Algorithms Comparison

Alfonso R. Reyes

2019-09-18

Contents

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 1

Introduction to h2o. Bad Loans dataset. *(GLM, RF, GBM, DL, NB)*

Source: https://github.com/h2oai/h2o-tutorials/blob/master/h2o-open-tour-2016/chicago/intro-to-h2o.R

1.1 Introduction

Introductory H2O Machine Learning Tutorial Prepared for H2O Open Chicago 2016: http://open.h2o.ai/chicago.html

1.1.1 Install and download `h2o`

First step is to download & install the h2o R library The latest version is available by clicking on the R tab here: http://h2o-release.s3.amazonaws.com/h2o/latest_stable.html

Load the H2O library and start up the H2O cluster locally on your machine:

```
# Load the H2O library and start up the H2O cluster locally on your machine
library(h2o)
h2o.init(nthreads = -1, #Number of threads -1 means use all cores on your machine
         max_mem_size = "8G")  #max mem size is the maximum memory to allocate to H2O
#>
#> H2O is not running yet, starting it now...
#>
#> Note:  In case of errors look at the following log files:
#>     /tmp/RtmpRaWpdk/h2o_datascience_started_from_r.out
#>     /tmp/RtmpRaWpdk/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 192 milliseconds
#>     H2O cluster timezone:       America/Chicago
#>     H2O data parsing timezone:  UTC
#>     H2O cluster version:        3.22.1.1
```

```
#>     H2O cluster version age:    8 months and 20 days !!!
#>     H2O cluster name:           H2O_started_from_R_datascience_mwl453
#>     H2O cluster total nodes:    1
#>     H2O cluster total memory:   7.11 GB
#>     H2O cluster total cores:    8
#>     H2O cluster allowed cores:  8
#>     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 3.6.0 (2019-04-26)
```

1.1.2 Load the dataset

Next we will import a cleaned up version of the Lending Club "Bad Loans" dataset The purpose here is to predict whether a loan will be bad (not repaid to the lender). The response column, bad_loan, is 1 if the loan was bad, and 0 otherwise

Import the data `loan_csv <- "/Volumes/H2OTOUR/loan.csv"`
Alternatively, you can import the data directly from a URL

```
# modify this for your machine
loan_csv <- "https://raw.githubusercontent.com/h2oai/app-consumer-loan/master/data/loan.csv"
data <- h2o.importFile(loan_csv)  # 163,987 rows x 15 columns
#>
  |
  |                                                                      |   0%
  |
  |============                                                          |  19%
  |
  |========================                                              |  41%
  |
  |================================================                      |  75%
  |
  |========================================================              |  91%
  |
  |======================================================================| 100%
dim(data)
#> [1] 163987     15
# [1] 163987     15
```

```
url <- "https://raw.githubusercontent.com/h2oai/app-consumer-loan/master/data/loan.csv"
loans <- read.csv(url)
write.csv(loans, file = file.path(data_raw_dir, "loan.csv"))
```

1.1.3 Feature Engineering

Since we want to train a binary classification model, we must ensure that the response is coded as a factor If the response is 0/1, H2O will assume it's numeric, which means that H2O will train a regression model instead

```
data$bad_loan <- as.factor(data$bad_loan)  #encode the binary repsonse as a factor
h2o.levels(data$bad_loan)  #optional: after encoding, this shows the two factor levels, '0' and '1'
#> [1] "0" "1"
# [1] "0" "1"
```

1.1.4 Partition data

Partition the data into training, validation and test sets

```
# Partition the data into training, validation and test sets
splits <- h2o.splitFrame(data = data,
                         ratios = c(0.7, 0.15),  #partition data into 70%, 15%, 15% chunks
                         seed = 1)  #setting a seed will guarantee reproducibility
train <- splits[[1]]
valid <- splits[[2]]
test <- splits[[3]]
```

Take a look at the size of each partition Notice that h2o.splitFrame uses approximate splitting not exact splitting (for efficiency) so these are not exactly 70%, 15% and 15% of the total rows

```
nrow(train)  # 114908
#> [1] 114908
nrow(valid) # 24498
#> [1] 24498
nrow(test)  # 24581
#> [1] 24581
```

1.1.5 Identify response and predictor variables

```
# Identify response and predictor variables
y <- "bad_loan"
x <- setdiff(names(data), c(y, "int_rate"))  # remove the interest rate column because it's correlated
print(x)
#>  [1] "loan_amnt"             "term"
#>  [3] "emp_length"            "home_ownership"
#>  [5] "annual_inc"            "purpose"
#>  [7] "addr_state"            "dti"
#>  [9] "delinq_2yrs"           "revol_util"
#> [11] "total_acc"             "longest_credit_length"
#> [13] "verification_status"
# [1] "loan_amnt"             "term"
# [3] "emp_length"            "home_ownership"
# [5] "annual_inc"            "verification_status"
# [7] "purpose"               "addr_state"
# [9] "dti"                   "delinq_2yrs"
# [11] "revol_util"            "total_acc"
# [13] "longest_credit_length"
```

1.2 Algorithms

Now that we have prepared the data, we can train some models We will start by training a single model from each of the H2O supervised algos:

1. Generalized Linear Model (GLM)
2. Random Forest (RF)
3. Gradient Boosting Machine (GBM)
4. Deep Learning (DL)
5. Naive Bayes (NB)

1.3 GLM

Let's start with a basic binomial Generalized Linear Model By default, h2o.glm uses a regularized, elastic net model

```
glm_fit1 <- h2o.glm(x = x,
                    y = y,
                    training_frame = train,
                    model_id = "glm_fit1",
                    family = "binomial")  #similar to R's glm, h2o.glm has the family argument
#>
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

Next we will do some automatic tuning by passing in a validation frame and setting `lambda_search = True`. Since we are training a GLM with regularization, we should try to find the right amount of regularization (to avoid overfitting). The model parameter, `lambda`, controls the amount of regularization in a GLM model and we can find the optimal value for `lambda` automatically by setting `lambda_search = TRUE` and passing in a validation frame (which is used to evaluate model performance using a particular value of lambda).

```
glm_fit2 <- h2o.glm(x = x,
                    y = y,
                    training_frame = train,
                    model_id = "glm_fit2",
                    validation_frame = valid,
                    family = "binomial",
                    lambda_search = TRUE)
#>
  |
  |                                                                      |   0%
  |
  |============                                                          |  19%
  |
  |=========================                                             |  42%
  |
  |=====================================                                 |  63%
  |
  |======================================================================| 100%
```

Let's compare the performance of the two GLMs

```
# Let's compare the performance of the two GLMs
glm_perf1 <- h2o.performance(model = glm_fit1,
                             newdata = test)
glm_perf2 <- h2o.performance(model = glm_fit2,
                             newdata = test)

# Print model performance
glm_perf1
#> H2OBinomialMetrics: glm
#>
#> MSE:  0.142
#> RMSE:  0.377
#> LogLoss:  0.451
#> Mean Per-Class Error:  0.37
#> AUC:  0.677
#> pr_auc:  0.327
#> Gini:  0.355
#> R^2:  0.0639
#> Residual Deviance:  22176
#> AIC:  22280
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0    1    Error         Rate
#> 0      13647 6344 0.317343  =6344/19991
#> 1       1939 2651 0.422440   =1939/4590
#> Totals 15586 8995 0.336968  =8283/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                        max f1  0.193323 0.390283 222
#> 2                        max f2  0.118600 0.556655 307
#> 3                  max f0point5  0.276272 0.354086 146
#> 4                  max accuracy  0.494244 0.814410  29
#> 5                 max precision  0.744500 1.000000   0
#> 6                    max recall  0.001225 1.000000 399
#> 7                max specificity  0.744500 1.000000   0
#> 8               max absolute_mcc  0.198334 0.210606 216
#> 9   max min_per_class_accuracy  0.180070 0.627783 236
#> 10 max mean_per_class_accuracy  0.193323 0.630109 222
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
glm_perf2
#> H2OBinomialMetrics: glm
#>
#> MSE:  0.142
#> RMSE:  0.377
#> LogLoss:  0.451
#> Mean Per-Class Error:  0.372
#> AUC:  0.677
#> pr_auc:  0.326
#> Gini:  0.354
#> R^2:  0.0635
#> Residual Deviance:  22186
```

```
#> AIC:  22282
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0    1    Error         Rate
#> 0      13699 6292 0.314742  =6292/19991
#> 1       1968 2622 0.428758   =1968/4590
#> Totals 15667 8914 0.336032  =8260/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                       max f1  0.194171 0.388329 216
#> 2                       max f2  0.119200 0.555998 306
#> 3                 max f0point5  0.256488 0.351893 153
#> 4                 max accuracy  0.474001 0.814654  32
#> 5                max precision  0.736186 1.000000   0
#> 6                   max recall  0.001255 1.000000 399
#> 7              max specificity  0.736186 1.000000   0
#> 8             max absolute_mcc  0.198114 0.208337 212
#> 9   max min_per_class_accuracy  0.180131 0.625181 231
#> 10 max mean_per_class_accuracy  0.194171 0.628250 216
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

Instead of printing the entire model performance metrics object, it is probably easier to print just the metric that you are interested in comparing. Retreive test set AUC

```
h2o.auc(glm_perf1)  #0.677449084114
#> [1] 0.677
h2o.auc(glm_perf2)  #0.6776758582776
#> [1] 0.677
```

Compare test AUC to the training AUC and validation AUC

```
# Compare test AUC to the training AUC and validation AUC
h2o.auc(glm_fit2, train = TRUE)  #0.674306164325
#> [1] 0.673
h2o.auc(glm_fit2, valid = TRUE)  #0.6755122216705
#> [1] 0.675
glm_fit2@model$validation_metrics  #0.6755122216705
#> H2OBinomialMetrics: glm
#> ** Reported on validation data. **
#>
#> MSE:  0.142
#> RMSE:  0.377
#> LogLoss:  0.451
#> Mean Per-Class Error:  0.37
#> AUC:  0.675
#> pr_auc:  0.316
#> Gini:  0.351
#> R^2:  0.0597
#> Residual Deviance:  22101
#> AIC:  22197
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0    1    Error         Rate
```

```
#> 0      13591 6365 0.318952  =6365/19956
#> 1       1916 2626 0.421841   =1916/4542
#> Totals 15507 8991 0.338028  =8281/24498
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                       max f1  0.193519 0.388088 217
#> 2                       max f2  0.116436 0.555055 308
#> 3                 max f0point5  0.288405 0.343386 132
#> 4                 max accuracy  0.487882 0.815250  29
#> 5                max precision  0.576333 0.681818   9
#> 6                   max recall  0.004789 1.000000 398
#> 7              max specificity  0.715719 0.999950   0
#> 8             max absolute_mcc  0.195417 0.209494 215
#> 9   max min_per_class_accuracy  0.180760 0.627731 230
#> 10 max mean_per_class_accuracy  0.192639 0.629672 218
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

1.4 Random Forest

H2O's Random Forest (RF) implements a distributed version of the standard Random Forest algorithm and variable importance measures. First we will train a basic Random Forest model with default parameters. The Random Forest model will infer the response distribution from the response encoding. A seed is required for reproducibility.

```
rf_fit1 <- h2o.randomForest(x = x,
                            y = y,
                            training_frame = train,
                            model_id = "rf_fit1",
                            seed = 1)
#>
  |
  |                                                                      |   0%
  |
  |=                                                                     |   2%
  |
  |=====                                                                 |   8%
  |
  |==========                                                            |  16%
  |
  |===============                                                       |  24%
  |
  |=======================                                               |  36%
  |
  |==============================                                        |  48%
  |
  |======================================                                |  60%
  |
  |=============================================                         |  72%
  |
  |====================================================                  |  84%
```

```
  |
  |============================================================    |  96%
  |
  |=============================================================| 100%
```

Next we will increase the number of trees used in the forest by setting `ntrees = 100`. The default number of trees in an H2O Random Forest is 50, so this RF will be twice as big as the default. Usually increasing the number of trees in a RF will increase performance as well. Unlike Gradient Boosting Machines (GBMs), Random Forests are fairly resistant (although not free from) overfitting. See the GBM example below for additional guidance on preventing overfitting using H2O's early stopping functionality.

```
rf_fit2 <- h2o.randomForest(x = x,
                            y = y,
                            training_frame = train,
                            model_id = "rf_fit2",
                            #validation_frame = valid,  #only used if stopping_rounds > 0
                            ntrees = 100,
                            seed = 1)
#>
  |
  |                                                              |   0%
  |
  |===                                                           |   4%
  |
  |=====                                                         |   8%
  |
  |========                                                      |  13%
  |
  |===========                                                   |  17%
  |
  |==============                                                |  22%
  |
  |==================                                            |  27%
  |
  |====================                                          |  31%
  |
  |=======================                                       |  35%
  |
  |===========================                                   |  40%
  |
  |===============================                               |  45%
  |
  |===================================                           |  51%
  |
  |=======================================                       |  57%
  |
  |===========================================                   |  62%
  |
  |===============================================               |  68%
  |
  |===================================================           |  74%
  |
  |=======================================================       |  79%
  |
```

```
  |=============================================================          |  85%
  |
  |================================================================       |  91%
  |
  |===================================================================    |  96%
  |
  |=======================================================================| 100%
```

Let's compare the performance of the two RFs

```
# Let's compare the performance of the two RFs
rf_perf1 <- h2o.performance(model = rf_fit1,
                            newdata = test)
rf_perf2 <- h2o.performance(model = rf_fit2,
                            newdata = test)

# Print model performance
rf_perf1
#> H2OBinomialMetrics: drf
#>
#> MSE:  0.144
#> RMSE:  0.379
#> LogLoss:  0.459
#> Mean Per-Class Error:  0.379
#> AUC:  0.663
#> pr_auc:  0.311
#> Gini:  0.327
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0    1    Error         Rate
#> 0      12842 7149 0.357611  =7149/19991
#> 1       1839 2751 0.400654   =1839/4590
#> Totals 14681 9900 0.365648  =8988/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                       max f1  0.193397 0.379710 229
#> 2                       max f2  0.077797 0.547982 344
#> 3                 max f0point5  0.277524 0.344251 158
#> 4                 max accuracy  0.543639 0.813799  29
#> 5                max precision  0.817454 1.000000   0
#> 6                   max recall  0.001181 1.000000 399
#> 7              max specificity  0.817454 1.000000   0
#> 8             max absolute_mcc  0.252480 0.195090 178
#> 9   max min_per_class_accuracy  0.186549 0.619826 235
#> 10 max mean_per_class_accuracy  0.192603 0.620890 230
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
rf_perf2
#> H2OBinomialMetrics: drf
#>
#> MSE:  0.143
#> RMSE:  0.378
#> LogLoss:  0.454
```

```
#> Mean Per-Class Error:  0.377
#> AUC:  0.669
#> pr_auc:  0.32
#> Gini:  0.339
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0    1    Error         Rate
#> 0      13172 6819 0.341103  =6819/19991
#> 1       1891 2699 0.411983   =1891/4590
#> Totals 15063 9518 0.354339  =8710/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                       max f1  0.196407 0.382620 225
#> 2                       max f2  0.092270 0.549691 331
#> 3                 max f0point5  0.291396 0.349342 144
#> 4                 max accuracy  0.555908 0.813840  20
#> 5                max precision  0.651522 0.785714   5
#> 6                   max recall  0.004212 1.000000 398
#> 7              max specificity  0.711667 0.999950   0
#> 8             max absolute_mcc  0.229251 0.204236 194
#> 9   max min_per_class_accuracy  0.184594 0.619829 235
#> 10 max mean_per_class_accuracy  0.196407 0.623457 225
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/

# Retreive test set AUC
h2o.auc(rf_perf1)  # 0.662266990734
#> [1] 0.663
h2o.auc(rf_perf2)  # 0.66525468051
#> [1] 0.669
```

1.4.1 Cross-validate performance

Rather than using held-out test set to evaluate model performance, a user may wish to estimate model performance using cross-validation. Using the RF algorithm (with default model parameters) as an example, we demonstrate how to perform k-fold cross-validation using H2O. No custom code or loops are required, you simply specify the number of desired folds in the nfolds argument. Since we are not going to use a test set here, we can use the original (full) dataset, which we called data rather than the subsampled `train` dataset. Note that this will take approximately k (nfolds) times longer than training a single RF model, since it will train k models in the cross-validation process (trained on n(k-1)/k rows), in addition to the final model trained on the full training_frame dataset with n rows.

```
rf_fit3 <- h2o.randomForest(x = x,
                            y = y,
                            training_frame = train,
                            model_id = "rf_fit3",
                            seed = 1,
                            nfolds = 5)
#>
  |
  |                                                                      |   0%
  |
```

```
|=                                                    |   1%
|
|==                                                   |   2%
|
|==                                                   |   3%
|
|===                                                  |   5%
|
|====                                                 |   6%
|
|=====                                                |   8%
|
|======                                               |   9%
|
|=======                                              |  11%
|
|========                                             |  13%
|
|=========                                            |  14%
|
|===========                                          |  16%
|
|===========                                          |  17%
|
|============                                         |  18%
|
|============                                         |  19%
|
|=============                                        |  20%
|
|==============                                       |  21%
|
|===============                                      |  23%
|
|================                                     |  25%
|
|=================                                    |  26%
|
|==================                                   |  27%
|
|==================                                   |  28%
|
|===================                                  |  29%
|
|====================                                 |  31%
|
|=====================                                |  32%
|
|=====================                                |  33%
|
|======================                               |  34%
|
|=======================                              |  35%
```

```
|
|=======================                                         |  36%
|
|=======================                                         |  37%
|
|========================                                        |  39%
|
|=========================                                       |  41%
|
|==========================                                      |  43%
|
|==========================                                      |  45%
|
|===========================                                     |  46%
|
|===========================                                     |  48%
|
|============================                                    |  50%
|
|============================                                    |  51%
|
|=============================                                   |  52%
|
|=============================                                   |  53%
|
|=============================                                   |  54%
|
|==============================                                  |  56%
|
|==============================                                  |  58%
|
|===============================                                 |  59%
|
|===============================                                 |  61%
|
|================================                                |  62%
|
|================================                                |  64%
|
|=================================                               |  65%
|
|=================================                               |  67%
|
|==================================                              |  68%
|
|==================================                              |  69%
|
|===================================                             |  70%
|
|====================================                            |  72%
|
|====================================                            |  73%
|
```

```
  |=============================================                    |  74%
  |
  |==============================================                   |  76%
  |
  |===============================================                  |  77%
  |
  |================================================                 |  78%
  |
  |=================================================                |  80%
  |
  |==================================================               |  82%
  |
  |==================================================               |  82%
  |
  |==================================================               |  83%
  |
  |===================================================              |  85%
  |
  |====================================================             |  86%
  |
  |=====================================================            |  88%
  |
  |======================================================           |  89%
  |
  |======================================================           |  90%
  |
  |=======================================================          |  92%
  |
  |=======================================================          |  93%
  |
  |========================================================         |  95%
  |
  |========================================================         |  96%
  |
  |=========================================================        |  98%
  |
  |=================================================================| 100%
```

To evaluate the cross-validated AUC, do the following:

```
# To evaluate the cross-validated AUC, do the following:
h2o.auc(rf_fit3, xval = TRUE)  # 0.661201482614
#> [1] 0.659
```

1.5 Gradient Boosting Machine (GBM)

H2O's Gradient Boosting Machine (GBM) offers a Stochastic GBM, which can increase performance quite a bit compared to the original GBM implementation.

Now we will train a basic GBM model The GBM model will infer the response distribution from the response encoding if not specified explicitly through the `distribution` argument. A seed is required for reproducibility.

```
gbm_fit1 <- h2o.gbm(x = x,
                    y = y,
                    training_frame = train,
                    model_id = "gbm_fit1",
                    seed = 1)
#>
  |
  |                                                                      |   0%
  |
  |=========                                                             |  14%
  |
  |=====================                                                 |  34%
  |
  |==================================                                    |  54%
  |
  |================================================                      |  78%
  |
  |======================================================================| 100%
```

Next we will increase the number of trees used in the GBM by setting `ntrees=500`. The default number of trees in an H2O GBM is 50, so this GBM will trained using ten times the default. Increasing the number of trees in a GBM is one way to increase performance of the model, however, you have to be careful not to overfit your model to the training data by using too many trees. To automatically find the optimal number of trees, you must use H2O's early stopping functionality. This example will not do that, however, the following # example will.

```
gbm_fit2 <- h2o.gbm(x = x,
                    y = y,
                    training_frame = train,
                    model_id = "gbm_fit2",
                    #validation_frame = valid,  #only used if stopping_rounds > 0
                    ntrees = 500,
                    seed = 1)
#>
  |
  |                                                                      |   0%
  |
  |=                                                                     |   2%
  |
  |===                                                                   |   4%
  |
  |====                                                                  |   6%
  |
  |=====                                                                 |   8%
  |
  |=======                                                               |  11%
  |
  |==========                                                            |  15%
  |
  |=============                                                         |  20%
  |
  |================                                                      |  24%
  |
  |=====================                                                 |  34%
```

```
  |
  |================================                                 |  51%
  |
  |============================================                     |  69%
  |
  |========================================================         |  84%
  |
  |================================================================ |  98%
  |
  |=================================================================| 100%
```

We will again set `ntrees = 500`, however, this time we will use early stopping in order to prevent overfitting (from too many trees). All of H2O's algorithms have early stopping available, however early stopping is not enabled by default (with the exception of Deep Learning). There are several parameters that should be used to control early stopping. The three that are common to all the algorithms are: `stopping_rounds`, `stopping_metric` and `stopping_tolerance`. The stopping metric is the metric by which you'd like to measure performance, and so we will choose AUC here. The `score_tree_interval` is a parameter specific to the Random Forest model and the GBM. Setting `score_tree_interval = 5` will score the model after every five trees. The parameters we have set below specify that the model will stop training after there have been three scoring intervals where the AUC has not increased more than 0.0005. Since we have specified a validation frame, the stopping tolerance will be computed on validation AUC rather than training AUC.

```
gbm_fit3 <- h2o.gbm(x = x,
                    y = y,
                    training_frame = train,
                    model_id = "gbm_fit3",
                    validation_frame = valid,  #only used if stopping_rounds > 0
                    ntrees = 500,
                    score_tree_interval = 5,      #used for early stopping
                    stopping_rounds = 3,          #used for early stopping
                    stopping_metric = "AUC",      #used for early stopping
                    stopping_tolerance = 0.0005,  #used for early stopping
                    seed = 1)
#>
  |
  |                                                                 |   0%
  |
  |==                                                               |   2%
  |
  |===                                                              |   4%
  |
  |=====                                                            |   7%
  |
  |======                                                           |  10%
  |
  |========                                                         |  13%
  |
  |==========                                                       |  16%
  |
  |============                                                     |  19%
  |
  |=================================================================| 100%
```

Let's compare the performance of the two GBMs

```
# Let's compare the performance of the two GBMs
gbm_perf1 <- h2o.performance(model = gbm_fit1,
                             newdata = test)
gbm_perf2 <- h2o.performance(model = gbm_fit2,
                             newdata = test)
gbm_perf3 <- h2o.performance(model = gbm_fit3,
                             newdata = test)

# Print model performance
gbm_perf1
#> H2OBinomialMetrics: gbm
#>
#> MSE:  0.141
#> RMSE:  0.376
#> LogLoss:  0.448
#> Mean Per-Class Error:  0.367
#> AUC:  0.684
#> pr_auc:  0.332
#> Gini:  0.368
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0     1    Error         Rate
#> 0      12143  7848 0.392577  =7848/19991
#> 1       1571  3019 0.342266   =1571/4590
#> Totals 13714 10867 0.383182  =9419/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                       max f1  0.171139 0.390632 251
#> 2                       max f2  0.108885 0.560103 328
#> 3                 max f0point5  0.285149 0.356430 145
#> 4                 max accuracy  0.510077 0.814410  27
#> 5                max precision  0.601699 0.636364   8
#> 6                   max recall  0.037543 1.000000 398
#> 7              max specificity  0.719189 0.999950   0
#> 8             max absolute_mcc  0.220471 0.215401 199
#> 9   max min_per_class_accuracy  0.176689 0.628540 244
#> 10 max mean_per_class_accuracy  0.170225 0.632624 252
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
gbm_perf2
#> H2OBinomialMetrics: gbm
#>
#> MSE:  0.142
#> RMSE:  0.376
#> LogLoss:  0.449
#> Mean Per-Class Error:  0.367
#> AUC:  0.684
#> pr_auc:  0.329
#> Gini:  0.368
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0    1    Error         Rate
```

```
#> 0      13661 6330 0.316642  =6330/19991
#> 1       1918 2672 0.417865   =1918/4590
#> Totals 15579 9002 0.335544  =8248/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                       max f1  0.189615 0.393172 234
#> 2                       max f2  0.096969 0.558918 333
#> 3                 max f0point5  0.278776 0.359560 160
#> 4                 max accuracy  0.521287 0.814287  37
#> 5                max precision  0.901295 1.000000   0
#> 6                   max recall  0.018504 1.000000 398
#> 7              max specificity  0.901295 1.000000   0
#> 8             max absolute_mcc  0.231089 0.217255 196
#> 9   max min_per_class_accuracy  0.174914 0.630834 249
#> 10 max mean_per_class_accuracy  0.156944 0.633792 267
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
gbm_perf3
#> H2OBinomialMetrics: gbm
#>
#> MSE:  0.141
#> RMSE:  0.376
#> LogLoss:  0.448
#> Mean Per-Class Error:  0.367
#> AUC:  0.684
#> pr_auc:  0.331
#> Gini:  0.369
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0    1    Error         Rate
#> 0      13652 6339 0.317093  =6339/19991
#> 1       1917 2673 0.417647   =1917/4590
#> Totals 15569 9012 0.335869  =8256/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                       max f1  0.189870 0.393030 234
#> 2                       max f2  0.109837 0.559580 321
#> 3                 max f0point5  0.294571 0.356508 148
#> 4                 max accuracy  0.510869 0.814572  40
#> 5                max precision  0.797620 1.000000   0
#> 6                   max recall  0.026373 1.000000 397
#> 7              max specificity  0.797620 1.000000   0
#> 8             max absolute_mcc  0.231530 0.218255 197
#> 9   max min_per_class_accuracy  0.176566 0.631808 248
#> 10 max mean_per_class_accuracy  0.175468 0.633400 249
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/

# Retreive test set AUC
h2o.auc(gbm_perf1)  # 0.682765594191
#> [1] 0.684
```

```
h2o.auc(gbm_perf2)  # 0.671854616713
#> [1] 0.684
h2o.auc(gbm_perf3)  # 0.68309902855
#> [1] 0.684
```

To examine the scoring history, use the `scoring_history` method on a trained model. If `score_tree_interval` is not specified, it will score at various intervals, as we can see for `h2o.scoreHistory()` below. However, regular 5-tree intervals are used for `h2o.scoreHistory()`. The `gbm_fit2` was trained only using a training set (no validation set), so the scoring history is calculated for training set performance metrics only.

