales, xlab = "TV", ylab = "Sales", col = "red")
abline(tv_model, col = "blue")
plot(advertising$radio, advertising$sales, xlab="Radio", ylab="Sales", col="red")
abline(radio_model)
```

```
plot(advertising$newspaper, advertising$sales, xlab="Newspaper",
     ylab="Sales", col="red")
abline(newspaper_model)
```

Recall the Advertising data from Chapter 2. Figure 2.1 displays sales (in thousands of units) for a particular product as a function of advertis- ing budgets (in thousands of dollars) for TV, radio, and newspaper media. Suppose that in our role as statistical consultants we are asked to suggest, on the basis of this data, a marketing plan for next year that will result in high product sales. What information would be useful in order to provide such a recommendation? Here are a few important questions that we might seek to address:

1. Is there a relationship between advertising budget and sales?

2. How strong is the relationship between advertising budget and sales?

3. Which media contribute to sales?

4. How accurately can we estimate the effect of each medium on sales?

For the Advertising data, the least squares fit for the regression of sales onto TV is shown. The fit is found by minimizing the sum of squared errors. Each grey line segment represents an error, and the fit makes a compro- mise by averaging their squares. In this case a linear fit captures the essence of the relationship, although it is somewhat deficient in the left of the plot.

```
tv_model <- lm(sales ~ TV, data = advertising)
plot(advertising$TV, advertising$sales, xlab = "TV", ylab = "Sales",
     col = "red", pch=16)
abline(tv_model, col = "blue", lwd=2)
segments(advertising$TV, advertising$sales, advertising$TV, predict(tv_model),
         col = "gray")
```

```r
smry <- summary(tv_model)
smry
```

```
##
## Call:
## lm(formula = sales ~ TV, data = advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.3860 -1.9545 -0.1913  2.0671  7.2124
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 7.032594   0.457843   15.36   <2e-16 ***
## TV          0.047537   0.002691   17.67   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.259 on 198 degrees of freedom
## Multiple R-squared:  0.6119, Adjusted R-squared:  0.6099
## F-statistic: 312.1 on 1 and 198 DF,  p-value: < 2.2e-16
```

```r
library(lattice)

minRss <- sqrt(abs(min(smry$residuals))) * sign(min(smry$residuals))
maxRss <- sqrt(max(smry$residuals))

twovar <- function(x, y) {
  x^2 + y^2 }

mat <- outer( seq(minRss, maxRss, length = 100),
                seq(minRss, maxRss, length = 100),
                Vectorize( function(x,y) twovar(x, y) ) )
```

```r
contourplot(mat, at = c(1,2,3))
```

```r
tv_model
```

```
##
## Call:
## lm(formula = sales ~ TV, data = advertising)
##
## Coefficients:
## (Intercept)           TV
##     7.03259      0.04754
```

```r
tv.lm <- lm(sales ~ poly(sales, TV, degree=2), data = advertising)
# contour(tv.lm, sales ~ TV)
```

```r
library(rsm)
mpg.lm <- lm(mpg ~ poly(hp, disp, degree = 3), data = mtcars)
contour(mpg.lm, hp ~ disp)
```

```
x <- -6:16
op <- par(mfrow = c(2, 2))
contour(outer(x, x), method = "flattest", vfont = c("sans serif", "plain"))
```

Chapter 22

Data Visualization. 6 Work with models

https://socviz.co/modeling.html

Data visualization is about more than generating figures that display the raw numbers from a table of data. Right from the beginning, it involves summarizing or transforming parts of the data, and then plotting the results. Statistical models are a central part of that process. In this Chapter, we will begin by looking briefly at how `ggplot` can use various modeling techniques directly within geoms. Then we will see how to use the `broom` and `margins` libraries to tidily extract and plot estimates from models that we fit ourselves.

```
library(ggplot2)
```

```
#> Error in library(ggplot2): there is no package called 'ggplot2'
```

```
library(dplyr)
```

```
#> Error in library(dplyr): there is no package called 'dplyr'
```

```
library(tidyr)
```

```
#> Error in library(tidyr): there is no package called 'tidyr'
```

```
library(purrr)
library(socviz)       # devtools::install_github("kjhealy/socviz")
```

```
#> Error in library(socviz): there is no package called 'socviz'
```

```
library(gapminder)
```

```
#> Error in library(gapminder): there is no package called 'gapminder'
```

```
p <- ggplot(data = gapminder,
            mapping = aes(x = log(gdpPercap), y = lifeExp))
```

```
#> Error in ggplot(data = gapminder, mapping = aes(x = log(gdpPercap), y = lifeExp)): could not find fu
```

```
p + geom_point(alpha=0.1) +
    geom_smooth(color = "tomato", fill="tomato", method = MASS::rlm) +
    geom_smooth(color = "steelblue", fill="steelblue", method = "lm")
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

```
p + geom_point(alpha=0.1) +
    geom_smooth(color = "tomato", method = "lm", size = 1.2,
                formula = y ~ splines::bs(x, 3), se = FALSE)
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

```
p + geom_point(alpha=0.1) +
    geom_quantile(color = "tomato", size = 1.2, method = "rqss",
                  lambda = 1, quantiles = c(0.20, 0.5, 0.85))
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

Histograms, density plots, boxplots, and other geoms compute either single numbers or new variables before plotting them. As we saw in Section 4.4, these calculations are done by `stat_` functions, each of which works hand-in-hand with its default `geom_` function, and vice versa. Moreover, from the smoothing lines we drew from almost the very first plots we made, we have seen that `stat_` functions can do a fair amount of calculation and even model estimation on the fly. The `geom_smooth()` function can take a range of method arguments to fit LOESS, OLS, and robust regression lines, amongst others.

Both the `geom_smooth()` and `geom_quantile()` functions can also be instructed to use different formulas to produce their fits. In the top panel of Figure 6.1, we access the `MASS` library's `rlm` function to fit a robust regression line. In the second panel, the `bs` function is invoked directly from the `splines` library in the same way, to fit a polynominal curve to the data. This is the same approach to directly accessing functions without loading a whole library that we have already used several times when using functions from the `scales` library. The `geom_quantile()` function, meanwhile, is like a specialized version of `geom_smooth()` that can fit quantile regression lines using a variety of methods. The `quantiles` argument takes a vector specifying the quantiles at which to fit the lines.

22.1 6.1 Show several fits at once, with a legend

As we just saw in the first panel of Figure 6.1, where we plotted both an OLS and a robust regression line, we can look at several fits at once on the same plot by layering on new smoothers with `geom_smooth()`. As long as we set the color and fill aesthetics to different values for each fit, we can easily distinguish them visually. However, `ggplot` will not draw a legend that guides us about which fit is which. This is because the smoothers are not logically connected to one another. They exist as separate layers. What if we are comparing several different fits and want a legend describing them?

As it turns out, `geom_smooth()` can do this via the slightly unusual route of mapping the color and fill aesthetics to a string describing the model we are fitting, and then using `scale_color_manual()` and `scale_fill_manual()` to create the legend. First we use `brewer.pal()` from the `RColorBrewer` library to extract three qualitatively different colors from a larger palette. The colors are represented as hex values. As before use the `::` convention to use the function without loading the whole library:

```
model_colors <- RColorBrewer::brewer.pal(3, "Set1")
```

```
#> Error in loadNamespace(name): there is no package called 'RColorBrewer'
```

```
model_colors
```

```
#> Error in eval(expr, envir, enclos): object 'model_colors' not found
```

Then we create a plot with three different smoothers, mapping the color and fill within the `aes()` function as the name of the smoother:

```
p0 <- ggplot(data = gapminder,
             mapping = aes(x = log(gdpPercap), y = lifeExp))
```

```
#> Error in ggplot(data = gapminder, mapping = aes(x = log(gdpPercap), y = lifeExp)): could not find fu
```

```
p1 <- p0 + geom_point(alpha = 0.2) +
    geom_smooth(method = "lm", aes(color = "OLS", fill = "OLS")) +
    geom_smooth(method = "lm", formula = y ~ splines::bs(x, df = 3),
                aes(color = "Cubic Spline", fill = "Cubic Spline")) +
    geom_smooth(method = "loess",
                aes(color = "LOESS", fill = "LOESS"))
```

```
#> Error in eval(expr, envir, enclos): object 'p0' not found
```

```
p1 + scale_color_manual(name = "Models", values = model_colors) +
    scale_fill_manual(name = "Models", values = model_colors) +
    theme(legend.position = "top")
```

```
#> Error in eval(expr, envir, enclos): object 'p1' not found
```

In a way we have cheated a little here to make the plot work. Until now, we have always mapped aesthetics to the names of variables, not to strings like "OLS" or "Cubic Splines". In Chapter 3, when we discussed mapping versus setting aesthetics, we saw what happened when we tried to change the color of the points in a scatterplot by setting them to "purple" inside the `aes()`function. The result was that the points turned red instead, as `ggplot` in effect created a new variable and labeled it with the word "purple". We learned there that the `aes()` function was for mapping variables to aesthetics.

Here we take advantage of that behavior, creating a new single-value variable for the name of each of our models. Ggplot will properly construct the relevant guide if we call `scale_color_manual()` and `scale_fill_manual()`. Remember that we have to call two scale functions because we have two mappings. The result is a single plot containing not just our three smoothers, but also an appropriate legend to guide the reader.

These model-fitting features make `ggplot` very useful for exploratory work, and make it straightforward to generate and compare model-based trends and other summaries as part of the process of descriptive data visualization. The various `stat_` functions are a flexible way to add summary estimates of various kinds to plots. But we will also want more than this, including presenting results from models we fit ourselves.

22.2 6.2 Look inside model objects

Covering the details of fitting statistical models in R is beyond the scope of this book. For a comprehensive, modern introduction to that topic you should work your way through (Gelman & Hill, 2018). (Harrell, 2016) is also very good on the many practical connections between modeling and graphing data. Similarly, (Gelman, 2004) provides a detailed discussion of the use of graphics as a tool in model-checking and validation. Here we will discuss some ways to take the models that you fit and extract information that is easy to work with in ggplot. Our goal, as always, is to get from however the object is stored to a tidy table of numbers that we can plot. Most classes of statistical model in R will contain the information we need, or will have a special set of functions, or methods, designed to extract it.

We can start by learning a little more about how the output of models is stored in R. Remember, we are always working with objects, and objects have an internal structure consisting of named pieces. Sometimes these are single numbers, sometimes vectors, and sometimes lists of things like vectors, matrices, or formulas.

We have been working extensively with tibbles and data frames. These store tables of data with named columns, perhaps consisting of different classes of variable, such as integers, characters, dates, or factors.

Model objects are a little more complicated again.

```
gapminder
```

```
#> Error in eval(expr, envir, enclos): object 'gapminder' not found
```

Remember, we can use the `str()` function to learn more about the internal structure of any object. For example, we can get some information on what class (or classes) of object `gapminder` is, how large it is, and what components it has. The output from `str(gapminder)` is somewhat dense:

```
str(gapminder)
```

```
#> Error in str(gapminder): object 'gapminder' not found
```

There is a lot of information here about the object as a whole and each variable in it. In the same way, statistical models in R have an internal structure. But because models are more complex entities than data tables, their structure is correspondingly more complicated. There are more pieces of information, and more kinds of information, that we might want to use. All of this information is generally stored in or is computable from parts of a model object.

We can create a linear model, an ordinary OLS regression, using the `gapminder` data. This dataset has a country-year structure that makes an OLS specification like this the wrong one to use. But never mind that for now. We use the `lm()` function to run the model, and store it in an object called `out`:

```
out <- lm(formula = lifeExp ~ gdpPercap + pop + continent,
          data = gapminder)
```

```
#> Error in is.data.frame(data): object 'gapminder' not found
```

The first argument is the formula for the `model. lifeExp` is the dependent variable and the tilde `~` operator is used to designate the left- and right-hand sides of a model (including in cases, as we saw with `facet_wrap()` where the model just has a right-hand side.)

Let's look at the results by asking R to print a summary of the model.

```
summary(out)
```

```
#> Error in summary(out): object 'out' not found
```

When we use the `summary()` function on out, we are not getting a simple feed of what's in the model object. Instead, like any function, `summary()` takes its input, performs some actions, and produces output. In this case, what is printed to the console is partly information that is stored inside the model object, and partly information that the `summary()` function has calculated and formated for display on the screen. Behind the scenes, `summary()` gets help from other functions. Objects of different classes have default methods associated with them, so that when the generic `summary()` function is applied to a linear model object, the function knows to pass the work on to a more specialized function that does a bunch of calculations and formatting appropriate to a linear model object. We use the same generic `summary()` function on data frames, as in `summary(gapminder)`, but in that case a different default method is applied.

Schematic view of a linear model object.Figure 6.3: Schematic view of a linear model object.

The output from `summary()` gives a precis of the model, but we can't really do any further analysis with it directly. For example, what if we want to plot something from the model? The information necessary to make plots is inside the out object, but it is not obvious how to use it.