```
h2o.scoreHistory(gbm_fit2)
#> Scoring History:
#>             timestamp   duration number_of_trees training_rmse
#> 1 2019-09-18 13:26:39  0.009 sec               0       0.38563
#> 2 2019-09-18 13:26:39  0.156 sec               1       0.38370
#> 3 2019-09-18 13:26:39  0.207 sec               2       0.38206
#> 4 2019-09-18 13:26:39  0.277 sec               3       0.38069
#> 5 2019-09-18 13:26:39  0.357 sec               4       0.37954
#>   training_logloss training_auc training_pr_auc training_lift
#> 1          0.47403      0.50000         0.00000       1.00000
#> 2          0.46913      0.65779         0.30174       2.68330
#> 3          0.46512      0.66583         0.31164       2.79399
#> 4          0.46184      0.66851         0.31500       2.97100
#> 5          0.45912      0.67011         0.31822       2.97544
#>   training_classification_error
#> 1                       0.81825
#> 2                       0.40069
#> 3                       0.33325
#> 4                       0.34475
#> 5                       0.33180
#>
#> ---
#>              timestamp   duration number_of_trees training_rmse
#> 37 2019-09-18 13:26:42  3.660 sec              36       0.36864
#> 38 2019-09-18 13:26:43  3.825 sec              37       0.36849
#> 39 2019-09-18 13:26:43  3.951 sec              38       0.36834
#> 40 2019-09-18 13:26:47  7.990 sec             116       0.36240
#> 41 2019-09-18 13:26:51 12.012 sec             404       0.36129
#> 42 2019-09-18 13:26:52 13.455 sec             500       0.36129
#>    training_logloss training_auc training_pr_auc training_lift
#> 37          0.43381      0.70801         0.36444       3.55969
#> 38          0.43345      0.70872         0.36547       3.58361
#> 39          0.43311      0.70955         0.36656       3.56447
#> 40          0.42008      0.73512         0.41026       4.12426
#> 41          0.41770      0.74024         0.41866       4.25823
#> 42          0.41770      0.74024         0.41866       4.25823
#>    training_classification_error
#> 37                       0.29517
#> 38                       0.29555
#> 39                       0.29962
#> 40                       0.27824
#> 41                       0.25719
#> 42                       0.25719
```

When early stopping is used, we see that training stopped at 105 trees instead of the full 500. Since we used a validation set in `gbm_fit3`, both training and validation performance metrics are stored in the scoring history object. Take a look at the validation AUC to observe that the correct stopping tolerance was enforced.

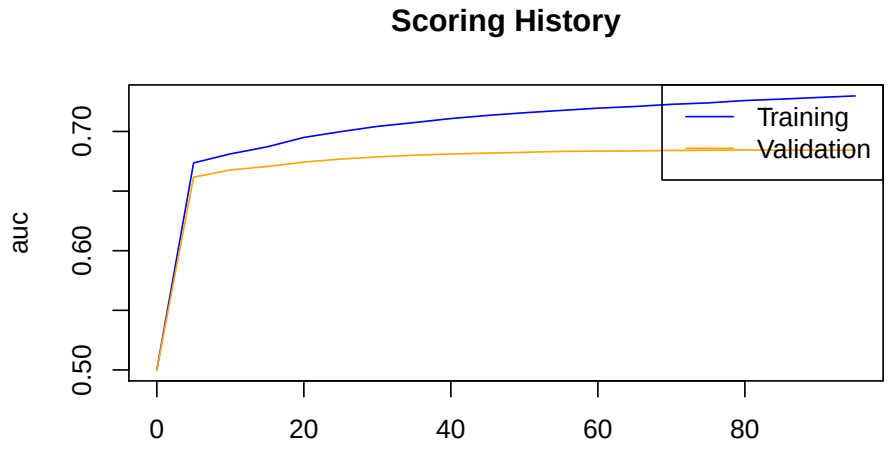
```
h2o.scoreHistory(gbm_fit3)
#> Scoring History:
#>              timestamp   duration number_of_trees training_rmse
#> 1  2019-09-18 13:26:53  0.005 sec               0       0.38563
#> 2  2019-09-18 13:26:54  0.435 sec               5       0.37853
#> 3  2019-09-18 13:26:54  0.840 sec              10       0.37508
#> 4  2019-09-18 13:26:55  1.215 sec              15       0.37307
#> 5  2019-09-18 13:26:55  1.803 sec              20       0.37152
#> 6  2019-09-18 13:26:56  2.441 sec              25       0.37041
#> 7  2019-09-18 13:26:56  2.751 sec              30       0.36947
#> 8  2019-09-18 13:26:57  3.107 sec              35       0.36877
#> 9  2019-09-18 13:26:57  3.415 sec              40       0.36808
#> 10 2019-09-18 13:26:57  3.787 sec              45       0.36748
#> 11 2019-09-18 13:26:58  4.157 sec              50       0.36699
#> 12 2019-09-18 13:26:58  4.503 sec              55       0.36651
#> 13 2019-09-18 13:26:58  4.839 sec              60       0.36607
#> 14 2019-09-18 13:26:59  5.159 sec              65       0.36574
#> 15 2019-09-18 13:26:59  5.501 sec              70       0.36531
#> 16 2019-09-18 13:26:59  5.794 sec              75       0.36503
#> 17 2019-09-18 13:27:00  6.243 sec              80       0.36460
#> 18 2019-09-18 13:27:00  6.537 sec              85       0.36426
#> 19 2019-09-18 13:27:00  6.839 sec              90       0.36394
#> 20 2019-09-18 13:27:01  7.128 sec              95       0.36362
#>    training_logloss training_auc training_pr_auc training_lift
#> 1           0.47403      0.50000         0.00000       1.00000
#> 2           0.45676      0.67362         0.32194       3.04518
#> 3           0.44884      0.68117         0.33330       3.19128
#> 4           0.44424      0.68708         0.34105       3.30132
#> 5           0.44060      0.69498         0.34930       3.46878
#> 6           0.43800      0.69983         0.35479       3.47356
#> 7           0.43578      0.70425         0.36014       3.48792
#> 8           0.43410      0.70746         0.36373       3.53576
#> 9           0.43252      0.71082         0.36837       3.61710
#> 10          0.43116      0.71346         0.37174       3.66495
#> 11          0.43002      0.71568         0.37493       3.66016
#> 12          0.42899      0.71765         0.37852       3.70801
#> 13          0.42801      0.71953         0.38157       3.73193
#> 14          0.42726      0.72093         0.38411       3.78934
#> 15          0.42632      0.72277         0.38733       3.82762
#> 16          0.42572      0.72396         0.38955       3.85154
#> 17          0.42477      0.72596         0.39282       3.95202
#> 18          0.42407      0.72713         0.39550       3.99508
#> 19          0.42338      0.72852         0.39819       4.02857
#> 20          0.42272      0.72984         0.40069       4.02378
#>    training_classification_error validation_rmse validation_logloss
#> 1                        0.81825         0.38864            0.47953
#> 2                        0.32117         0.38233            0.46398
#> 3                        0.32202         0.37958            0.45742
#> 4                        0.32027         0.37828            0.45428
#> 5                        0.33371         0.37739            0.45210
```

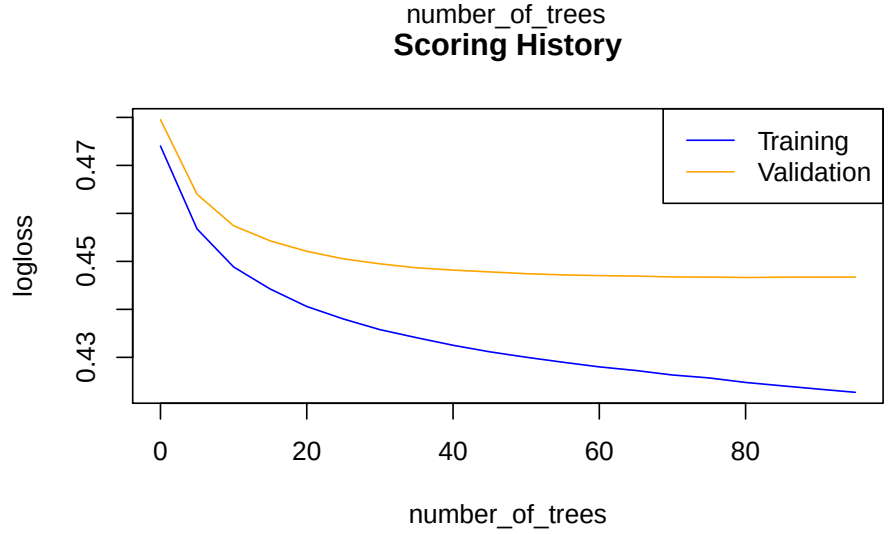
```
#> 6                       0.32537         0.37676          0.45053
#> 7                       0.29722         0.37636          0.44949
#> 8                       0.29544         0.37604          0.44866
#> 9                       0.28871         0.37587          0.44818
#> 10                      0.30181         0.37574          0.44781
#> 11                      0.30179         0.37560          0.44744
#> 12                      0.29464         0.37552          0.44718
#> 13                      0.30343         0.37547          0.44703
#> 14                      0.28692         0.37543          0.44694
#> 15                      0.28579         0.37536          0.44676
#> 16                      0.26903         0.37536          0.44673
#> 17                      0.28476         0.37534          0.44664
#> 18                      0.26950         0.37537          0.44671
#> 19                      0.27036         0.37538          0.44671
#> 20                      0.26573         0.37540          0.44673
#>    validation_auc validation_pr_auc validation_lift
#> 1         0.50000           0.00000         1.00000
#> 2         0.66168           0.30421         2.75098
#> 3         0.66767           0.30959         2.68582
#> 4         0.67061           0.31287         2.70784
#> 5         0.67426           0.31757         2.79590
#> 6         0.67685           0.32121         2.99403
#> 7         0.67866           0.32312         2.97202
#> 8         0.68006           0.32477         3.03806
#> 9         0.68113           0.32493         2.90597
#> 10        0.68183           0.32549         2.88396
#> 11        0.68251           0.32659         2.86194
#> 12        0.68326           0.32626         2.86194
#> 13        0.68354           0.32686         2.88396
#> 14        0.68364           0.32705         2.83993
#> 15        0.68405           0.32754         2.97202
#> 16        0.68423           0.32747         2.88396
#> 17        0.68454           0.32735         2.86194
#> 18        0.68431           0.32727         2.83993
#> 19        0.68434           0.32718         2.95000
#> 20        0.68436           0.32678         2.92799
#>    validation_classification_error
#> 1                          0.81460
#> 2                          0.35387
#> 3                          0.35285
#> 4                          0.39028
#> 5                          0.36770
#> 6                          0.35240
#> 7                          0.34848
#> 8                          0.34386
#> 9                          0.34807
#> 10                         0.38681
#> 11                         0.33774
#> 12                         0.34215
#> 13                         0.34431
#> 14                         0.34146
#> 15                         0.34329
#> 16                         0.33872
```

```
#> 17                          0.34276
#> 18                          0.34256
#> 19                          0.34289
#> 20                          0.35097
```

Look at scoring history for third GBM model

```
# Look at scoring history for third GBM model
plot(gbm_fit3,
     timestep = "number_of_trees",
     metric = "AUC")
plot(gbm_fit3,
     timestep = "number_of_trees",
     metric = "logloss")
```





1.6 Deep Learning

H2O's Deep Learning algorithm is a multilayer feed-forward artificial neural network. It can also be used to train an autoencoder. In this example we will train a standard supervised prediction model.

1.6.1 Train a default DL

First we will train a basic DL model with default parameters. The DL model will infer the response distribution from the response encoding if it is not specified explicitly through the `distribution` argument. H2O's DL will not be reproducible if it is run on more than a single core, so in this example, the performance metrics below may vary slightly from what you see on your machine. In H2O's DL, early stopping is enabled by default, so below, it will use the training set and default stopping parameters to perform early stopping.

```
dl_fit1 <- h2o.deeplearning(x = x,
                            y = y,
                            training_frame = train,
                            model_id = "dl_fit1",
                            seed = 1)
#>
  |
  |                                                                      |   0%
  |
  |=====                                                                 |   7%
  |
  |=========                                                             |  14%
  |
  |==============                                                        |  22%
  |
  |====================                                                  |  29%
  |
  |=======================                                               |  36%
  |
  |===========================                                           |  43%
  |
  |================================                                      |  50%
  |
  |====================================                                  |  58%
  |
  |========================================                              |  65%
  |
  |============================================                          |  72%
  |
  |================================================                      |  79%
  |
  |========================================================              |  87%
  |
  |============================================================          |  94%
  |
  |======================================================================| 100%
```

1.6.2 Train a DL with new architecture and more epochs.

Next we will increase the number of epochs used in the GBM by setting `epochs=20` (the default is 10). Increasing the number of epochs in a deep neural net may increase performance of the model, however, you have to be careful not to overfit your model to your training data. To automatically find the optimal number of epochs, you must use H2O's early stopping functionality. Unlike the rest of the H2O algorithms, H2O's DL will use early stopping by default, so for comparison we will first turn off early stopping. We do this in the next example by setting `stopping_rounds=0`.

```
dl_fit2 <- h2o.deeplearning(x = x,
                            y = y,
                            training_frame = train,
                            model_id = "dl_fit2",
                            #validation_frame = valid,  #only used if stopping_rounds > 0
                            epochs = 20,
                            hidden= c(10,10),
                            stopping_rounds = 0,  # disable early stopping
                            seed = 1)
#>
  |
  |                                                                      |   0%
  |
  |===========                                                           |  17%
  |
  |======================                                                |  35%
  |
  |==================================                                    |  57%
  |
  |==============================================                        |  83%
  |
  |======================================================================| 100%
```

1.6.3 Train a DL with early stopping

This example will use the same model parameters as `dl_fit2`. This time, we will turn on early stopping and specify the stopping criterion. We will also pass a validation set, as is recommended for early stopping.

```
dl_fit3 <- h2o.deeplearning(x = x,
                            y = y,
                            training_frame = train,
                            model_id = "dl_fit3",
                            validation_frame = valid,  #in DL, early stopping is on by default
                            epochs = 20,
                            hidden = c(10,10),
                            score_interval = 1,           #used for early stopping
                            stopping_rounds = 3,          #used for early stopping
                            stopping_metric = "AUC",      #used for early stopping
                            stopping_tolerance = 0.0005,  #used for early stopping
                            seed = 1)
#>
  |
  |                                                                      |   0%
  |
  |========                                                              |  13%
  |
  |======================                                                |  35%
  |
  |=======================================                               |  61%
  |
  |==============================================                        |  83%
  |
  |======================================================================| 100%
```

Let's compare the performance of the three DL models

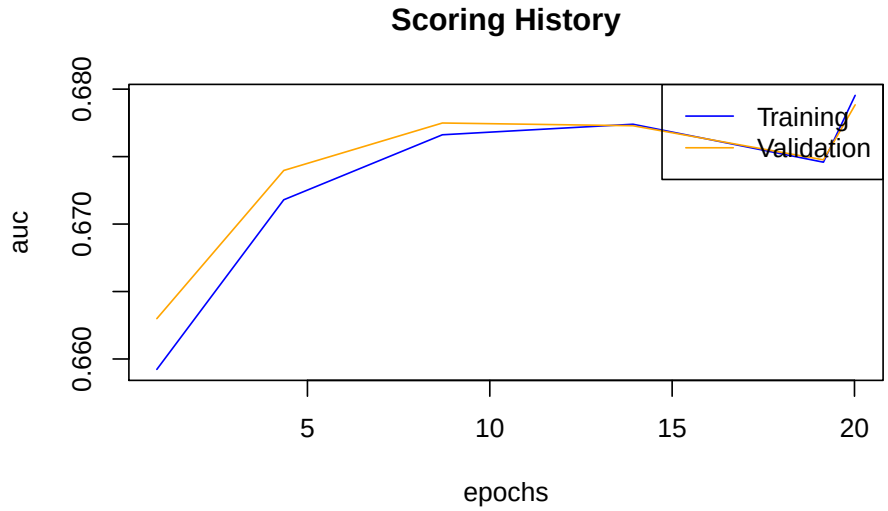
```
# Let's compare the performance of the three DL models
dl_perf1 <- h2o.performance(model = dl_fit1,
                            newdata = test)
dl_perf2 <- h2o.performance(model = dl_fit2,
                            newdata = test)
dl_perf3 <- h2o.performance(model = dl_fit3,
                            newdata = test)

# Print model performance
dl_perf1
#> H2OBinomialMetrics: deeplearning
#>
#> MSE:  0.142
#> RMSE:  0.377
#> LogLoss:  0.453
#> Mean Per-Class Error:  0.368
#> AUC:  0.68
#> pr_auc:  0.328
#> Gini:  0.361
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0     1    Error         Rate
#> 0      12340  7651 0.382722  =7651/19991
#> 1       1619  2971 0.352723   =1619/4590
#> Totals 13959 10622 0.377121  =9270/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                       max f1  0.160846 0.390613 242
#> 2                       max f2  0.089911 0.557712 319
#> 3                 max f0point5  0.260368 0.358963 155
#> 4                 max accuracy  0.451630 0.814165  48
#> 5                max precision  0.916585 1.000000   0
#> 6                   max recall  0.001856 1.000000 399
#> 7              max specificity  0.916585 1.000000   0
#> 8             max absolute_mcc  0.203948 0.214742 204
#> 9   max min_per_class_accuracy  0.164615 0.629194 238
#> 10 max mean_per_class_accuracy  0.160846 0.632277 242
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
dl_perf2
#> H2OBinomialMetrics: deeplearning
#>
#> MSE:  0.142
#> RMSE:  0.377
#> LogLoss:  0.451
#> Mean Per-Class Error:  0.371
#> AUC:  0.678
#> pr_auc:  0.323
#> Gini:  0.356
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
```

```
#>            0     1    Error         Rate
#> 0      11963  8028 0.401581  =8028/19991
#> 1       1563  3027 0.340523   =1563/4590
#> Totals 13526 11055 0.390179  =9591/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                        max f1  0.174451 0.386961 239
#> 2                        max f2  0.108823 0.557393 311
#> 3                  max f0point5  0.264272 0.354205 156
#> 4                  max accuracy  0.496031 0.813799  29
#> 5                 max precision  0.630382 0.714286   5
#> 6                    max recall  0.008995 1.000000 397
#> 7                max specificity  0.703701 0.999950   0
#> 8               max absolute_mcc  0.223873 0.207801 190
#> 9   max min_per_class_accuracy  0.182983 0.626362 230
#> 10 max mean_per_class_accuracy  0.174451 0.628948 239
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
dl_perf3
#> H2OBinomialMetrics: deeplearning
#>
#> MSE:  0.142
#> RMSE:  0.377
#> LogLoss:  0.45
#> Mean Per-Class Error:  0.369
#> AUC:  0.68
#> pr_auc:  0.331
#> Gini:  0.36
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0    1    Error         Rate
#> 0      13127 6864 0.343355  =6864/19991
#> 1       1808 2782 0.393900   =1808/4590
#> Totals 14935 9646 0.352793  =8672/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                        max f1  0.213309 0.390840 211
#> 2                        max f2  0.115230 0.556916 307
#> 3                  max f0point5  0.315823 0.355585 130
#> 4                  max accuracy  0.548601 0.814694  25
#> 5                 max precision  0.669774 0.750000   2
#> 6                    max recall  0.005888 1.000000 398
#> 7                max specificity  0.762541 0.999950   0
#> 8               max absolute_mcc  0.228065 0.210358 199
#> 9   max min_per_class_accuracy  0.204577 0.628433 218
#> 10 max mean_per_class_accuracy  0.213309 0.631373 211
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/

# Retreive test set AUC
h2o.auc(dl_perf1)  # 0.6774335
```

```
#> [1] 0.68
h2o.auc(dl_perf2)  # 0.678446
#> [1] 0.678
h2o.auc(dl_perf3)  # 0.6770498
#> [1] 0.68
```

Look at scoring history for third DL model

```
# Look at scoring history for third DL model
plot(dl_fit3,
     timestep = "epochs",
     metric = "AUC")
```



1.6.4 Scoring history

```
# Scoring history
h2o.scoreHistory(dl_fit3)
#> Scoring History:
#>             timestamp   duration training_speed   epochs iterations
#> 1 2019-09-18 13:27:52  0.000 sec              NA  0.00000          0
#> 2 2019-09-18 13:27:52  0.481 sec 270981 obs/sec  0.87019          1
#> 3 2019-09-18 13:27:53  1.515 sec 379657 obs/sec  4.35139          5
#> 4 2019-09-18 13:27:54  2.520 sec 446895 obs/sec  8.70006         10
#> 5 2019-09-18 13:27:55  3.609 sec 493255 obs/sec 13.92523         16
#> 6 2019-09-18 13:27:56  4.640 sec 525771 obs/sec 19.14884         22
#> 7 2019-09-18 13:27:56  4.882 sec 531737 obs/sec 20.01859         23
#>          samples training_rmse training_logloss training_r2 training_auc
#> 1       0.000000            NA               NA          NA           NA
#> 2   99992.000000       0.38629          0.48694     0.01270      0.65923
#> 3  500009.000000       0.37744          0.45198     0.05740      0.67180
#> 4  999706.000000       0.37667          0.45021     0.06126      0.67662
#> 5 1600120.000000       0.37650          0.44937     0.06212      0.67740
#> 6 2200355.000000       0.37782          0.45380     0.05550      0.67459
#> 7 2300296.000000       0.37637          0.44942     0.06276      0.67954
#>   training_pr_auc training_lift training_classification_error
#> 1              NA            NA                           NA
#> 2         0.30388       2.66706                      0.36701
```

```
#> 3         0.31138       2.39491                       0.35558
#> 4         0.32077       2.83035                       0.35528
#> 5         0.32044       2.72149                       0.34122
#> 6         0.31749       3.04807                       0.41970
#> 7         0.32694       3.04807                       0.36863
#>   validation_rmse validation_logloss validation_r2 validation_auc
#> 1              NA                NA            NA             NA
#> 2         0.38590           0.48541       0.01399        0.66299
#> 3         0.37698           0.45096       0.05904        0.67397
#> 4         0.37650           0.45014       0.06143        0.67749
#> 5         0.37640           0.44951       0.06191        0.67729
#> 6         0.37780           0.45375       0.05493        0.67476
#> 7         0.37644           0.44997       0.06172        0.67884
#>   validation_pr_auc validation_lift validation_classification_error
#> 1                NA              NA                             NA
#> 2           0.30441         2.61978                        0.38591
#> 3           0.31460         2.55373                        0.36354
#> 4           0.31899         2.57575                        0.35754
#> 5           0.31988         2.64179                        0.36117
#> 6           0.31464         2.59776                        0.35934
#> 7           0.32263         2.86194                        0.34954
# Scoring History:
#   timestamp   duration  training_speed   epochs
# 1 2016-05-03 10:33:29  0.000 sec                  0.00000
# 2 2016-05-03 10:33:29  0.347 sec 424697 rows/sec  0.86851
# 3 2016-05-03 10:33:30  1.356 sec 601925 rows/sec  6.09185
# 4 2016-05-03 10:33:31  2.348 sec 717617 rows/sec 13.05168
# 5 2016-05-03 10:33:32  3.281 sec 777538 rows/sec 20.00783
# 6 2016-05-03 10:33:32  3.345 sec 777275 rows/sec 20.00783
# iterations        samples training_MSE training_r2
# 1          0       0.000000
# 2          1   99804.000000      0.14402     0.03691
# 3          7  700039.000000      0.14157     0.05333
# 4         15 1499821.000000      0.14033     0.06159
# 5         23 2299180.000000      0.14079     0.05853
# 6         23 2299180.000000      0.14157     0.05333
# training_logloss training_AUC training_lift
# 1
# 2          0.45930      0.66685       2.20727
# 3          0.45220      0.68133       2.59354
# 4          0.44710      0.67993       2.70390
# 5          0.45100      0.68192       2.81426
# 6          0.45220      0.68133       2.59354
# training_classification_error validation_MSE validation_r2
# 1
# 2                       0.36145        0.14682       0.03426
# 3                       0.33647        0.14500       0.04619
# 4                       0.37126        0.14411       0.05204
# 5                       0.32868        0.14474       0.04793
# 6                       0.33647        0.14500       0.04619
# validation_logloss validation_AUC validation_lift
# 1
# 2           0.46692        0.66582         2.53209
```

```
# 3            0.46256        0.67354         2.64124
# 4            0.45789        0.66986         2.44478
# 5            0.46292        0.67117         2.70672
# 6            0.46256        0.67354         2.64124
# validation_classification_error
# 1
# 2                          0.37197
# 3                          0.34716
# 4                          0.34385
# 5                          0.36544
# 6                          0.34716
```

1.7 Naive Bayes model

The Naive Bayes (NB) algorithm does not usually beat an algorithm like a Random Forest or GBM, however it is still a popular algorithm, especially in the text domain (when your input is text encoded as "Bag of Words", for example). The Naive Bayes algorithm is for binary or multiclass classification problems only, not regression. Therefore, your response must be a factor instead of a numeric.

```
# First we will train a basic NB model with default parameters.
nb_fit1 <- h2o.naiveBayes(x = x,
                          y = y,
                          training_frame = train,
                          model_id = "nb_fit1")
#>
  |
  |                                                                      |   0%
  |
  |===========                                                           |  17%
  |
  |======================================================================| 100%
```

1.7.1 Train a NB model with Laplace Smoothing

One of the few tunable model parameters for the Naive Bayes algorithm is the amount of Laplace smoothing. The H2O Naive Bayes model will not use any Laplace smoothing by default.

```
nb_fit2 <- h2o.naiveBayes(x = x,
                          y = y,
                          training_frame = train,
                          model_id = "nb_fit2",
                          laplace = 6)
#>
  |
  |                                                                      |   0%
  |
  |=================================================                     |  83%
  |
  |======================================================================| 100%

# Let's compare the performance of the two NB models
nb_perf1 <- h2o.performance(model = nb_fit1,
```

```
                            newdata = test)
nb_perf2 <- h2o.performance(model = nb_fit2,
                            newdata = test)

# Print model performance
nb_perf1
#> H2OBinomialMetrics: naivebayes
#>
#> MSE:  0.15
#> RMSE:  0.387
#> LogLoss:  0.489
#> Mean Per-Class Error:  0.39
#> AUC:  0.651
#> pr_auc:  0.297
#> Gini:  0.303
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0    1    Error         Rate
#> 0      13184 6807 0.340503  =6807/19991
#> 1       2021 2569 0.440305   =2021/4590
#> Totals 15205 9376 0.359139  =8828/24581
#>
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                       max f1  0.225948 0.367893 236
#> 2                       max f2  0.090634 0.545538 346
#> 3                 max f0point5  0.340471 0.335807 166
#> 4                 max accuracy  0.999554 0.812945   0
#> 5                max precision  0.559607 0.428747  70
#> 6                   max recall  0.000196 1.000000 399
#> 7              max specificity  0.999554 0.999550   0
#> 8             max absolute_mcc  0.287231 0.188641 196
#> 9   max min_per_class_accuracy  0.208724 0.602832 250
#> 10 max mean_per_class_accuracy  0.225948 0.609596 236
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
nb_perf2
#> H2OBinomialMetrics: naivebayes
#>
#> MSE:  0.15
#> RMSE:  0.387
#> LogLoss:  0.489
#> Mean Per-Class Error:  0.39
#> AUC:  0.651
#> pr_auc:  0.297
#> Gini:  0.303
#>
#> Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
#>            0    1    Error         Rate
#> 0      14002 5989 0.299585  =5989/19991
#> 1       2207 2383 0.480828   =2207/4590
#> Totals 16209 8372 0.333428  =8196/24581
#>
```

```
#> Maximum Metrics: Maximum metrics at their respective thresholds
#>                         metric threshold    value idx
#> 1                       max f1  0.242206 0.367690 222
#> 2                       max f2  0.088660 0.545677 347
#> 3                 max f0point5  0.362995 0.336012 152
#> 4                 max accuracy  0.999564 0.812945   0
#> 5                max precision  0.574610 0.428775  63
#> 6                   max recall  0.000207 1.000000 399
#> 7              max specificity  0.999564 0.999550   0
#> 8             max absolute_mcc  0.286635 0.189479 192
#> 9   max min_per_class_accuracy  0.207609 0.604357 248
#> 10 max mean_per_class_accuracy  0.247906 0.609878 218
#>
#> Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/

# Retreive test set AUC
h2o.auc(nb_perf1)  # 0.6488014
#> [1] 0.651
h2o.auc(nb_perf2)  # 0.6490678
#> [1] 0.651
```

Chapter 2

Classification algorithms comparison. Diabetes datset. (*CART*, *LDA*, *SVM*, *KNN*, *RF*)

2.1 `PimaIndiansDiabetes` dataset

2.2 Introduction

We compare the following clasification algorithms:

- CART
- LDA
- SVM
- KNN
- RF

2.3 Workflow

1. Load dataset
2. Create the train dataset
3. Train the models
4. Collect resamples
5. Plot comparison
6. Summarize p-values

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

```
dplyr::glimpse(PimaIndiansDiabetes)
#> Observations: 768
#> Variables: 9
#> $ pregnant <dbl> 6, 1, 8, 1, 0, 5, 3, 10, 2, 8, 4, 10, 10, 1, 5, 7, 0,...
```

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

```
tibble::as_tibble(PimaIndiansDiabetes)
#> # A tibble: 768 x 9
#>   pregnant glucose pressure triceps insulin  mass pedigree   age diabetes
#>      <dbl>   <dbl>    <dbl>   <dbl>   <dbl> <dbl>    <dbl> <dbl> <fct>
#> 1        6     148       72      35       0  33.6    0.627    50 pos
#> 2        1      85       66      29       0  26.6    0.351    31 neg
#> 3        8     183       64       0       0  23.3    0.672    32 pos
#> 4        1      89       66      23      94  28.1    0.167    21 neg
#> 5        0     137       40      35     168  43.1    2.29     33 pos
#> 6        5     116       74       0       0  25.6    0.201    30 neg
#> # ... with 762 more rows
```

2.4 Train the models using cross-validation

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

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

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

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

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

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

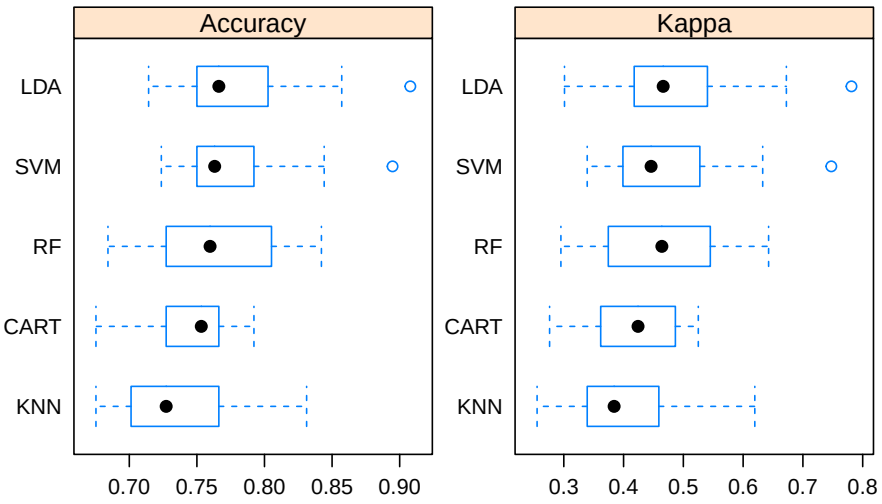
```
# collect resamples
results <- resamples(list(CART=fit.cart,
                          LDA=fit.lda,
                          SVM=fit.svm,
                          KNN=fit.knn,
                          RF=fit.rf))
```

2.5 Compare models

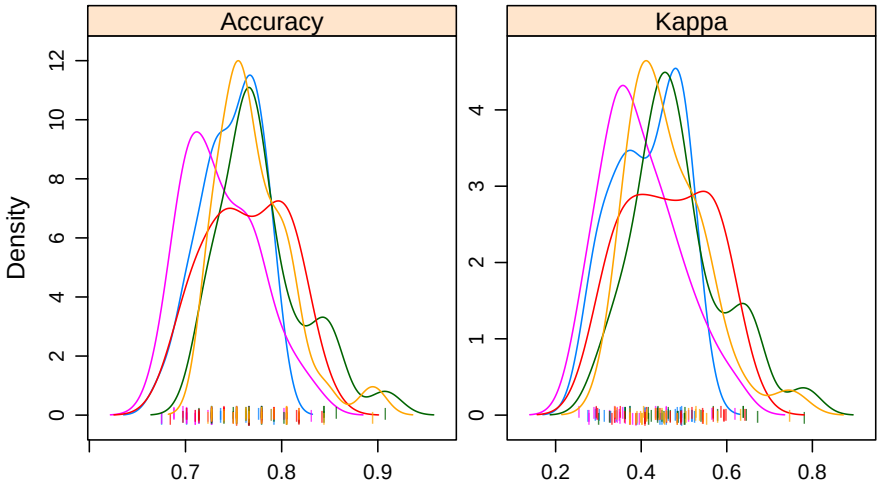
```
# summarize differences between models
summary(results)
#>
#> Call:
#> summary.resamples(object = results)
#>
#> Models: CART, LDA, SVM, KNN, RF
#> Number of resamples: 30
#>
#> Accuracy
#>       Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
#> CART 0.675   0.727  0.753 0.747   0.766 0.792    0
#> LDA  0.714   0.751  0.766 0.779   0.800 0.908    0
#> SVM  0.724   0.751  0.763 0.771   0.792 0.895    0
#> KNN  0.675   0.704  0.727 0.737   0.766 0.831    0
#> RF   0.684   0.731  0.760 0.764   0.802 0.842    0
#>
#> Kappa
#>       Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
#> CART 0.276   0.362  0.424 0.415   0.486 0.525    0
#> LDA  0.301   0.419  0.466 0.486   0.531 0.781    0
#> SVM  0.339   0.400  0.446 0.462   0.523 0.748    0
#> KNN  0.255   0.341  0.384 0.398   0.454 0.620    0
#> RF   0.295   0.378  0.464 0.463   0.545 0.643    0
```

2.6 Plot comparison

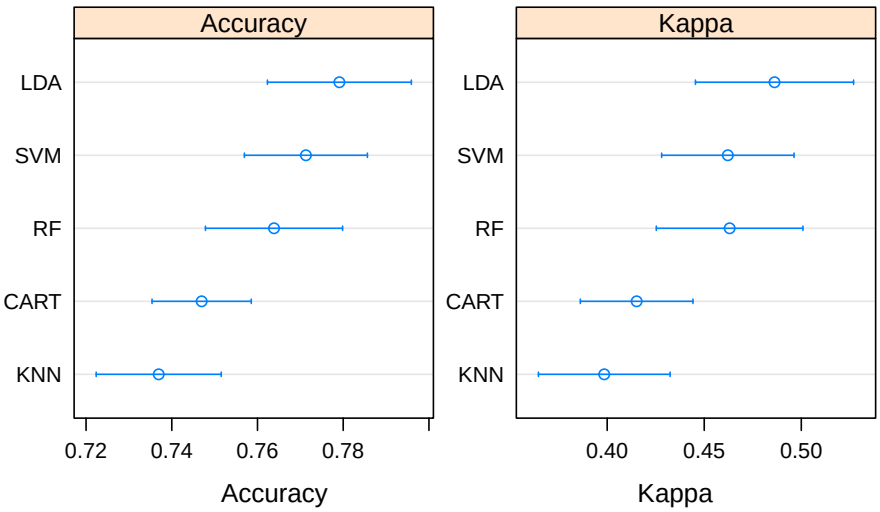
```
# box and whisker plots to compare models
scales <- list(x=list(relation="free"), y=list(relation="free"))
bwplot(results, scales=scales)
```



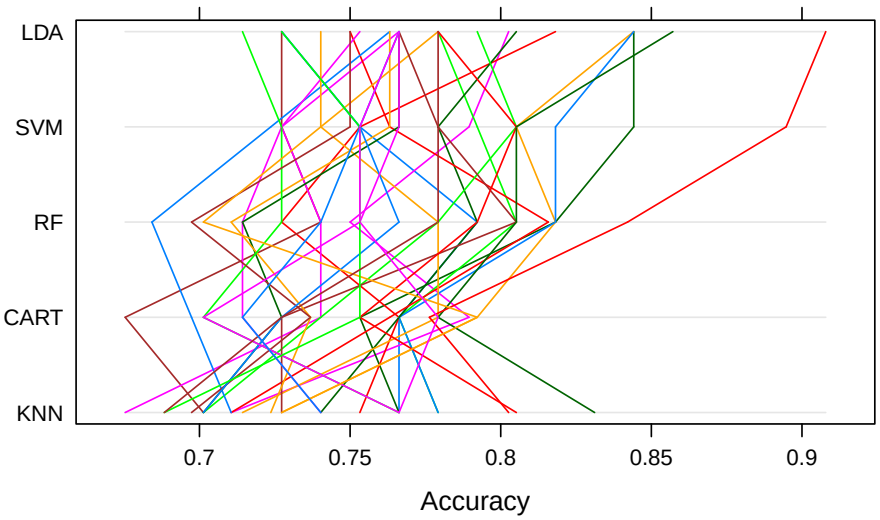