If we take a look at the structure of the model object with `str(out)` we will find that there is a lot of information in there. Like most complex objects in R, out is organized as a list of components or elements. Several of these elements are themselves lists. Figure 6.3 gives you a schematic view of the contents of a

linear model object. In this list of items, elements are single values, some are data frames, and some are additional lists of simpler items. Again, remember our earlier discussion where we said objects could be thought of as being organized like a filing system: cabinets contain drawers, and drawer may contain which may contain pages of information, whole documents, or groups of folders with more documents inside. As an alternative analogy, and sticking with the image of a list, you can think of a master to-do list for a project, where the top-level headings lead to contain additional lists of tasks of different kinds.

The `out` object created by `lm` contains several different named elements. Some, like the residual degrees of freedom in the model, are just a single number. Try `out$df.residual` at the console. Others are much larger entities, such as the data frame used to fit the model, which is retained by default. Try `out$model`, but be prepared for a lot of stuff to be printed at the console. Other elements have been computed by R and then stored, such as the coefficients of the model and other quantities. You can try `out$coefficients`, `out$residuals`, and `out$fitted.values`, for instance. Others are lists themselves (like `qr`). So you can see that the `summary()` function is selecting and printing only a small amount of core information, in comparison to what is stored in the model object.

Just like the tables of data we saw earlier in Section A.1.3, the output of `summary()` is presented in a way that is compact and efficient in terms of getting information across, but also untidy when considered from the point of view of further manipulation. There is a table of coefficients, but the variable names are in the rows. The column names are awkward, and some information (e.g. at the bottom of the output) has been calculated and printed out, but is not stored in the model object.

22.3 6.3 Get model-based graphics right

Figures based on statistical models face all the ordinary challenges of effective data visualization, and then some. This is because model results usually carry a considerable extra burden of interpretation and necessary background knowledge. The more complex the model, the trickier it becomes to convey this information effectively, and the easier it becomes to lead one's audience or oneself into error. Within the social sciences, our ability to clearly and honestly present model-based graphics has greatly improved over the past ten or fifteen years. Over the same period, it has become clearer that some kinds of models are quite tricky to understand, even ones that had previously been seen as straightforward elements of the modeling toolkit (Ai & Norton, 2003; Brambor, Clark, & Golder, 2006).

Plotting model estimates is closely connected to properly estimating models in the first place. This means there is no substitute for learning the statistics. You should not use graphical methods as a substitute for understanding the model used to produce them. While this book cannot teach you that material, we can make a few general points about what good model-based graphics look like, and work through some examples of how `ggplot` and some additional libraries can make it easier to get good results.

22.3.1 6.3.1 Present your findings in substantive terms

Useful model-based plots show results in ways that are substantively meaningful and directly interpretable with respect to the questions the analysis is trying to answer. This means showing results in a context where other variables in the analysis are held at sensible values, such as their means or medians. With continuous variables, it can often be useful to generate predicted values that cover some substantively meaningful move across the distribution, such as from the 25th to the 75th percentile, rather than a single-unit increment in the variable of interest. For unordered categorical variables, predicted values might be presented with respect to the modal category in the data, or for a particular category of theoretical interest. Presenting substantively interpretable findings often also means using (and sometimes converting to) a scale that readers can easily understand. If your model reports results in log-odds, for example, converting the estimates to predicted probabilities will make it easier to interpret. All of this advice is quite general. Each of these points applies equally well to the presentation of summary results in a table rather than a graph. There is nothing distinctively graphical about putting the focus on the substantive meaning of your findings.

22.3.2 6.3.2 Show your degree of confidence

Much the same applies to presenting the degree of uncertainty or confidence you have in your results. Model estimates come with various measures of precision, confidence, credence, or significance. Presenting and interpreting these measures is notoriously prone to misinterpretation, or over-interpretation, as researchers and audiences both demand more from things like confidence intervals and p-values than these statistics can deliver. At a minimum, having decided on an appropriate measure of model fit or the right assessment of confidence, you should show their range when you present your results. A family of related `ggplot` geoms allow you to show a range or interval defined by position on the x-axis and then a `ymin` and `ymax` range on the y-axis. These geoms include `geom_pointrange()` and `geom_errorbar()`, which we will see in action shortly. A related geom, `geom_ribbon()` uses the same arguments to draw filled areas, and is useful for plotting ranges of y-axis values along some continuously varying x-axis.

22.3.3 6.3.3 Show your data when you can

Plotting the results from a multivariate model generally means one of two things. First, we can show what is in effect a table of coefficients with associated measures of confidence, perhaps organizing the coefficients into meaningful groups, or by the size of the predicted association, or both. Second, we can show the predicted values of some variables (rather than just a model's coefficients) across some range of interest. The latter approach lets us show the original data points if we wish. The way `ggplot` builds graphics layer by layer allows us to easily combine model estimates (e.g. a regression line and an associated range) and the underlying data. In effect these are manually-constructed versions of the automatically-generated plots that we have been producing with `geom_smooth()` since the beginning of this book.

22.4 6.4 Generate predictions to graph

Having fitted a model, then, we might want to get a picture of the estimates it produces over the range of some particular variable, holding other covariates constant at some sensible values. The `predict()` function is a generic way of using model objects to produce this kind of prediction. In R, "generic" functions take their inputs and pass them along to more specific functions behind the scenes, ones that are suited to working with the particular kind of model object we have. The details of getting predicted values from a OLS model, for instance, will be somewhat different from getting predictions out of a logistic regression. But in each case we can use the same `predict()` function, taking care to check the documentation to see what form the results are returned in for the kind of model we are working with. Many of the most commonly-used functions in R are generic in this way. The `summary()` function, for example, works on objects of many different classes, from vectors to data frames and statistical models, producing appropriate output in each case by way of a class-specific function in the background.

For `predict()` to calculate the new values for us, it needs some new data to fit the model to. We will generate a new data frame whose columns have the same names as the variables in the model's original data, but where the rows have new values. A very useful function called `expand.grid()` will help us do this. We will give it a list of variables, specifying the range of values we want each variable to take. Then `expand.grid()` will generate the will multiply out the full range of values for all combinations of the values we give it, thus creating a new data frame with the new data we need.

In the following bit of code, we use `min()` and `max()` to get the minimum and maximum values for per capita GDP, and then create a vector with one hundred evenly-spaced elements between the minimum and the maximum. We hold population constant at its median, and we let continent take all of its five available values.

```
min_gdp <- min(gapminder$gdpPercap)
```

```
#> Error in eval(expr, envir, enclos): object 'gapminder' not found
```

```
max_gdp <- max(gapminder$gdpPercap)
```

```
#> Error in eval(expr, envir, enclos): object 'gapminder' not found
```

```
med_pop <- median(gapminder$pop)
```

```
#> Error in median(gapminder$pop): object 'gapminder' not found
```

```
pred_df <- expand.grid(gdpPercap = (seq(from = min_gdp,
                                        to = max_gdp,
                                        length.out = 100)),
                       pop = med_pop,
                       continent = c("Africa", "Americas",
                                     "Asia", "Europe", "Oceania"))
```

```
#> Error in seq(from = min_gdp, to = max_gdp, length.out = 100): object 'min_gdp' not found
```

```
dim(pred_df)
```

```
#> Error in eval(expr, envir, enclos): object 'pred_df' not found
```

```
head(pred_df)
```

```
#> Error in head(pred_df): object 'pred_df' not found
```

Now we can use `predict()`. If we give the function our new data and model, without any further argument, it will calculate the fitted values for every row in the data frame. If we specify `interval = 'predict'` as an argument, it will calculate 95% prediction intervals in addition to the point estimate.

```
pred_out <- predict(object = out,
                    newdata = pred_df,
                    interval = "predict")
```

```
#> Error in predict(object = out, newdata = pred_df, interval = "predict"): object 'out' not found
```

```
head(pred_out)
```

```
#> Error in head(pred_out): object 'pred_out' not found
```

Because we know that, by construction, the cases in `pred_df` and `pred_out` correspond row for row, we can bind the two data frames together by column. This method of joining or merging tables is definitely not recommended when you are dealing with data.

```
pred_df <- cbind(pred_df, pred_out)
```

```
#> Error in cbind(pred_df, pred_out): object 'pred_df' not found
```

```
head(pred_df)
```

```
#> Error in head(pred_df): object 'pred_df' not found
```

The end result is a tidy data frame, containing the predicted values from the model for the range of values we specified. Now we can plot the results. Because we produced a full range of predicted values, we can decide whether or not to use all of them. Here we further subset the predictions to just those for Europe and Africa.

```
p <- ggplot(data = subset(pred_df, continent %in% c("Europe", "Africa")),
            aes(x = gdpPercap,
                y = fit, ymin = lwr, ymax = upr,
                color = continent,
```

```
                 fill = continent,
                 group = continent))
```

```
#> Error in ggplot(data = subset(pred_df, continent %in% c("Europe", "Africa")), : could not find functi
```

```
p + geom_point(data = subset(gapminder,
                             continent %in% c("Europe", "Africa")),
               aes(x = gdpPercap, y = lifeExp,
                   color = continent),
               alpha = 0.5,
               inherit.aes = FALSE) +
    geom_line() +
    geom_ribbon(alpha = 0.2, color = FALSE) +
    scale_x_log10(labels = scales::dollar)
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

We use a new geom here to draw the area covered by the prediction intervals: `geom_ribbon()`. It takes an x argument like a line, but a ymin and ymax argument as specified in the `ggplot()` aesthetic mapping. This defines the lower and upper limits of the prediction interval.

In practice, you may not use `predict()` directly all that often. Instead, you might write code using additional libraries that encapsulate the process of producing predictions and plots from models. These are especially useful when your model is a little more complex and the interpretation of coefficients becomes trickier. This happens, for instance, when you have a binary outcome variable and need to convert the results of a logistic regression into predicted probabilities, or when you have interaction terms amongst your predictions. We will discuss some of these helper libraries in the next few sections. However, bear in mind that `predict()` and its ability to work safely with different classes of model underpins many of those libraries. So it's useful to see it in action first hand in order to understand what it is doing.

22.5 6.5 Tidy model objects with `broom`

The `predict` method is very useful, but there are a lot of other things we might want to do with our model output. We will use David Robinson's `broom` package to help us out. It is a library of functions that help us get from the model results that R generates to numbers that we can plot. It will take model objects and turn pieces of them into data frames that you can use easily with `ggplot`.

```
library(broom)
```

```
#> Error in library(broom): there is no package called 'broom'
```

Broom takes ggplot's approach to tidy data and extends it to the model objects that R produces. Its methods can tidily extract three kinds of information. First, we can see component-level information about aspects of the model itself, such as coefficients and t-statistics. Second, we can obtain observation-level information about the model's connection to the underlying data. This includes the fitted values and residuals for each observation in the data. And finally we can get model-level information that summarizes the fit as a whole, such as an F-statistic, the model deviance, or the r-squared. There is a `broom` function for each of these tasks.

22.5.1 6.5.1 Get component-level statistics with `tidy()`

The `tidy()` function takes a model object and returns a data frame of component-level information. We can work with this to make plots in a familiar way, and much more easily than fishing inside the model object to extract the various terms. Here is an example, using the default results as just returned. For a

more convenient display of the results, we will pipe the object we create with `tidy()` through a function that rounds the numeric columns of the data frame to two decimal places. This doesn't change anything about the object itself, of course.

```
out_comp <- tidy(out)
```

```
#> Error in tidy(out): could not find function "tidy"
```

```
out_comp %>% round_df()
```

```
#> Error in eval(lhs, parent, parent): object 'out_comp' not found
```

We are now able to treat this data frame just like all the other data that we have seen so far.

```
p <- ggplot(out_comp, mapping = aes(x = term,
                                    y = estimate))
```

```
#> Error in ggplot(out_comp, mapping = aes(x = term, y = estimate)): could not find function "ggplot"
```

```
p + geom_point() + coord_flip()
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

We can extend and clean up this plot in a variety of ways. For example, we can tell `tidy()` to calculate confidence intervals for the estimates, using R's `confint()` function.

```
out_conf <- tidy(out, conf.int = TRUE)
```

```
#> Error in tidy(out, conf.int = TRUE): could not find function "tidy"
```

```
out_conf %>% round_df()
```

```
#> Error in eval(lhs, parent, parent): object 'out_conf' not found
```

The convenience "not in" operator `%nin%` is available via the `socviz` library. It does the opposite of `%in%` and selects only the items in a first vector of characters that are not in the second. We'll use it to drop the intercept term from the table. We also want to something about the labels. When fitting a model with categorical variables, R will create coefficient names based on the variable name and the category name, like continentAmericas. Normally we like to clean these up before plotting. Most commonly, we just want to strip away the variable name at the beginning of the coefficient label. For this we can use `prefix_strip()`, a convenience function in the `socviz` library. We tell it which prefixes to drop, using it to create a new column variable in `out_conf` that corresponds to the terms column, but that has nicer labels.

```
out_conf <- subset(out_conf, term %nin% "(Intercept)")
```

```
#> Error in subset(out_conf, term %nin% "(Intercept)"): object 'out_conf' not found
```

```
out_conf$nicelabs <- prefix_strip(out_conf$term, "continent")
```

```
#> Error in prefix_strip(out_conf$term, "continent"): could not find function "prefix_strip"
```

Now we can use `geom_pointrange()`to make a figure that displays some information about our confidence in the variable estimates, as opposed to just the coefficients. As with the boxplots earlier, we use `reorder()` to sort the names of the model's terms by the estimate variable, thus arranging our plot of effects from largest to smallest in magnitude.

```
p <- ggplot(out_conf, mapping = aes(x = reorder(nicelabs, estimate),
                                    y = estimate, ymin = conf.low, ymax = conf.high))
```

```
#> Error in ggplot(out_conf, mapping = aes(x = reorder(nicelabs, estimate), : could not find function "
```

```
p + geom_pointrange() + coord_flip() + labs(x="", y="OLS Estimate")
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

Dotplots of this kind can be very compact. The vertical axis can often be compressed quite a bit, with no loss in comprehension. In fact, they are often easier to read with much less room between the rows than given by a default square shape.

22.5.2 6.5.2 Get observation-level statistics with `augment()`

The values returned by `augment()` are all statistics calculated at the level of the original observations. As such, they can be added on to the data frame that the model is based on. Working from a call to `augment()` will return a data frame with all the original observations used in the estimation of the model, together with columns like the following:

- **.fitted** — The fitted values of the model.
- **.se.fit** — The standard errors of the fitted values.
- **.resid** — The residuals.
- **.hat** — The diagonal of the hat matrix.
- **.sigma** — An estimate of residual standard deviation when the corresponding observation is dropped from the model.
- **.cooksd** — Cook's distance, a common regression diagnostic; and
- **.std.resid** — The standardized residuals.

Each of these variables is named with a leading `dot`, for example `.hat` rather than hat, and so on. This is to guard against accidentally confusing it with (or accidentally overwriting) an existing variable in your data with this name. The columns of values return will differ slightly depending on the class of model being fitted.

```
out_aug <- augment(out)
```

```
#> Error in augment(out): could not find function "augment"
```

```
head(out_aug) %>% round_df()
```

```
#> Error in head(out_aug): object 'out_aug' not found
```

By default, `augment()` will extract the available data from the model object. This will usually include the variables used in the model itself, but not any additional ones contained in the original data frame. Sometimes it is useful to have these. We can add them by specifying the data argument:

```
out_aug <- augment(out, data = gapminder)
```

```
#> Error in augment(out, data = gapminder): could not find function "augment"
```

```
head(out_aug) %>% round_df()
```

```
#> Error in head(out_aug): object 'out_aug' not found
```

If some rows containing missing data were dropped to fit the model, then these will not be carried over to the augmented data frame.

The new columns created by `augment()` can be used to create some standard regression plots. For example, we can plot the residuals versus the fitted values. Figure 6.7 suggests, unsurprisingly, that our country-year data has rather more structure than is captured by our OLS model.

```
p <- ggplot(data = out_aug,
            mapping = aes(x = .fitted, y = .resid))
```

```
#> Error in ggplot(data = out_aug, mapping = aes(x = .fitted, y = .resid)): could not find function "gg
```

```
p + geom_point()
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

22.5.3 6.5.3 Get model-level statistics with `glance()`

This function organizes the information typically presented at the bottom of a model's `summary()` output. By itself, it usually just returns a table with a single row in it. But as we shall see in a moment, the real power of broom's approach is the way that it can scale up to cases where we are grouping or subsampling our data.

```
glance(out) %>% round_df()
```

```
#> Error in glance(out): could not find function "glance"
```

Broom is able to `tidy` (and `augment`, and `glance` at) a wide range of model types. Not all functions are available for all classes of model. Consult broom's documentation for more details on what is available. For example, here is a plot created from the tidied output of an event-history analysis. First we generate a Cox proportional hazards model of some `survival` data.

```
library(survival)

out_cph <- coxph(Surv(time, status) ~ age + sex, data = lung)
out_surv <- survfit(out_cph)
```

The details of the fit are not important here, but in the first step the `Surv()` function creates the response or outcome variable for the proportional hazards model that is then fitted by the `coxph()`function. Then the `survfit()` function creates the survival curve from the model, much like we used `predict()` to generate predicted values earlier. Try `summary(out_cph)` to see the model, and `summary(out_surv)` to see the table of predicted values that will form the basis for our plot. Next we tidy `out_surv` to get a data frame, and plot it.

```
# Figure 6.8: A Kaplan-Meier plot.
out_tidy <- tidy(out_surv)
```

```
#> Error in tidy(out_surv): could not find function "tidy"
```

```
p <- ggplot(data = out_tidy, mapping = aes(time, estimate))
```

```
#> Error in ggplot(data = out_tidy, mapping = aes(time, estimate)): could not find function "ggplot"
```

```
p + geom_line() +
    geom_ribbon(mapping = aes(ymin = conf.low, ymax = conf.high), alpha = .2)
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

22.6 6.6 Grouped analysis and list-columns

Broom makes it possible to quickly fit models to different subsets of your data and get consistent and usable tables of results out the other end. For example, let's say we wanted to look at the gapminder data by examining the relationship between life expectancy and GDP by continent, for each year in the data.

The `gapminder` data is at bottom organized by country-years. That is the unit of observation in the rows. If we wanted, we could take a slice of the data manually, such as "all countries observed in Asia, in 1962" or "all in Africa, 2002". Here is "Europe, 1977":

```
eu77 <- gapminder %>% filter(continent == "Europe", year == 1977)
```

```
#> Error in eval(lhs, parent, parent): object 'gapminder' not found
```

We could then see what the relationship between life expectancy and GDP looked like for that continent-year group:

```
fit <- lm(lifeExp ~ log(gdpPercap), data = eu77)
```

```
#> Error in is.data.frame(data): object 'eu77' not found
```

```
summary(fit)
```

```
#> Error in summary(fit): object 'fit' not found
```

With `dplyr` and `broom` we can do this for every continent-year slice of the data in a compact and tidy way. We start with our table of data, and then (`%>%`) group the countries by continent and year using the `group_by()` function. We introduced this grouping operation in Chapter 4. Our data is reorganized first by continent, and within continent by year. Here we will take one further step and nest the data that make up each group:

```
out_le <- gapminder %>%
    group_by(continent, year) %>%
    nest()
```

```
#> Error in eval(lhs, parent, parent): object 'gapminder' not found
```

```
out_le
```

```
#> Error in eval(expr, envir, enclos): object 'out_le' not found
```

Think of what `nest()` does as a more intensive version what `group_by()` does. The resulting object is has the tabular form we expect (it is a tibble) but it looks a little unusual. The first two columns are the familiar continent and year. But we now also have a new column, data, that contains a small table of data corresponding to each continent-year group. This is a list-column, something we have not seen before. It turns out to be very useful for bundling together complex objects (structured, in this case, as a list of tibbles, each being a 33x4 table of data) within the rows of our data (which remains tabular). Our "Europe 1977" fit is in there. We can look at it, if we like, by filtering the data and then unnesting the list column.

```
out_le %>% filter(continent == "Europe" & year == 1977) %>% unnest()
```

```
#> Error in eval(lhs, parent, parent): object 'out_le' not found
```

List-columns are useful because we can act on them in a compact and tidy way. In particular, we can pass functions along to each row of the list-column and make something happen. For example, a moment ago we ran a regression of life expectancy and logged GDP for European countries in 1977. We can do that for every continent-year combination in the data. We first create a convenience function called `fit_ols()` that takes a single argument, `df` (for data frame) and that fits the linear model we are interested in. Then we map that function to each of our list-column rows in turn. Recall from Chapter 4 that mutate creates new variables or columns on the fly within a pipeline.

The `map` action is an important idea in functional programming. If you have written code in other, more imperative languages you can think of it as a compact alternative to writing `for … next` loops. You can of course write loops like this in R. Computationally they are often not any less efficient than their functional

alternatives. But mapping functions to arrays is more easily integrated into a sequence of data transformations.

```
fit_ols <- function(df) {
    lm(lifeExp ~ log(gdpPercap), data = df)
}

out_le <- gapminder %>%
    group_by(continent, year) %>%
    nest() %>%
    mutate(model = map(data, fit_ols))
```

```
#> Error in eval(lhs, parent, parent): object 'gapminder' not found
```

```
out_le
```

```
#> Error in eval(expr, envir, enclos): object 'out_le' not found
```

Before starting the pipeline we create a new function: It is a convenience function whose only job is to estimate a particular OLS model on some data. Like almost everything in R, functions are a kind of object. To make a new one, we use the slightly special `function()` function. (Nerds love that sort of thing.) There is a little more detail on creating functions in the Appendix. To see what fit_ols() looks like once it is created, type `fit_ols` without parentheses at the Console. To see what it does, try `fit_ols(df = gapminder)`, or `summary(fit_ols(gapminder))`.

Now we have two list-columns: `data`, and `model`. The latter was created by mapping the `fit_ols()` function to each row of data. Inside each element of model is a linear model for that continent-year. So we now have sixty OLS fits, one for every continent-year grouping. Having the models inside the list column is not much use to us in and of itself. But we can extract the information we want while keeping things in a tidy tabular form. For clarity we will run the pipeline from the beginning again, this time adding a few new steps.

First we extract summary statistics from each model by mapping the `tidy()` function from `broom` to the model list column. Then we unnest the result, dropping the other columns in the process. Finally, we filter out all the Intercept terms, and also drop all observations from Oceania. In the case of the Intercepts we do this just out of convenience. Oceania we drop just because there are so few observations. We put the results in an object called `out_tidy`.

```
fit_ols <- function(df) {
    lm(lifeExp ~ log(gdpPercap), data = df)
}

out_tidy <- gapminder %>%
    group_by(continent, year) %>%
    nest() %>%
    mutate(model = map(data, fit_ols),
           tidied = map(model, tidy)) %>%
    unnest(tidied, .drop = TRUE) %>%
    filter(term %nin% "(Intercept)" &
           continent %nin% "Oceania")
```

```
#> Error in eval(lhs, parent, parent): object 'gapminder' not found
```

```
out_tidy %>% sample_n(5)
```

```
#> Error in eval(lhs, parent, parent): object 'out_tidy' not found
```

We now have tidy regression output with an estimate of the association between log GDP per capita and life expectancy for each year, within continents. We can plot these estimates in a way that takes advantage

of their groupiness.

```
# Figure 6.9: Yearly estimates of the association between GDP and Life Expectancy, pooled by continent.
p <- ggplot(data = out_tidy,
            mapping = aes(x = year, y = estimate,
                          ymin = estimate - 2*std.error,
                          ymax = estimate + 2*std.error,
                          group = continent, color = continent))
```

```
#> Error in ggplot(data = out_tidy, mapping = aes(x = year, y = estimate, : could not find function "gg
```

```
p + geom_pointrange(position = position_dodge(width = 1)) +
    scale_x_continuous(breaks = unique(gapminder$year)) +
    theme(legend.position = "top") +
    labs(x = "Year", y = "Estimate", color = "Continent")
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

The call to `position_dodge()` within `geom_pointrange()` allows the point ranges for each continent to be near each other within years, instead of being plotted right on top of one another. We could have faceted the results by continent, but doing it this way lets us see differences in the yearly estimates much more easily. This technique is very useful not just for cases like this, but also when you want to compare the coefficients given by different kinds of statistical model. This sometimes happens when we're interested in seeing how, say, OLS performs against some other model specification.

22.7 6.7 Plot marginal effects

Our earlier discussion of `predict()` was about obtaining estimates of the average effect of some coefficient, net of the other terms in the model. Over the past decade, estimating and plotting partial or marginal effects from a model has become an increasingly common way of presenting accurate and interpretively useful predictions. Interest in marginal effects plots was stimulated by the realization that the interpretation of terms in logistic regression models, in particular, was trickier than it seemed—especially when there were interaction terms in the model (Ai & Norton, 2003). Thomas Leeper's `margins` package can make these plots for us.

```
library(margins)
```

```
#> Error in library(margins): there is no package called 'margins'
```

To see it in action, we'll take another look at the General Social Survey data in `gss_sm`, this time focusing on the binary variable, obama.As is common with retrospective questions on elections, rather more people claim to have voted for Obama than is consistent with the vote share he received in the election. It is coded 1 if the respondent said they voted for Barack Obama in the 2012 presidential election, and 0 otherwise. In this case, mostly for convenience here, the zero code includes all other answers to the question, including those who said they voted for Mitt Romney, those who said they did not vote, those who refused to answer, and those who said they didn't know who they voted for. We will fit a logistic regression on obama, with `age`, `polviews`, `race`, and `sex` as the predictors. The `age` variable is the respondent's age in years. The `sex` variable is coded as "Male" or "Female" with "Male" as the reference category. The `race` variable is coded as "White", "Black", or "Other" with "White" as the reference category. The polviews measure is a self-reported scale of the respondent's political orientation from "Extremely Conservative" through "Extremely Liberal", with "Moderate" in the middle. We take `polviews` and create a new variable, `polviews_m`, using the `relevel()` function to recode "Moderate" to be the reference category. We fit the model with the `glm()` function, and specify an interaction between race and sex.

```
gss_sm$polviews_m <- relevel(gss_sm$polviews, ref = "Moderate")
```

```
#> Error in relevel(gss_sm$polviews, ref = "Moderate"): object 'gss_sm' not found
```

```
out_bo <- glm(obama ~ polviews_m + sex*race,
              family = "binomial", data = gss_sm)
```

```
#> Error in is.data.frame(data): object 'gss_sm' not found
```

```
summary(out_bo)
```

```
#> Error in summary(out_bo): object 'out_bo' not found
```

The summary reports the coefficients and other information. We can now graph the data in any one of several ways. Using `margins()` we calculate the marginal effects for each variable:

```
bo_m <- margins(out_bo)
```

```
#> Error in margins(out_bo): could not find function "margins"
```

```
summary(bo_m)
```

```
#> Error in summary(bo_m): object 'bo_m' not found
```

The `margins` library comes with several plot methods of its own. If you wish, at this point you can just try `plot(bo_m)` to see a plot of the average marginal effects, produced with the general look of a Stata graphic. Other plot methods in the margins library include `cplot()`, which visualizes marginal effects conditional on a second variable, and `image()`, which shows predictions or marginal effects as a filled heatmap or contour plot.