```
# density plots of accuracy
scales <- list(x=list(relation="free"), y=list(relation="free"))
densityplot(results, scales=scales, pch = "|")
```



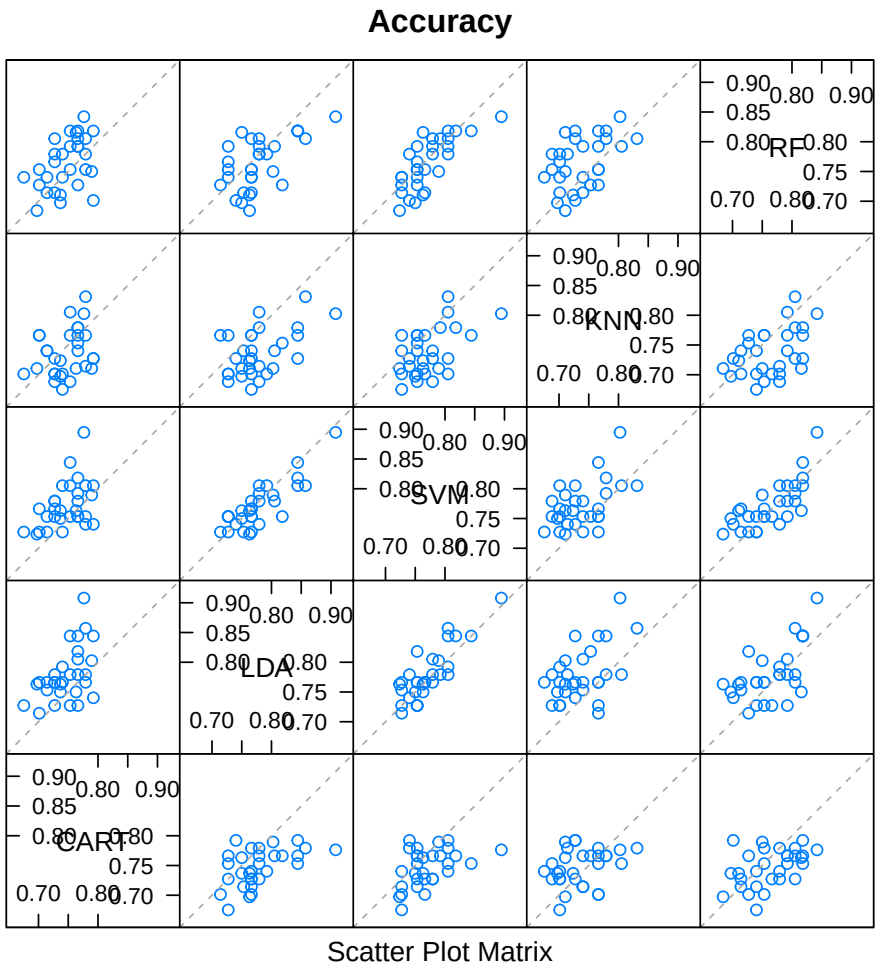

```
# dot plots of accuracy
scales <- list(x=list(relation="free"), y=list(relation="free"))
dotplot(results, scales=scales)
```



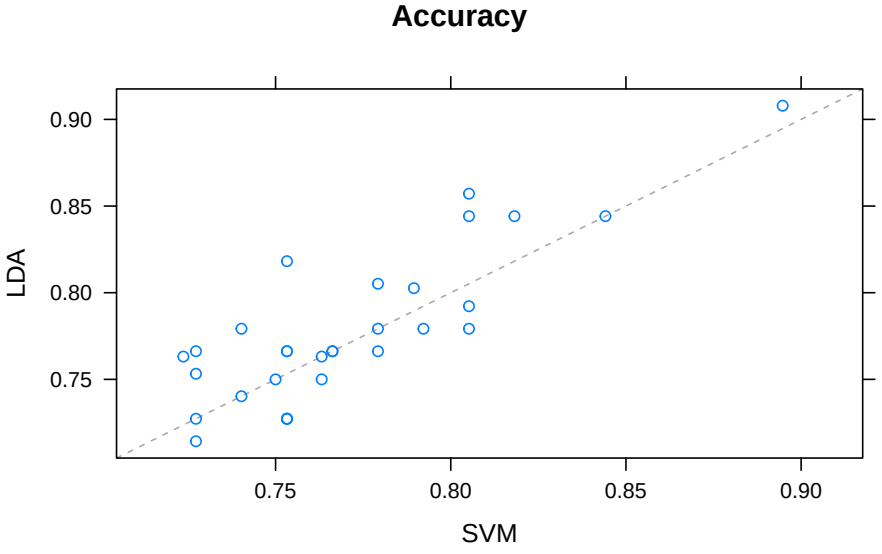

```
# parallel plots to compare models
parallelplot(results)
```




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




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




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

```r
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          -0.03214 -0.02432  0.01002 -0.01688
#> LDA  0.001186           0.00781  0.04216  0.01525
#> SVM  0.011640 0.915689           0.03434  0.00744
#> KNN  1.000000 6.68e-05 0.000294          -0.02690
#> RF   0.272754 0.449062 1.000000 0.018379
#>
#> Kappa
#>      CART     LDA       SVM       KNN       RF
#> CART          -0.071016 -0.046972  0.016687 -0.047894
#> LDA  0.000809            0.024044  0.087703  0.023122
#> SVM  0.025808 0.356273             0.063659 -0.000922
#> KNN  1.000000 0.000386  0.004082            -0.064581
#> RF   0.021176 1.000000  1.000000  0.015897
```

Chapter 3

Multiclass classification comparison. Diabetes dataset. (*LDA, CART, KNN, SVM, RF*)

3.1 `iris` dataset

3.2 Introduction

These are the algorithms used:

1. LDA
2. CART
3. KNN
4. SVM
5. RF

```
# load the caret package
library(caret)
#> Loading required package: lattice
#> Loading required package: ggplot2
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang
# attach the iris dataset to the environment
data(iris)
# rename the dataset
dataset <- iris
```

3.3 Workflow

1. Load dataset
2. Create train and test datasets, 80/20

3. Inspect dataset
4. Visualize features
5. Set the train control to

- 10 cross-validations
- Metric: accuracy

6. Train the models
7. Compare accuracy of models
8. Visual comparison
9. Make predictions on `validation` set

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"
```

3.4 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)
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
```

```
glimpse(dataset)
#> Observations: 120
#> Variables: 5
#> $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9,...
#> $ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1,...
#> $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5,...
#> $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1,...
#> $ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, s...
```

```
library(skimr)
#>
#> Attaching package: 'skimr'
#> The following object is masked from 'package:stats':
#>
#>     filter

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 3.76 1.78 1   1.58 4.35 5.1   6.9
#>   Petal.Width       0      120 120 1.2  0.76 0.1 0.3  1.3  1.8   2.5
#>  Sepal.Length       0      120 120 5.86 0.84 4.3 5.1  5.8  6.4   7.9
#>   Sepal.Width       0      120 120 3.06 0.44 2   2.8  3    3.32  4.4
#>      hist
#>
#>
#>
#>
```

3.5 Levels of the class

```
# list the levels for the class
levels(dataset$Species)
#> [1] "setosa"     "versicolor" "virginica"
```

3.6 class distribution

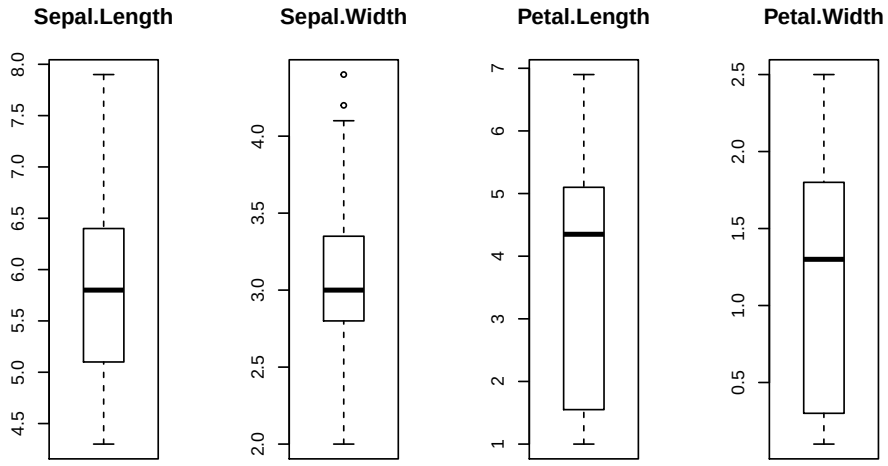
```
# summarize the class distribution
percentage <- prop.table(table(dataset$Species)) * 100
```

```
cbind(freq=table(dataset$Species), percentage=percentage)
#>            freq percentage
#> setosa       40       33.3
#> versicolor   40       33.3
#> virginica    40       33.3
```

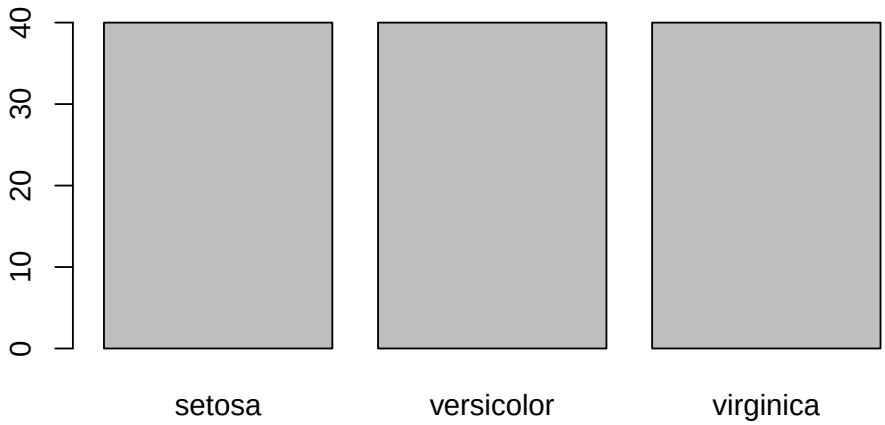
3.7 Visualize the dataset

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

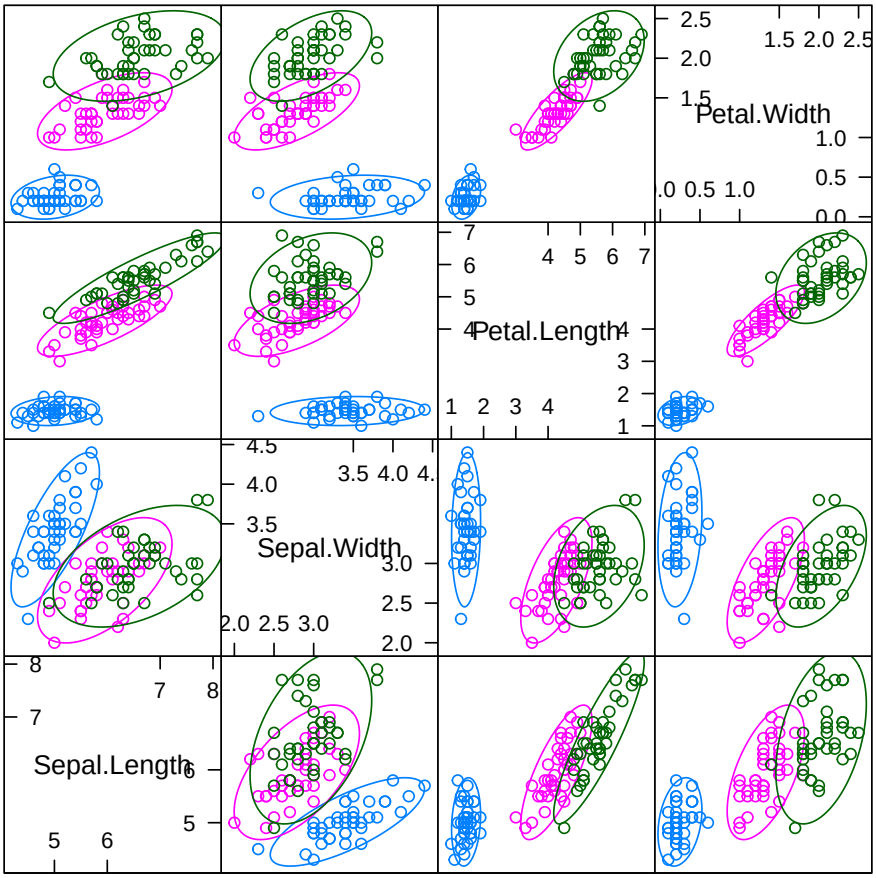
```
# boxplot for each attribute on one image
par(mfrow=c(1,4))
for(i in 1:4) {
    boxplot(x[,i], main=names(dataset)[i])
}
```




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

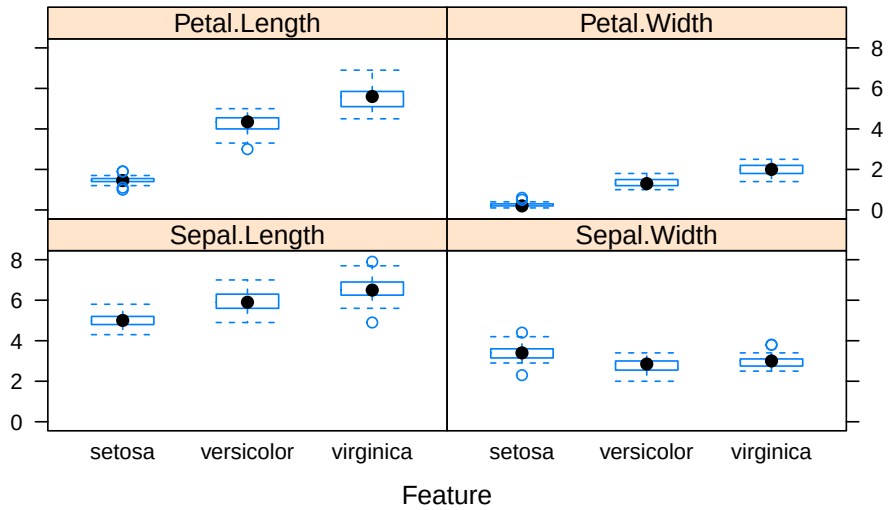



```
# scatter plot matrix
featurePlot(x=x, y=y, plot="ellipse")
```



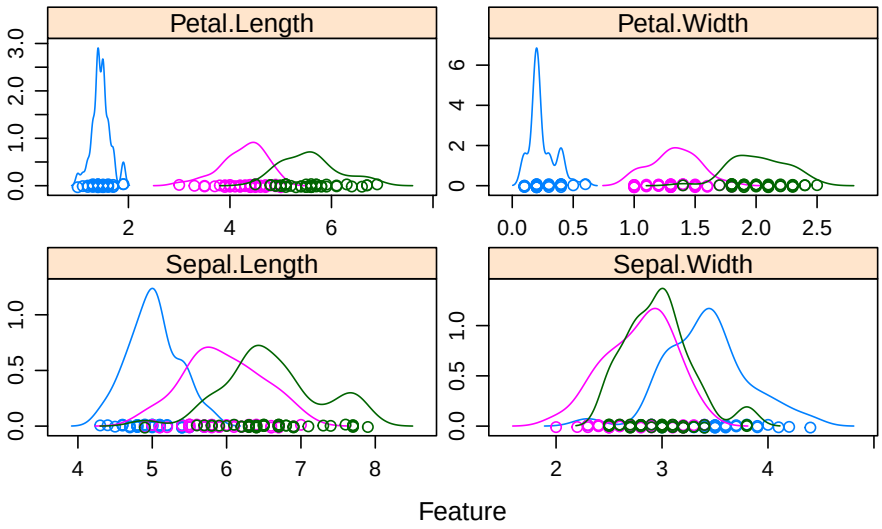
Scatter Plot Matrix

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



Feature

```
# 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)
```



3.8 Evaluate algorithms

3.8.1 split and metrics

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

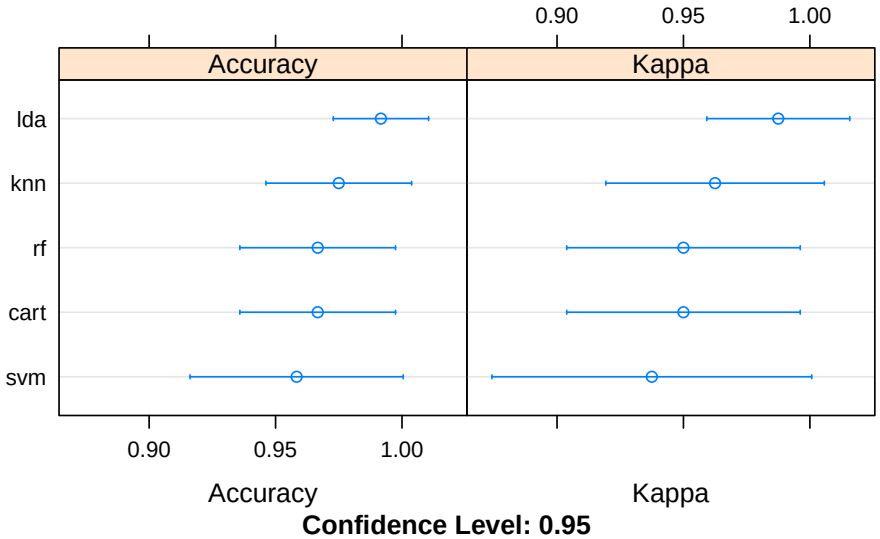
3.8.2 build models

```
# 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)
```

3.8.3 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  0.917   1.000      1 0.992       1    1    0
#> cart 0.917   0.917      1 0.967       1    1    0
#> knn  0.917   0.938      1 0.975       1    1    0
#> svm  0.833   0.917      1 0.958       1    1    0
#> rf   0.917   0.917      1 0.967       1    1    0
#>
#> Kappa
#>       Min. 1st Qu. Median  Mean 3rd Qu. Max. NA's
#> lda  0.875   1.000      1 0.987       1    1    0
#> cart 0.875   0.875      1 0.950       1    1    0
#> knn  0.875   0.906      1 0.962       1    1    0
#> svm  0.750   0.875      1 0.937       1    1    0
#> rf   0.875   0.875      1 0.950       1    1    0
```

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



```
# summarize Best Model
print(fit.lda)
#> Linear Discriminant Analysis
#>
#> 120 samples
#>   4 predictor
#>   3 classes: 'setosa', 'versicolor', 'virginica'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold)
#> Summary of sample sizes: 108, 108, 108, 108, 108, 108, ...
#> Resampling results:
#>
#>   Accuracy  Kappa
#>   0.992     0.987
```

3.9 Make predictions

```
# estimate skill of LDA on the validation dataset
predictions <- predict(fit.lda, validation)
confusionMatrix(predictions, validation$Species)
#> Confusion Matrix and Statistics
#>
#>             Reference
#> Prediction   setosa versicolor virginica
#>   setosa         10          0         0
#>   versicolor      0          9         1
#>   virginica       0          1         9
#>
#> Overall Statistics
#>
#>                Accuracy : 0.933
#>                  95% CI : (0.779, 0.992)
#>     No Information Rate : 0.333
#>     P-Value [Acc > NIR] : 8.75e-12
#>
#>                   Kappa : 0.9
#>
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: setosa Class: versicolor Class: virginica
#> Sensitivity                  1.000            0.900           0.900
#> Specificity                  1.000            0.950           0.950
#> Pos Pred Value               1.000            0.900           0.900
#> Neg Pred Value               1.000            0.950           0.950
#> Prevalence                   0.333            0.333           0.333
#> Detection Rate               0.333            0.300           0.300
#> Detection Prevalence         0.333            0.333           0.333
#> Balanced Accuracy            1.000            0.925           0.925
```

Chapter 4

Regression algorithms comparison. Boston dataset. (*LM, GKM, GLMNET, SVM, CART, KNN*)

4.1 `Boston` dataset

- Comparison of various algorithms.

4.2 Introduction

These are the algorithms used:

1. LM
2. GLM
3. GLMNET
4. SVM
5. CART
6. KNN

```
# load packages
library(mlbench)
library(caret)
#> Loading required package: lattice
#> Loading required package: ggplot2
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang
library(corrplot)
#> corrplot 0.84 loaded

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

```
dplyr::glimpse(BostonHousing)
#> Observations: 506
#> Variables: 14
#> $ crim    <dbl> 0.00632, 0.02731, 0.02729, 0.03237, 0.06905, 0.02985, ...
#> $ zn      <dbl> 18.0, 0.0, 0.0, 0.0, 0.0, 0.0, 12.5, 12.5, 12.5, 12.5,...
#> $ indus   <dbl> 2.31, 7.07, 7.07, 2.18, 2.18, 2.18, 7.87, 7.87, 7.87, ...
#> $ chas    <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ nox     <dbl> 0.538, 0.469, 0.469, 0.458, 0.458, 0.458, 0.524, 0.524...
#> $ rm      <dbl> 6.58, 6.42, 7.18, 7.00, 7.15, 6.43, 6.01, 6.17, 5.63, ...
#> $ age     <dbl> 65.2, 78.9, 61.1, 45.8, 54.2, 58.7, 66.6, 96.1, 100.0,...
#> $ dis     <dbl> 4.09, 4.97, 4.97, 6.06, 6.06, 6.06, 5.56, 5.95, 6.08, ...
#> $ 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> 397, 397, 393, 395, 397, 394, 396, 397, 387, 387, 393,...
#> $ lstat   <dbl> 4.98, 9.14, 4.03, 2.94, 5.33, 5.21, 12.43, 19.15, 29.9...
#> $ medv    <dbl> 24.0, 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, ...
```

```
tibble::as_tibble(BostonHousing)
#> # A tibble: 506 x 14
#>       crim    zn indus chas    nox    rm   age   dis   rad   tax ptratio
#>      <dbl> <dbl> <dbl> <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
#> 1 0.00632    18  2.31 0     0.538  6.58  65.2  4.09     1   296    15.3
#> 2 0.0273      0  7.07 0     0.469  6.42  78.9  4.97     2   242    17.8
#> 3 0.0273      0  7.07 0     0.469  7.18  61.1  4.97     2   242    17.8
#> 4 0.0324      0  2.18 0     0.458  7.00  45.8  6.06     3   222    18.7
#> 5 0.0690      0  2.18 0     0.458  7.15  54.2  6.06     3   222    18.7
#> 6 0.0298      0  2.18 0     0.458  6.43  58.7  6.06     3   222    18.7
#> # ... with 500 more rows, and 3 more variables: b <dbl>, lstat <dbl>,
#> #   medv <dbl>
```

4.3 Workflow

1. Load dataset
2. Create train and test datasets, 80/20
3. Inspect dataset:

- Dimension
- classes
- `skimr`

4. Analyze features

- correlation

5. Visualize features

- histograms
- density plots
- pairwise
- correlogram

5. Train as-is

- Set the train control to
  - 10 cross-validations
  - 3 repetitions
  - Metric: RMSE
- Train the models
- Compare accuracy of models
- Visual comparison
  - dot plot

6. Train with Feature selection

- Feature selection
  - `findCorrelation`
  - generate new dataset
- Train models again

- Compare RMSE again
- Visual comparison
  - dot plot

7. Train with dataset transformation

- data transformatiom
  - Center
  - Scale
  - BoxCox
- Train models
- Compare RMSE
- Visual comparison
  - dot plot

8. Tune the best model

- Set the train control to
  - 10 cross-validations
  - 3 repetitions
  - Metric: RMSE
- Train the models
  - Radial SVM
  - Sigma vector
  - .C
  - BoxCox 9, Ensembling
- Select the algorithms
  - Random Forest
  - Stochastic Gradient Boosting
  - Cubist
- Numeric comparison
  - resample
  - summary
- Visual comparison
- dot plot

10. Tune the best model: Cubist

- Set the train control to
  - 10 cross-validations
  - 3 repetitions

  - Metric: RMSE
- Train the models
  - Cubist
  - `.committees`
  - `.neighbors`
  - BoxCox
- Evaluate the tuning parameters
  - Numeric comparison
    - print tuned model
  - Visual comparison
    - scatter plot

11. Finalize the model

- Back transformation
- Summary

12. Apply model to validation set
    - Transform the dataset
    - Make prediction
    - Calculate the RMSE

```
# 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 lstat
#> 1  0.00632 18.0  2.31    0 0.538 6.58  65.2 4.09   1 296    15.3 397  4.98
#> 2  0.02731  0.0  7.07    0 0.469 6.42  78.9 4.97   2 242    17.8 397  9.14
#> 3  0.02729  0.0  7.07    0 0.469 7.18  61.1 4.97   2 242    17.8 393  4.03
#> 4  0.03237  0.0  2.18    0 0.458 7.00  45.8 6.06   3 222    18.7 395  2.94
#> 5  0.06905  0.0  2.18    0 0.458 7.15  54.2 6.06   3 222    18.7 397  5.33
#> 6  0.02985  0.0  2.18    0 0.458 6.43  58.7 6.06   3 222    18.7 394  5.21
#> 7  0.08829 12.5  7.87    0 0.524 6.01  66.6 5.56   5 311    15.2 396 12.43
```

```
#> 10 0.17004 12.5  7.87    0 0.524 6.00  85.9 6.59   5 311    15.2 387 17.10
#> 11 0.22489 12.5  7.87    0 0.524 6.38  94.3 6.35   5 311    15.2 393 20.45
#> 12 0.11747 12.5  7.87    0 0.524 6.01  82.9 6.23   5 311    15.2 397 13.27
#> 13 0.09378 12.5  7.87    0 0.524 5.89  39.0 5.45   5 311    15.2 390 15.71
#> 14 0.62976  0.0  8.14    0 0.538 5.95  61.8 4.71   4 307    21.0 397  8.26
#> 15 0.63796  0.0  8.14    0 0.538 6.10  84.5 4.46   4 307    21.0 380 10.26
#> 17 1.05393  0.0  8.14    0 0.538 5.93  29.3 4.50   4 307    21.0 387  6.58
#> 20 0.72580  0.0  8.14    0 0.538 5.73  69.5 3.80   4 307    21.0 391 11.28
#> 21 1.25179  0.0  8.14    0 0.538 5.57  98.1 3.80   4 307    21.0 377 21.02
#> 22 0.85204  0.0  8.14    0 0.538 5.96  89.2 4.01   4 307    21.0 393 13.83
#> 23 1.23247  0.0  8.14    0 0.538 6.14  91.7 3.98   4 307    21.0 397 18.72
#> 24 0.98843  0.0  8.14    0 0.538 5.81 100.0 4.10   4 307    21.0 395 19.88
#> 25 0.75026  0.0  8.14    0 0.538 5.92  94.1 4.40   4 307    21.0 394 16.30
#>    medv
#> 1  24.0
#> 2  21.6
#> 3  34.7
#> 4  33.4
#> 5  36.2
#> 6  28.7
#> 7  22.9
#> 10 18.9
#> 11 15.0
#> 12 18.9
#> 13 21.7
#> 14 20.4
#> 15 18.2
#> 17 23.1
#> 20 18.2
#> 21 13.6
#> 22 19.6
#> 23 15.2
#> 24 14.5
#> 25 15.6
```

```
library(skimr)
#>
#> Attaching package: 'skimr'
#> The following object is masked from 'package:stats':
#>
#>     filter
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: 378, 1: 29, NA: 0   FALSE
#>
#> -- Variable type:numeric ------------------------------------------------------------
#>  variable missing complete   n   mean     sd       p0    p25    p50    p75
#>       age       0      407 407  68.38  28.16   6.2     42.7   77.3   94.2
```

```
#>          b       0      407 407 357.19  89.67   0.32   373.81 391.27 396.02
#>       crim       0      407 407   3.64   8.8    0.0063   0.08   0.27   3.69
#>        dis       0      407 407   3.82   2.12   1.13     2.11   3.15   5.21
#>      indus       0      407 407  11      6.87   0.74     4.93   8.56  18.1
#>      lstat       0      407 407  12.56   7.03   1.92     7.06  11.32  16.45
#>       medv       0      407 407  22.52   8.96   5       17.05  21.2   25
#>        nox       0      407 407   0.55   0.12   0.39     0.45   0.54   0.63
#>    ptratio       0      407 407  18.42   2.18  12.6     17     19     20.2
#>        rad       0      407 407   9.59   8.77   1        4      5     24
#>         rm       0      407 407   6.29   0.7    3.56     5.89   6.21   6.62
#>        tax       0      407 407 408.75 168.72 187      280.5  334    666
#>         zn       0      407 407  11.92  24.19   0        0      0     15
#>     p100
#>  100
#>  396.9
#>   88.98
#>   12.13
#>   27.74
#>   37.97
#>   50
#>    0.87
#>   22
#>   24
#>    8.78
#>  711
#>  100
```

```
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  68.38   28.16   6.2     42.7   77.3
#>         b       0      407 407 357.19   89.67   0.32   373.81 391.27
#>      chas       0      407 407   0.071   0.26   0        0      0
#>      crim       0      407 407   3.64    8.8    0.0063   0.08   0.27
#>       dis       0      407 407   3.82    2.12   1.13     2.11   3.15
#>     indus       0      407 407  11       6.87   0.74     4.93   8.56
#>     lstat       0      407 407  12.56    7.03   1.92     7.06  11.32
#>      medv       0      407 407  22.52    8.96   5       17.05  21.2
#>       nox       0      407 407   0.55    0.12   0.39     0.45   0.54
#>   ptratio       0      407 407  18.42    2.18  12.6     17     19
#>       rad       0      407 407   9.59    8.77   1        4      5
#>        rm       0      407 407   6.29    0.7    3.56     5.89   6.21
#>       tax       0      407 407 408.75  168.72 187      280.5  334
#>        zn       0      407 407  11.92   24.19   0        0      0
#>     p75   p100
#>   94.2  100
#>  396.02 396.9
#>    0      1
```

```
#>    3.69  88.98
#>    5.21  12.13
#>   18.1   27.74
#>   16.45  37.97
#>   25     50
#>    0.63   0.87
#>   20.2   22
#>   24     24
#>    6.62   8.78
#>  666    711
#>   15    100
```

no more factors or character variables

```
# find correlation between variables
cor(dataset[,1:13])
#>             crim      zn   indus     chas      nox     rm      age     dis
#> crim     1.0000 -0.1996  0.4076 -0.05507  0.4099 -0.194  0.3524 -0.376
#> zn      -0.1996  1.0000 -0.5314 -0.02987 -0.5202  0.311 -0.5845  0.680
#> indus    0.4076 -0.5314  1.0000  0.06583  0.7733 -0.383  0.6512 -0.711
#> chas    -0.0551 -0.0299  0.0658  1.00000  0.0934  0.127  0.0735 -0.099
#> nox      0.4099 -0.5202  0.7733  0.09340  1.0000 -0.296  0.7338 -0.769
#> rm      -0.1940  0.3111 -0.3826  0.12677 -0.2961  1.000 -0.2262  0.207
#> age      0.3524 -0.5845  0.6512  0.07350  0.7338 -0.226  1.0000 -0.749
#> dis     -0.3756  0.6799 -0.7113 -0.09905 -0.7693  0.207 -0.7492  1.000
#> rad      0.6083 -0.3227  0.6200 -0.00245  0.6276 -0.221  0.4690 -0.504
#> tax      0.5711 -0.3184  0.7185 -0.03064  0.6758 -0.295  0.5058 -0.526
#> ptratio  0.2897 -0.3888  0.3782 -0.12283  0.1888 -0.365  0.2709 -0.228
#> b       -0.3442  0.1747 -0.3644  0.03782 -0.3684  0.126 -0.2742  0.284
#> lstat    0.4229 -0.4219  0.6136 -0.08430  0.5839 -0.612  0.6066 -0.501
#>              rad     tax ptratio       b   lstat
#> crim     0.60834  0.5711   0.290 -0.3442  0.4229
#> zn      -0.32273 -0.3184  -0.389  0.1747 -0.4219
#> indus    0.61998  0.7185   0.378 -0.3644  0.6136
#> chas    -0.00245 -0.0306  -0.123  0.0378 -0.0843
#> nox      0.62760  0.6758   0.189 -0.3684  0.5839
#> rm      -0.22126 -0.2953  -0.365  0.1260 -0.6120
#> age      0.46896  0.5058   0.271 -0.2742  0.6066
#> dis     -0.50372 -0.5264  -0.228  0.2843 -0.5013
#> rad      1.00000  0.9201   0.480 -0.4231  0.5025
#> tax      0.92005  1.0000   0.469 -0.4303  0.5538
#> ptratio  0.47971  0.4691   1.000 -0.1700  0.4093
#> b       -0.42314 -0.4303  -0.170  1.0000 -0.3509
#> lstat    0.50251  0.5538   0.409 -0.3509  1.0000
```