Alternatively, we can take results from `margins()` and plot them ourselves. To clean up the summary a little a little, we convert it to a tibble, then use `prefix_strip()` and `prefix_replace()` to tidy the labels. We want to strip the polviews_m and sex prefixes, and (to avoid ambiguity about "Other"), adjust the race prefix.

```
bo_gg <- as_tibble(summary(bo_m))
```

```
#> Error in as_tibble(summary(bo_m)): could not find function "as_tibble"
```

```
prefixes <- c("polviews_m", "sex")
bo_gg$factor <- prefix_strip(bo_gg$factor, prefixes)
```

```
#> Error in prefix_strip(bo_gg$factor, prefixes): could not find function "prefix_strip"
```

```
bo_gg$factor <- prefix_replace(bo_gg$factor, "race", "Race: ")
```

```
#> Error in prefix_replace(bo_gg$factor, "race", "Race: "): could not find function "prefix_replace"
```

```
bo_gg %>% select(factor, AME, lower, upper)
```

```
#> Error in eval(lhs, parent, parent): object 'bo_gg' not found
```

Now we have a table that we can plot as we have learned:

```
p <- ggplot(data = bo_gg, aes(x = reorder(factor, AME),
                              y = AME, ymin = lower, ymax = upper))
```

```
#> Error in ggplot(data = bo_gg, aes(x = reorder(factor, AME), y = AME, ymin = lower, : could not find
```

```
p + geom_hline(yintercept = 0, color = "gray80") +
    geom_pointrange() + coord_flip() +
    labs(x = NULL, y = "Average Marginal Effect")
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

If we are just interested in getting conditional effects for a particular variable, then conveniently we can ask the plot methods in the margins library to do the work calculating effects for us but without drawing their plot. Instead, they can return the results in a format we can easily use in ggplot, and with less need for clean up, for the clean-up. For example, with `cplot()`:

```
pv_cp <- cplot(out_bo, x = "sex", draw = FALSE)
```

```
#> Error in cplot(out_bo, x = "sex", draw = FALSE): could not find function "cplot"
```

```
p <- ggplot(data = pv_cp, aes(x = reorder(xvals, yvals),
                              y = yvals, ymin = lower, ymax = upper))
```

```
#> Error in ggplot(data = pv_cp, aes(x = reorder(xvals, yvals), y = yvals, : could not find function "gg
```

```
p + geom_hline(yintercept = 0, color = "gray80") +
    geom_pointrange() + coord_flip() +
    labs(x = NULL, y = "Conditional Effect")
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

The `margins` package is under active development. It can do much more than described here. The vignettes that come with the package provide more extensive discussion and numerous examples.

22.8 6.8 Plots from complex surveys

Social scientists often work with data collected using a complex survey design. Survey instruments may be stratified by region or some other characteristic, contain replicate weights to make them comparable to a reference population, have a clustered structure, and so on. In Chapter 4 we learned how calculate and then plot frequency tables of categorical variables, using some data from the General Social Survey (GSS). However, if we want accurate estimates of US households from the GSS, we will need to take the survey's design into account, and use the survey weights provided in the dataset. Thomas Lumley's `survey` library provides a comprehensive set of tools for addressing these issues. The tools and the theory behind them are discussed in detail in Lumley (2010), and an overview of the package is provided in Lumley (2004). While the functions in the `survey` package are straightforward to use and return results in a generally tidy form, the package predates the tidyverse and its conventions by several years. This means we cannot use survey functions directly with `dplyr`. However, Greg Freedman Ellis has written a helper package, `srvyr`, that solves this problem for us, and lets us use the `survey` library's functions within a data analysis pipeline in a familiar way.

For example, the `gss_lon` data contains a small subset of measures from every wave of the GSS since its inception in 1972. It also contains several variables that describe the design of the survey and provide replicate weights for observations in various years. These technical details are described in the GSS documentation. Similar information is typically provided by other complex surveys. Here we will use this design information to calculate weighted estimates of the distribution of educational attainment by race, for selected survey years from 1976 to 2016.

To begin, we load the `survey` and `srvyr` libraries.

```
library(survey)
```

```
#> Error in library(survey): there is no package called 'survey'
```

```
library(srvyr)
```

```
#> Error in library(srvyr): there is no package called 'srvyr'
```

Next, we take our gss_lon dataset and use the survey tools to create a new object that contains the data, as before, but with some additional information about the survey's design:

```
options(survey.lonely.psu = "adjust")
options(na.action="na.pass")

gss_wt <- subset(gss_lon, year > 1974) %>%
    mutate(stratvar = interaction(year, vstrat)) %>%
    as_survey_design(ids = vpsu,
                     strata = stratvar,
                     weights = wtssall,
                     nest = TRUE)
```

```
#> Error in subset(gss_lon, year > 1974): object 'gss_lon' not found
```

The two options set at the beginning provide some information to the survey library about how to behave. You should consult Lumley (2010) and the survey package documentation for details. The subsequent operations create gss_wt, an object with one additional column (stratvar), describing the yearly sampling strata. We use the interaction() function to do this. It multiplies the vstrat variable by the year variable to get a vector of stratum information for each year. We have to do this because of the way the GSS codes its stratum information. In the next step, we use the as_survey_design() function to add the key pieces of information about the survey design. It adds information about the sampling identifiers (ids), the strata (strata), and the replicate weights (weights). With those in place we can take advantage of a large number of specialized functions in the survey library that allow us to calculate properly weighted survey means or estimate models with the correct sampling specification. For example, we can easily calculate the distribution of education by race for a series of years from 1976 to 2016. We use survey_mean() to do this:

```
out_grp <- gss_wt %>%
    filter(year %in% seq(1976, 2016, by = 4)) %>%
    group_by(year, race, degree) %>%
    summarize(prop = survey_mean(na.rm = TRUE))
```

```
#> Error in eval(lhs, parent, parent): object 'gss_wt' not found
```

```
out_grp
```

```
#> Error in eval(expr, envir, enclos): object 'out_grp' not found
```

The results returned in out_grp include standard errors. We can also ask survey_mean() to calculate confidence intervals for us, if we wish.

Grouping with group_by() lets us calculate counts or means for the innermost variable, grouped by the next variable "up" or "out", in this case, degree by race, such that the proportions for degree will sum to one for each group in race, and this will be done separately for each value of year. If we want the marginal frequencies, such that the values for all combinations of race and degree sum to one within each year, we first have to interact the variables we are cross-classifying. Then we group by the new interacted variable and do the calculation as before:

```
out_mrg <- gss_wt %>%
    filter(year %in% seq(1976, 2016, by = 4)) %>%
    mutate(racedeg = interaction(race, degree)) %>%
    group_by(year, racedeg) %>%
    summarize(prop = survey_mean(na.rm = TRUE))
```

```
#> Error in eval(lhs, parent, parent): object 'gss_wt' not found
```

```
out_mrg
```

```
#> Error in eval(expr, envir, enclos): object 'out_mrg' not found
```

This gives us the numbers that we want and returns them in a tidy data frame. The `interaction()` function produces variable labels that are a compound of the two variables we interacted, with each combination of categories separated by a period, (such as White.Graduate. However, perhaps we would like to see these categories as two separate columns, one for race and one for education, as before. Because the variable labels are organized in a predictable way, we can use one of the convenient functions in the tidyverse's `tidyr` library to separate the single variable into two columns while correctly preserving the row values. Appropriately, this function is called `separate()`.

```
out_mrg <- gss_wt %>%
    filter(year %in% seq(1976, 2016, by = 4)) %>%
    mutate(racedeg = interaction(race, degree)) %>%
    group_by(year, racedeg) %>%
    summarize(prop = survey_mean(na.rm = TRUE)) %>%
    separate(racedeg, sep = "\\.", into = c("race", "degree"))
```

```
#> Error in eval(lhs, parent, parent): object 'gss_wt' not found
```

```
out_mrg
```

```
#> Error in eval(expr, envir, enclos): object 'out_mrg' not found
```

The call to `separate()` says to take the `racedeg` column, split each value when it sees a period, and reorganize the results into two columns, race and degree. This gives us a tidy table much like `out_grp`, but for the marginal frequencies.

Reasonable people can disagree over how best to plot a small multiple of a frequency table while faceting by year, especially when there is some measure of uncertainty attached. A barplot is the obvious approach for a single case, but when there are many years it can become difficult to compare bars across panels. This is especially the case when standard errors or confidence intervals are used in conjunction with bars.Sometimes it may be preferable to show that the underlying variable is categorical, as a bar chart makes clear, and not continuous, as a line graph suggests. Here the trade-off is in favor of the line graphs as the bars are very hard to compare across facets. This is sometimes called a "dynamite plot", not because it looks amazing but because the t-shaped error bars on the tops of the columns make them look like cartoon dynamite plungers. An alternative is to use a line graph to join up the time observations, faceting on educational categories instead of year. Figure 6.12 shows the results for our GSS data in dynamite-plot form, where the error bars are defined as twice the standard error in either direction around the point estimate.

```
p <- ggplot(data = subset(out_grp, race %nin% "Other"),
            mapping = aes(x = degree, y = prop,
                          ymin = prop - 2*prop_se,
                          ymax = prop + 2*prop_se,
                          fill = race,
                          color = race,
                          group = race))
```

```
#> Error in ggplot(data = subset(out_grp, race %nin% "Other"), mapping = aes(x = degree, : could not fi
```

```
dodge <- position_dodge(width=0.9)
```

```
#> Error in position_dodge(width = 0.9): could not find function "position_dodge"
```

```
p + geom_col(position = dodge, alpha = 0.2) +
    geom_errorbar(position = dodge, width = 0.2) +
    scale_x_discrete(labels = scales::wrap_format(10)) +
```

```
    scale_y_continuous(labels = scales::percent) +
    scale_color_brewer(type = "qual", palette = "Dark2") +
    scale_fill_brewer(type = "qual", palette = "Dark2") +
    labs(title = "Educational Attainment by Race",
         subtitle = "GSS 1976-2016",
         fill = "Race",
         color = "Race",
         x = NULL, y = "Percent") +
    facet_wrap(~ year, ncol = 2) +
    theme(legend.position = "top")
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

This plot has a few cosmetic details and adjustments that we will learn more about in Chapter 8. As before, I encourage you to peel back the plot from the bottom, one instruction at a time, to see what changes. One useful adjustment to notice is the new call to the `scales` library to adjust the labels on the x-axis. The adjustment on the y-axis is familiar, `scales::percent` to convert the proportion to a percentage. On the x-axis, the issue is that several of the labels are rather long. If we do not adjust them they will print over one another. The `scales::wrap_format()` function will break long labels into lines. It takes a single numerical argument (here 10) that is the maximum length a string can be before it is wrapped onto a new line.

Faceting by education instead.Figure 6.13: Faceting by education instead. A graph like this is true to the categorical nature of the data, while showing the breakdown of groups within each year. But you should experiment with some alternatives. For example, we might decide that it is better to facet by degree category instead, and put the year on the x-axis within each panel. If we do that, then we can use `geom_line()` to show a time trend, which is more natural, and `geom_ribbon()` to show the error range. This is perhaps a better way to show the data, especially as it brings out the time trends within each degree category, and allows us to see the similarities and differences by racial classification at the same time.

```
p <- ggplot(data = subset(out_grp, race %nin% "Other"),
            mapping = aes(x = year, y = prop, ymin = prop - 2*prop_se,
                          ymax = prop + 2*prop_se, fill = race, color = race,
                          group = race))
```

```
#> Error in ggplot(data = subset(out_grp, race %nin% "Other"), mapping = aes(x = year, : could not find
```

```
p + geom_ribbon(alpha = 0.3, aes(color = NULL)) +
    geom_line() +
    facet_wrap(~ degree, ncol = 1) +
    scale_y_continuous(labels = scales::percent) +
    scale_color_brewer(type = "qual", palette = "Dark2") +
    scale_fill_brewer(type = "qual", palette = "Dark2") +
    labs(title = "Educational Attainment\nby Race",
         subtitle = "GSS 1976-2016", fill = "Race",
         color = "Race", x = NULL, y = "Percent") +
    theme(legend.position = "top")
```

```
#> Error in eval(expr, envir, enclos): object 'p' not found
```

22.9 6.9 Where to go next

In general, when you estimate models and want to plot the results, the difficult step is not the plotting but rather calculating and extracting the right numbers. Generating predicted values and measures of confidence or uncertainty from models requires that you understand the model you are fitting, and the function you

use to fit it, especially when it involves interactions, cross-level effects, or transformations of the predictor or response scales. The details can vary substantially from model type to model type, and also with the goals of any particular analysis. It is unwise to approach them mechanically. That said, several tools exist to help you work with model objects and produce a default set of plots from them.

22.9.1 6.9.1 Default plots for models

Just as model objects in R usually have a default `summary()` method, printing out an overview tailored to the type of model it is, they will usually have a default `plot()` method, too. Figures produced by `plot()` are typically not generated via `ggplot`, but it is usually worth exploring them. They typically make use of either R's base graphics or the `lattice` library (Sarkar, 2008). These are two plotting systems that we do not cover in this book. Default plot methods are easy to examine. Let's take a look again at our simple OLS model.

```
out <- lm(formula = lifeExp ~ log(gdpPercap) + pop + continent, data = gapminder)
```

```
#> Error in is.data.frame(data): object 'gapminder' not found
```

To look at some of R's default plots for this model, use the `plot()` function.

```
# Plot not shown
plot(out, which = c(1,2), ask=FALSE)
```

```
#> Error in plot(out, which = c(1, 2), ask = FALSE): object 'out' not found
```

The `which()` statement here selects the first two of four default plots for this kind of model. If you want to easily reproduce base R's default model graphics using `ggplot`, the `ggfortify` library is worth examining. It is in some ways similar to `broom`, in that it tidies the output of model objects, but it focuses on producing a standard plot (or group of plots) for a wide variety of model types. It does this by defining a function called `autoplot()`. The idea is to be able to use `autoplot()` with the output of many different kinds of model.

A second option worth looking at is the `coefplot` library. It provides a quick way to produce good-quality plots of point estimates and confidence intervals. It has the advantage of managing the estimation of interaction effects and other occasionally tricky calculations.

```
library(coefplot)
```

```
#> Error in library(coefplot): there is no package called 'coefplot'
```

```
out <- lm(formula = lifeExp ~ log(gdpPercap) + log(pop) + continent, data = gapminder)
```

```
#> Error in is.data.frame(data): object 'gapminder' not found
```

```
coefplot(out, sort = "magnitude", intercept = FALSE)
```

```
#> Error in coefplot(out, sort = "magnitude", intercept = FALSE): could not find function "coefplot"
```

22.9.2 6.9.2 Tools in development

Tidyverse tools for modeling and model exploration are being actively developed. The `broom` and `margins` libraries continue to get more and more useful. There are also other projects worth paying attention to. The `infer` packageinfer.netlify.com is in its early stages but can already do useful things in a pipeline-friendly way. You can install it from CRAN with `install.packages("infer")`.

22.9.3 6.9.3 Extensions to ggplot

The `GGally` package provides a suite of functions designed to make producing standard but somewhat complex plots a little easier. For instance, it can produce generalized pairs plots, a useful way of quickly examining possible relationships between several different variables at once. This sort of plot is like the visual version of a correlation matrix. It shows a bivariate plot for all pairs of variables in the data. This is relatively straightforward when all the variables are continuous measures. Things get more complex when, as is often the case in the social sciences, some or all variables are categorical or otherwise limited in the range of values they can take. A generalized pairs plot can handle these cases. For example, Figure ?? shows a generalized pairs plot for five variables from the `organdata` dataset.

```
library(GGally)
```

```
#> Error in library(GGally): there is no package called 'GGally'
```

```
organdata_sm <- organdata %>%
    select(donors, pop_dens, pubhealth,
           roads, consent_law)
```

```
#> Error in eval(lhs, parent, parent): object 'organdata' not found
```

```
ggpairs(data = organdata_sm,
        mapping = aes(color = consent_law),
        upper = list(continuous = wrap("density"), combo = "box_no_facet"),
        lower = list(continuous = wrap("points"), combo = wrap("dot_no_facet")))
```

```
#> Error in ggpairs(data = organdata_sm, mapping = aes(color = consent_law), : could not find function
```

Multi-panel plots like this are intrinsically very rich in information. When combined with several within-panel types of representation, or any more than a modest number of variables, they can become quite complex. They should be used less for the presentation of finished work, although it is possible. More often they are a useful tool for the working researcher to quickly investigate aspects of a data set. The goal is not to pithily summarize a single point one already knows, but to open things up for further exploration.

Chapter 23

Lab 3A: Regression

https://www.matthewrenze.com/workshops/practical-machine-learning-with-r/lab-3a-regression.html

23.1 1. Explore the Data

Load Iris data

```
data(iris)
```

Create scatterplot matrix

```
plot(iris[1:4])
```

```
library(corrgram)
corrgram(iris[1:4])
```

```r
cor(iris[1:4])
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

```r
cor(
  x = iris$Petal.Length,
  y = iris$Petal.Width)
```

```
## [1] 0.9628654
```

```r
plot(
  x = iris$Petal.Length,
  y = iris$Petal.Width,
  xlim = c(0.25, 7),
  ylim = c(0.25, 2.5))
```

23.2 2. Create Training and Test Sets

```
set.seed(42)
```

```
indexes <- sample(
  x = 1:150,
  size = 100)
```

```
train <- iris[indexes, ]
test <- iris[-indexes, ]
```

23.3 3. Predict with Simple Linear Regression

```
simpleModel <- lm(
  formula = Petal.Width ~ Petal.Length,
  data = train)
```

```
plot(
  x = iris$Petal.Length,
  y = iris$Petal.Width,
  xlim = c(0.25, 7),
  ylim = c(0.25, 2.5))

lines(
  x = train$Petal.Length,
  y = simpleModel$fitted,
  col = "red",
  lwd = 3)
```

```r
summary(simpleModel)
```

```
##
## Call:
## lm(formula = Petal.Width ~ Petal.Length, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.56627 -0.12399 -0.01245  0.13209  0.64001
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.34411    0.04996  -6.888 5.47e-10 ***
## Petal.Length  0.41257    0.01192  34.617  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2119 on 98 degrees of freedom
## Multiple R-squared:  0.9244, Adjusted R-squared:  0.9236
## F-statistic:  1198 on 1 and 98 DF,  p-value: < 2.2e-16
```

```r
simplePredictions <- predict(
  object = simpleModel,
  newdata = test)
```

```r
plot(
  x = iris$Petal.Length,
  y = iris$Petal.Width,
  xlim = c(0.25, 7),
  ylim = c(0.25, 2.5))

points(
  x = test$Petal.Length,
```

```
  y = simplePredictions,
  col = "blue",
  pch = 4,
  lwd = 2)

points(
  x = test$Petal.Length,
  y = test$Petal.Width,
  col = "red",
  pch = 16)
```

```
simpleRMSE <- sqrt(mean((test$Petal.Width - simplePredictions)^2))
print(simpleRMSE)
```

```
## [1] 0.1960294
```

23.4 4. Predict with Multiple Regression

```
multipleModel <- lm(
  formula = Petal.Width ~ .,
  data = train)
```

```
summary(multipleModel)
```

```
##
## Call:
## lm(formula = Petal.Width ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.61056 -0.08035 -0.01463  0.09287  0.45260
```

```
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        -0.28170    0.22889  -1.231 0.221503
## Sepal.Length       -0.10639    0.05639  -1.887 0.062271 .
## Sepal.Width         0.19705    0.05894   3.343 0.001189 **
## Petal.Length        0.26896    0.06718   4.004 0.000125 ***
## Speciesversicolor  0.55074    0.17149   3.211 0.001808 **
## Speciesvirginica   0.92273    0.23227   3.973 0.000139 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1788 on 94 degrees of freedom
## Multiple R-squared:  0.9484, Adjusted R-squared:  0.9456
## F-statistic: 345.4 on 5 and 94 DF,  p-value: < 2.2e-16
```

```r
multiplePredictions <- predict(
  object = multipleModel,
  newdata = test)
```

```r
plot(
  x = iris$Petal.Length,
  y = iris$Petal.Width,
  xlim = c(0.25, 7),
  ylim = c(0.25, 2.5))

points(
  x = test$Petal.Length,
  y = multiplePredictions,
  col = "blue",
  pch = 4,
  lwd = 2)

points(
  x = test$Petal.Length,
  y = test$Petal.Width,
  col = "red",
  pch = 16)
```

```
multipleRMSE <- sqrt(mean((test$Petal.Width - multiplePredictions)^2))
print(multipleRMSE)
```

```
## [1] 0.144159
```

23.5 5. Predict with Neural Network Regression

```
normalize <- function(x) {
  (x - min(x)) / (max(x) - min(x)) - 0.5
}
```

```
denormalize <- function(x, y) {
  ((x + 0.5) * (max(y) - min(y))) + min(y)
}
```

```
scaledIris <- data.frame(
  Sepal.Length = normalize(iris$Sepal.Length),
  Sepal.Width = normalize(iris$Sepal.Width),
  Petal.Length = normalize(iris$Petal.Length),
  Petal.Width = normalize(iris$Petal.Width),
  Species = iris$Species)
```

```
scaledTrain <- scaledIris[indexes, ]
scaledTest <- scaledIris[-indexes, ]
```

```
library(nnet)
```

```
neuralRegressor <- nnet(
  formula = Petal.Width ~ .,
  data = scaledTrain,
  linout = TRUE,
  skip = TRUE,
```

```r
  size = 4,
  decay = 0.0001,
  maxit = 500)
```

```
## # weights:  34
## initial  value 97.925222
## iter  10 value 0.557610
## iter  20 value 0.519856
## iter  30 value 0.511567
## iter  40 value 0.492292
## iter  50 value 0.484937
## iter  60 value 0.477067
## iter  70 value 0.473046
## iter  80 value 0.471776
## iter  90 value 0.468606
## iter 100 value 0.459243
## iter 110 value 0.448696
## iter 120 value 0.443403
## iter 130 value 0.439474
## iter 140 value 0.437018
## iter 150 value 0.435668
## iter 160 value 0.433006
## iter 170 value 0.428821
## iter 180 value 0.426469
## iter 190 value 0.425468
## iter 200 value 0.424296
## iter 210 value 0.423836
## iter 220 value 0.423537
## iter 230 value 0.423190
## iter 240 value 0.422704
## iter 250 value 0.421557
## iter 260 value 0.420794
## iter 270 value 0.420288
## iter 280 value 0.419967
## iter 290 value 0.419680
## iter 300 value 0.419337
## iter 310 value 0.419080
## iter 320 value 0.418105
## iter 330 value 0.417495
## iter 340 value 0.416167
## iter 350 value 0.416022
## iter 360 value 0.415458
## iter 370 value 0.415073
## iter 380 value 0.414178
## iter 390 value 0.412626
## iter 400 value 0.409767
## iter 410 value 0.406429
## iter 420 value 0.404543
## iter 430 value 0.403254
## iter 440 value 0.401789
## iter 450 value 0.401003
## iter 460 value 0.400334
## iter 470 value 0.399434
## iter 480 value 0.399090
```

```
## iter 490 value 0.398894
## iter 500 value 0.398773
## final  value 0.398773
## stopped after 500 iterations
```

```
library(NeuralNetTools)
```

```
plotnet(neuralRegressor)
```

```
scaledPredictions <- predict(
  object = neuralRegressor,
  newdata = scaledTest)
```

```
neuralPredictions <- denormalize(
  x = scaledPredictions,
  y = iris$Petal.Width)
```

```
plot(
  x = iris$Petal.Length,
  y = iris$Petal.Width,
  xlim = c(0.25, 7),
  ylim = c(0.25, 2.5))

points(
  x = test$Petal.Length,
  y = neuralPredictions,
  col = "blue",
  pch = 4,
  lwd = 2)

points(
  x = test$Petal.Length,
  y = test$Petal.Width,
  col = "red",
  pch = 16)
```

```
neuralRMSE <- sqrt(mean((test$Petal.Width - neuralPredictions)^2))
print(neuralRMSE)
```

```
## [1] 0.136935
```

23.6 6. Evaluate Regression Models

```
print(simpleRMSE)
```

```
## [1] 0.1960294
```

```
print(multipleRMSE)
```

```
## [1] 0.144159
```

```
print(neuralRMSE)
```

```
## [1] 0.136935
```