```
library(dplyr)
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
```

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

```
# select variables with correlation 0.7 and above
threshold <- 0.7
ok <- apply(abs(m) >= threshold, 1, any)
m[ok, ok]
#>         indus    nox    age    dis    rad    tax
#> indus  0.000  0.773  0.651 -0.711  0.620  0.719
#> nox    0.773  0.000  0.734 -0.769  0.628  0.676
#> age    0.651  0.734  0.000 -0.749  0.469  0.506
#> dis   -0.711 -0.769 -0.749  0.000 -0.504 -0.526
#> rad    0.620  0.628  0.469 -0.504  0.000  0.920
#> tax    0.719  0.676  0.506 -0.526  0.920  0.000
```

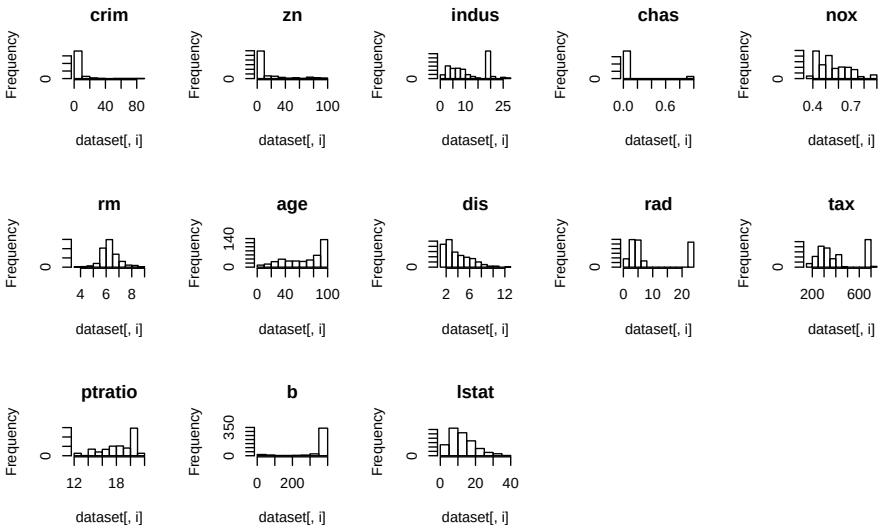
```
# values of correlation >= 0.7
ind <- sapply(1:13, function(x) abs(m[, x]) > 0.7)
m[ind]
#>  [1]  0.773 -0.711  0.719  0.773  0.734 -0.769  0.734 -0.749 -0.711 -0.769
#> [11] -0.749  0.920  0.719  0.920
```

```
# 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     tax
#> crim      0.4076  0.4099  0.3524 -0.376  0.60834  0.5711
#> zn       -0.5314 -0.5202 -0.5845  0.680 -0.32273 -0.3184
#> indus     0.0000  0.7733  0.6512 -0.711  0.61998  0.7185
#> chas      0.0658  0.0934  0.0735 -0.099 -0.00245 -0.0306
#> nox       0.7733  0.0000  0.7338 -0.769  0.62760  0.6758
#> rm       -0.3826 -0.2961 -0.2262  0.207 -0.22126 -0.2953
#> age       0.6512  0.7338  0.0000 -0.749  0.46896  0.5058
#> dis      -0.7113 -0.7693 -0.7492  0.000 -0.50372 -0.5264
#> rad       0.6200  0.6276  0.4690 -0.504  0.00000  0.9201
#> tax       0.7185  0.6758  0.5058 -0.526  0.92005  0.0000
#> ptratio   0.3782  0.1888  0.2709 -0.228  0.47971  0.4691
#> b        -0.3644 -0.3684 -0.2742  0.284 -0.42314 -0.4303
#> lstat     0.6136  0.5839  0.6066 -0.501  0.50251  0.5538
```

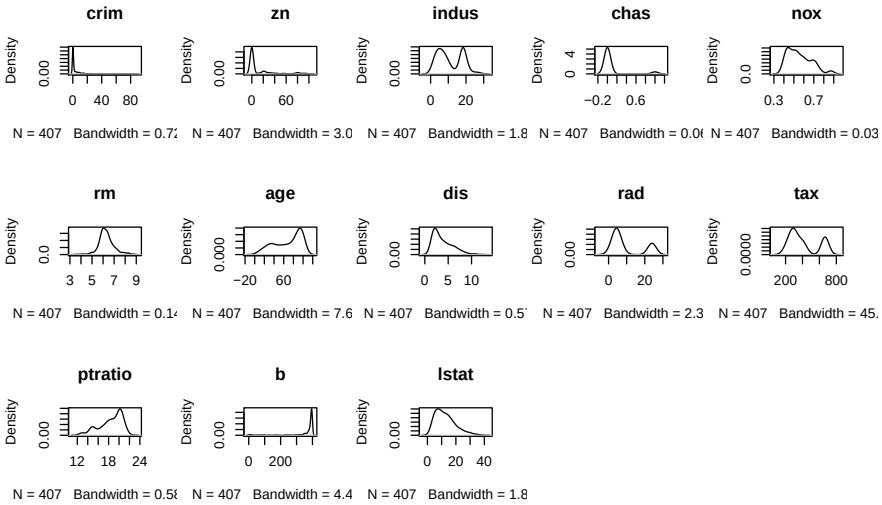
```
rind <- apply(cm, 1, function(x) any(abs(x) > 0.7))
rm <- cm[rind, ]
rm
#>         indus    nox    age    dis    rad    tax
#> indus  0.000  0.773  0.651 -0.711  0.620  0.719
#> nox    0.773  0.000  0.734 -0.769  0.628  0.676
#> age    0.651  0.734  0.000 -0.749  0.469  0.506
#> dis   -0.711 -0.769 -0.749  0.000 -0.504 -0.526
#> rad    0.620  0.628  0.469 -0.504  0.000  0.920
#> tax    0.719  0.676  0.506 -0.526  0.920  0.000
```

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

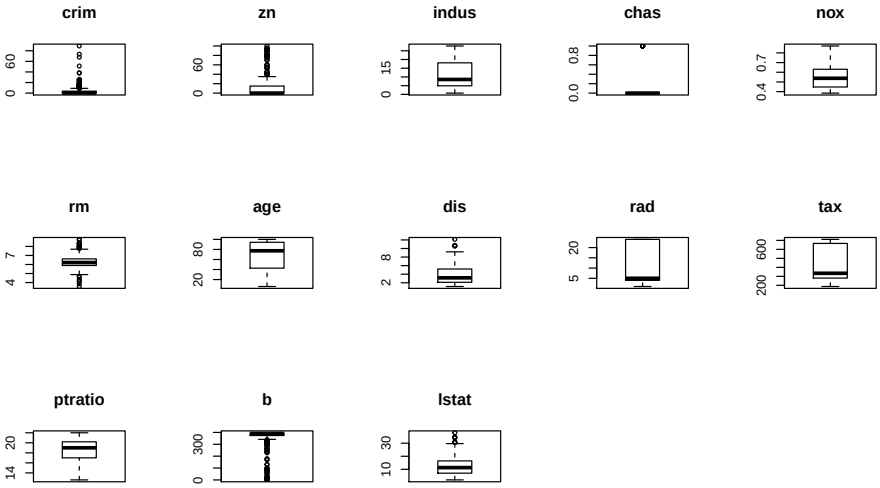
```
}
```




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

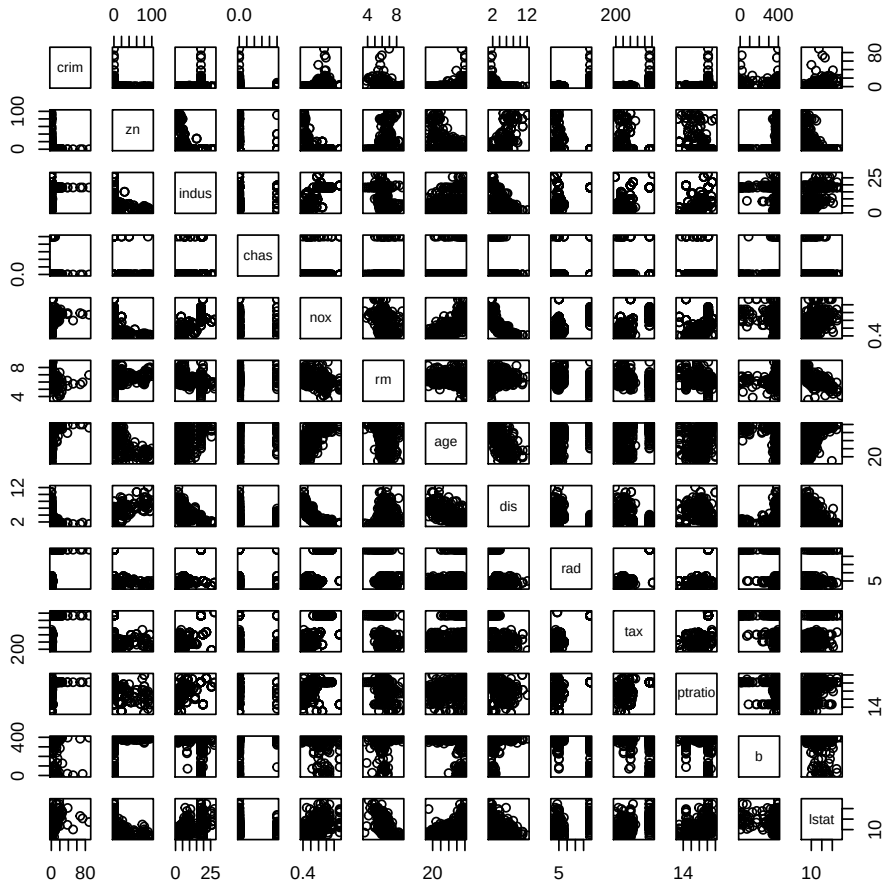



```
# boxplots for each attribute
par(mfrow=c(3,5))
for(i in 1:13) {
boxplot(dataset[,i], main=names(dataset)[i])
}
```

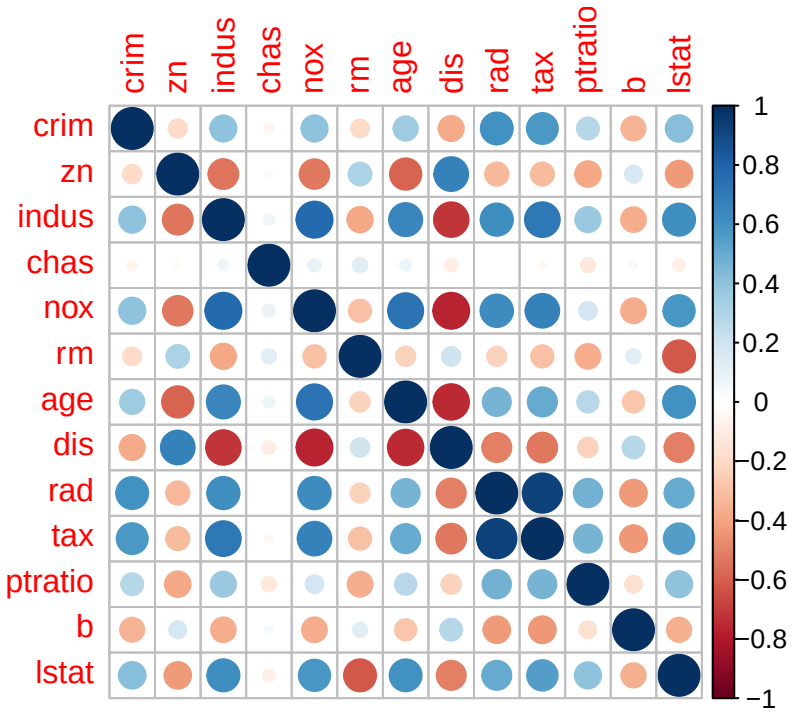


-1.bb

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




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



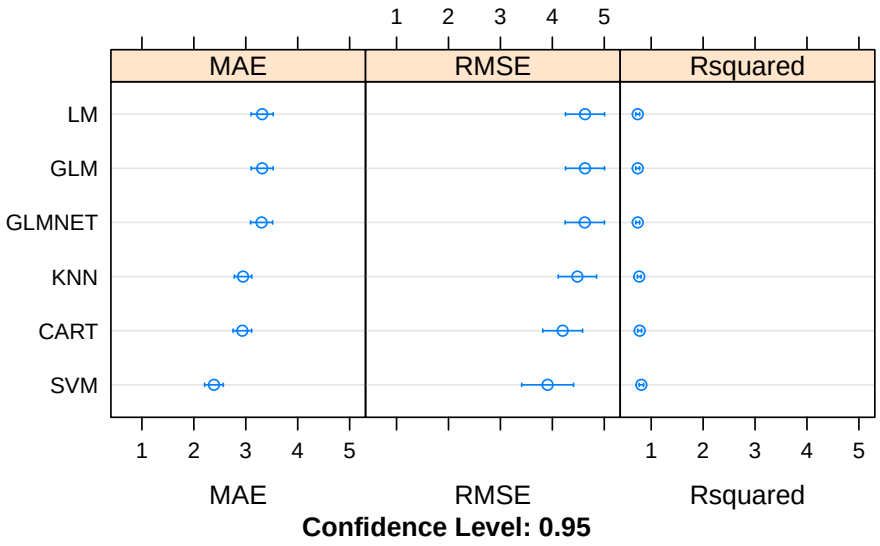
4.4 Evaluation

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

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

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

```
# Compare algorithms
results <- resamples(list(LM     = fit.lm,
                          GLM    = fit.glm,
                          GLMNET = fit.glmnet,
                          SVM    = fit.svm,
                          CART   = fit.cart,
                          KNN    = fit.knn))
summary(results)
#>
#> Call:
#> summary.resamples(object = results)
#>
#> Models: LM, GLM, GLMNET, SVM, CART, KNN
#> Number of resamples: 30
#>
#> MAE
#>        Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
#> LM     2.30    2.90   3.37 3.32    3.70 4.64    0
#> GLM    2.30    2.90   3.37 3.32    3.70 4.64    0
#> GLMNET 2.30    2.88   3.34 3.30    3.70 4.63    0
#> SVM    1.42    1.99   2.52 2.39    2.65 3.35    0
#> CART   2.22    2.62   2.88 2.93    3.08 4.16    0
#> KNN    1.98    2.69   2.87 2.95    3.24 4.00    0
#>
#> RMSE
#>        Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
#> LM     2.99    3.87   4.63 4.63    5.32 6.69    0
#> GLM    2.99    3.87   4.63 4.63    5.32 6.69    0
#> GLMNET 2.99    3.88   4.62 4.62    5.32 6.69    0
#> SVM    2.05    2.95   3.81 3.91    4.46 6.98    0
#> CART   2.77    3.38   4.00 4.20    4.60 7.09    0
#> KNN    2.65    3.74   4.42 4.48    5.06 6.98    0
#>
#> Rsquared
#>         Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
#> LM     0.505   0.674  0.747 0.740   0.813 0.900    0
#> GLM    0.505   0.674  0.747 0.740   0.813 0.900    0
#> GLMNET 0.503   0.673  0.747 0.741   0.816 0.904    0
#> SVM    0.519   0.762  0.845 0.810   0.896 0.970    0
#> CART   0.514   0.737  0.816 0.778   0.842 0.899    0
#> KNN    0.519   0.748  0.804 0.770   0.829 0.931    0
dotplot(results)
```



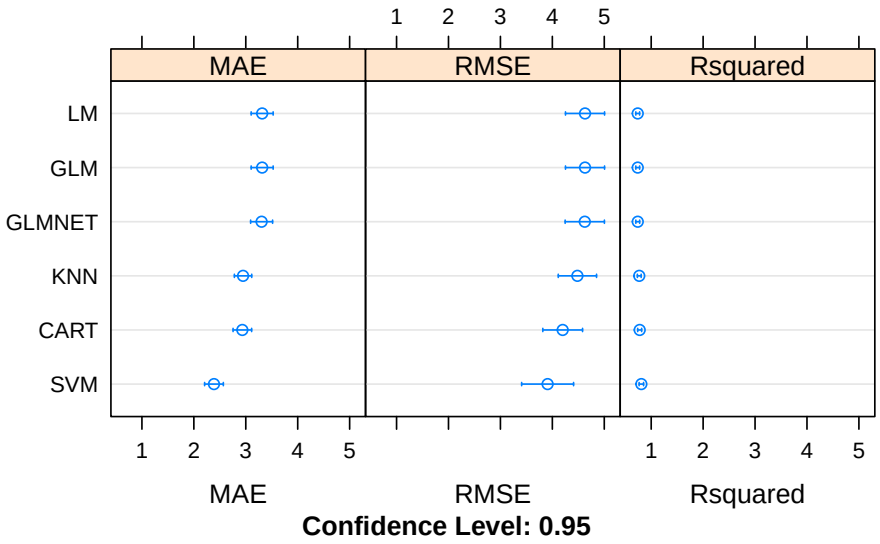
4.5 Feature selection

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

for (value in highlyCorrelated) {
    print(names(dataset)[value])
}
#> [1] "indus"
#> [1] "nox"
#> [1] "tax"
#> [1] "dis"

# create a new dataset without highly correlated features
datasetFeatures <- dataset[,-highlyCorrelated]
dim(datasetFeatures)
#> [1] 407  10
```

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

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

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

# Compare algorithms
feature_results <- resamples(list(LM     = fit.lm,
                                  GLM    = fit.glm,
                                  GLMNET = fit.glmnet,
                                  SVM    = fit.svm,
                                  CART   = fit.cart,
                                  KNN    = fit.knn))
summary(feature_results)
#>
#> Call:
#> summary.resamples(object = feature_results)
#>
#> Models: LM, GLM, GLMNET, SVM, CART, KNN
#> Number of resamples: 30
#>
#> MAE
#>        Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
#> LM     2.30    2.90   3.37 3.32    3.70 4.64    0
#> GLM    2.30    2.90   3.37 3.32    3.70 4.64    0
#> GLMNET 2.30    2.88   3.34 3.30    3.70 4.63    0
#> SVM    1.42    1.99   2.52 2.39    2.65 3.35    0
#> CART   2.22    2.62   2.88 2.93    3.08 4.16    0
#> KNN    1.98    2.69   2.87 2.95    3.24 4.00    0
#>
#> RMSE
#>        Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
```

```
#> LM     2.99    3.87   4.63 4.63    5.32 6.69    0
#> GLM    2.99    3.87   4.63 4.63    5.32 6.69    0
#> GLMNET 2.99    3.88   4.62 4.62    5.32 6.69    0
#> SVM    2.05    2.95   3.81 3.91    4.46 6.98    0
#> CART   2.77    3.38   4.00 4.20    4.60 7.09    0
#> KNN    2.65    3.74   4.42 4.48    5.06 6.98    0
#>
#> Rsquared
#>          Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
#> LM     0.505   0.674  0.747 0.740   0.813 0.900    0
#> GLM    0.505   0.674  0.747 0.740   0.813 0.900    0
#> GLMNET 0.503   0.673  0.747 0.741   0.816 0.904    0
#> SVM    0.519   0.762  0.845 0.810   0.896 0.970    0
#> CART   0.514   0.737  0.816 0.778   0.842 0.899    0
#> KNN    0.519   0.748  0.804 0.770   0.829 0.931    0
dotplot(feature_results)
```



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

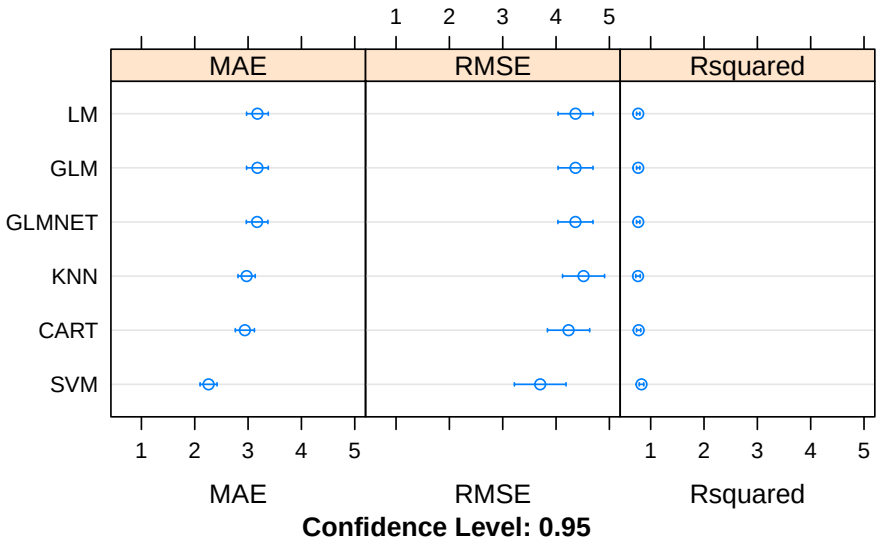
4.6 Evaluate Algorithms: Box-Cox Transform

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

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

# lm
set.seed(7)
```

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

# Compare algorithms
transformResults <- resamples(list(LM     = fit.lm,
                                   GLM    = fit.glm,
                                   GLMNET = fit.glmnet,
                                   SVM    = fit.svm,
                                   CART   = fit.cart,
                                   KNN    = fit.knn))
summary(transformResults)
#>
#> Call:
#> summary.resamples(object = transformResults)
#>
#> Models: LM, GLM, GLMNET, SVM, CART, KNN
#> Number of resamples: 30
#>
#> MAE
#>        Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
#> LM     2.10    2.80   3.21 3.18    3.45 4.44    0
#> GLM    2.10    2.80   3.21 3.18    3.45 4.44    0
#> GLMNET 2.11    2.80   3.21 3.17    3.45 4.43    0
#> SVM    1.30    1.95   2.25 2.26    2.48 3.19    0
#> CART   2.22    2.62   2.89 2.94    3.11 4.16    0
```

```
#> KNN    2.33    2.66   2.82 2.97    3.27 3.96    0
#>
#> RMSE
#>        Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
#> LM     2.82    3.81   4.43 4.37    5.00 6.16    0
#> GLM    2.82    3.81   4.43 4.37    5.00 6.16    0
#> GLMNET 2.83    3.79   4.42 4.36    5.00 6.18    0
#> SVM    1.80    2.73   3.41 3.70    4.24 6.73    0
#> CART   2.77    3.38   4.00 4.23    4.83 7.09    0
#> KNN    3.01    3.73   4.37 4.52    5.02 7.30    0
#>
#> Rsquared
#>         Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
#> LM     0.560   0.712  0.775 0.766   0.825 0.910    0
#> GLM    0.560   0.712  0.775 0.766   0.825 0.910    0
#> GLMNET 0.556   0.713  0.775 0.766   0.826 0.910    0
#> SVM    0.524   0.778  0.854 0.827   0.907 0.979    0
#> CART   0.514   0.727  0.816 0.774   0.843 0.899    0
#> KNN    0.492   0.723  0.792 0.762   0.842 0.937    0
dotplot(transformResults)
```



4.7 Tune SVM

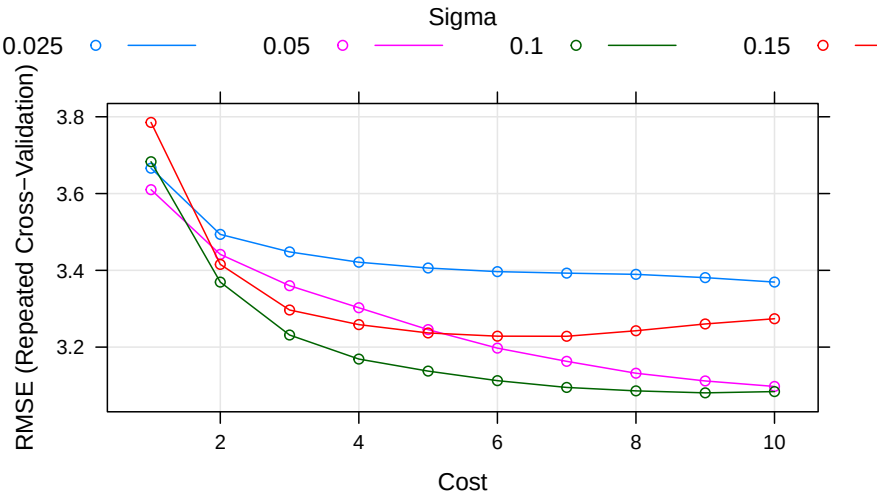
```
print(fit.svm)
#> Support Vector Machines with Radial Basis Function Kernel
#>
#> 407 samples
#>  13 predictor
#>
#> Pre-processing: centered (13), scaled (13), Box-Cox transformation (11)
#> Resampling: Cross-Validated (10 fold, repeated 3 times)
#> Summary of sample sizes: 365, 366, 366, 367, 366,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

```
#>   C     RMSE  Rsquared  MAE
#>   0.25  4.54  0.772     2.73
#>   0.50  4.07  0.802     2.46
#>   1.00  3.70  0.827     2.26
#>
#> Tuning parameter 'sigma' was held constant at a value of 0.116
#> RMSE was used to select the optimal model using the smallest value.
#> The final values used for the model were sigma = 0.116 and C = 1.
```

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

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

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

fit.svm <- train(medv~., data=dataset, method="svmRadial", metric=metric,
                 tuneGrid=grid,
                 preProc=c("BoxCox"), trControl=trainControl)
print(fit.svm)
#> Support Vector Machines with Radial Basis Function Kernel
#>
#> 407 samples
#>  13 predictor
#>
#> Pre-processing: Box-Cox transformation (11)
#> Resampling: Cross-Validated (10 fold, repeated 3 times)
#> Summary of sample sizes: 365, 366, 366, 367, 366, 366, ...
#> Resampling results across tuning parameters:
#>
#>   sigma  C   RMSE  Rsquared  MAE
#>   0.025   1  3.67  0.830     2.34
#>   0.025   2  3.49  0.840     2.21
#>   0.025   3  3.45  0.842     2.17
#>   0.025   4  3.42  0.844     2.14
#>   0.025   5  3.41  0.845     2.13
#>   0.025   6  3.40  0.846     2.12
#>   0.025   7  3.39  0.846     2.11
#>   0.025   8  3.39  0.846     2.11
#>   0.025   9  3.38  0.846     2.11
#>   0.025  10  3.37  0.847     2.10
#>   0.050   1  3.61  0.833     2.25
#>   0.050   2  3.44  0.843     2.17
#>   0.050   3  3.36  0.848     2.11
#>   0.050   4  3.30  0.852     2.08
#>   0.050   5  3.25  0.856     2.05
#>   0.050   6  3.20  0.860     2.03
#>   0.050   7  3.16  0.862     2.02
```

```
#>   0.050   8  3.13  0.865     2.02
#>   0.050   9  3.11  0.866     2.01
#>   0.050  10  3.10  0.867     2.01
#>   0.100   1  3.68  0.829     2.26
#>   0.100   2  3.37  0.848     2.12
#>   0.100   3  3.23  0.858     2.06
#>   0.100   4  3.17  0.862     2.04
#>   0.100   5  3.14  0.865     2.04
#>   0.100   6  3.11  0.866     2.04
#>   0.100   7  3.09  0.868     2.04
#>   0.100   8  3.09  0.868     2.04
#>   0.100   9  3.08  0.868     2.04
#>   0.100  10  3.08  0.868     2.05
#>   0.150   1  3.79  0.822     2.30
#>   0.150   2  3.42  0.846     2.14
#>   0.150   3  3.30  0.854     2.09
#>   0.150   4  3.26  0.857     2.09
#>   0.150   5  3.24  0.858     2.09
#>   0.150   6  3.23  0.858     2.10
#>   0.150   7  3.23  0.857     2.12
#>   0.150   8  3.24  0.856     2.13
#>   0.150   9  3.26  0.855     2.15
#>   0.150  10  3.27  0.854     2.17
#>
#> RMSE was used to select the optimal model using the smallest value.
#> The final values used for the model were sigma = 0.1 and C = 9.
plot(fit.svm)
```

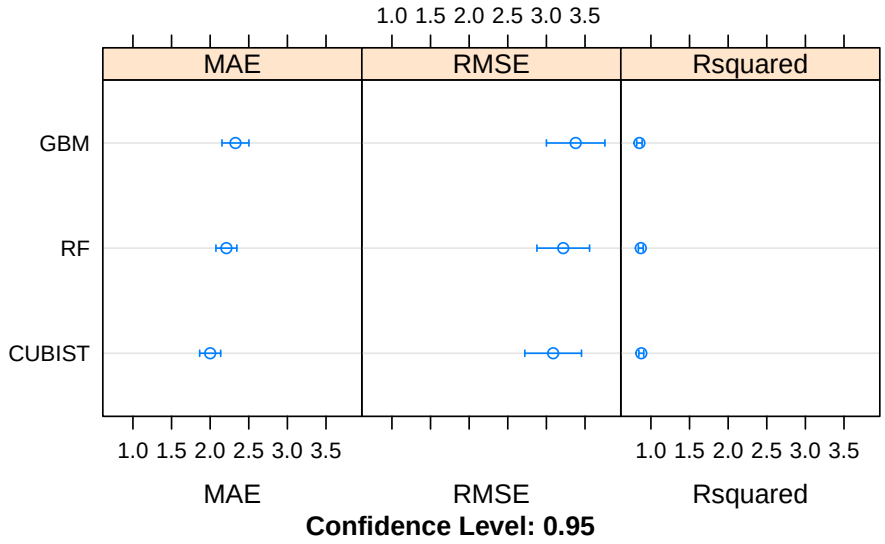


4.8 Ensembling

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     1.64    1.98   2.20 2.21    2.30 3.22    0
#> GBM    1.65    1.98   2.27 2.33    2.55 3.75    0
#> CUBIST 1.31    1.75   1.95 2.00    2.17 2.89    0
#>
#> RMSE
#>        Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
#> RF     2.11    2.58   3.08 3.22    3.70 6.52    0
#> GBM    1.92    2.54   3.31 3.38    3.67 6.85    0
#> CUBIST 1.79    2.38   2.74 3.09    3.78 5.79    0
#>
#> Rsquared
#>         Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
#> RF     0.597   0.852  0.900 0.869   0.919 0.972    0
#> GBM    0.558   0.815  0.889 0.851   0.912 0.964    0
#> CUBIST 0.681   0.818  0.909 0.875   0.932 0.970    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).

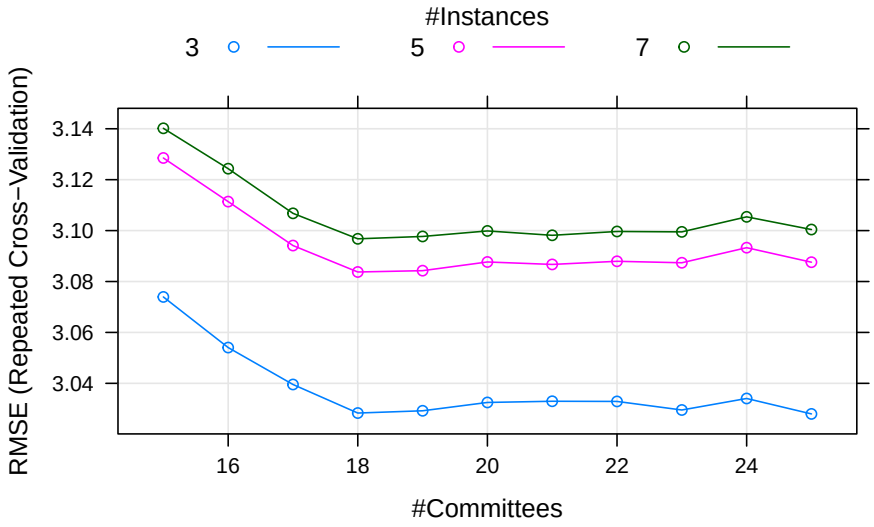
```
# look at parameters used for Cubist
print(fit.cubist)
#> Cubist
#>
#> 407 samples
#>  13 predictor
#>
#> Pre-processing: Box-Cox transformation (11)
#> Resampling: Cross-Validated (10 fold, repeated 3 times)
#> Summary of sample sizes: 365, 366, 366, 367, 366, 366, ...
#> Resampling results across tuning parameters:
#>
#>   committees  neighbors  RMSE  Rsquared  MAE
#>    1          0          3.94  0.805     2.50
#>    1          5          3.66  0.828     2.24
#>    1          9          3.69  0.825     2.26
#>   10          0          3.45  0.848     2.29
#>   10          5          3.19  0.868     2.04
#>   10          9          3.23  0.864     2.07
#>   20          0          3.34  0.858     2.25
#>   20          5          3.09  0.875     2.00
#>   20          9          3.12  0.872     2.03
#>
#> 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: 365, 366, 366, 367, 366, 366, ...
#> Resampling results across tuning parameters:
#>
#>   committees  neighbors  RMSE  Rsquared  MAE
#>   15          3          3.07  0.877     2.00
#>   15          5          3.13  0.873     2.02
#>   15          7          3.14  0.871     2.03
#>   16          3          3.05  0.878     1.99
#>   16          5          3.11  0.874     2.01
#>   16          7          3.12  0.872     2.02
#>   17          3          3.04  0.879     1.98
#>   17          5          3.09  0.875     2.00
#>   17          7          3.11  0.873     2.01
#>   18          3          3.03  0.880     1.97
#>   18          5          3.08  0.876     2.00
#>   18          7          3.10  0.874     2.01
#>   19          3          3.03  0.880     1.97
#>   19          5          3.08  0.876     1.99
#>   19          7          3.10  0.874     2.01
#>   20          3          3.03  0.879     1.98
#>   20          5          3.09  0.875     2.00
#>   20          7          3.10  0.874     2.01
#>   21          3          3.03  0.879     1.98
#>   21          5          3.09  0.876     2.00
#>   21          7          3.10  0.874     2.02
#>   22          3          3.03  0.879     1.98
#>   22          5          3.09  0.875     2.00
#>   22          7          3.10  0.874     2.02
#>   23          3          3.03  0.880     1.98
#>   23          5          3.09  0.876     2.01
#>   23          7          3.10  0.874     2.02
#>   24          3          3.03  0.879     1.98
#>   24          5          3.09  0.875     2.01
#>   24          7          3.11  0.873     2.02
#>   25          3          3.03  0.880     1.98
#>   25          5          3.09  0.876     2.01
#>   25          7          3.10  0.874     2.02
```