Chapter 24

Regression 3b

line 29 does not plot

https://www.matthewrenze.com/workshops/practical-machine-learning-with-r/lab-3b-regression.html

```
library(readr)

policies <- read_csv(file.path(data_raw_dir, "Rates.csv"))
```

```
## Parsed with column specification:
## cols(
##   Gender = col_character(),
##   State = col_character(),
##   State.Rate = col_double(),
##   Height = col_double(),
##   Weight = col_double(),
##   BMI = col_double(),
##   Age = col_double(),
##   Rate = col_double()
## )
```

```
policies
```

```
## # A tibble: 1,942 x 8
##    Gender State State.Rate Height Weight   BMI   Age   Rate
##    <chr>  <chr>      <dbl>  <dbl>  <dbl> <dbl> <dbl>  <dbl>
##  1 Male   MA        0.100     184   67.8  20.0    77 0.332
##  2 Male   VA        0.142     163   89.4  33.6    82 0.869
##  3 Male   NY        0.0908    170   81.2  28.1    31 0.01
##  4 Male   TN        0.120     175   99.7  32.6    39 0.0215
##  5 Male   FL        0.110     184   72.1  21.3    68 0.150
##  6 Male   WA        0.163     166   98.4  35.7    64 0.211
##  7 Male   NY        0.0908    178   98.9  31.2    67 0.146
##  8 Male   NJ        0.132     182   97.1  29.3    46 0.03
##  9 Male   UT        0.106     174   79.3  26.2    73 0.261
## 10 Male   PA        0.129     168   73.9  26.2    81 0.628
## # ... with 1,932 more rows
```

```
summary(policies)
```

```
##     Gender             State             State.Rate         Height
```

```
##  Length:1942        Length:1942        Min.   :0.0010   Min.   :150.0
##  Class :character   Class :character   1st Qu.:0.1103   1st Qu.:162.0
##  Mode  :character   Mode  :character   Median :0.1276   Median :170.0
##                                        Mean   :0.1381   Mean   :169.7
##                                        3rd Qu.:0.1443   3rd Qu.:176.0
##                                        Max.   :0.3181   Max.   :190.0
##      Weight            BMI              Age             Rate
##  Min.   : 44.10   Min.   :16.02   Min.   :18.00   Min.   :0.00100
##  1st Qu.: 68.60   1st Qu.:23.74   1st Qu.:34.00   1st Qu.:0.01475
##  Median : 81.30   Median :28.06   Median :51.00   Median :0.04628
##  Mean   : 81.16   Mean   :28.29   Mean   :50.84   Mean   :0.13806
##  3rd Qu.: 93.80   3rd Qu.:32.46   3rd Qu.:68.00   3rd Qu.:0.17269
##  Max.   :116.50   Max.   :46.80   Max.   :84.00   Max.   :0.99900
```

```r
library(RColorBrewer)
palette <- brewer.pal(9, "Reds")
```

```r
# plot(
#   x = policies,
#   col = palette[cut(x = policies$Rate, breaks = 9)]
#   )
```

```r
library(corrgram)
corrgram(policies)
```

```r
cor(policies[3:8])
```

```
##              State.Rate      Height      Weight         BMI         Age
## State.Rate  1.000000000 -0.01652294 0.009233267  0.01924141  0.11234748
## Height     -0.016522938  1.00000000 0.238085304 -0.31696110 -0.16478131
## Weight      0.009233267  0.23808530 1.000000000  0.83962760  0.01167918
## BMI         0.019241409 -0.31696110 0.839627602  1.00000000  0.10231657
## Age         0.112347476 -0.16478131 0.011679178  0.10231657  1.00000000
## Rate        0.226852143 -0.12858150 0.060939196  0.14050657  0.78007905
```

```
##                   Rate
## State.Rate  0.2268521
## Height     -0.1285815
## Weight      0.0609392
## BMI         0.1405066
## Age         0.7800790
## Rate        1.0000000
```

```
cor(
  x = policies$Age,
  y = policies$Rate)
```

```
## [1] 0.780079
```

```
plot(
  x = policies$Age,
  y = policies$Rate)
```

Chapter 25

2. Split the Data into Test and Training Sets

```r
set.seed(42)
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'lattice'

## The following object is masked from 'package:corrgram':
##
##     panel.fill

## Loading required package: ggplot2
```

```r
indexes <- createDataPartition(
  y = policies$Rate,
  p = 0.80,
  list = FALSE)

train <- policies[indexes, ]
test <- policies[-indexes, ]
```

```r
print(nrow(train))
```

```
## [1] 1555
```

```r
print(nrow(test))
```

```
## [1] 387
```

Chapter 26

3. Predict with Simple Linear Regression

```
simpleModel <- lm(
  formula = Rate ~ Age,
  data = train)
```

```
plot(
  x = policies$Age,
  y = policies$Rate)

lines(
  x = train$Age,
  y = simpleModel$fitted,
  col = "red",
  lwd = 3)
```

```r
summary(simpleModel)
```

```
##
## Call:
## lm(formula = Rate ~ Age, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.18237 -0.09092 -0.02208  0.06002  0.62697
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.2650115  0.0087679  -30.23   <2e-16 ***
## Age          0.0079630  0.0001609   49.50   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1236 on 1553 degrees of freedom
## Multiple R-squared:  0.6121, Adjusted R-squared:  0.6118
## F-statistic:  2450 on 1 and 1553 DF,  p-value: < 2.2e-16
```

```r
simplePredictions <- predict(
  object = simpleModel,
  newdata = test)
```

```r
plot(
  x = policies$Age,
  y = policies$Rate)


points(
  x = test$Age,
  y = simplePredictions,
  col = "blue",
  pch = 4,
  lwd = 2)
```

```
simpleRMSE <- sqrt(mean((test$Rate - simplePredictions)^2))
print(simpleRMSE)
```

```
## [1] 0.1148266
```

Chapter 27

4. Predict with Multiple Linear Regression

```r
multipleModel <- lm(
  formula = Rate ~ Age + Gender + State.Rate + BMI,
  data = train)
```

```r
summary(multipleModel)
```

```
##
## Call:
## lm(formula = Rate ~ Age + Gender + State.Rate + BMI, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.24620 -0.08738 -0.02936  0.05979  0.60437
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.4233471  0.0189843 -22.300  < 2e-16 ***
## Age          0.0077309  0.0001561  49.538  < 2e-16 ***
## GenderMale   0.0355968  0.0060606   5.873 5.21e-09 ***
## State.Rate   0.6258740  0.0686779   9.113  < 2e-16 ***
## BMI          0.0023340  0.0005240   4.454 9.03e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1188 on 1550 degrees of freedom
## Multiple R-squared:  0.6424, Adjusted R-squared:  0.6415
## F-statistic: 696.3 on 4 and 1550 DF,  p-value: < 2.2e-16
```

```r
multiplePredictions <- predict(
  object = multipleModel,
  newdata = test)
```

```r
plot(
  x = policies$Age,
  y = policies$Rate)
```

```
points(
  x = test$Age,
  y = multiplePredictions,
  col = "blue",
  pch = 4,
  lwd = 2)
```

```
multipleRMSE <- sqrt(mean((test$Rate - multiplePredictions)^2))
print(multipleRMSE)
```

```
## [1] 0.1102457
```

Chapter 28

6. Predict with Neural Network Regression

```
normalize <- function(x) {
  (x - min(x)) / (max(x) - min(x)) - 0.5
}
```

```
denormalize <- function(x, y) {
  ((x + 0.5) * (max(y) - min(y))) + min(y)
}
```

```
scaledPolicies <- data.frame(
  Gender = policies$Gender,
  State.Rate = normalize(policies$State.Rate),
  BMI = normalize(policies$BMI),
  Age = normalize(policies$Age),
  Rate = normalize(policies$Rate))
```

```
scaledTrain <- scaledPolicies[indexes, ]
scaledTest <- scaledPolicies[-indexes, ]
```

```
library(nnet)

neuralRegressor <- nnet(
  formula = Rate ~ .,
  data = scaledTrain,
  linout = TRUE,
  size = 5,
  decay = 0.0001,
  maxit = 1000)
```

```
## # weights:  31
## initial  value 132.761775
## iter  10 value 21.598965
## iter  20 value 12.422322
## iter  30 value 3.800375
## iter  40 value 3.408680
## iter  50 value 3.314877
## iter  60 value 2.906282
```

```
## iter  70 value 2.717530
## iter  80 value 2.397142
## iter  90 value 2.353781
## iter 100 value 2.305473
## iter 110 value 2.276631
## iter 120 value 2.247389
## iter 130 value 2.232213
## iter 140 value 2.216797
## iter 150 value 2.167034
## iter 160 value 2.104010
## iter 170 value 2.042041
## iter 180 value 2.000514
## iter 190 value 1.987659
## iter 200 value 1.983122
## iter 210 value 1.950894
## iter 220 value 1.929749
## iter 230 value 1.914172
## iter 240 value 1.903245
## iter 250 value 1.895931
## iter 260 value 1.894051
## iter 270 value 1.891183
## iter 280 value 1.888449
## iter 290 value 1.883886
## iter 300 value 1.880534
## iter 310 value 1.877332
## iter 320 value 1.877112
## iter 330 value 1.876228
## iter 340 value 1.874656
## iter 350 value 1.872564
## iter 360 value 1.871248
## iter 370 value 1.870875
## iter 380 value 1.870699
## iter 380 value 1.870699
## iter 380 value 1.870699
## final  value 1.870699
## converged
```

```r
scaledPredictions <- predict(
  object = neuralRegressor,
  newdata = scaledTest)
```

```r
neuralPredictions <- denormalize(
  x = scaledPredictions,
  y = policies$Rate)
```

```r
plot(
  x = train$Age,
  y = train$Rate)

points(
  x = test$Age,
  y = neuralPredictions,
  col = "blue",
  pch = 4,
  lwd = 2)
```

```
library(NeuralNetTools)
plotnet(neuralRegressor)
```

```
neuralRMSE <- sqrt(mean((test$Rate - neuralPredictions)^2))
print(neuralRMSE)
```

```
## [1] 0.03990543
```

Chapter 29

7. Evaluate the Regression Models

```r
print(simpleRMSE)
```

```
## [1] 0.1148266
```

```r
print(multipleRMSE)
```

```
## [1] 0.1102457
```

```r
print(neuralRMSE)
```

```
## [1] 0.03990543
```

Chapter 30

Employee attrition

Article: https://www.business-science.io/business/2017/09/18/hr_employee_attrition.html Data: https://www.ibm.com/communities/analytics/watson-analytics-blog/hr-employee-attrition/

30.1 EMPLOYEE ATTRITION: A MAJOR PROBLEM

Bill Gates was once quoted as saying,

> "You take away our top 20 employees and we [Microsoft] become a mediocre company".

His statement cuts to the core of a major problem: employee attrition. An organization is only as good as its employees, and these people are the true source of its competitive advantage.

Organizations face huge costs resulting from employee turnover. Some costs are tangible such as training expenses and the time it takes from when an employee starts to when they become a productive member. However, the most important costs are intangible. Consider what's lost when a productive employee quits: new product ideas, great project management, or customer relationships.

With advances in machine learning and data science, its possible to not only predict employee attrition but to understand the key variables that influence turnover. We'll take a look at two cutting edge techniques:

1. Machine Learning with `h2o.automl()` from the h2o package: This function takes automated machine learning to the next level by testing a number of advanced algorithms such as random forests, ensemble methods, and deep learning along with more traditional algorithms such as logistic regression. The main takeaway is that we can now easily achieve predictive performance that is in the same ball park (and in some cases even better than) commercial algorithms and ML/AI software.

2. Feature Importance with the `lime` package: The problem with advanced machine learning algorithms such as deep learning is that it's near impossible to understand the algorithm because of its complexity. This has all changed with the lime package. The major advancement with lime is that, by recursively analyzing the models locally, it can extract feature importance that repeats globally. What this means to us is that lime has opened the door to understanding the ML models regardless of complexity. Now the best (and typically very complex) models can also be investigated and potentially understood as to what variables or features make the model tick.

30.2 EMPLOYEE ATTRITION: MACHINE LEARNING ANALYSIS

With these new automated ML tools combined with tools to uncover critical variables, we now have capabilities for both extreme predictive accuracy and understandability, which was previously impossible! We'll investigate an HR Analytic example of employee attrition that was evaluated by IBM Watson.

30.2.1 IBM WATSON (WHERE WE GOT THE DATA)

The example comes from IBM Watson Analytics website. You can download the data and read the analysis here:

Get data used in this post here. Read IBM Watson Analytics article here. To summarize, the article makes a usage case for IBM Watson as an automated ML platform. The article shows that using Watson, the analyst was able to detect features that led to increased probability of attrition.

30.2.2 AUTOMATED MACHINE LEARNING (WHAT WE DID WITH THE DATA)

In this example we'll show how we can use the combination of H2O for developing a complex model with high predictive accuracy on unseen data and then how we can use LIME to understand important features related to employee attrition.

30.2.3 PACKAGES

Load the following packages.

```
# Load the following packages
library(tidyquant)  # Loads tidyverse and several other pkgs
```

```
## Loading required package: lubridate

##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date

## Loading required package: PerformanceAnalytics

## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

##
## Attaching package: 'PerformanceAnalytics'
```

```
## The following object is masked from 'package:graphics':
##
##     legend

## Loading required package: quantmod

## Loading required package: TTR

## Version 0.4-0 included new data defaults. See ?getSymbols.

## Loading required package: tidyverse

## -- Attaching packages --------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.1.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0

## -- Conflicts ------------------------------------ tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()        masks base::date()
## x dplyr::filter()          masks stats::filter()
## x dplyr::first()           masks xts::first()
## x lubridate::intersect()   masks base::intersect()
## x dplyr::lag()             masks stats::lag()
## x dplyr::last()            masks xts::last()
## x lubridate::setdiff()     masks base::setdiff()
## x lubridate::union()       masks base::union()
```

```r
library(readxl)     # Super simple excel reader
library(h2o)        # Professional grade ML pkg
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## ----------------------------------------------------------------------

##
## Attaching package: 'h2o'

## The following objects are masked from 'package:lubridate':
##
##     day, hour, month, week, year

## The following objects are masked from 'package:stats':
##
##     cor, sd, var

## The following objects are masked from 'package:base':
##
```

Table 30.1: First 10 rows

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educat
41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sc
49	No	Travel_Frequently	279	Research & Development	8	1	Life Sc
37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other
33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sc
27	No	Travel_Rarely	591	Research & Development	2	1	Medica
32	No	Travel_Frequently	1005	Research & Development	2	2	Life Sc
59	No	Travel_Rarely	1324	Research & Development	3	3	Medica
30	No	Travel_Rarely	1358	Research & Development	24	1	Life Sc
38	No	Travel_Frequently	216	Research & Development	23	3	Life Sc
36	No	Travel_Rarely	1299	Research & Development	27	3	Medica

```
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```r
library(lime)       # Explain complex black-box ML models
```

```
##
## Attaching package: 'lime'

## The following object is masked from 'package:dplyr':
##
##     explain
```

30.2.4 DATA

Download the data here. You can load the data using read_excel(), pointing the path to your local file.

```r
# Read excel data
hr_data_raw <- read_excel(path = file.path(data_raw_dir,
                                           "WA_Fn-UseC_-HR-Employee-Attrition.xlsx"))
```

Let's check out the raw data. It's 1470 rows (observations) by 35 columns (features). The "Attrition" column is our target. We'll use all other columns as features to our model.

```r
# View first 10 rows
hr_data_raw[1:10,] %>%
    knitr::kable(caption = "First 10 rows")
```

The only pre-processing we'll do in this example is change all character data types to factors. This is needed for H2O. We could make a number of other numeric data that is actually categorical factors, but this tends to increase modeling time and can have little improvement on model performance.

```r
hr_data <- hr_data_raw %>%
    mutate_if(is.character, as.factor) %>%
    select(Attrition, everything())
```

Let's take a glimpse at the processed dataset. We can see all of the columns. Note our target ("Attrition") is the first column.

```r
glimpse(hr_data)
```

```
## Observations: 1,470
## Variables: 35
```

```
## $ Attrition                <fct> Yes, No, Yes, No, No, No, No, No, No,...
## $ Age                      <dbl> 41, 49, 37, 33, 27, 32, 59, 30, 38, 3...
## $ BusinessTravel           <fct> Travel_Rarely, Travel_Frequently, Tra...
## $ DailyRate                <dbl> 1102, 279, 1373, 1392, 591, 1005, 132...
## $ Department               <fct> Sales, Research & Development, Resear...
## $ DistanceFromHome         <dbl> 1, 8, 2, 3, 2, 2, 3, 24, 23, 27, 16, ...
## $ Education                <dbl> 2, 1, 2, 4, 1, 2, 3, 1, 3, 3, 3, 2, 1...
## $ EducationField           <fct> Life Sciences, Life Sciences, Other, ...
## $ EmployeeCount            <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ EmployeeNumber           <dbl> 1, 2, 4, 5, 7, 8, 10, 11, 12, 13, 14,...
## $ EnvironmentSatisfaction  <dbl> 2, 3, 4, 4, 1, 4, 3, 4, 4, 3, 1, 4, 1...
## $ Gender                   <fct> Female, Male, Male, Female, Male, Mal...
## $ HourlyRate               <dbl> 94, 61, 92, 56, 40, 79, 81, 67, 44, 9...
## $ JobInvolvement           <dbl> 3, 2, 2, 3, 3, 3, 4, 3, 2, 3, 4, 2, 3...
## $ JobLevel                 <dbl> 2, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 2, 1...
## $ JobRole                  <fct> Sales Executive, Research Scientist, ...
## $ JobSatisfaction          <dbl> 4, 2, 3, 3, 2, 4, 1, 3, 3, 3, 2, 3, 3...
## $ MaritalStatus            <fct> Single, Married, Single, Married, Mar...
## $ MonthlyIncome            <dbl> 5993, 5130, 2090, 2909, 3468, 3068, 2...
## $ MonthlyRate              <dbl> 19479, 24907, 2396, 23159, 16632, 118...
## $ NumCompaniesWorked       <dbl> 8, 1, 6, 1, 9, 0, 4, 1, 0, 6, 0, 0, 1...
## $ Over18                   <fct> Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y...
## $ OverTime                 <fct> Yes, No, Yes, Yes, No, No, Yes, No, N...
## $ PercentSalaryHike        <dbl> 11, 23, 15, 11, 12, 13, 20, 22, 21, 1...
## $ PerformanceRating        <dbl> 3, 4, 3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3...
## $ RelationshipSatisfaction <dbl> 1, 4, 2, 3, 4, 3, 1, 2, 2, 2, 3, 4, 4...
## $ StandardHours            <dbl> 80, 80, 80, 80, 80, 80, 80, 80, 80, 8...
## $ StockOptionLevel         <dbl> 0, 1, 0, 0, 1, 0, 3, 1, 0, 2, 1, 0, 1...
## $ TotalWorkingYears        <dbl> 8, 10, 7, 8, 6, 8, 12, 1, 10, 17, 6, ...
## $ TrainingTimesLastYear    <dbl> 0, 3, 3, 3, 3, 2, 3, 2, 2, 3, 5, 3, 1...
## $ WorkLifeBalance          <dbl> 1, 3, 3, 3, 3, 2, 2, 3, 3, 2, 3, 3, 2...
## $ YearsAtCompany           <dbl> 6, 10, 0, 8, 2, 7, 1, 1, 9, 7, 5, 9, ...
## $ YearsInCurrentRole       <dbl> 4, 7, 0, 7, 2, 7, 0, 0, 7, 7, 4, 5, 2...
## $ YearsSinceLastPromotion  <dbl> 0, 1, 0, 3, 2, 3, 0, 0, 1, 7, 0, 0, 4...
## $ YearsWithCurrManager     <dbl> 5, 7, 0, 0, 2, 6, 0, 0, 8, 7, 3, 8, 3...
```

30.3 MODELING EMPLOYEE ATTRITION

We are going to use the h2o.automl() function from the H2O platform to model employee attrition.

30.3.1 H2O

First, we need to initialize the Java Virtual Machine (JVM) that H2O uses locally.

```r
# Initialize H2O JVM
h2o.init()
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /tmp/RtmpGhv8nn/h2o_datascience_started_from_r.out
```

```
##     /tmp/RtmpGhv8nn/h2o_datascience_started_from_r.err
##
##
## Starting H2O JVM and connecting: .. Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 seconds 274 milliseconds
##     H2O cluster timezone:       America/Chicago
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.22.1.1
##     H2O cluster version age:    4 months and 15 days !!!
##     H2O cluster name:           H2O_started_from_R_datascience_bid370
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.73 GB
##     H2O cluster total cores:    6
##     H2O cluster allowed cores:  6
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.5.3 (2019-03-11)
```

```
## Warning in h2o.clusterInfo():
## Your H2O cluster version is too old (4 months and 15 days)!
## Please download and install the latest version from http://h2o.ai/download/
```

```r
h2o.no_progress() # Turn off output of progress bars
```

Next, we change our data to an h2o object that the package can interpret. We also split the data into training, validation, and test sets. Our preference is to use 70%, 15%, 15%, respectively.

```r
# Split data into Train/Validation/Test Sets
hr_data_h2o <- as.h2o(hr_data)

split_h2o <- h2o.splitFrame(hr_data_h2o, c(0.7, 0.15), seed = 1234 )

train_h2o <- h2o.assign(split_h2o[[1]], "train" ) # 70%
valid_h2o <- h2o.assign(split_h2o[[2]], "valid" ) # 15%
test_h2o  <- h2o.assign(split_h2o[[3]], "test" )  # 15%
```

30.4 MODEL

Now we are ready to model. We'll set the target and feature names. The target is what we aim to predict (in our case "Attrition"). The features (every other column) are what we will use to model the prediction.

```r
# Set names for h2o
y <- "Attrition"
x <- setdiff(names(train_h2o), y)
```

Now the fun begins. We run the h2o.automl() setting the arguments it needs to run models against. For more information, see the h2o.automl documentation.

- x = x: The names of our feature columns.

- y = y: The name of our target column.
- training_frame = train_h2o: Our training set consisting of 70% of the data.
- leaderboard_frame = valid_h2o: Our validation set consisting of 15% of the data. H2O uses this to ensure the model does not overfit the data.
- max_runtime_secs = 30: We supply this to speed up H2O's modeling. The algorithm has a large number of complex models so we want to keep things moving at the expense of some accuracy.

```
# Run the automated machine learning
automl_models_h2o <- h2o.automl(
    x = x,
    y = y,
    training_frame    = train_h2o,
    leaderboard_frame = valid_h2o,
    max_runtime_secs  = 30
    )
```

All of the models are stored the automl_models_h2o object. However, we are only concerned with the leader, which is the best model in terms of accuracy on the validation set. We'll extract it from the models object.

```
# Extract leader model
automl_leader <- automl_models_h2o@leader
```

30.5 PREDICT

Now we are ready to predict on our test set, which is unseen from during our modeling process. This is the true test of performance. We use the h2o.predict() function to make predictions.

```
# Predict on hold-out set, test_h2o
pred_h2o <- h2o.predict(object = automl_leader, newdata = test_h2o)
```

30.6 PERFORMANCE

Now we can evaluate our leader model. We'll reformat the test set an add the predictions as column so we have the actual and prediction columns side-by-side.

```
# Prep for performance assessment
test_performance <- test_h2o %>%
    tibble::as_tibble() %>%
    select(Attrition) %>%
    add_column(pred = as.vector(pred_h2o$predict)) %>%
    mutate_if(is.character, as.factor)
test_performance
```

```
## # A tibble: 211 x 2
##    Attrition pred
##    <fct>     <fct>
##  1 No        No
##  2 No        No
##  3 Yes       Yes
##  4 No        No
##  5 No        No
##  6 No        No
```

```
##  7 Yes       Yes
##  8 No        No
##  9 No        No
## 10 Yes       Yes
## # ... with 201 more rows
```

We can use the table() function to quickly get a confusion table of the results. We see that the leader model wasn't perfect, but it did a decent job identifying employees that are likely to quit. For perspective, a logistic regression would not perform nearly this well.

```r
# Confusion table counts
confusion_matrix <- test_performance %>%
    table()
confusion_matrix
```

```
##          pred
## Attrition  No Yes
##       No  165  17
##       Yes   9  20
```

We'll run through a binary classification analysis to understand the model performance.

```r
# Performance analysis
tn <- confusion_matrix[1]
tp <- confusion_matrix[4]
fp <- confusion_matrix[3]
fn <- confusion_matrix[2]

accuracy <- (tp + tn) / (tp + tn + fp + fn)
misclassification_rate <- 1 - accuracy
recall <- tp / (tp + fn)
precision <- tp / (tp + fp)
null_error_rate <- tn / (tp + tn + fp + fn)

tibble(
    accuracy,
    misclassification_rate,
    recall,
    precision,
    null_error_rate
) %>%
    transpose()
```

```
## [[1]]
## [[1]]$accuracy
## [1] 0.8767773
##
## [[1]]$misclassification_rate
## [1] 0.1232227
##
## [[1]]$recall
## [1] 0.6896552
##
## [[1]]$precision
## [1] 0.5405405
##
## [[1]]$null_error_rate
```

```
## [1] 0.7819905
```

It is important to understand is that the accuracy can be misleading: 88% sounds pretty good especially for modeling HR data, but if we just pick Attrition = NO we would get an accuracy of about 79%. Doesn't sound so great now.

Before we make our final judgement, let's dive a little deeper into precision and recall. Precision is when the model predicts yes, how often is it actually yes. Recall (also true positive rate or specificity) is when the actual value is yes how often is the model correct. Confused yet? Let's explain in terms of what's important to HR.

Most HR groups would probably prefer to incorrectly classify folks not looking to quit as high potential of quiting rather than classify those that are likely to quit as not at risk. Because it's important to not miss at risk employees, HR will really care about recall or when the actual value is Attrition = YES how often the model predicts YES.

Recall for our model is 62%. In an HR context, this is 62% more employees that could potentially be targeted prior to quiting. From that standpoint, an organization that loses 100 people per year could possibly target 62 implementing measures to retain.

30.7 LIME

We have a very good model that is capable of making very accurate predictions on unseen data, but what can it tell us about what causes attrition? Let's find out using LIME.

30.7.1 SETUP

The lime package implements LIME in R. One thing to note is that it's not setup out-of-the-box to work with h2o. The good news is with a few functions we can get everything working properly. We'll need to make two custom functions:

- model_type: Used to tell lime what type of model we are dealing with. It could be classification, regression, survival, etc.
- predict_model: Used to allow lime to perform predictions that its algorithm can interpret.