```
#>
#> RMSE was used to select the optimal model using the smallest value.
#> The final values used for the model were committees = 25 and neighbors = 3.
plot(tune.cubist)
```



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

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

4.9 Finalize the model

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

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

# train the final model
finalModel <- cubist(x = transX, y=y, committees=18)
summary(finalModel)
#>
#> Call:
#> cubist.default(x = transX, y = y, committees = 18)
#>
#>
#> Cubist [Release 2.07 GPL Edition]  Wed Sep 18 13:36:54 2019
#> -------------------------------
#>
#>     Target attribute `outcome'
```

```
#>
#> Read 407 cases (14 attributes) from undefined.data
#>
#> Model 1:
#>
#>   Rule 1/1: [84 cases, mean 14.29, range 5 to 27.5, est err 1.97]
#>
#>     if
#>  nox > -0.4864544
#>     then
#>  outcome = 35.08 - 2.45 crim - 4.31 lstat + 2.1e-05 b
#>
#>   Rule 1/2: [163 cases, mean 19.37, range 7 to 31, est err 2.10]
#>
#>     if
#>  nox <= -0.4864544
#>  lstat > 2.848535
#>     then
#>  outcome = 186.8 - 2.34 lstat - 3.3 dis - 88 tax + 2 rad + 4.4 rm
#>            - 0.033 ptratio - 0.0116 age + 3.3e-05 b
#>
#>   Rule 1/3: [24 cases, mean 21.65, range 18.2 to 25.3, est err 1.19]
#>
#>     if
#>  rm <= 3.326479
#>  dis > 1.345056
#>  lstat <= 2.848535
#>     then
#>  outcome = 43.83 + 14.5 rm - 2.29 lstat - 3.8 dis - 30 tax
#>            - 0.014 ptratio - 1.4 nox + 0.017 zn + 0.4 rad + 0.15 crim
#>            - 0.0025 age + 8e-06 b
#>
#>   Rule 1/4: [7 cases, mean 27.66, range 20.7 to 50, est err 7.89]
#>
#>     if
#>  rm > 3.326479
#>  ptratio > 193.545
#>  lstat <= 2.848535
#>     then
#>  outcome = 19.64 + 7.8 rm - 3.4 dis - 1.62 lstat + 0.27 crim - 0.006 age
#>            + 0.023 zn - 7 tax - 0.003 ptratio
#>
#>   Rule 1/5: [141 cases, mean 30.60, range 15 to 50, est err 2.09]
#>
#>     if
#>  rm > 3.326479
#>  ptratio <= 193.545
#>     then
#>  outcome = 137.95 + 21.7 rm - 3.43 lstat - 4.9 dis - 87 tax - 0.0162 age
#>            - 0.039 ptratio + 0.06 crim + 0.005 zn
#>
#>   Rule 1/6: [8 cases, mean 32.16, range 22.1 to 50, est err 8.67]
#>
```

```
#>     if
#>  rm <= 3.326479
#>  dis <= 1.345056
#>  lstat <= 2.848535
#>     then
#>  outcome = -19.71 + 18.58 lstat - 15.9 dis + 5.6 rm
#>
#> Model 2:
#>
#>   Rule 2/1: [23 cases, mean 10.57, range 5 to 15, est err 3.06]
#>
#>     if
#>  crim > 2.086391
#>  dis <= 0.6604174
#>  b > 67032.41
#>     then
#>  outcome = 37.22 - 4.83 crim - 7 dis - 1.9 lstat - 1.9e-05 b - 0.7 rm
#>
#>   Rule 2/2: [70 cases, mean 14.82, range 5 to 50, est err 3.90]
#>
#>     if
#>  rm <= 3.620525
#>  dis <= 0.6604174
#>     then
#>  outcome = 74.6 - 21 dis - 5.09 lstat - 15 tax - 0.0017 age + 6e-06 b
#>
#>   Rule 2/3: [18 cases, mean 18.03, range 7.5 to 50, est err 6.81]
#>
#>     if
#>  crim > 2.086391
#>  dis <= 0.6604174
#>  b <= 67032.41
#>     then
#>  outcome = 94.95 - 40.1 dis - 8.15 crim - 7.14 lstat - 3.5e-05 b - 1.3 rm
#>
#>   Rule 2/4: [258 cases, mean 20.74, range 9.5 to 36.2, est err 1.92]
#>
#>     if
#>  rm <= 3.620525
#>  dis > 0.6604174
#>  lstat > 1.805082
#>     then
#>  outcome = 61.89 - 2.56 lstat + 5.5 rm - 2.8 dis + 7.3e-05 b - 0.0132 age
#>            - 26 tax - 0.11 indus - 0.004 ptratio + 0.05 crim
#>
#>   Rule 2/5: [37 cases, mean 31.66, range 10.4 to 50, est err 3.70]
#>
#>     if
#>  rm > 3.620525
#>  lstat > 1.805082
#>     then
#>  outcome = 370.03 - 180 tax - 2.19 lstat - 1.7 dis + 2.6 rm
#>            - 0.016 ptratio - 0.25 indus + 0.12 crim - 0.0021 age
```

```
#>            + 9e-06 b - 0.5 nox
#>
#>   Rule 2/6: [42 cases, mean 38.23, range 22.8 to 50, est err 3.70]
#>
#>     if
#>  lstat <= 1.805082
#>     then
#>  outcome = -73.87 + 32.4 rm - 9.4e-05 b - 1.8 dis + 0.028 zn
#>            - 0.013 ptratio
#>
#>   Rule 2/7: [4 cases, mean 40.20, range 37.6 to 42.8, est err 7.33]
#>
#>     if
#>  rm > 4.151791
#>  dis > 1.114486
#>     then
#>  outcome = 35.8
#>
#>   Rule 2/8: [8 cases, mean 47.45, range 41.3 to 50, est err 10.01]
#>
#>     if
#>  dis <= 1.114486
#>  lstat <= 1.805082
#>     then
#>  outcome = 48.96 + 7.53 crim - 4.1e-05 b - 0.8 dis + 1.2 rm + 0.008 zn
#>
#> Model 3:
#>
#>   Rule 3/1: [81 cases, mean 13.93, range 5 to 23.2, est err 2.24]
#>
#>     if
#>  nox > -0.4864544
#>  lstat > 2.848535
#>     then
#>  outcome = 55.03 - 0.0631 age - 2.11 crim + 12 nox - 4.16 lstat
#>            + 3.2e-05 b
#>
#>   Rule 3/2: [163 cases, mean 19.37, range 7 to 31, est err 2.29]
#>
#>     if
#>  nox <= -0.4864544
#>  lstat > 2.848535
#>     then
#>  outcome = 77.73 - 0.059 ptratio + 5.8 rm - 3.2 dis - 0.0139 age
#>            - 1.15 lstat - 30 tax - 1.1 nox + 0.4 rad
#>
#>   Rule 3/3: [62 cases, mean 24.01, range 18.2 to 50, est err 3.56]
#>
#>     if
#>  rm <= 3.448196
#>  lstat <= 2.848535
#>     then
#>  outcome = 94.86 + 18.2 rm + 0.63 crim - 68 tax - 2.3 dis - 3 nox
```

```
#>            - 0.0098 age - 0.41 indus - 0.011 ptratio
#>
#>   Rule 3/4: [143 cases, mean 28.76, range 16.5 to 50, est err 2.53]
#>
#>     if
#>  dis > 0.9547035
#>  lstat <= 2.848535
#>     then
#>  outcome = 269.46 + 17.9 rm - 6.1 dis - 153 tax + 0.96 crim - 0.0217 age
#>            - 5.5 nox - 0.62 indus - 0.028 ptratio - 0.89 lstat + 0.4 rad
#>            + 0.004 zn
#>
#>   Rule 3/5: [10 cases, mean 35.13, range 21.9 to 50, est err 9.31]
#>
#>     if
#>  dis <= 0.6492998
#>  lstat <= 2.848535
#>     then
#>  outcome = 58.69 - 56.8 dis - 8.4 nox
#>
#>   Rule 3/6: [10 cases, mean 41.67, range 22 to 50, est err 9.89]
#>
#>     if
#>  dis > 0.6492998
#>  dis <= 0.9547035
#>  lstat <= 2.848535
#>     then
#>  outcome = 47.93
#>
#> Model 4:
#>
#>   Rule 4/1: [69 cases, mean 12.69, range 5 to 27.5, est err 2.55]
#>
#>     if
#>  dis <= 0.719156
#>  lstat > 3.508535
#>     then
#>  outcome = 180.13 - 7.2 dis + 0.039 age - 3.78 lstat - 83 tax
#>
#>   Rule 4/2: [164 cases, mean 19.42, range 12 to 31, est err 1.96]
#>
#>     if
#>  dis > 0.719156
#>  lstat > 2.848535
#>     then
#>  outcome = 52.75 + 7.1 rm - 2.05 lstat - 3.6 dis + 8.2e-05 b - 0.0152 age
#>            - 25 tax + 0.5 rad - 1.2 nox - 0.008 ptratio
#>
#>   Rule 4/3: [11 cases, mean 20.39, range 15 to 27.9, est err 3.51]
#>
#>     if
#>  dis <= 0.719156
#>  lstat > 2.848535
```

```
#>   lstat <= 3.508535
#>      then
#>   outcome = 21.69
#>
#>    Rule 4/4: [63 cases, mean 23.22, range 16.5 to 31.5, est err 1.67]
#>
#>      if
#>   rm <= 3.483629
#>   dis > 0.9731624
#>   lstat <= 2.848535
#>      then
#>   outcome = 59.35 - 3.96 lstat - 3.1 dis + 1 rm - 14 tax + 0.3 rad
#>             - 0.7 nox - 0.005 ptratio + 6e-06 b
#>
#>    Rule 4/5: [8 cases, mean 33.08, range 22 to 50, est err 23.91]
#>
#>      if
#>   rm > 3.369183
#>   dis <= 0.9731624
#>   lstat > 2.254579
#>   lstat <= 2.848535
#>      then
#>   outcome = -322.28 + 64.9 lstat + 56.8 rm - 30.2 dis
#>
#>    Rule 4/6: [7 cases, mean 33.87, range 22.1 to 50, est err 13.21]
#>
#>      if
#>   rm <= 3.369183
#>   dis <= 0.9731624
#>   lstat <= 2.848535
#>      then
#>   outcome = -52.11 + 43.45 lstat - 30.8 dis
#>
#>    Rule 4/7: [91 cases, mean 34.43, range 21.9 to 50, est err 3.32]
#>
#>      if
#>   rm > 3.483629
#>   lstat <= 2.848535
#>      then
#>   outcome = -33.09 + 22 rm - 5.02 lstat - 0.038 ptratio - 0.9 dis
#>             + 0.005 zn
#>
#>    Rule 4/8: [22 cases, mean 36.99, range 21.9 to 50, est err 13.21]
#>
#>      if
#>   dis <= 0.9731624
#>   lstat <= 2.848535
#>      then
#>   outcome = 80.3 - 17.43 lstat - 0.134 ptratio + 2.5 rm - 1.2 dis
#>             + 0.008 zn
#>
#> Model 5:
#>
```

```
#>   Rule 5/1: [84 cases, mean 14.29, range 5 to 27.5, est err 2.81]
#>
#>     if
#>  nox > -0.4864544
#>     then
#>  outcome = 56.48 + 28.5 nox - 0.0875 age - 3.58 crim - 5.9 dis
#>            - 2.96 lstat + 0.073 ptratio + 1.7e-05 b
#>
#>   Rule 5/2: [163 cases, mean 19.37, range 7 to 31, est err 2.38]
#>
#>     if
#>  nox <= -0.4864544
#>  lstat > 2.848535
#>     then
#>  outcome = 61.59 - 0.064 ptratio + 5.9 rm - 3.1 dis - 0.0142 age
#>            - 0.77 lstat - 21 tax
#>
#>   Rule 5/3: [163 cases, mean 29.94, range 16.5 to 50, est err 3.65]
#>
#>     if
#>  lstat <= 2.848535
#>     then
#>  outcome = 264.17 + 21.9 rm - 8 dis - 155 tax - 0.0317 age
#>            - 0.032 ptratio + 0.29 crim - 1.6 nox - 0.25 indus
#>
#>   Rule 5/4: [10 cases, mean 35.13, range 21.9 to 50, est err 11.79]
#>
#>     if
#>  dis <= 0.6492998
#>  lstat <= 2.848535
#>     then
#>  outcome = 68.19 - 73.4 dis + 1.1 rm + 0.11 crim - 0.6 nox - 0.1 indus
#>            - 0.0017 age - 0.12 lstat
#>
#> Model 6:
#>
#>   Rule 6/1: [71 cases, mean 15.57, range 5 to 50, est err 4.42]
#>
#>     if
#>  dis <= 0.6443245
#>  lstat > 1.793385
#>     then
#>  outcome = 45.7 - 20.6 dis - 5.38 lstat
#>
#>   Rule 6/2: [159 cases, mean 19.53, range 8.3 to 36.2, est err 2.08]
#>
#>     if
#>  rm <= 3.329365
#>  dis > 0.6443245
#>     then
#>  outcome = 24.33 + 8.8 rm + 0.000118 b - 0.0146 age - 2.5 dis
#>            - 0.95 lstat + 0.37 crim - 0.32 indus + 0.02 zn - 16 tax
#>            + 0.2 rad - 0.5 nox - 0.004 ptratio
```

```
#>
#>   Rule 6/3: [175 cases, mean 27.80, range 9.5 to 50, est err 2.95]
#>
#>     if
#>  rm > 3.329365
#>  dis > 0.6443245
#>     then
#>  outcome = 0.11 + 18.7 rm - 3.11 lstat + 8.1e-05 b - 1.1 dis + 0.19 crim
#>            - 20 tax - 0.19 indus + 0.3 rad - 0.7 nox - 0.005 ptratio
#>            + 0.006 zn
#>
#>   Rule 6/4: [8 cases, mean 32.50, range 21.9 to 50, est err 10.34]
#>
#>     if
#>  dis <= 0.6443245
#>  lstat > 1.793385
#>  lstat <= 2.894121
#>     then
#>  outcome = 69.38 - 71.2 dis - 0.14 lstat
#>
#>   Rule 6/5: [34 cases, mean 37.55, range 22.8 to 50, est err 3.55]
#>
#>     if
#>  rm <= 4.151791
#>  lstat <= 1.793385
#>     then
#>  outcome = -125.14 + 41.7 rm + 4.3 rad + 1.48 indus - 0.014 ptratio
#>
#>   Rule 6/6: [7 cases, mean 43.66, range 37.6 to 50, est err 3.12]
#>
#>     if
#>  rm > 4.151791
#>  lstat <= 1.793385
#>     then
#>  outcome = -137.67 + 44.6 rm - 0.064 ptratio
#>
#> Model 7:
#>
#>   Rule 7/1: [84 cases, mean 14.29, range 5 to 27.5, est err 2.91]
#>
#>     if
#>  nox > -0.4864544
#>     then
#>  outcome = 46.85 - 3.45 crim - 0.0621 age + 14.2 nox + 4.4 dis
#>            - 2.01 lstat + 2.5e-05 b
#>
#>   Rule 7/2: [323 cases, mean 24.66, range 7 to 50, est err 3.68]
#>
#>     if
#>  nox <= -0.4864544
#>     then
#>  outcome = 57.59 - 0.065 ptratio - 4.4 dis + 6.8 rm - 0.0143 age
#>            - 1.36 lstat - 19 tax - 0.8 nox - 0.12 crim + 0.09 indus
```

```
#>
#>   Rule 7/3: [132 cases, mean 28.24, range 16.5 to 50, est err 2.55]
#>
#>     if
#>  dis > 1.063503
#>  lstat <= 2.848535
#>     then
#>  outcome = 270.92 + 24.5 rm - 0.0418 age - 165 tax - 5.7 dis
#>            - 0.028 ptratio + 0.26 crim + 0.017 zn
#>
#>   Rule 7/4: [7 cases, mean 36.01, range 23.3 to 50, est err 3.87]
#>
#>     if
#>  dis <= 0.6002641
#>  lstat <= 2.848535
#>     then
#>  outcome = 57.18 - 69.5 dis - 6.5 nox + 1.9 rm - 0.015 ptratio
#>
#>   Rule 7/5: [24 cases, mean 37.55, range 21.9 to 50, est err 8.66]
#>
#>     if
#>  dis > 0.6002641
#>  dis <= 1.063503
#>  lstat <= 2.848535
#>     then
#>  outcome = -3.76 - 14.8 dis - 2.93 crim - 0.16 ptratio + 17.5 rm - 15 nox
#>
#> Model 8:
#>
#>   Rule 8/1: [80 cases, mean 13.75, range 5 to 27.9, est err 3.51]
#>
#>     if
#>  dis <= 0.719156
#>  lstat > 2.848535
#>     then
#>  outcome = 123.46 - 11.3 dis - 5.06 lstat - 45 tax + 0.9 rad + 1.7e-05 b
#>
#>   Rule 8/2: [164 cases, mean 19.42, range 12 to 31, est err 2.05]
#>
#>     if
#>  dis > 0.719156
#>  lstat > 2.848535
#>     then
#>  outcome = 227.11 - 120 tax + 6.4 rm + 9.3e-05 b - 3.3 dis + 2 rad
#>            - 0.0183 age - 0.93 lstat + 0.05 crim - 0.3 nox
#>
#>   Rule 8/3: [163 cases, mean 29.94, range 16.5 to 50, est err 3.54]
#>
#>     if
#>  lstat <= 2.848535
#>     then
#>  outcome = 158.14 - 5.73 lstat + 10.8 rm - 4 dis - 83 tax - 4.1 nox
#>            + 0.61 crim - 0.54 indus + 1 rad + 3.6e-05 b
```

```
#>
#>   Rule 8/4: [7 cases, mean 36.01, range 23.3 to 50, est err 11.44]
#>
#>     if
#>  dis <= 0.6002641
#>  lstat <= 2.848535
#>     then
#>  outcome = 72.89 - 87.2 dis + 0.6 rm - 0.13 lstat
#>
#>   Rule 8/5: [47 cases, mean 38.44, range 15 to 50, est err 5.71]
#>
#>     if
#>  rm > 3.726352
#>     then
#>  outcome = 602.95 - 10.4 lstat + 21 rm - 326 tax - 0.093 ptratio
#>
#> Model 9:
#>
#>   Rule 9/1: [81 cases, mean 13.93, range 5 to 23.2, est err 2.91]
#>
#>     if
#>  nox > -0.4864544
#>  lstat > 2.848535
#>     then
#>  outcome = 41.11 - 3.98 crim - 4.42 lstat + 6.7 nox
#>
#>   Rule 9/2: [163 cases, mean 19.37, range 7 to 31, est err 2.49]
#>
#>     if
#>  nox <= -0.4864544
#>  lstat > 2.848535
#>     then
#>  outcome = 44.98 - 0.068 ptratio - 4.4 dis + 6.6 rm - 1.25 lstat
#>            - 0.0118 age - 0.9 nox - 12 tax - 0.08 crim + 0.06 indus
#>
#>   Rule 9/3: [132 cases, mean 28.24, range 16.5 to 50, est err 2.35]
#>
#>     if
#>  dis > 1.063503
#>  lstat <= 2.848535
#>     then
#>  outcome = 157.67 + 22.2 rm - 0.0383 age - 104 tax - 0.033 ptratio
#>            - 2.2 dis
#>
#>   Rule 9/4: [7 cases, mean 30.76, range 21.9 to 50, est err 6.77]
#>
#>     if
#>  dis <= 1.063503
#>  b <= 66469.73
#>  lstat <= 2.848535
#>     then
#>  outcome = 48.52 - 56.1 dis - 12.9 nox - 0.032 ptratio + 2.7 rm
#>
```

```
#>   Rule 9/5: [24 cases, mean 39.09, range 22 to 50, est err 6.20]
#>
#>     if
#>  dis <= 1.063503
#>  b > 66469.73
#>  lstat <= 2.848535
#>     then
#>  outcome = -5.49 - 34.8 dis - 20.7 nox + 18.2 rm - 0.051 ptratio
#>
#> Model 10:
#>
#>   Rule 10/1: [327 cases, mean 19.45, range 5 to 50, est err 2.77]
#>
#>     if
#>  rm <= 3.617282
#>  lstat > 1.805082
#>     then
#>  outcome = 270.78 - 4.09 lstat - 131 tax + 2.9 rad + 5.3e-05 b - 0.6 dis
#>            - 0.16 indus + 0.7 rm - 0.3 nox
#>
#>   Rule 10/2: [38 cases, mean 31.57, range 10.4 to 50, est err 4.71]
#>
#>     if
#>  rm > 3.617282
#>  lstat > 1.805082
#>     then
#>  outcome = 308.44 - 150 tax - 2.63 lstat + 1.6 rad - 1.9 dis - 0.49 indus
#>            + 2.5 rm + 3e-05 b - 1.2 nox + 0.14 crim - 0.005 ptratio
#>
#>   Rule 10/3: [35 cases, mean 37.15, range 22.8 to 50, est err 2.76]
#>
#>     if
#>  rm <= 4.151791
#>  lstat <= 1.805082
#>     then
#>  outcome = -71.65 + 33.4 rm - 0.017 ptratio - 0.34 lstat + 0.2 rad
#>            - 0.3 dis - 7 tax - 0.4 nox
#>
#>   Rule 10/4: [10 cases, mean 42.63, range 21.9 to 50, est err 7.11]
#>
#>     if
#>  rm > 4.151791
#>     then
#>  outcome = -92.51 + 32.8 rm - 0.03 ptratio
#>
#> Model 11:
#>
#>   Rule 11/1: [84 cases, mean 14.29, range 5 to 27.5, est err 4.13]
#>
#>     if
#>  nox > -0.4864544
#>     then
#>  outcome = 42.75 - 4.12 crim + 18.1 nox - 0.045 age + 6.8 dis
```

```
#>            - 1.86 lstat
#>
#>   Rule 11/2: [244 cases, mean 17.56, range 5 to 31, est err 4.29]
#>
#>     if
#>  lstat > 2.848535
#>     then
#>  outcome = 34.83 - 5.2 dis - 0.058 ptratio - 0.0228 age + 5.8 rm
#>            - 0.56 lstat - 0.07 crim - 0.4 nox - 5 tax
#>
#>   Rule 11/3: [163 cases, mean 29.94, range 16.5 to 50, est err 3.49]
#>
#>     if
#>  lstat <= 2.848535
#>     then
#>  outcome = 151.5 + 23.3 rm - 5.5 dis + 1.01 crim - 0.0211 age
#>            - 0.052 ptratio - 98 tax + 0.031 zn
#>
#>   Rule 11/4: [10 cases, mean 35.13, range 21.9 to 50, est err 25.19]
#>
#>     if
#>  dis <= 0.6492998
#>  lstat <= 2.848535
#>     then
#>  outcome = 130.87 - 157.1 dis - 15.76 crim
#>
#> Model 12:
#>
#>   Rule 12/1: [80 cases, mean 13.75, range 5 to 27.9, est err 4.76]
#>
#>     if
#>  dis <= 0.719156
#>  lstat > 2.894121
#>     then
#>  outcome = 182.68 - 6.03 lstat - 7.6 dis - 76 tax + 1.3 rad - 0.52 indus
#>            + 2.6e-05 b
#>
#>   Rule 12/2: [300 cases, mean 19.10, range 5 to 50, est err 2.76]
#>
#>     if
#>  rm <= 3.50716
#>  lstat > 1.793385
#>     then
#>  outcome = 83.61 - 3 lstat + 9.6e-05 b - 0.0072 age - 33 tax + 0.7 rad
#>            + 0.32 indus
#>
#>   Rule 12/3: [10 cases, mean 24.25, range 15.7 to 36.2, est err 13.88]
#>
#>     if
#>  rm <= 3.50716
#>  tax <= 1.865769
#>     then
#>  outcome = 35.46
```

```
#>
#>   Rule 12/4: [10 cases, mean 32.66, range 21.9 to 50, est err 6.28]
#>
#>     if
#>  dis <= 0.719156
#>  lstat > 1.793385
#>  lstat <= 2.894121
#>     then
#>  outcome = 82.78 - 69.5 dis - 3.66 indus
#>
#>   Rule 12/5: [89 cases, mean 32.75, range 13.4 to 50, est err 3.39]
#>
#>     if
#>  rm > 3.50716
#>  dis > 0.719156
#>     then
#>  outcome = 313.22 + 13.7 rm - 174 tax - 3.06 lstat + 4.8e-05 b - 1.5 dis
#>            - 0.41 indus + 0.7 rad - 0.0055 age + 0.22 crim
#>
#>   Rule 12/6: [34 cases, mean 37.55, range 22.8 to 50, est err 3.25]
#>
#>     if
#>  rm <= 4.151791
#>  lstat <= 1.793385
#>     then
#>  outcome = -86.8 + 36 rm - 0.3 lstat - 5 tax
#>
#>   Rule 12/7: [7 cases, mean 43.66, range 37.6 to 50, est err 5.79]
#>
#>     if
#>  rm > 4.151791
#>  lstat <= 1.793385
#>     then
#>  outcome = -158.68 + 47.4 rm - 0.02 ptratio
#>
#> Model 13:
#>
#>   Rule 13/1: [84 cases, mean 14.29, range 5 to 27.5, est err 2.87]
#>
#>     if
#>  nox > -0.4864544
#>     then
#>  outcome = 54.69 - 3.79 crim - 0.0644 age + 11.4 nox - 2.53 lstat
#>
#>   Rule 13/2: [8 cases, mean 17.76, range 7 to 27.9, est err 13.69]
#>
#>     if
#>  nox <= -0.4864544
#>  age > 296.3423
#>  b <= 60875.57
#>     then
#>  outcome = -899.55 + 3.0551 age
#>
```

```
#>   Rule 13/3: [31 cases, mean 17.94, range 7 to 27.9, est err 5.15]
#>
#>     if
#>  nox <= -0.4864544
#>  b <= 60875.57
#>  lstat > 2.848535
#>     then
#>  outcome = 44.43 - 3.51 lstat - 0.054 ptratio - 1.4 dis - 0.26 crim
#>            - 0.0042 age - 0.21 indus + 0.9 rm
#>
#>   Rule 13/4: [163 cases, mean 19.37, range 7 to 31, est err 3.37]
#>
#>     if
#>  nox <= -0.4864544
#>  lstat > 2.848535
#>     then
#>  outcome = -5.76 + 0.000242 b + 8.9 rm - 5.2 dis - 0.0209 age
#>            - 0.042 ptratio - 0.63 indus
#>
#>   Rule 13/5: [163 cases, mean 29.94, range 16.5 to 50, est err 3.45]
#>
#>     if
#>  lstat <= 2.848535
#>     then
#>  outcome = 178.84 + 23.8 rm - 0.0343 age - 4.5 dis - 114 tax + 0.88 crim
#>            - 0.048 ptratio + 0.026 zn
#>
#>   Rule 13/6: [7 cases, mean 36.01, range 23.3 to 50, est err 14.09]
#>
#>     if
#>  dis <= 0.6002641
#>  lstat <= 2.848535
#>     then
#>  outcome = 45.82 - 70.3 dis - 9.9 nox + 5.1 rm + 1.5 rad
#>
#>   Rule 13/7: [31 cases, mean 37.21, range 21.9 to 50, est err 7.73]
#>
#>     if
#>  dis <= 1.063503
#>  lstat <= 2.848535
#>     then
#>  outcome = 95.05 - 4.52 lstat - 7.5 dis + 8.8 rm - 0.064 ptratio
#>            - 6.2 nox - 36 tax
#>
#> Model 14:
#>
#>   Rule 14/1: [49 cases, mean 16.06, range 8.4 to 22.7, est err 3.17]
#>
#>     if
#>  nox > -0.4205732
#>  lstat > 2.848535
#>     then
#>  outcome = 12.83 + 42.3 nox - 4.77 lstat + 9.7 rm + 7.8e-05 b
```

```
#>
#>   Rule 14/2: [78 cases, mean 16.36, range 5 to 50, est err 5.17]
#>
#>     if
#>  dis <= 0.6604174
#>     then
#>  outcome = 110.6 - 10.4 dis - 4.85 lstat + 0.0446 age - 46 tax + 0.8 rad
#>
#>   Rule 14/3: [57 cases, mean 18.40, range 9.5 to 31, est err 2.43]
#>
#>     if
#>  nox > -0.9365134
#>  nox <= -0.4205732
#>  age > 245.2507
#>  dis > 0.6604174
#>  lstat > 2.848535
#>     then
#>  outcome = 206.69 - 0.1012 age - 7.05 lstat + 12.2 nox - 67 tax + 0.3 rad
#>            + 0.5 rm - 0.3 dis
#>
#>   Rule 14/4: [230 cases, mean 20.19, range 9.5 to 36.2, est err 2.09]
#>
#>     if
#>  rm <= 3.483629
#>  dis > 0.6492998
#>     then
#>  outcome = 119.15 - 2.61 lstat + 5.2 rm - 57 tax - 1.8 dis - 2.4 nox
#>            + 0.7 rad + 0.24 crim + 0.003 age - 0.007 ptratio + 9e-06 b
#>
#>   Rule 14/5: [48 cases, mean 20.28, range 10.2 to 24.5, est err 2.13]
#>
#>     if
#>  nox > -0.9365134
#>  nox <= -0.4205732
#>  age <= 245.2507
#>  dis > 0.6604174
#>  lstat > 2.848535
#>     then
#>  outcome = 19.4 - 1.91 lstat + 1.02 indus - 0.013 age + 2.7 rm + 2.6 nox
#>            - 0.009 ptratio
#>
#>   Rule 14/6: [44 cases, mean 20.69, range 14.4 to 29.6, est err 2.26]
#>
#>     if
#>  nox <= -0.9365134
#>  lstat > 2.848535
#>     then
#>  outcome = 87.55 - 0.000315 b - 6.5 dis + 2.6 rad - 0.59 lstat - 18 tax
#>
#>   Rule 14/7: [102 cases, mean 32.44, range 13.4 to 50, est err 3.35]
#>
#>     if
#>  rm > 3.483629
```

```
#>   dis > 0.6492998
#>      then
#>   outcome = 126.92 + 22.7 rm - 4.68 lstat - 85 tax - 0.036 ptratio
#>             - 1.1 dis + 0.007 zn
#>
#>    Rule 14/8: [84 cases, mean 33.40, range 21 to 50, est err 2.44]
#>
#>      if
#>   rm > 3.483629
#>   tax <= 1.896025
#>      then
#>   outcome = 347.12 + 25.2 rm - 213 tax - 3.5 lstat - 0.013 ptratio
#>
#>    Rule 14/9: [10 cases, mean 35.13, range 21.9 to 50, est err 12.13]
#>
#>      if
#>   dis <= 0.6492998
#>   lstat <= 2.848535
#>      then
#>   outcome = 72.65 - 77.8 dis
#>
#> Model 15:
#>
#>    Rule 15/1: [28 cases, mean 12.35, range 5 to 27.9, est err 4.09]
#>
#>      if
#>   crim > 2.405809
#>   b > 16084.5
#>      then
#>   outcome = 53.45 - 7.8 crim - 3.5 lstat - 0.0189 age
#>
#>    Rule 15/2: [11 cases, mean 13.56, range 8.3 to 27.5, est err 5.99]
#>
#>      if
#>   crim > 2.405809
#>   b <= 16084.5
#>      then
#>   outcome = 8.73 + 0.001756 b
#>
#>    Rule 15/3: [244 cases, mean 17.56, range 5 to 31, est err 2.73]
#>
#>      if
#>   lstat > 2.848535
#>      then
#>   outcome = 103.02 - 0.0251 age - 2.37 lstat - 3.5 dis + 6.8e-05 b + 4 rm
#>             - 0.035 ptratio - 41 tax - 0.25 crim
#>
#>    Rule 15/4: [131 cases, mean 28.22, range 16.5 to 50, est err 2.59]
#>
#>      if
#>   dis > 1.086337
#>   lstat <= 2.848535
#>      then
```

```
#>   outcome = 267.07 + 17.7 rm - 0.0421 age - 150 tax - 5.5 dis + 0.88 crim
#>             - 0.035 ptratio + 0.031 zn - 0.12 lstat - 0.3 nox
#>
#>    Rule 15/5: [13 cases, mean 33.08, range 22 to 50, est err 4.44]
#>
#>      if
#>   nox <= -0.7229691
#>   dis <= 1.086337
#>   lstat <= 2.848535
#>      then
#>   outcome = 148.52 - 0.002365 b - 85.9 nox - 1 dis + 0.16 crim + 0.8 rm
#>             + 0.007 zn - 0.0016 age - 7 tax - 0.003 ptratio
#>
#>    Rule 15/6: [7 cases, mean 36.01, range 23.3 to 50, est err 7.00]
#>
#>      if
#>   dis <= 0.6002641
#>   lstat <= 2.848535
#>      then
#>   outcome = 50.55 - 68.1 dis - 11.4 nox + 0.00012 b + 1 rm - 0.008 ptratio
#>
#>    Rule 15/7: [12 cases, mean 41.77, range 21.9 to 50, est err 9.73]
#>
#>      if
#>   nox > -0.7229691
#>   dis > 0.6002641
#>   lstat <= 2.848535
#>      then
#>   outcome = 13.74 - 92 nox - 40.5 dis - 0.023 ptratio + 2.6 rm
#>
#> Model 16:
#>
#>    Rule 16/1: [60 cases, mean 15.95, range 7.2 to 27.5, est err 3.16]
#>
#>      if
#>   nox > -0.4344906
#>      then
#>   outcome = 46.98 - 6.53 lstat - 6.9 dis - 1.1 rm
#>
#>    Rule 16/2: [45 cases, mean 16.89, range 5 to 50, est err 5.45]
#>
#>      if
#>   nox <= -0.4344906
#>   dis <= 0.6557049
#>      then
#>   outcome = 35.33 - 37 dis - 51.7 nox - 7.38 lstat - 0.4 rm
#>
#>    Rule 16/3: [128 cases, mean 19.97, range 9.5 to 36.2, est err 2.52]
#>
#>      if
#>   rm <= 3.626081
#>   dis > 0.6557049
#>   dis <= 1.298828
```

```
#>   lstat > 2.133251
#>     then
#>   outcome = 61.65 - 3.35 lstat + 4.9 dis + 1.6 rm - 1.3 nox - 22 tax
#>             + 0.5 rad + 1.8e-05 b + 0.09 crim - 0.004 ptratio
#>
#>    Rule 16/4: [140 cases, mean 21.93, range 12.7 to 35.1, est err 2.19]
#>
#>     if
#>   rm <= 3.626081
#>   dis > 1.298828
#>     then
#>   outcome = 54.16 - 3.58 lstat + 2.2 rad - 1.6 dis - 1.9 nox + 1.8 rm
#>             - 17 tax + 1.3e-05 b + 0.06 crim - 0.003 ptratio
#>
#>    Rule 16/5: [30 cases, mean 21.97, range 14.4 to 29.1, est err 2.41]
#>
#>     if
#>   rm <= 3.626081
#>   dis > 1.298828
#>   tax <= 1.879832
#>   lstat > 2.133251
#>     then
#>   outcome = -1065.35 + 566 tax + 8.7 rm - 0.13 lstat - 0.2 dis - 0.3 nox
#>
#>    Rule 16/6: [22 cases, mean 30.88, range 10.4 to 50, est err 4.51]
#>
#>     if
#>   rm > 3.626081
#>   lstat > 2.133251
#>     then
#>   outcome = 42.24 + 18.7 rm - 1.5 indus - 1.84 lstat - 2.5 nox - 1.6 dis
#>             - 39 tax + 0.7 rad - 0.012 ptratio + 0.0035 age + 1.2e-05 b
#>             + 0.11 crim
#>
#>    Rule 16/7: [73 cases, mean 34.52, range 20.6 to 50, est err 3.36]
#>
#>     if
#>   lstat <= 2.133251
#>     then
#>   outcome = 50.6 + 19.6 rm - 2.77 lstat - 3.2 nox - 1.7 dis - 45 tax
#>             + 1 rad + 0.007 age - 0.014 ptratio
#>
#> Model 17:
#>
#>    Rule 17/1: [116 cases, mean 15.37, range 5 to 27.9, est err 2.55]
#>
#>     if
#>   crim > 0.4779842
#>   lstat > 2.944963
#>     then
#>   outcome = 35.96 - 3.68 crim - 3.41 lstat + 0.3 nox
#>
#>    Rule 17/2: [112 cases, mean 19.13, range 7 to 31, est err 2.14]
```

```
#>
#>     if
#>  crim <= 0.4779842
#>  lstat > 2.944963
#>     then
#>  outcome = 184.65 - 0.0365 age + 9 rm - 4.1 dis - 97 tax + 8.4e-05 b
#>            - 0.024 ptratio
#>
#>   Rule 17/3: [9 cases, mean 28.37, range 15 to 50, est err 11.17]
#>
#>     if
#>  dis <= 0.9547035
#>  b <= 66469.73
#>  lstat <= 2.944963
#>     then
#>  outcome = -1.12 + 0.000454 b
#>
#>   Rule 17/4: [179 cases, mean 29.28, range 15 to 50, est err 3.35]
#>
#>     if
#>  lstat <= 2.944963
#>     then
#>  outcome = 278.16 + 20 rm - 7.4 dis - 0.0356 age - 161 tax + 0.051 zn
#>            - 0.61 lstat + 0.17 crim - 0.008 ptratio
#>
#>   Rule 17/5: [23 cases, mean 36.10, range 15 to 50, est err 10.83]
#>
#>     if
#>  dis <= 0.9547035
#>  lstat <= 2.944963
#>     then
#>  outcome = 233.74 - 8.5 dis + 12.1 rm + 1.15 crim - 2.42 lstat - 113 tax
#>            - 0.0221 age + 0.068 zn - 0.031 ptratio
#>
#> Model 18:
#>
#>   Rule 18/1: [84 cases, mean 14.29, range 5 to 27.5, est err 2.44]
#>
#>     if
#>  nox > -0.4864544
#>     then
#>  outcome = 41.55 - 6.2 lstat + 14.6 nox + 3.8e-05 b
#>
#>   Rule 18/2: [163 cases, mean 19.37, range 7 to 31, est err 2.44]
#>
#>     if
#>  nox <= -0.4864544
#>  lstat > 2.848535
#>     then
#>  outcome = 172.79 - 3.67 lstat + 3.1 rad - 3.5 dis - 72 tax - 0.72 indus
#>            - 0.033 ptratio - 1.2 nox + 0.0027 age + 0.6 rm + 0.05 crim
#>            + 5e-06 b
#>
```

```
#>   Rule 18/3: [106 cases, mean 25.41, range 16.5 to 50, est err 2.76]
#>
#>     if
#>  rm <= 3.626081
#>  lstat <= 2.848535
#>     then
#>  outcome = 10.71 - 4.6 dis - 2.21 lstat + 2.3 rad + 5.5 rm - 5.3 nox
#>            - 0.83 indus - 0.003 ptratio
#>
#>   Rule 18/4: [4 cases, mean 33.47, range 30.1 to 36.2, est err 5.61]
#>
#>     if
#>  rm <= 3.626081
#>  tax <= 1.863917
#>  lstat <= 2.848535
#>     then
#>  outcome = 36.84
#>
#>   Rule 18/5: [10 cases, mean 35.13, range 21.9 to 50, est err 17.40]
#>
#>     if
#>  dis <= 0.6492998
#>  lstat <= 2.848535
#>     then
#>  outcome = 84.58 - 94.7 dis - 0.15 lstat
#>
#>   Rule 18/6: [57 cases, mean 38.38, range 21.9 to 50, est err 3.97]
#>
#>     if
#>  rm > 3.626081
#>  lstat <= 2.848535
#>     then
#>  outcome = 100.34 + 22.3 rm - 5.79 lstat - 0.062 ptratio - 69 tax
#>            + 0.3 rad - 0.5 nox - 0.3 dis + 0.0011 age
#>
#>
#> Evaluation on training data (407 cases):
#>
#>     Average  |error|               1.72
#>     Relative |error|               0.26
#>     Correlation coefficient        0.96
#>
#>
#>  Attribute usage:
#>    Conds  Model
#>
#>     72%    84%    lstat
#>     38%    85%    dis
#>     35%    80%    rm
#>     27%    55%    nox
#>      4%    58%    crim
#>      2%    49%    b
#>      2%    68%    ptratio
```

```
#>      1%    78%    tax
#>      1%    67%    age
#>            41%    rad
#>            36%    indus
#>            20%    zn
#>
#>
#> Time: 0.2 secs
```