The first thing we need to do is identify the class of our model leader object. We do this with the class() function.

```r
class(automl_leader)
```

```
## [1] "H2OBinomialModel"
## attr(,"package")
## [1] "h2o"
```

Next we create our model_type function. It's only input is x the h2o model. The function simply returns "classification", which tells LIME we are classifying.

```r
# Setup lime::model_type() function for h2o
model_type.H2OBinomialModel <- function(x, ...) {
    # Function tells lime() what model type we are dealing with
    # 'classification', 'regression', 'survival', 'clustering', 'multilabel', etc
    #
    # x is our h2o model

    return("classification")
}
```

Now we can create our predict_model function. The trick here is to realize that it's inputs must be x a model, newdata a dataframe object (this is important), and type which is not used but can be use to switch the output type. The output is also a little tricky because it must be in the format of probabilities by classification (this is important; shown next). Internally we just call the h2o.predict() function.

```
# Setup lime::predict_model() function for h2o
predict_model.H2OBinomialModel <- function(x, newdata, type, ...) {
    # Function performs prediction and returns dataframe with Response
    #
    # x is h2o model
    # newdata is data frame
    # type is only setup for data frame

    pred <- h2o.predict(x, as.h2o(newdata))

    # return probs
    return(as.data.frame(pred[,-1]))

}
```

Run this next script to show you what the output looks like and to test our predict_model function. See how it's the probabilities by classification. It must be in this form for model_type = "classification".

```
# Test our predict_model() function
predict_model(x = automl_leader, newdata = as.data.frame(test_h2o[,-1]), type = 'raw') %>%
    tibble::as_tibble()
```

```
## # A tibble: 211 x 2
##       No     Yes
##    <dbl>   <dbl>
##  1 0.812 0.188
##  2 0.940 0.0604
##  3 0.164 0.836
##  4 0.985 0.0152
##  5 0.808 0.192
##  6 0.991 0.00903
##  7 0.190 0.810
##  8 0.927 0.0731
##  9 0.860 0.140
## 10 0.604 0.396
## # ... with 201 more rows
```

Now the fun part, we create an explainer using the lime() function. Just pass the training data set without the "Attribution column". The form must be a data frame, which is OK since our predict_model function will switch it to an h2o object. Set model = automl_leader our leader model, and bin_continuous = FALSE. We could tell the algorithm to bin continuous variables, but this may not make sense for categorical numeric data that we didn't change to factors.

```
# Run lime() on training set
explainer <- lime::lime(
    as.data.frame(train_h2o[,-1]),
    model          = automl_leader,
    bin_continuous = FALSE)
```

```
## Warning: Data contains numeric columns with zero variance
```

Now we run the explain() function, which returns our explanation. This can take a minute to run so we

limit it to just the first ten rows of the test data set. We set n_labels = 1 because we care about explaining a single class. Setting n_features = 4 returns the top four features that are critical to each case. Finally, setting kernel_width = 0.5 allows us to increase the "model_r2" value by shrinking the localized evaluation.

```
# Run explain() on explainer
explanation <- lime::explain(
    as.data.frame(test_h2o[1:10,-1]),
    explainer    = explainer,
    n_labels     = 1,
    n_features   = 4,
    kernel_width = 0.5)
```

```
## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
```

```
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range

## Warning in gower_work(x = x, y = y, pair_x = pair_x, pair_y = pair_y, n =
## NULL, : skipping variable with zero or non-finite range
```

30.8 FEATURE IMPORTANCE VISUALIZATION

The payoff for the work we put in using LIME is this feature importance plot. This allows us to visualize each of the ten cases (observations) from the test data. The top four features for each case are shown. Note that they are not the same for each case. The green bars mean that the feature supports the model conclusion, and the red bars contradict. We'll focus in on Cases with Label = Yes, which are predicted to have attrition. We can see a common theme with Case 3 and Case 7: Training Time, Job Role, and Over Time are among the top factors influencing attrition. These are only two cases, but they can be used to potentially generalize to the larger population as we will see next.

```r
plot_features(explanation) +
    labs(title = "HR Predictive Analytics: LIME Feature Importance Visualization",
         subtitle = "Hold Out (Test) Set, First 10 Cases Shown")
```

HR Predictive Analytics: LIME Feature Importance Visualizatio
Hold Out (Test) Set, First 10 Cases Shown
Case: 1
Label: No
Probability: 0.81
Explanation Fit: 0.31
OverTime = Yes
YearsSinceLastPromotion
YearsWithCurrManager
Education
Case: 2
Label: No
Probability: 0.94
Explanation Fit: 0.33
OverTime = No
YearsSinceLastPromotion
YearsWithCurrManager
EmployeeNumber
Case: 3
Label: Yes
Probability: 0.84
Explanation Fit: 0.32
OverTime = Yes
YearsSinceLastPromotion
YearsWithCurrManager
EmployeeNumber
Case: 4
Label: No
Probability: 0.98
Explanation Fit: 0.28
OverTime = No
NumCompaniesWorked
YearsSinceLastPromotion
PerformanceRating
Case: 5
Label: No
Probability: 0.81
Explanation Fit: 0.30
OverTime = Yes
YearsSinceLastPromotion
YearsWithCurrManager
Education
Case: 6
Label: No
Probability: 0.99
Explanation Fit: 0.31
OverTime = No
NumCompaniesWorked
YearsSinceLastPromotion
MonthlyRate
Case: 7
Label: Yes
Probability: 0.81
Explanation Fit: 0.33
OverTime = Yes
YearsSinceLastPromotion
YearsWithCurrManager
Education
Case: 8
Label: No
Probability: 0.93
Explanation Fit: 0.29
OverTime = No
YearsSinceLastPromotion
YearsWithCurrManager
EmployeeNumber
Case: 9
Label: No
Probability: 0.86
Explanation Fit: 0.30
OverTime = No
YearsSinceLastPromotion
YearsWithCurrManager
MonthlyRate
Case: 10
Label: No
Probability: 0.60
Explanation Fit: 0.32
OverTime = Yes
YearsSinceLastPromotion
YearsWithCurrManager
EmployeeNumber
Feature
−0.2 −0.1 0.0 0.1 0.2
Weight
Supports
Contradicts

30.9 WHAT FEATURES ARE LINKED TO EMPLOYEE ATTRITION?

Now we turn to our three critical features from the LIME Feature Importance Plot:

- Training Time
- Job Role
- Over Time

We'll subset this data and visualize to detect trends.

```
# Focus on critical features of attrition
attrition_critical_features <- hr_data %>%
    tibble::as_tibble() %>%
    select(Attrition, TrainingTimesLastYear, JobRole, OverTime) %>%
    rowid_to_column(var = "Case")
attrition_critical_features
```

```
## # A tibble: 1,470 x 5
##     Case Attrition TrainingTimesLastYear JobRole                   OverTime
##    <int> <fct>                     <dbl> <fct>                     <fct>
##  1     1 Yes                           0 Sales Executive           Yes
##  2     2 No                            3 Research Scientist        No
##  3     3 Yes                           3 Laboratory Technician     Yes
##  4     4 No                            3 Research Scientist        Yes
##  5     5 No                            3 Laboratory Technician     No
##  6     6 No                            2 Laboratory Technician     No
##  7     7 No                            3 Laboratory Technician     Yes
##  8     8 No                            2 Laboratory Technician     No
##  9     9 No                            2 Manufacturing Director    No
## 10    10 No                            3 Healthcare Representative No
## # ... with 1,460 more rows
```

30.10 TRAINING

From the violin plot, the employees that stay tend to have a large peaks at two and three trainings per year whereas the employees that leave tend to have a large peak at two trainings per year. This suggests that employees with more trainings may be less likely to leave.

```
ggplot(attrition_critical_features, aes(x = Attrition,
                                        y = TrainingTimesLastYear)) +
    geom_violin()  +
    geom_jitter(alpha = 0.25)
```

```
attrition_critical_features %>%
    ggplot(aes(Attrition, TrainingTimesLastYear)) +
    geom_jitter(alpha = 0.5, fill = palette_light()[[1]]) +
    geom_violin(alpha = 0.7, fill = palette_light()[[1]]) +
    theme_tq() +
    labs(
    title = "Prevalance of Training is Lower in Attrition = Yes",
    subtitle = "Suggests that increased training is related to lower attrition"
    )
```

30.11 OVERTIME

The plot below shows a very interesting relationship: a very high proportion of employees that turnover are working over time. The opposite is true for employees that stay.

```
attrition_critical_features %>%
    mutate(OverTime = case_when(
                        OverTime == "Yes" ~ 1,
                        OverTime == "No"  ~ 0 )) %>%
    ggplot(aes(Attrition, OverTime)) +
    geom_jitter(alpha = 0.5, fill = palette_light()[[1]]) +
    geom_violin(alpha = 0.7, fill = palette_light()[[1]]) +
    theme_tq() +
    labs(
    title = "Prevalance of Over Time is Higher in Attrition = Yes",
    subtitle = "Suggests that increased overtime is related to higher attrition")
```

```
ggplot(attrition_critical_features, aes(x = Attrition,
                                        y = OverTime,
                                        )) +
    # geom_violin(aes(y = ..prop.., group = 1)) +
    geom_jitter(alpha = 0.5)
```

30.12 JOB ROLE

Several job roles are experiencing more turnover. Sales reps have the highest turnover at about 40% followed by Lab Technician, Human Resources, Sales Executive, and Research Scientist. It may be worthwhile to investigate what localized issues could be creating the high turnover among these groups within the organization.

```
p <- ggplot(data = subset(attrition_critical_features, Attrition == "Yes"),
            mapping = aes(x = JobRole))
p + geom_bar(mapping = aes(y = ..prop.., group = 1)) +
    coord_flip()
```

```
# geom_bar(mapping = aes(y = ..prop.., group = 1))
```

```
p <- ggplot(data = attrition_critical_features,
            mapping = aes(x = JobRole))
p + geom_bar(mapping = aes(y = ..prop.., group = 1)) +
    coord_flip() +
    facet_wrap(Attrition ~ .)
```

```
attrition_critical_features %>%
    group_by(JobRole, Attrition) %>%
    summarize(total = n())
```

```
## # A tibble: 18 x 3
## # Groups:   JobRole [9]
##    JobRole                   Attrition total
##    <fct>                     <fct>     <int>
##  1 Healthcare Representative No          122
##  2 Healthcare Representative Yes           9
##  3 Human Resources           No           40
##  4 Human Resources           Yes          12
##  5 Laboratory Technician     No          197
##  6 Laboratory Technician     Yes          62
##  7 Manager                   No           97
##  8 Manager                   Yes           5
##  9 Manufacturing Director    No          135
## 10 Manufacturing Director    Yes          10
## 11 Research Director         No           78
## 12 Research Director         Yes           2
## 13 Research Scientist        No          245
## 14 Research Scientist        Yes          47
## 15 Sales Executive           No          269
## 16 Sales Executive           Yes          57
## 17 Sales Representative      No           50
## 18 Sales Representative      Yes          33
```

```r
attrition_critical_features %>%
    group_by(JobRole, Attrition) %>%
    summarize(total = n()) %>%
    spread(key = Attrition, value = total) %>%
    mutate(pct_attrition = Yes / (Yes + No))
```

```
## # A tibble: 9 x 4
## # Groups:   JobRole [9]
##   JobRole                      No   Yes pct_attrition
##   <fct>                     <int> <int>         <dbl>
## 1 Healthcare Representative   122     9        0.0687
## 2 Human Resources              40    12        0.231
## 3 Laboratory Technician       197    62        0.239
## 4 Manager                      97     5        0.0490
## 5 Manufacturing Director      135    10        0.0690
## 6 Research Director            78     2        0.025
## 7 Research Scientist          245    47        0.161
## 8 Sales Executive             269    57        0.175
## 9 Sales Representative         50    33        0.398
```

```r
attrition_critical_features %>%
    group_by(JobRole, Attrition) %>%
    summarize(total = n()) %>%
    spread(key = Attrition, value = total) %>%
    mutate(pct_attrition = Yes / (Yes + No)) %>%
    ggplot(aes(x = forcats::fct_reorder(JobRole, pct_attrition), y = pct_attrition)) +
    geom_bar(stat = "identity", alpha = 1, fill = palette_light()[[1]]) +
    expand_limits(y = c(0, 1)) +
    coord_flip() +
    theme_tq() +
    labs(
        title = "Attrition Varies By Job Role",
        subtitle = "Sales Rep, Lab Tech, HR, Sales Exec, and Research Scientist
        have much higher turnover",
        y = "Attrition Percentage (Yes / Total)",
        x = "JobRole"
    )
```

30.13 CONCLUSIONS

There's a lot to take away from this article. We showed how you can use predictive analytics to develop sophisticated models that very accurately detect employees that are at risk of turnover. The autoML algorithm from H2O.ai worked well for classifying attrition with an accuracy around 87% on unseen / unmodeled data. We then used LIME to breakdown the complex ensemble model returned from H2O into critical features that are related to attrition. Overall, this is a really useful example where we can see how machine learning and data science can be used in business applications.