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

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

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

# calculate RMSE
rmse <- RMSE(predictions, valY)
r2 <- R2(predictions, valY)
print(rmse)
#> [1] 3.24
```

> 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 5

Classification algorithms comparison. Breast cancer dataset, (*LG, LDA, GLMNET, KNN, CARTM NB, SVM*)

5.1 `BreastCancer` dataset

5.2 Introduction

In this classification problem we apply these algorithms:

- Linear
  1. LG (logistic regression)
  2. LDA (linear discriminant analysis)
  3. GLMNET (Regularized logistic regression)
- Non-linear
  4. KNN (k-Nearest Neighbors)
  5. CART (Classification and Regression Trees)
  6. NB (Naive Bayes)
  7. SVM (Support Vector Machines)

```
# load packages
library(mlbench)
library(caret)
#> Loading required package: lattice
#> Loading required package: ggplot2
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang
library(tictoc)

# Load data
```

```
data(BreastCancer)
```

5.3 Workflow

1. Load dataset
2. Create train and validation datasets, 80/20
3. Inspect dataset:

- dimension
- class of variables
- `skimr`

4. Clean up features

- Convert character to numeric
- Frequency table on class
- remove NAs

5. Visualize features

- histograms (loop on variables)
- density plots (loop)
- boxplot (loop)
- Pairwise jittered plot
- Barplots for all features (loop)

6. Train as-is

- Set the train control to
  - 10 cross-validations
  - 3 repetitions
  - Metric: Accuracy
- Train the models
- Numeric comparison of model results
- Visual comparison
  - dot plot

7. Train with data transformation

- data transformatiom
  - BoxCox
- Train models
- Numeric comparison
- Visual comparison
  - dot plot

8. Tune the best model: SVM

- Set the train control to
  - 10 cross-validations
  - 3 repetitions
  - Metric: Accuracy
- Train the models
  - Radial SVM
  - Sigma vector
  - `.C`

  - BoxCox
- Evaluate tuning parameters

9. Tune the best model: KNN

- Set the train control to
  - 10 cross-validations
  - 3 repetitions
  - Metric: Accuracy
- Train the models
  - .k
  - BoxCox
- Evaluate tuning parameters
  - Scatter plot 10, Ensembling
- Select the algorithms
  - Bagged CART
  - Random Forest
  - Stochastic Gradient Boosting
  - C5.0
- Numeric comparison
  - resamples
  - summary
- Visual comparison
  - dot plot

11. Finalize the model

- Back transformation
  - `preProcess`
  - `predict`

12. Apply model to validation set

- Prepare validation set
- Transform the dataset
- Make prediction
  - `knn3Train`
- Calculate accuracy
  - Confusion Matrix

5.4 Inspect the dataset

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

```
#> $ Mitoses         <fct> 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 4, 1...
#> $ Class           <fct> benign, benign, benign, benign, benign, malign...
```

```
tibble::as_tibble(BreastCancer)
#> # A tibble: 699 x 11
#>    Id    Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
#>    <chr> <ord>        <ord>     <ord>      <ord>         <ord>
#> 1 1000~ 5            1         1          1             2
#> 2 1002~ 5            4         4          5             7
#> 3 1015~ 3            1         1          1             2
#> 4 1016~ 6            8         8          1             3
#> 5 1017~ 4            1         1          3             2
#> 6 1017~ 8            10        10         8             7
#> # ... with 693 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
#> 1  1000025            5         1          1             1            2
#> 2  1002945            5         4          4             5            7
#> 3  1015425            3         1          1             1            2
#> 5  1017023            4         1          1             3            2
#> 6  1017122            8        10         10             8            7
#> 7  1018099            1         1          1             1            2
#> 8  1018561            2         1          2             1            2
#> 9  1033078            2         1          1             1            2
#> 10 1033078            4         2          1             1            2
#> 11 1035283            1         1          1             1            1
#> 13 1041801            5         3          3             3            2
#> 14 1043999            1         1          1             1            2
#> 15 1044572            8         7          5            10            7
#> 16 1047630            7         4          6             4            6
#> 18 1049815            4         1          1             1            2
#> 19 1050670           10         7          7             6            4
#> 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
#> 1            1           3               1       1    benign
#> 2           10           3               2       1    benign
#> 3            2           3               1       1    benign
#> 5            1           3               1       1    benign
#> 6           10           9               7       1 malignant
#> 7           10           3               1       1    benign
#> 8            1           3               1       1    benign
#> 9            1           1               1       5    benign
#> 10           1           2               1       1    benign
#> 11           1           3               1       1    benign
#> 13           3           4               4       1 malignant
#> 14           3           3               1       1    benign
#> 15           9           5               5       4 malignant
#> 16           1           4               3       1 malignant
#> 18           1           3               1       1    benign
#> 19          10           4               1       2 malignant
#> 21          10           5               4       4 malignant
#> 22           7           7              10       1 malignant
#> 23           1           2               1       1    benign
#> 24        <NA>           7               3       1 malignant
```

```
library(skimr)
#>
#> Attaching package: 'skimr'
#> The following object is masked from 'package:stats':
#>
#>     filter
skim(dataset)
#> Skim summary statistics
#>  n obs: 560
#>  n variables: 11
#>
#> -- Variable type:character -------------------------------------------------------
#>  variable missing complete   n min max empty n_unique
#>        Id       0      560 560   5   8     0      523
#>
#> -- Variable type:factor ----------------------------------------------------------
#>         variable missing complete   n n_unique
#>      Bare.nuclei      12      548 560       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: 102, 2: 28, 5: 24   FALSE
#>  3: 136, 2: 134, 1: 118, 7: 64   FALSE
```

```
#>   1: 281, 10: 48, 3: 47, 2: 46    TRUE
#>   1: 307, 10: 55, 3: 43, 2: 36    TRUE
#>    1: 113, 3: 94, 5: 94, 4: 62    TRUE
#>      ben: 367, mal: 193, NA: 0   FALSE
#>    2: 304, 3: 60, 4: 40, 1: 38    TRUE
#>   1: 331, 2: 44, 10: 44, 3: 42    TRUE
#>    1: 466, 3: 29, 2: 23, 4: 10   FALSE
#>   1: 347, 10: 50, 3: 36, 2: 29   FALSE
```

```
# types
sapply(dataset, class)
#> $Id
#> [1] "character"
#>
#> $Cl.thickness
#> [1] "ordered" "factor"
#>
#> $Cell.size
#> [1] "ordered" "factor"
#>
#> $Cell.shape
#> [1] "ordered" "factor"
#>
#> $Marg.adhesion
#> [1] "ordered" "factor"
#>
#> $Epith.c.size
#> [1] "ordered" "factor"
#>
#> $Bare.nuclei
#> [1] "factor"
#>
#> $Bl.cromatin
#> [1] "factor"
#>
#> $Normal.nucleoli
#> [1] "factor"
#>
#> $Mitoses
#> [1] "factor"
#>
#> $Class
#> [1] "factor"
```

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

5.5 clean up

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

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

```
# summary
summary(dataset)
#>   Cl.thickness     Cell.size       Cell.shape    Marg.adhesion
#>  Min.   : 1.00   Min.   : 1.00   Min.   : 1.00   Min.   : 1.00
#>  1st Qu.: 2.00   1st Qu.: 1.00   1st Qu.: 1.00   1st Qu.: 1.00
#>  Median : 4.00   Median : 1.00   Median : 1.00   Median : 1.00
#>  Mean   : 4.44   Mean   : 3.14   Mean   : 3.21   Mean   : 2.79
#>  3rd Qu.: 6.00   3rd Qu.: 5.00   3rd Qu.: 5.00   3rd Qu.: 4.00
#>  Max.   :10.00   Max.   :10.00   Max.   :10.00   Max.   :10.00
#>
#>   Epith.c.size    Bare.nuclei     Bl.cromatin    Normal.nucleoli
#>  Min.   : 1.00   Min.   : 1.00   Min.   : 1.00   Min.   : 1.00
#>  1st Qu.: 2.00   1st Qu.: 1.00   1st Qu.: 2.00   1st Qu.: 1.00
#>  Median : 2.00   Median : 1.00   Median : 3.00   Median : 1.00
#>  Mean   : 3.23   Mean   : 3.48   Mean   : 3.45   Mean   : 2.94
#>  3rd Qu.: 4.00   3rd Qu.: 5.25   3rd Qu.: 5.00   3rd Qu.: 4.00
#>  Max.   :10.00   Max.   :10.00   Max.   :10.00   Max.   :10.00
#>                  NA's   :12
#>     Mitoses            Class
#>  Min.   : 1.00   benign   :367
#>  1st Qu.: 1.00   malignant:193
#>  Median : 1.00
#>  Mean   : 1.59
#>  3rd Qu.: 1.00
#>  Max.   :10.00
#>
```

```
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      12      548 560 3.48 3.62  1   1   1 5.25   10
#>      Bl.cromatin       0      560 560 3.45 2.43  1   2   3 5      10
#>       Cell.shape       0      560 560 3.21 2.97  1   1   1 5      10
#>        Cell.size       0      560 560 3.14 3.07  1   1   1 5      10
#>     Cl.thickness       0      560 560 4.44 2.83  1   2   4 6      10
#>     Epith.c.size       0      560 560 3.23 2.22  1   2   2 4      10
```

```
#>    Marg.adhesion       0      560 560 2.79 2.85  1   1   1 4      10
#>          Mitoses       0      560 560 1.59 1.7   1   1   1 1      10
#>  Normal.nucleoli       0      560 560 2.94 3.08  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.

5.6 Analyze the class variable

```
# class distribution
cbind(freq = table(dataset$Class),
      percentage = prop.table(table(dataset$Class))*100)
#>           freq percentage
#> benign     367       65.5
#> malignant  193       34.5
```

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.

5.6.1 remove NAs

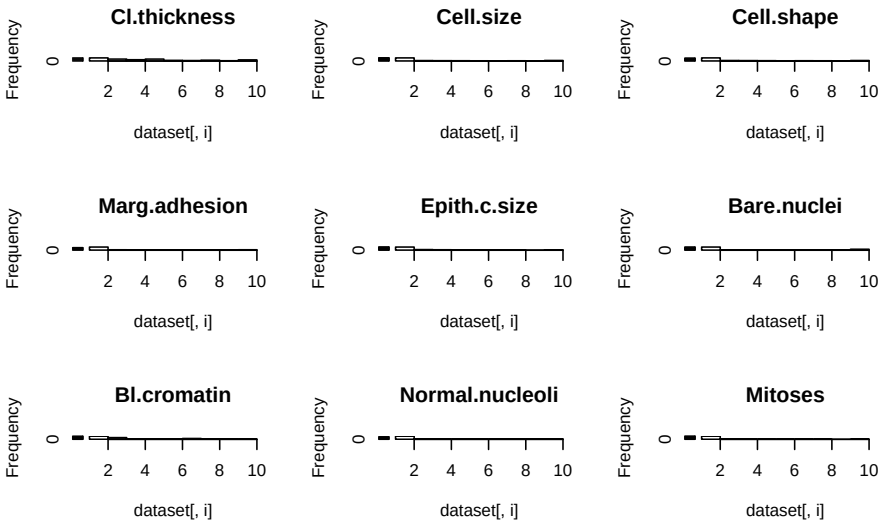
```
# summarize correlations between input variables
complete_cases <- complete.cases(dataset)
cor(dataset[complete_cases,1:9])
#>                 Cl.thickness Cell.size Cell.shape Marg.adhesion
#> Cl.thickness           1.000     0.665      0.667         0.486
#> Cell.size              0.665     1.000      0.904         0.722
#> Cell.shape             0.667     0.904      1.000         0.694
#> Marg.adhesion          0.486     0.722      0.694         1.000
#> Epith.c.size           0.543     0.773      0.739         0.643
#> Bare.nuclei            0.598     0.700      0.721         0.669
#> Bl.cromatin            0.565     0.752      0.739         0.692
#> Normal.nucleoli        0.570     0.737      0.741         0.644
#> Mitoses                0.347     0.453      0.432         0.419
#>                 Epith.c.size Bare.nuclei Bl.cromatin Normal.nucleoli
#> Cl.thickness           0.543       0.598       0.565           0.570
#> Cell.size              0.773       0.700       0.752           0.737
#> Cell.shape             0.739       0.721       0.739           0.741
#> Marg.adhesion          0.643       0.669       0.692           0.644
#> Epith.c.size           1.000       0.614       0.628           0.642
#> Bare.nuclei            0.614       1.000       0.685           0.605
```

```
#> Bl.cromatin            0.628       0.685       1.000           0.692
#> Normal.nucleoli        0.642       0.605       0.692           1.000
#> Mitoses                0.485       0.351       0.356           0.432
#>                   Mitoses
#> Cl.thickness        0.347
#> Cell.size           0.453
#> Cell.shape          0.432
#> Marg.adhesion       0.419
#> Epith.c.size        0.485
#> Bare.nuclei         0.351
#> Bl.cromatin         0.356
#> Normal.nucleoli     0.432
#> Mitoses             1.000
```

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.

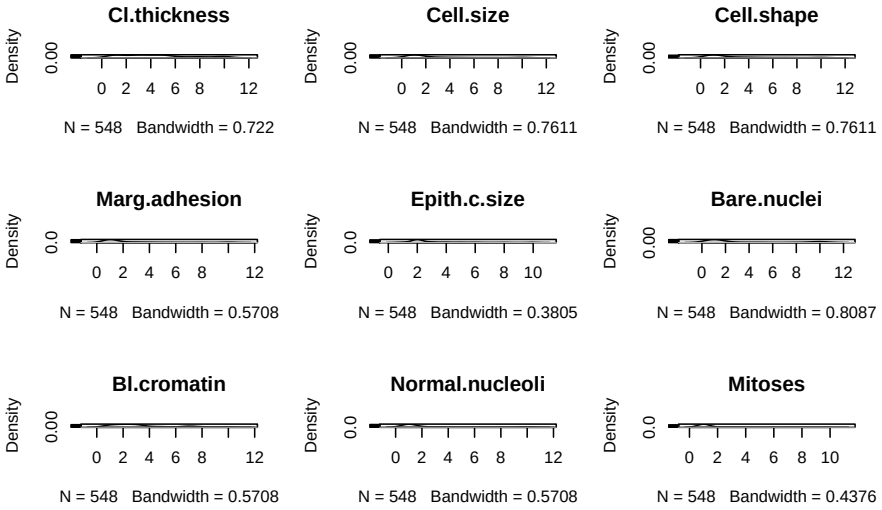
5.7 Unimodal visualization

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



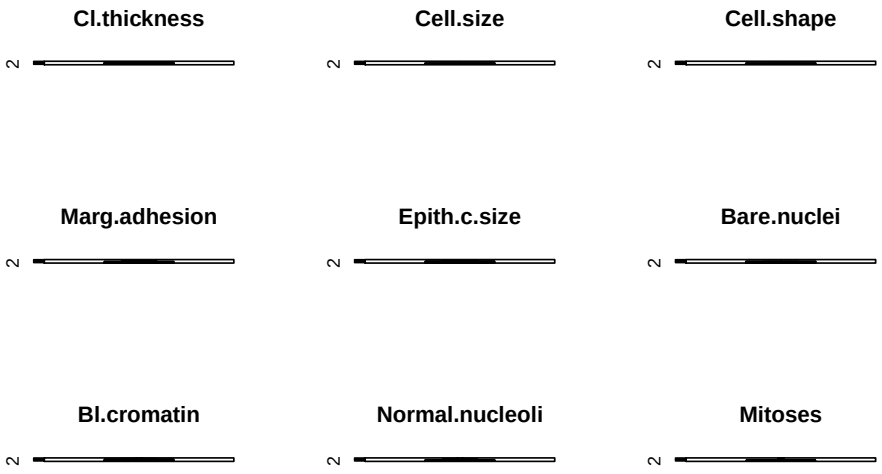
We can see that almost all of the distributions have an exponential or bimodal shape to them.

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



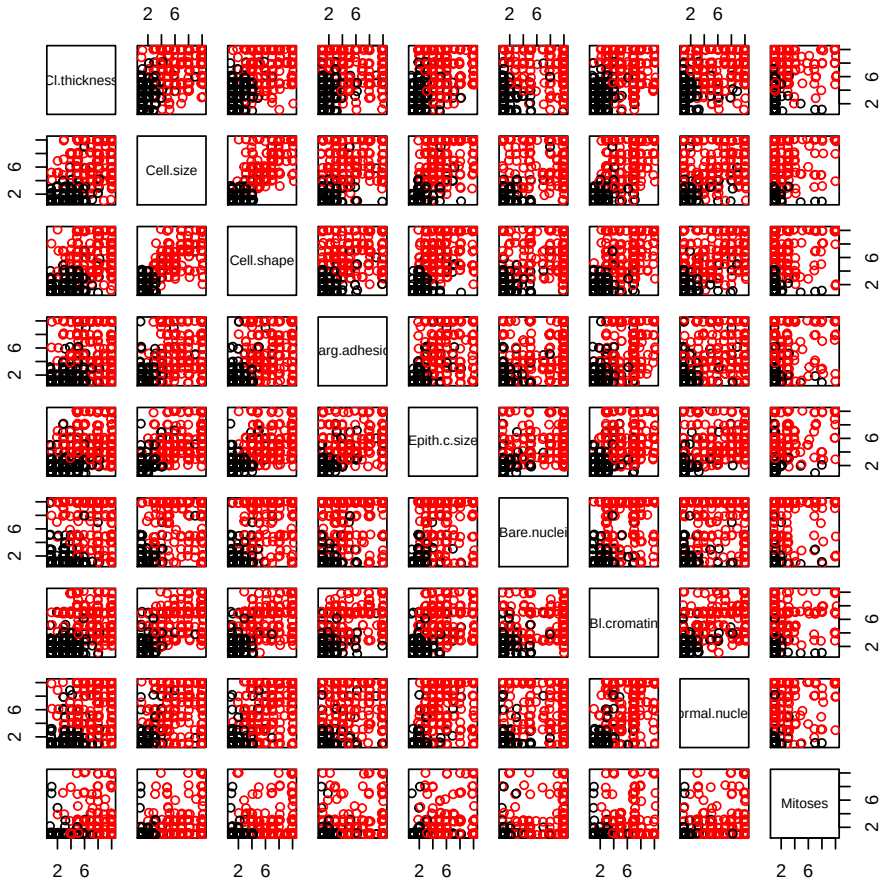
These plots add more support to our initial ideas. We can see bimodal distributions (two bumps) and exponential-looking distributions.

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



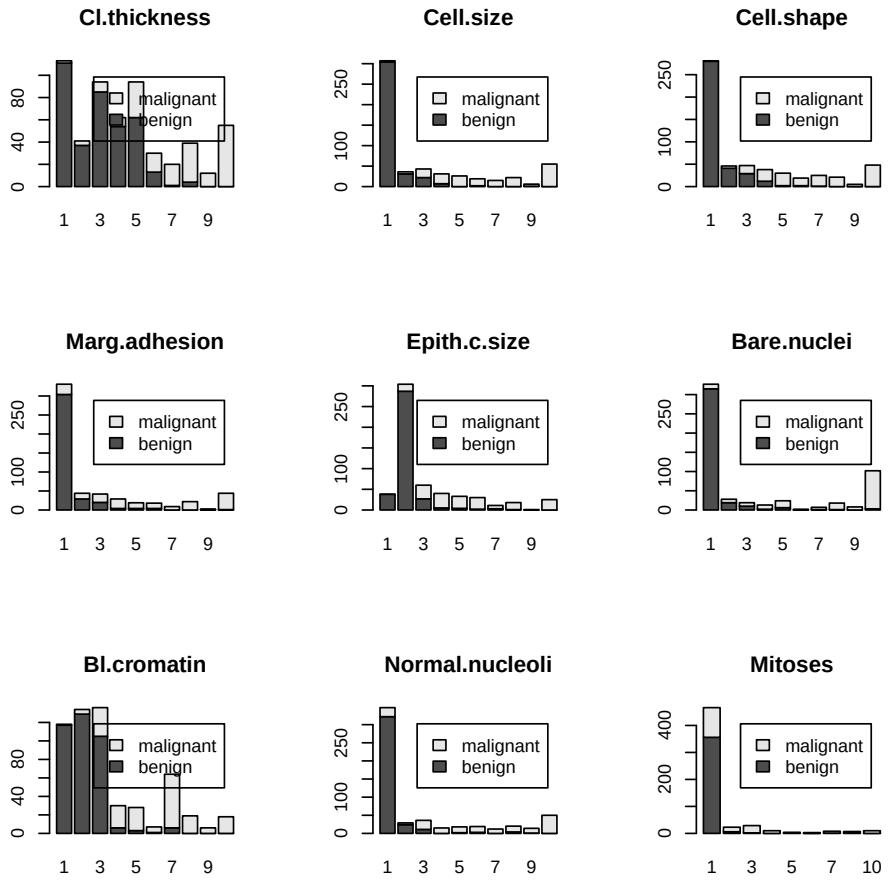
5.8 Multimodal visualization

```
# 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))
}
```



5.9 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).

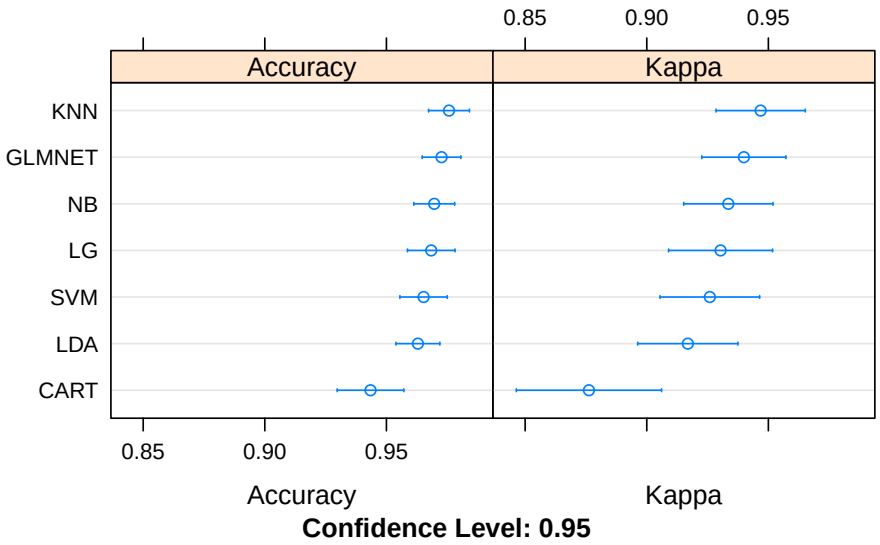
For simplicity, we will use Accuracy and Kappa metrics. Given that it is a medical test, we could have gone with the Area Under ROC Curve (AUC) and looked at the sensitivity and specificity to select the best algorithms.

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

```
tic()
# LG
set.seed(7)
fit.glm <- train(Class~., data=dataset, method="glm", metric=metric,
                 trControl=trainControl, na.action=na.omit)
# LDA
set.seed(7)
fit.lda <- train(Class~., data=dataset, method="lda", metric=metric,
                 trControl=trainControl, na.action=na.omit)
# GLMNET
```

```
set.seed(7)
fit.glmnet <- train(Class~., data=dataset, method="glmnet", metric=metric,
                    trControl=trainControl, na.action=na.omit)
# KNN
set.seed(7)
fit.knn <- train(Class~., data=dataset, method="knn", metric=metric,
                 trControl=trainControl, na.action=na.omit)
# CART
set.seed(7)
fit.cart <- train(Class~., data=dataset, method="rpart", metric=metric,
                  trControl=trainControl, na.action=na.omit)
# Naive Bayes
set.seed(7)
fit.nb <- train(Class~., data=dataset, method="nb", metric=metric,
                trControl=trainControl, na.action=na.omit)
# 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))
toc()
#> 14.804 sec elapsed
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     0.909   0.945  0.964 0.968   0.995    1    0
#> LDA    0.907   0.945  0.964 0.963   0.982    1    0
#> GLMNET 0.927   0.964  0.964 0.973   0.995    1    0
#> KNN    0.927   0.964  0.982 0.976   1.000    1    0
#> CART   0.833   0.927  0.945 0.943   0.964    1    0
#> NB     0.927   0.963  0.981 0.970   0.982    1    0
#> SVM    0.907   0.945  0.964 0.965   0.982    1    0
#>
#> Kappa
#>         Min. 1st Qu. Median  Mean 3rd Qu. Max. NA's
#> LG     0.806   0.880  0.921 0.930   0.990    1    0
#> LDA    0.786   0.880  0.918 0.917   0.959    1    0
#> GLMNET 0.843   0.918  0.922 0.940   0.990    1    0
```

```
#> KNN    0.843   0.920  0.959 0.947   1.000    1    0
#> CART   0.630   0.840  0.879 0.876   0.920    1    0
#> NB     0.835   0.918  0.959 0.934   0.960    1    0
#> SVM    0.804   0.883  0.922 0.926   0.960    1    0
dotplot(results)
```



We can see good accuracy across the board. All algorithms have a mean accuracy above 90%, well above the baseline of 65% if we just predicted benign. The problem is learnable. We can see that KNN (97.08%) and logistic regression (NB was 96.2% and GLMNET was 96.4%) had the highest accuracy on the problem.

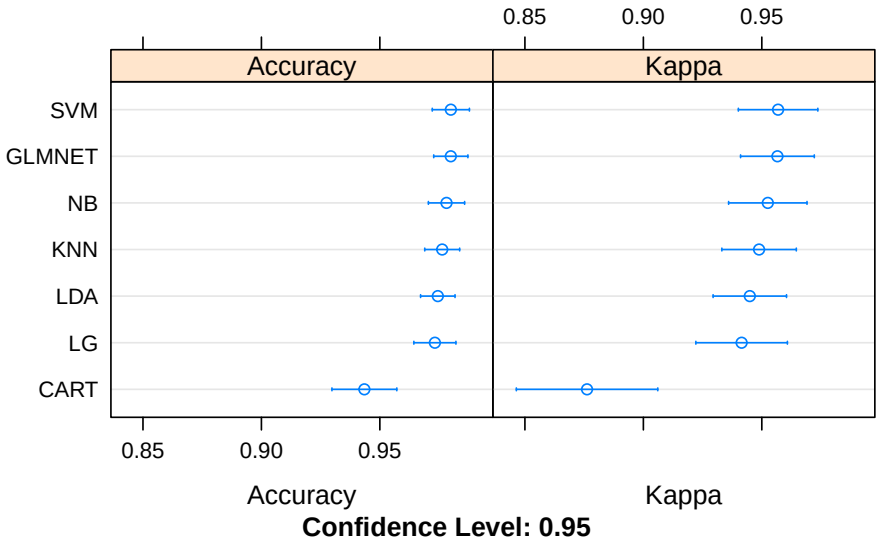
5.10 Data transform

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 1
#> 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 20
#> 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 18
#> 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 23
#> 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 27
#> 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 12
#> 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 2
#> 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 9
#> 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 23
#> Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
#> observation 32
# 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     0.909   0.963  0.982 0.973   0.996    1    0
#> LDA    0.927   0.964  0.981 0.974   0.982    1    0
#> GLMNET 0.944   0.964  0.982 0.980   1.000    1    0
#> KNN    0.909   0.964  0.981 0.976   0.982    1    0
#> CART   0.833   0.927  0.945 0.943   0.964    1    0
#> NB     0.927   0.964  0.982 0.978   1.000    1    0
#> SVM    0.927   0.964  0.982 0.980   1.000    1    0
#>
#> Kappa
#>         Min. 1st Qu. Median  Mean 3rd Qu. Max. NA's
#> LG     0.806   0.919  0.959 0.941   0.990    1    0
#> LDA    0.847   0.921  0.959 0.945   0.961    1    0
#> GLMNET 0.878   0.922  0.960 0.957   1.000    1    0
#> KNN    0.806   0.922  0.959 0.949   0.961    1    0
#> CART   0.630   0.840  0.879 0.876   0.920    1    0
#> NB     0.843   0.922  0.960 0.953   1.000    1    0
#> SVM    0.847   0.922  0.960 0.957   1.000    1    0
dotplot(transformResults)
```

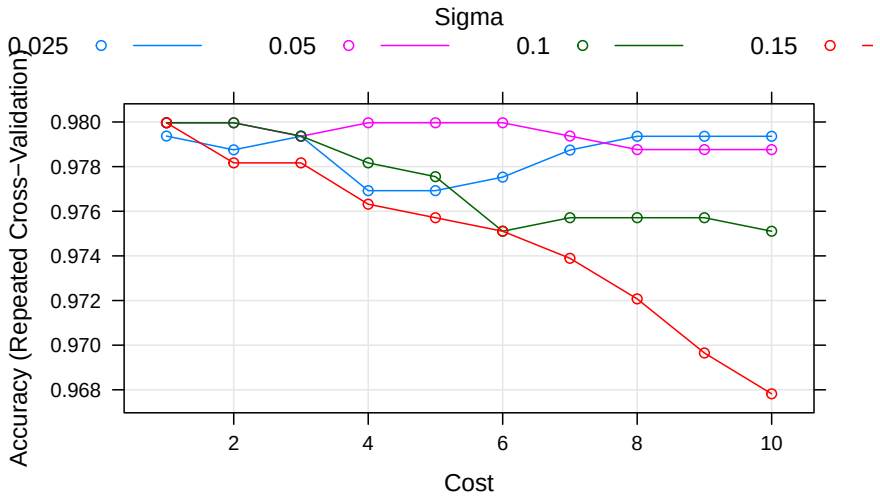


We can see that the accuracy of the previous best algorithm KNN was elevated to 97.14%. We have a new ranking, showing SVM with the most accurate mean accuracy at 97.20%.

5.11 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: 493, 492, 493, 493, 493, 494, ...
#> Resampling results across tuning parameters:
#>
#>   sigma  C   Accuracy  Kappa
#>   0.025   1  0.979     0.956
#>   0.025   2  0.979     0.954
#>   0.025   3  0.979     0.956
#>   0.025   4  0.977     0.950
```

```
#>   0.025   5  0.977     0.950
#>   0.025   6  0.978     0.951
#>   0.025   7  0.979     0.954
#>   0.025   8  0.979     0.956
#>   0.025   9  0.979     0.956
#>   0.025  10  0.979     0.956
#>   0.050   1  0.980     0.957
#>   0.050   2  0.980     0.957
#>   0.050   3  0.979     0.956
#>   0.050   4  0.980     0.957
#>   0.050   5  0.980     0.957
#>   0.050   6  0.980     0.957
#>   0.050   7  0.979     0.956
#>   0.050   8  0.979     0.954
#>   0.050   9  0.979     0.954
#>   0.050  10  0.979     0.954
#>   0.100   1  0.980     0.957
#>   0.100   2  0.980     0.957
#>   0.100   3  0.979     0.956
#>   0.100   4  0.978     0.953
#>   0.100   5  0.978     0.952
#>   0.100   6  0.975     0.946
#>   0.100   7  0.976     0.948
#>   0.100   8  0.976     0.948
#>   0.100   9  0.976     0.948
#>   0.100  10  0.975     0.946
#>   0.150   1  0.980     0.957
#>   0.150   2  0.978     0.953
#>   0.150   3  0.978     0.953
#>   0.150   4  0.976     0.949
#>   0.150   5  0.976     0.948
#>   0.150   6  0.975     0.946
#>   0.150   7  0.974     0.944
#>   0.150   8  0.972     0.939
#>   0.150   9  0.970     0.934
#>   0.150  10  0.968     0.930
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were sigma = 0.15 and C = 1.
plot(fit.svm)
```

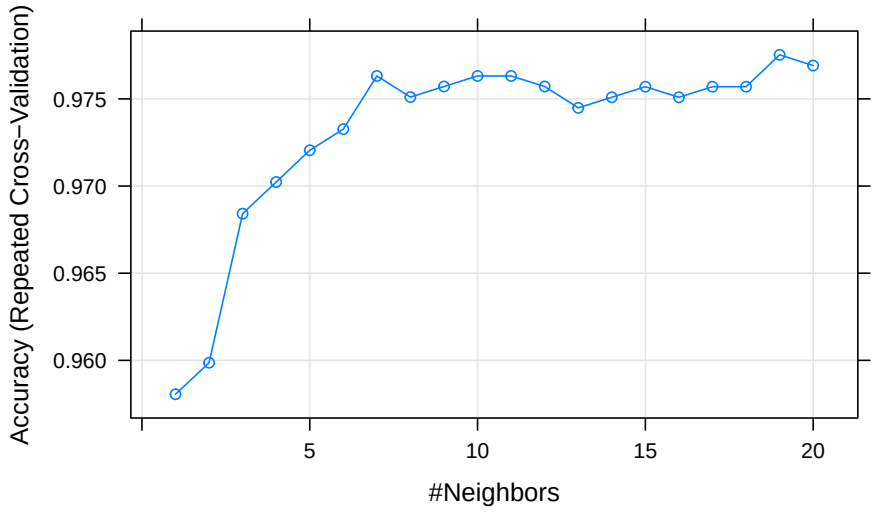


We can see that we have made very little difference to the results. The most accurate model had a score of 97.31% (the same as our previously rounded score of 97.20%) using a sigma = 0.1 and C = 1. We could tune further, but I don't expect a payoff.

5.12 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: 493, 492, 493, 493, 493, 494, ...
#> Resampling results across tuning parameters:
#>
#>   k   Accuracy  Kappa
#>    1  0.958     0.908
#>    2  0.960     0.912
#>    3  0.968     0.931
#>    4  0.970     0.935
#>    5  0.972     0.939
#>    6  0.973     0.942
#>    7  0.976     0.949
```

```
#>    8  0.975      0.946
#>    9  0.976      0.947
#>   10  0.976      0.949
#>   11  0.976      0.949
#>   12  0.976      0.947
#>   13  0.974      0.945
#>   14  0.975      0.946
#>   15  0.976      0.947
#>   16  0.975      0.946
#>   17  0.976      0.947
#>   18  0.976      0.947
#>   19  0.978      0.951
#>   20  0.977      0.950
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was k = 19.
plot(fit.knn)
```

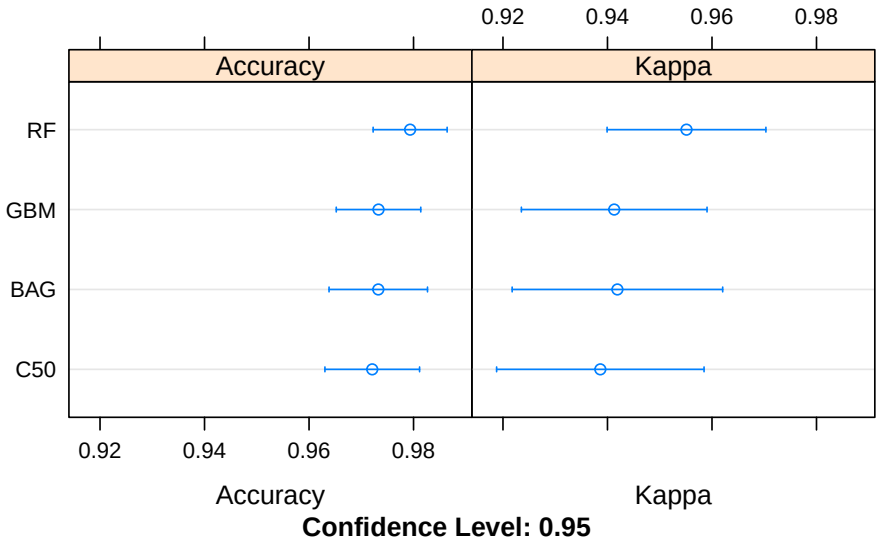


We can see again that tuning has made little difference, settling on a value of k = 7 with an accuracy of 97.19%. This is higher than the previous 97.14%, but very similar (or perhaps identical!) to the result achieved by the tuned SVM.

5.13 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)

# 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 0.907   0.963  0.982 0.973   0.982    1    0
#> RF  0.926   0.981  0.982 0.979   0.995    1    0
#> GBM 0.929   0.963  0.981 0.973   0.995    1    0
#> C50 0.907   0.964  0.982 0.972   0.982    1    0
#>
#> Kappa
#>      Min. 1st Qu. Median  Mean 3rd Qu. Max. NA's
#> BAG 0.804   0.920  0.959 0.942    0.96    1    0
#> RF  0.841   0.959  0.960 0.955    0.99    1    0
#> GBM 0.844   0.919  0.959 0.941    0.99    1    0
#> C50 0.795   0.918  0.959 0.939    0.96    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.

5.14 Finalize model

We now need to finalize the model, which really means choose which model we would like to use. For simplicity I would probably select the KNN method, at the expense of the memory required to store the training dataset. SVM would be a good choice to trade-off space and time complexity. I probably would not select the Random Forest algorithm given the complexity of the model. It seems overkill for this dataset, lots of trees with little benefit in Accuracy.

Let's go with the KNN algorithm. This is really simple, as we do not need to store a model. We do need to capture the parameters of the Box-Cox transform though. And we also need to prepare the data by removing the unused Id attribute and converting all of the inputs to numeric format.

The implementation of KNN (knn3()) belongs to the caret package and does not support missing values. We will have to remove the rows with missing values from the training dataset as well as the validation dataset. The code below shows the preparation of the pre-processing parameters using the training dataset.

```
# prepare parameters for data transform
set.seed(7)

datasetNoMissing <- dataset[complete.cases(dataset),]
x <- datasetNoMissing[,1:9]

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

5.15 Prepare the validation set

Next we need to prepare the validation dataset for making a prediction. We must:

1. Remove the Id attribute.
2. Remove those rows with missing data.
3. Convert all input attributes to numeric.
4. Apply the Box-Cox transform to the input attributes using parameters prepared on the training dataset.

```
# prepare the validation dataset
set.seed(7)

# remove id column
validation <- validation[,-1]

# remove missing values (not allowed in this implementation of knn)
validation <- validation[complete.cases(validation),]

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

# transform the validation dataset
validationX <- predict(preprocessParams, validation[,1:9])
```

```
# make predictions
set.seed(7)
# knn3Train(train, test, cl, k = 1, l = 0, prob = TRUE, use.all = TRUE)
# k: number of neighbours considered.
predictions <- knn3Train(x, validationX, datasetNoMissing$Class,
                         k = 9,
                         prob = FALSE)

# convert
confusionMatrix(as.factor(predictions), validation$Class)
#> Confusion Matrix and Statistics
#>
#>            Reference
#> Prediction  benign malignant
#>   benign        83         1
#>   malignant      4        47
#>
#>                Accuracy : 0.963
#>                  95% CI : (0.916, 0.988)
#>     No Information Rate : 0.644
#>     P-Value [Acc > NIR] : <2e-16
#>
#>                   Kappa : 0.92
#>
#>  Mcnemar's Test P-Value : 0.371
#>
#>             Sensitivity : 0.954
#>             Specificity : 0.979
#>          Pos Pred Value : 0.988
#>          Neg Pred Value : 0.922
#>              Prevalence : 0.644
#>          Detection Rate : 0.615
#>    Detection Prevalence : 0.622
```

```
#>       Balanced Accuracy : 0.967
#>
#>        '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 6

Classification algorithms comparison. outbreaks dataset. (*RF, GLMNET, KNN, PDA, LDA, NSC, C5, PLS*)

6.1 Introduction

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.

6.1.1 Algorithms

We compare these classification algorithms:

1. Random Forest
2. GLM net
3. k-Nearest Neighbors
4. Penalized Discriminant Analysis
5. Stabilized Linear Discriminant Analysis
6. Nearest Shrunken Centroids
7. Single C5.0 Tree
8. Partial Least Squares

6.1.2 Workflow

1. Load dataset

2. Data wrangling

- Many dates as one variable: `gather`
- Convert group to factor: `factor`
- Change dates descriptions: `dplyr::mapvalues`
- Set several provinces as Other: `mapvalues`
- Convert gender to factor
- Convert province to factor

3. Features visualization

- Area density plot of Month vs Age by Province, by date group, by outcome, by gender
- Number of flu cases by gender, by province
- Age density plot of flu cases by outcome
- Days passed from outset by province, by age, by gender, by outcome

4. Feature engineering

- Generate new features
- Gender to numeric
- Provinces to binary classifiers
- Outcome to binary classifier
- Impute missing data: `mice`

5. Split dataset in training, validation and test sets

6. Feature importance

- with Decision Trees
- with Random Forest
- Wrangling dataset
- Plot Importance vs Variable

7. Impact on Datasets

- Density plot of training, validation and test datasets
- Features vs outcome by age, days onset to hospital, day onset to outcome

8. Train models on training validation dataset

- Numeric and visual Comparison of models
  - Accuracy
  - Kappa
- Compare predictions on training validation set
  - Create dataset for results
  - New variable for outcome

11. Predicting on unknown outcomes on training data

- Numeric and visual Comparison of models
  - Accuracy
  - Kappa
- Compare predictions on training validation set
  - Create dataset for results
  - New variable for outcome
- Calculate predicted outcome
- Save to CSV file
- Calculate recovery cases

- Summarize outcome
- Plot month vs log2-ratio of recovery vs death, by gender, by age, by date group

6.1.3 Can we predict flu outcome with Machine Learning in R?

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. :-)

6.2 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")
#> Loading required package: 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.

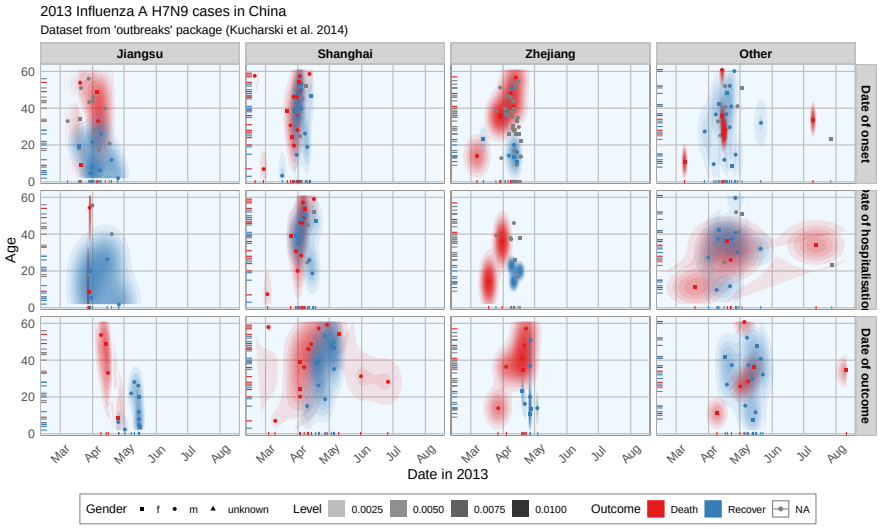
Most provinces have only a handful of cases, so I am combining them into the category "other" and keep only Jiangsu, Shanghai and Zhejian and separate provinces.

```
# 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")
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"))
```

```
library(ggplot2)
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
```

```
#>   print.quosures rlang
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)
  )
}
```

```
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")
#> Warning: Removed 149 rows containing non-finite values (stat_density2d).
#> Warning: Removed 149 rows containing missing values (geom_point).
```



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.

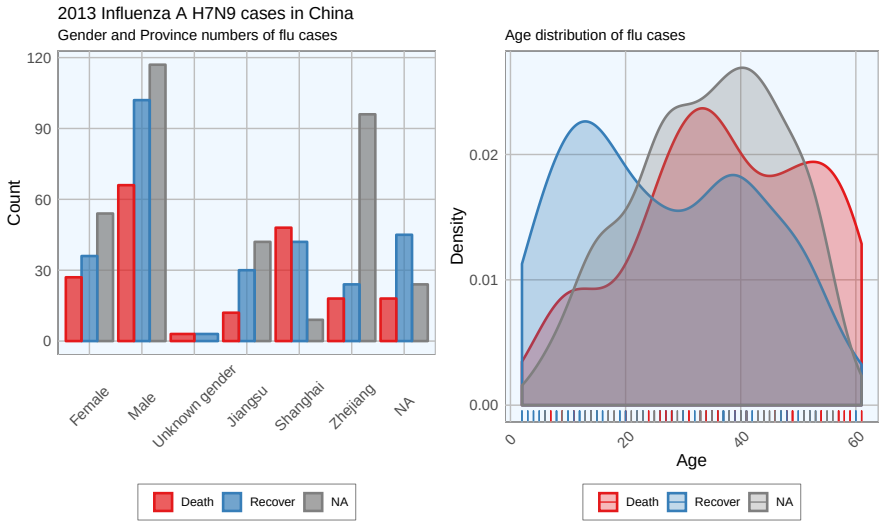
The density distribution of date by age for the cases seems to indicate that older people died more frequently in the Jiangsu and Zhejiang province than in Shanghai and in other provinces.

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)
#> Warning: attributes are not identical across measure variables;
#> they will be dropped

# 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)
#> Warning: Removed 6 rows containing non-finite values (stat_density).
```



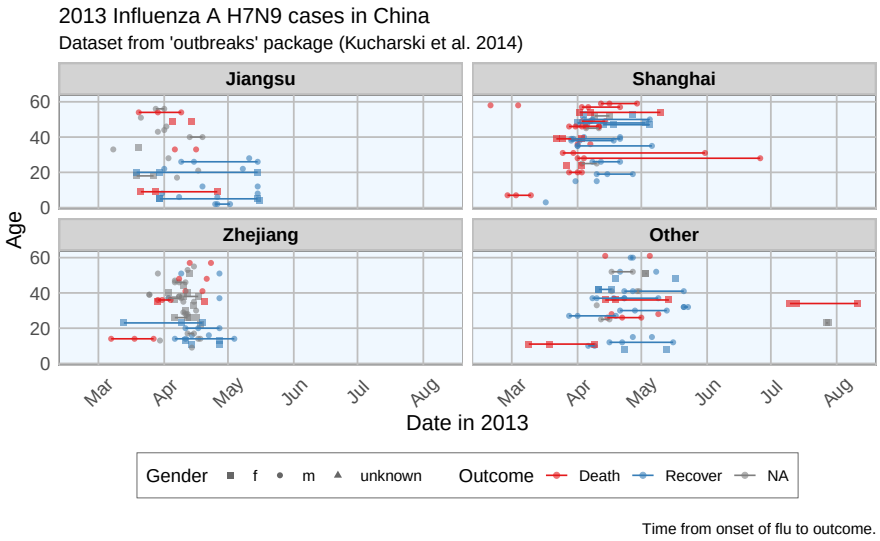
In the dataset, there are more male than female cases and correspondingly, we see more deaths, recoveries and unknown outcomes in men than in women. This is potentially a problem later on for modeling because the inherent likelihoods for outcome are not directly comparable between the sexes.

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."
  )
#> Warning: Removed 149 rows containing missing values (geom_point).
#> Warning: Removed 122 rows containing missing values (geom_path).
```



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

6.3 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)
#>
#> Attaching package: 'dplyr'
#> The following object is masked from 'package:gridExtra':
#>
#>     combine
#> The following objects are masked from 'package:plyr':
#>
#>     arrange, count, desc, failwith, id, mutate, rename, summarise, summarize
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union

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
```

6.3.1 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)
#> Loading required package: lattice
#>
#> Attaching package: 'mice'
#> The following object is masked from 'package:tidyr':
#>
#>     complete
#> The following objects are masked from 'package:base':
#>
#>     cbind, rbind

dataset_impute  <- mice(dataset[, -1],  print = FALSE)  # remove outcome
#> Warning: Number of logged events: 150
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 d
#> 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   54        0        0                0        1        0        0
#> case_75   83        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
```

```
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  54        0        0                0                1                0
#> case_75    Death  83        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
```

```
cat("NAs before imput: ", sum(is.na(dataset)), "\n")
#> NAs before imput:  277
summary(dataset$outcome)
#>   Death Recover    NA's
#>      32      47      57
```

```
cat("NAs after imput: ", sum(is.na(dataset_complete)), "\n")
#> NAs after imput:  57
summary(dataset_complete$outcome)
#>   Death Recover    NA's
#>      32      47      57
```

6.4 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.

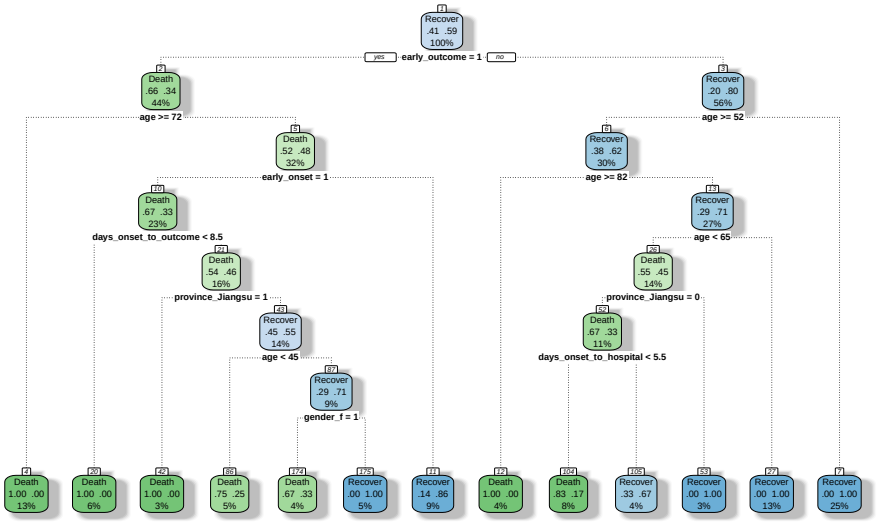
```
train_index <- which(is.na(dataset_complete$outcome))
train_data <- dataset_complete[-train_index, ]     # 79x12
test_data  <- dataset_complete[train_index, -1]    # 57x11

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]
```

6.4.1 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)
#> Rattle: A free graphical interface for data science with R.
#> Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
#> Type 'rattle()' to shake, rattle, and roll your data.
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−Sep−18 13:41:05 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.

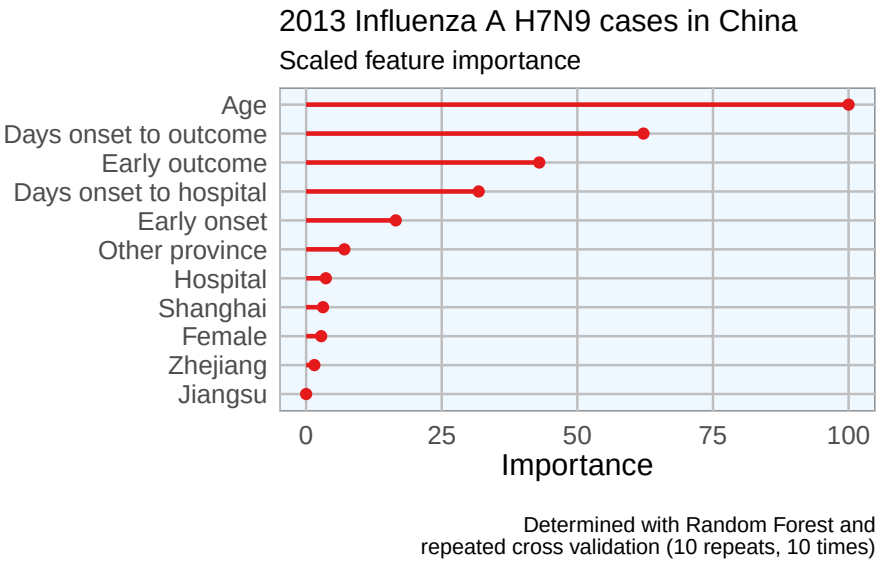
6.4.2 Feature Importance

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)"
  )
#> Warning in brewer.pal(1, "Set1"): minimal value for n is 3, returning requested palette with 3 diffe
```



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.

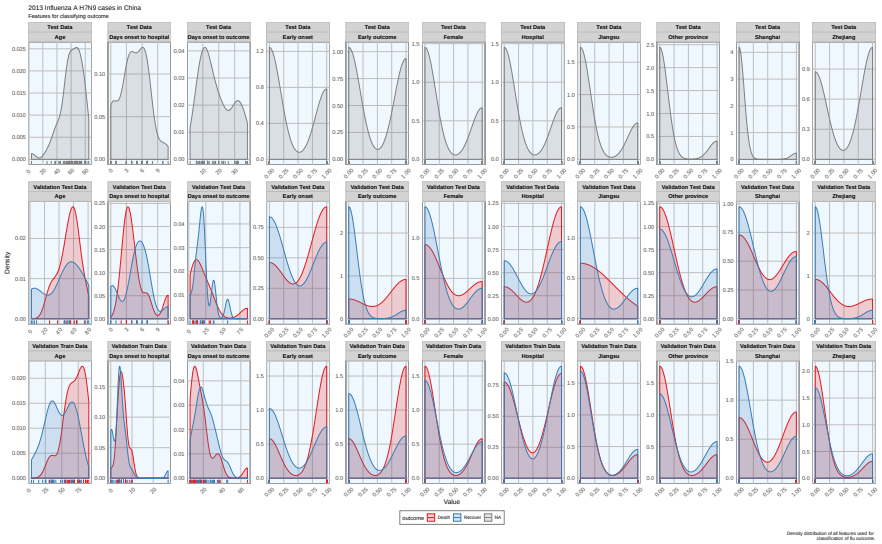
6.4.3 Feature Plot

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)
#> Warning: attributes are not identical across measure variables;
#> they will be dropped

# 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)
```

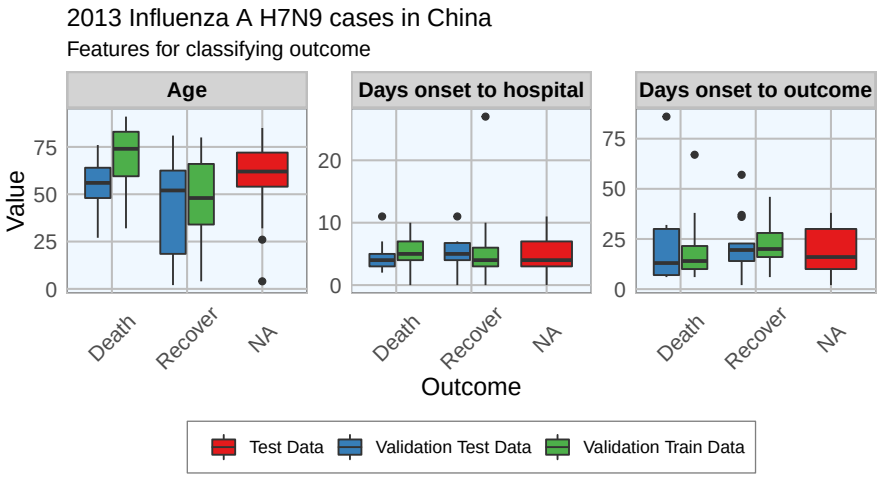
6.4.4 Plot distribution of features in each dataset

```
# 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."
  )
```



6.4.5 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."
  )
```



Boxplot 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.

6.5 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)
```

6.5.1 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: 51, 49, 50, 51, 49, 51, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy  Kappa
#>    2    0.687     0.340
#>    6    0.732     0.432
#>   11    0.726     0.423
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 6.
```

```
confusionMatrix(predict(model_rf, val_test_data[, -1]), val_test_data$outcome)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       3       0
#>    Recover     6      14
#>
#>                Accuracy : 0.739
#>                  95% CI : (0.516, 0.898)
#>     No Information Rate : 0.609
#>     P-Value [Acc > NIR] : 0.1421
#>
#>                   Kappa : 0.378
#>
#>  Mcnemar's Test P-Value : 0.0412
#>
#>             Sensitivity : 0.333
#>             Specificity : 1.000
#>          Pos Pred Value : 1.000
#>          Neg Pred Value : 0.700
#>              Prevalence : 0.391
#>          Detection Rate : 0.130
#>    Detection Prevalence : 0.130
#>       Balanced Accuracy : 0.667
#>
#>        'Positive' Class : Death
#>
```

6.5.2 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.

```
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: 51, 49, 50, 51, 49, 51, ...
#> Resampling results across tuning parameters:
#>
#>   alpha  lambda    Accuracy  Kappa
#>   0.10   0.000491  0.671     0.324
#>   0.10   0.004909  0.669     0.318
#>   0.10   0.049093  0.680     0.339
#>   0.55   0.000491  0.671     0.324
#>   0.55   0.004909  0.671     0.322
#>   0.55   0.049093  0.695     0.365
#>   1.00   0.000491  0.671     0.324
#>   1.00   0.004909  0.672     0.326
#>   1.00   0.049093  0.714     0.414
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final values used for the model were alpha = 1 and lambda = 0.0491.
```

```
confusionMatrix(predict(model_glmnet, val_test_data[, -1]), val_test_data$outcome)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       3       2
#>    Recover     6      12
#>
#>                Accuracy : 0.652
#>                  95% CI : (0.427, 0.836)
#>     No Information Rate : 0.609
#>     P-Value [Acc > NIR] : 0.422
#>
#>                   Kappa : 0.207
#>
#>  Mcnemar's Test P-Value : 0.289
```

```
#>
#>             Sensitivity : 0.333
#>             Specificity : 0.857
#>          Pos Pred Value : 0.600
#>          Neg Pred Value : 0.667
#>              Prevalence : 0.391
#>          Detection Rate : 0.130
#>    Detection Prevalence : 0.217
#>       Balanced Accuracy : 0.595
#>
#>        'Positive' Class : Death
#>
```

6.5.3 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: 51, 49, 50, 51, 49, 51, ...
#> Resampling results across tuning parameters:
#>
#>   kmax  Accuracy  Kappa
#>   5     0.666     0.313
#>   7     0.653     0.274
#>   9     0.648     0.263
#>
#> 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.
```

```
confusionMatrix(predict(model_kknn, val_test_data[, -1]), val_test_data$outcome)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       5       3
#>    Recover     4      11
```

```
#>
#>                Accuracy : 0.696
#>                  95% CI : (0.471, 0.868)
#>     No Information Rate : 0.609
#>     P-Value [Acc > NIR] : 0.264
#>
#>                   Kappa : 0.348
#>
#>  Mcnemar's Test P-Value : 1.000
#>
#>             Sensitivity : 0.556
#>             Specificity : 0.786
#>          Pos Pred Value : 0.625
#>          Neg Pred Value : 0.733
#>              Prevalence : 0.391
#>          Detection Rate : 0.217
#>    Detection Prevalence : 0.348
#>       Balanced Accuracy : 0.671
#>
#>        'Positive' Class : Death
#>
```

6.5.4 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)
#> Warning: predictions failed for Fold01.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
```

```
#> Warning: predictions failed for Fold10.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
```

```
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
```

```
#> Warning: predictions failed for Fold03.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
```

```
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, : There were missing
#> Warning in train.default(x, y, weights = w, ...): missing values found in aggregated results
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: 51, 49, 50, 51, 49, 51, ...
#> Resampling results across tuning parameters:
#>
#>   lambda  Accuracy  Kappa
#>   0e+00     NaN       NaN
#>   1e-04   0.681     0.343
#>   1e-01   0.681     0.343
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was lambda = 1e-04.
```

```
confusionMatrix(predict(model_pda, val_test_data[, -1]), val_test_data$outcome)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       3       2
#>    Recover     6      12
#>
#>                Accuracy : 0.652
```

```
#>                  95% CI : (0.427, 0.836)
#>     No Information Rate : 0.609
#>     P-Value [Acc > NIR] : 0.422
#>
#>                   Kappa : 0.207
#>
#>  Mcnemar's Test P-Value : 0.289
#>
#>             Sensitivity : 0.333
#>             Specificity : 0.857
#>          Pos Pred Value : 0.600
#>          Neg Pred Value : 0.667
#>              Prevalence : 0.391
#>          Detection Rate : 0.130
#>    Detection Prevalence : 0.217
#>       Balanced Accuracy : 0.595
#>
#>        'Positive' Class : Death
#>
```

6.5.5 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.

```
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: 51, 49, 50, 51, 49, 51, ...
#> Resampling results:
#>
#>   Accuracy  Kappa
#>   0.682     0.358
```

```
confusionMatrix(predict(model_slda, val_test_data[, -1]), val_test_data$outcome)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       3       3
#>    Recover     6      11
```

```
#>
#>                Accuracy : 0.609
#>                  95% CI : (0.385, 0.803)
#>     No Information Rate : 0.609
#>     P-Value [Acc > NIR] : 0.590
#>
#>                   Kappa : 0.127
#>
#>  Mcnemar's Test P-Value : 0.505
#>
#>             Sensitivity : 0.333
#>             Specificity : 0.786
#>          Pos Pred Value : 0.500
#>          Neg Pred Value : 0.647
#>              Prevalence : 0.391
#>          Detection Rate : 0.130
#>    Detection Prevalence : 0.261
#>       Balanced Accuracy : 0.560
#>
#>        'Positive' Class : Death
#>
```

6.5.6 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)
#> 1234567891011121314151617181920212223242526272829301111111111111111111111111111111111111111111111111111
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: 51, 49, 50, 51, 49, 51, ...
#> Resampling results across tuning parameters:
#>
#>   threshold  Accuracy  Kappa
#>   0.142      0.709     0.416
#>   2.065      0.714     0.382
#>   3.987      0.590     0.000
#>
```

```
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was threshold = 2.06.
```

```
confusionMatrix(predict(model_pam, val_test_data[, -1]), val_test_data$outcome)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       1       3
#>    Recover     8      11
#>
#>                Accuracy : 0.522
#>                  95% CI : (0.306, 0.732)
#>     No Information Rate : 0.609
#>     P-Value [Acc > NIR] : 0.857
#>
#>                   Kappa : -0.115
#>
#>  Mcnemar's Test P-Value : 0.228
#>
#>             Sensitivity : 0.1111
#>             Specificity : 0.7857
#>          Pos Pred Value : 0.2500
#>          Neg Pred Value : 0.5789
#>              Prevalence : 0.3913
#>          Detection Rate : 0.0435
#>    Detection Prevalence : 0.1739
#>       Balanced Accuracy : 0.4484
#>
#>        'Positive' Class : Death
#>
```

6.5.7 Single C5.0 Tree

C5.0 is another tree-based modeling algorithm.

This model classified 15 out of 23 cases correctly.

```
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: 51, 49, 50, 51, 49, 51, ...
```

```
#> Resampling results:
#>
#>   Accuracy  Kappa
#>   0.696     0.359
```

```
confusionMatrix(predict(model_C5.0Tree, val_test_data[, -1]), val_test_data$outcome)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       4       1
#>    Recover     5      13
#>
#>                Accuracy : 0.739
#>                  95% CI : (0.516, 0.898)
#>     No Information Rate : 0.609
#>     P-Value [Acc > NIR] : 0.142
#>
#>                   Kappa : 0.405
#>
#>  Mcnemar's Test P-Value : 0.221
#>
#>             Sensitivity : 0.444
#>             Specificity : 0.929
#>          Pos Pred Value : 0.800
#>          Neg Pred Value : 0.722
#>              Prevalence : 0.391
#>          Detection Rate : 0.174
#>    Detection Prevalence : 0.217
#>       Balanced Accuracy : 0.687
#>
#>        'Positive' Class : Death
#>
```

6.5.8 Partial Least Squares

modeling with correlated features.

This model classified 15 out of 23 cases correctly.

```
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: 51, 49, 50, 51, 49, 51, ...
#> Resampling results across tuning parameters:
#>
#>   ncomp  Accuracy  Kappa
#>   1      0.663     0.315
#>   2      0.676     0.341
#>   3      0.691     0.376
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was ncomp = 3.
```

```
confusionMatrix(predict(model_pls, val_test_data[, -1]), val_test_data$outcome)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction Death Recover
#>    Death       2       3
#>    Recover     7      11
#>
#>                Accuracy : 0.565
#>                  95% CI : (0.345, 0.768)
#>     No Information Rate : 0.609
#>     P-Value [Acc > NIR] : 0.742
#>
#>                   Kappa : 0.009
#>
#>  Mcnemar's Test P-Value : 0.343
#>
#>             Sensitivity : 0.222
#>             Specificity : 0.786
#>          Pos Pred Value : 0.400
#>          Neg Pred Value : 0.611
#>              Prevalence : 0.391
#>          Detection Rate : 0.087
#>    Detection Prevalence : 0.217
#>       Balanced Accuracy : 0.504
#>
#>        'Positive' Class : Death
#>
```

6.6 Comparing accuracy of models

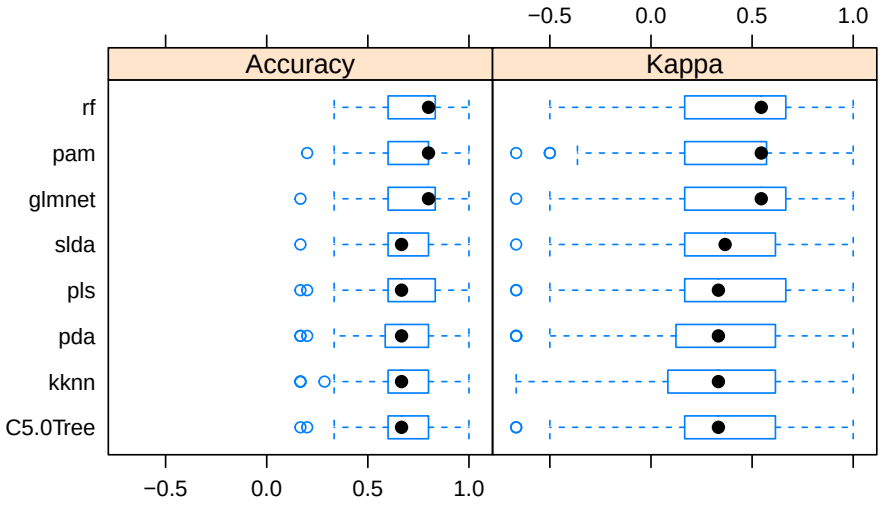
All models were similarly accurate.

6.6.1 Summary Accuracy and Kappa

```
# 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       -0.500   0.167  0.545 0.432   0.667    1    0
#> glmnet   -0.667   0.167  0.545 0.414   0.667    1    0
#> kknn     -0.667   0.125  0.333 0.313   0.615    1    0
#> pda      -0.667   0.142  0.333 0.343   0.615    1    0
#> slda     -0.667   0.167  0.367 0.358   0.615    1    0
#> pam      -0.667   0.167  0.545 0.382   0.571    1    0
#> C5.0Tree -0.667   0.167  0.333 0.359   0.615    1    0
#> pls      -0.667   0.167  0.333 0.376   0.667    1    0
#>
#> Accuracy
#>           Min. 1st Qu. Median  Mean 3rd Qu. Max. NA's
#> rf       0.333   0.600  0.800 0.732   0.833    1    0
#> glmnet   0.167   0.600  0.800 0.714   0.833    1    0
#> kknn     0.167   0.600  0.667 0.666   0.800    1    0
#> pda      0.167   0.593  0.667 0.681   0.800    1    0
#> slda     0.167   0.600  0.667 0.682   0.800    1    0
#> pam      0.200   0.600  0.800 0.714   0.800    1    0
#> C5.0Tree 0.167   0.600  0.667 0.696   0.800    1    0
#> pls      0.167   0.600  0.667 0.691   0.833    1    0
```

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



6.6.2 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_10      4.237        3.76    -0.1709        Death    uncertain  uncertain
#> case_11      5.181        2.82    -0.8778        Death    uncertain  uncertain
#> case_116     2.412        5.59     1.2123      Recover    uncertain  uncertain
#> case_12      5.219        2.78    -0.9085        Death    uncertain  uncertain
#> case_121     2.356        5.64     1.2606        Death    uncertain  uncertain
#> case_127     0.694        7.31     3.3972      Recover      Recover    CORRECT
#> case_131     0.685        7.31     3.4164      Recover      Recover    CORRECT
#> case_133     0.649        7.35     3.5024      Recover      Recover    CORRECT
#> case_135     2.027        5.97     1.5589        Death      Recover      wrong
#> case_2       2.161        5.84     1.4337        Death    uncertain  uncertain
```

```
#> case_20      3.144        4.86     0.6272      Recover    uncertain  uncertain
#> case_30      4.493        3.51    -0.3576      Recover    uncertain  uncertain
#> case_45      2.594        5.41     1.0590        Death    uncertain  uncertain
#> case_5       3.019        4.98     0.7227      Recover    uncertain  uncertain
#> case_55      3.925        4.08     0.0543      Recover    uncertain  uncertain
#> case_59      1.894        6.11     1.6886      Recover      Recover    CORRECT
#> case_72      2.545        5.46     1.1002      Recover    uncertain  uncertain
#> case_74      2.339        5.66     1.2748      Recover    uncertain  uncertain
#> case_77      0.845        7.15     3.0819      Recover      Recover    CORRECT
#> case_8       2.237        5.76     1.3650        Death    uncertain  uncertain
#> case_89      1.712        6.29     1.8772      Recover      Recover    CORRECT
#> case_97      2.959        5.04     0.7687      Recover    uncertain  uncertain
#> case_98      4.798        3.20    -0.5833        Death    uncertain  uncertain
```

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

6.7 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)
#> Warning: predictions failed for Fold01.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
```

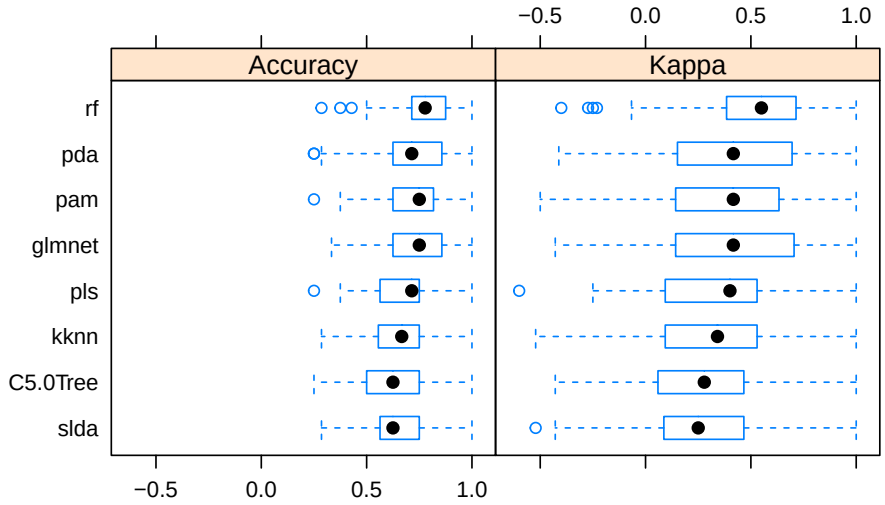
```
#> Warning: predictions failed for Fold04.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep01: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep02: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep03: lambda=0e+00 Error in mindist[l] <- ndist[l] :
```

```
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep04: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep05: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
```

```
#> Warning: predictions failed for Fold07.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep06: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep07: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep08: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
```

```
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep09: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold01.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold02.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold03.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold04.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold05.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold06.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold07.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold08.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold09.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning: predictions failed for Fold10.Rep10: lambda=0e+00 Error in mindist[l] <- ndist[l] :
#>   NAs are not allowed in subscripted assignments
#> Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, : There were missing
#> Warning in train.default(x, y, weights = w, ...): missing values found in aggregated results
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)
#> 12345678910111213141516171819202122232425262728293011111111111111111111111111111111111111111111111111
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.

6.7.1 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     1.854        6.15     1.7287           Recover
#> case_101     5.432        2.57    -1.0807         uncertain
#> case_102     2.806        5.19     0.8887         uncertain
#> case_103     2.342        5.66     1.2729         uncertain
#> case_104     1.744        6.26     1.8432           Recover
#> case_105     0.955        7.04     2.8828           Recover
#> case_108     4.489        3.51    -0.3543         uncertain
#> case_109     4.515        3.48    -0.3736         uncertain
#> case_110     2.411        5.59     1.2132         uncertain
#> case_112     2.632        5.37     1.0281         uncertain
#> case_113     2.198        5.80     1.4004         uncertain
#> case_114     3.339        4.66     0.4810         uncertain
#> case_115     1.112        6.89     2.6307           Recover
#> case_118     2.778        5.22     0.9109         uncertain
#> case_120     2.213        5.79     1.3868         uncertain
#> case_122     3.235        4.77     0.5590         uncertain
#> case_126     3.186        4.81     0.5952         uncertain
#> case_130     2.300        5.70     1.3091         uncertain
#> case_132     4.473        3.53    -0.3427         uncertain
#> case_136     3.281        4.72     0.5243         uncertain
#> case_15      2.270        5.73     1.3355         uncertain
#> case_16      2.820        5.18     0.8772         uncertain
#> case_22      4.779        3.22    -0.5689         uncertain
#> case_28      2.862        5.14     0.8445         uncertain
#> case_31      2.412        5.59     1.2117         uncertain
#> case_32      2.591        5.41     1.0616         uncertain
#> case_38      2.060        5.94     1.5280           Recover
#> case_39      4.749        3.25    -0.5466         uncertain
#> case_4       5.342        2.66    -1.0074         uncertain
#> case_40      6.550        1.45    -2.1750             Death
#> case_41      4.611        3.39    -0.4441         uncertain
#> case_42      5.570        2.43    -1.1966         uncertain
#> case_47      2.563        5.44     1.0850         uncertain
#> case_48      4.850        3.15    -0.6224         uncertain
#> case_52      4.709        3.29    -0.5173         uncertain
#> case_54      2.718        5.28     0.9586         uncertain
#> case_56      6.394        1.61    -1.9933             Death
#> case_62      6.048        1.95    -1.6319             Death
#> case_63      2.337        5.66     1.2766         uncertain
#> case_66      2.176        5.82     1.4205         uncertain
#> case_67      1.893        6.11     1.6895           Recover
#> case_68      3.907        4.09     0.0672         uncertain
#> case_69      4.465        3.53    -0.3370         uncertain
#> case_70      3.885        4.12     0.0833         uncertain
#> case_71      2.524        5.48     1.1172         uncertain
#> case_80      2.759        5.24     0.9261         uncertain
```

```
#> case_84      3.661        4.34     0.2448         uncertain
#> case_85      4.921        3.08    -0.6762         uncertain
#> case_86      3.563        4.44     0.3164         uncertain
#> case_88      0.566        7.43     3.7160           Recover
#> case_9       5.981        2.02    -1.5665             Death
#> case_90      3.570        4.43     0.3116         uncertain
#> case_92      4.664        3.34    -0.4835         uncertain
#> case_93      2.056        5.94     1.5319           Recover
#> case_95      4.495        3.50    -0.3589         uncertain
#> case_96      1.313        6.69     2.3482           Recover
#> case_99      4.799        3.20    -0.5839         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     1.854        6.15       1.73           Recover
#> 2     1.744        6.26       1.84           Recover
#> 3     0.955        7.04       2.88           Recover
#> 4     1.112        6.89       2.63           Recover
#> 5     2.060        5.94       1.53           Recover
#> 6     1.893        6.11       1.69           Recover
#> 7     0.566        7.43       3.72           Recover
#> 8     2.056        5.94       1.53           Recover
#> 9     1.313        6.69       2.35           Recover
```

6.7.2 Predicted outcome

```
results %>%
  group_by(predicted_outcome) %>%
  summarize(n = n())
#> # A tibble: 3 x 2
#>   predicted_outcome     n
#>   <chr>             <int>
#> 1 Death                 4
#> 2 Recover               9
#> 3 uncertain            44
```

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

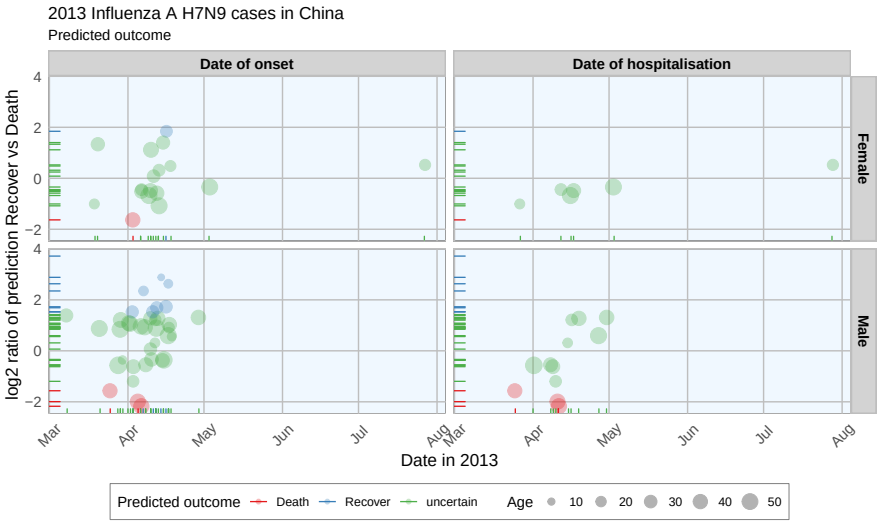
```
results_combined <- merge(results[, -c(1:16)],
                          fluH7N9_china_2013[which(fluH7N9_china_2013$case_ID
                                                   %in% rownames(results)), ],
                          by.x = "row.names", by.y = "case_ID")
# results_combined <- results_combined[, -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      1.7287           Recover     100    2013-04-16                    <NA>      m  58  Zhejiang
#> 2     -1.0807         uncertain     101    2013-04-13                    <NA>      f  79  Zhejiang
#> 3      0.8887         uncertain     102    2013-04-12                    <NA>      m  81  Zhejiang
```

```
#> 4     1.2729       uncertain   103   2013-04-13   2013-04-19   m  68   Jiangsu
#> 5     1.8432         Recover   104   2013-04-16         <NA>   f  54  Zhejiang
#> 6     2.8828         Recover   105   2013-04-14         <NA>   m  32  Zhejiang
#> 7    -0.3543       uncertain   108   2013-04-15         <NA>   m  84  Zhejiang
#> 8    -0.3736       uncertain   109   2013-04-15         <NA>   m  62  Zhejiang
#> 9     1.2132       uncertain   110   2013-04-12   2013-04-16   m  53    Taiwan
#> 10    1.0281       uncertain   112   2013-04-17         <NA>   m  69   Jiangxi
#> 11    1.4004       uncertain   113   2013-04-15         <NA>   f  60  Zhejiang
#> 12    0.4810       uncertain   114   2013-04-18         <NA>   f  50  Zhejiang
#> 13    2.6307         Recover   115   2013-04-17         <NA>   m  38  Zhejiang
#> 14    0.9109       uncertain   118   2013-04-17         <NA>   m  49   Jiangsu
#> 15    1.3868       uncertain   120   2013-03-08         <NA>   m  60   Jiangsu
#> 16    0.5590       uncertain   122   2013-04-18         <NA>   m  38  Zhejiang
#> 17    0.5952       uncertain   126   2013-04-17   2013-04-27   m  80    Fujian
#> 18    1.3091       uncertain   130   2013-04-29   2013-04-30   m  69    Fujian
#> 19   -0.3427       uncertain   132   2013-05-03   2013-05-03   f  79   Jiangxi
#> 20    0.5243       uncertain   136   2013-07-27   2013-07-28   f  51 Guangdong
#> 21    1.3355       uncertain    15   2013-03-20         <NA>   f  61   Jiangsu
#> 22    0.8772       uncertain    16   2013-03-21         <NA>   m  79   Jiangsu
#> 23   -0.5689       uncertain    22   2013-03-28   2013-04-01   m  85   Jiangsu
#> 24    0.8445       uncertain    28   2013-03-29         <NA>   m  79  Zhejiang
#> 25    1.2117       uncertain    31   2013-03-29         <NA>   m  70   Jiangsu
#> 26    1.0616       uncertain    32   2013-04-02         <NA>   m  74   Jiangsu
#> 27    1.5280         Recover    38   2013-04-03         <NA>   m  56   Jiangsu
#> 28   -0.5466       uncertain    39   2013-04-08   2013-04-08   m  66  Zhejiang
#> 29   -1.0074       uncertain     4   2013-03-19   2013-03-27   f  45   Jiangsu
#> 30   -2.1750           Death    40   2013-04-06   2013-04-11   m  74  Zhejiang
#> 31   -0.4441       uncertain    41   2013-04-06   2013-04-12   f  54  Zhejiang
#> 32   -1.1966       uncertain    42   2013-04-03   2013-04-10   m  53  Shanghai
#> 33    1.0850       uncertain    47   2013-04-01         <NA>   m  72   Jiangsu
#> 34   -0.6224       uncertain    48   2013-04-03   2013-04-09   m  65  Zhejiang
#> 35   -0.5173       uncertain    52   2013-04-06         <NA>   f  64  Zhejiang
#> 36    0.9586       uncertain    54   2013-04-06         <NA>   m  75  Zhejiang
#> 37   -1.9933           Death    56   2013-04-05   2013-04-11   m  73  Shanghai
#> 38   -1.6319           Death    62   2013-04-03         <NA>   f  68  Zhejiang
#> 39    1.2766       uncertain    63   2013-04-10         <NA>   m  60     Anhui
#> 40    1.4205       uncertain    66         <NA>         <NA>   m  72   Jiangsu
#> 41    1.6895         Recover    67   2013-04-12         <NA>   m  56  Zhejiang
#> 42    0.0672       uncertain    68   2013-04-10         <NA>   m  57  Zhejiang
#> 43   -0.3370       uncertain    69   2013-04-10         <NA>   m  62  Zhejiang
#> 44    0.0833       uncertain    70   2013-04-11         <NA>   f  58  Zhejiang
#> 45    1.1172       uncertain    71   2013-04-10         <NA>   f  72  Zhejiang
#> 46    0.9261       uncertain    80   2013-04-08         <NA>   m  74  Zhejiang
#> 47    0.2448       uncertain    84         <NA>         <NA>   f  26   Jiangsu
#> 48   -0.6762       uncertain    85   2013-04-09   2013-04-16   f  80  Shanghai
#> 49    0.3164       uncertain    86   2013-04-13         <NA>   f  54  Zhejiang
#> 50    3.7160         Recover    88         <NA>         <NA>   m   4   Beijing
#> 51   -1.5665           Death     9   2013-03-25   2013-03-25   m  67  Zhejiang
#> 52    0.3116       uncertain    90   2013-04-12   2013-04-15   m  43  Zhejiang
#> 53   -0.4835       uncertain    92   2013-04-10   2013-04-17   f  66  Zhejiang
#> 54    1.5319         Recover    93   2013-04-11         <NA>   m  56  Zhejiang
#> 55   -0.3589       uncertain    95   2013-03-30         <NA>   m  37  Zhejiang
#> 56    2.3482         Recover    96   2013-04-07         <NA>   m  43   Jiangsu
```

```
#> 57    -0.5839         uncertain      99    2013-04-12                    <NA>      f  68  Zhejiang
```

```
# 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"
```

```
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")
#> Warning: Removed 42 rows containing missing values (geom_point).
```



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.

6.8 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 3.6.0 (2019-04-26)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Ubuntu 18.04.3 LTS
#>
#> Matrix products: default
#> BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/libopenblasp-r0.2.20.so
#>
#> locale:
#>  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C                   LC_TIME=en_US.UTF-8        LC_COLLATE=en_
```

```
#>
#> 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_3.6.0
#> [50] pls_2.7-1           mitml_0.3-7         crayon_1.3.4        splines_3.6.0       zeallot_0.1.0
```