Applications of Machine Learning

Alfonso R. Reyes

2019-05-15

Contents

Chapter 1

Prerequisites

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

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

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

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

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

Chapter 2

Sonar Standalone Model with Random Forest

Classification problem

2.1 Introduction

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

2.2 Load libraries

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

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

2.3 Explore data

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

```
#> Observations: 208
#> Variables: 61
#> $ V1    <dbl> 0.0200, 0.0453, 0.0262, 0.0100, 0.0762, 0.0286, 0.0317, ...
#> $ V2    <dbl> 0.0371, 0.0523, 0.0582, 0.0171, 0.0666, 0.0453, 0.0956, ...
#> $ V3    <dbl> 0.0428, 0.0843, 0.1099, 0.0623, 0.0481, 0.0277, 0.1321, ...
#> $ V4    <dbl> 0.0207, 0.0689, 0.1083, 0.0205, 0.0394, 0.0174, 0.1408, ...
#> $ V5    <dbl> 0.0954, 0.1183, 0.0974, 0.0205, 0.0590, 0.0384, 0.1674, ...
#> $ V6    <dbl> 0.0986, 0.2583, 0.2280, 0.0368, 0.0649, 0.0990, 0.1710, ...
```

```
#> $ V7    <dbl> 0.1539, 0.2156, 0.2431, 0.1098, 0.1209, 0.1201, 0.0731, ...
#> $ V8    <dbl> 0.1601, 0.3481, 0.3771, 0.1276, 0.2467, 0.1833, 0.1401, ...
#> $ V9    <dbl> 0.3109, 0.3337, 0.5598, 0.0598, 0.3564, 0.2105, 0.2083, ...
#> $ V10   <dbl> 0.2111, 0.2872, 0.6194, 0.1264, 0.4459, 0.3039, 0.3513, ...
#> $ V11   <dbl> 0.1609, 0.4918, 0.6333, 0.0881, 0.4152, 0.2988, 0.1786, ...
#> $ V12   <dbl> 0.1582, 0.6552, 0.7060, 0.1992, 0.3952, 0.4250, 0.0658, ...
#> $ V13   <dbl> 0.2238, 0.6919, 0.5544, 0.0184, 0.4256, 0.6343, 0.0513, ...
#> $ V14   <dbl> 0.0645, 0.7797, 0.5320, 0.2261, 0.4135, 0.8198, 0.3752, ...
#> $ V15   <dbl> 0.0660, 0.7464, 0.6479, 0.1729, 0.4528, 1.0000, 0.5419, ...
#> $ V16   <dbl> 0.2273, 0.9444, 0.6931, 0.2131, 0.5326, 0.9988, 0.5440, ...
#> $ V17   <dbl> 0.3100, 1.0000, 0.6759, 0.0693, 0.7306, 0.9508, 0.5150, ...
#> $ V18   <dbl> 0.2999, 0.8874, 0.7551, 0.2281, 0.6193, 0.9025, 0.4262, ...
#> $ V19   <dbl> 0.5078, 0.8024, 0.8929, 0.4060, 0.2032, 0.7234, 0.2024, ...
#> $ V20   <dbl> 0.4797, 0.7818, 0.8619, 0.3973, 0.4636, 0.5122, 0.4233, ...
#> $ V21   <dbl> 0.5783, 0.5212, 0.7974, 0.2741, 0.4148, 0.2074, 0.7723, ...
#> $ V22   <dbl> 0.5071, 0.4052, 0.6737, 0.3690, 0.4292, 0.3985, 0.9735, ...
#> $ V23   <dbl> 0.4328, 0.3957, 0.4293, 0.5556, 0.5730, 0.5890, 0.9390, ...
#> $ V24   <dbl> 0.5550, 0.3914, 0.3648, 0.4846, 0.5399, 0.2872, 0.5559, ...
#> $ V25   <dbl> 0.6711, 0.3250, 0.5331, 0.3140, 0.3161, 0.2043, 0.5268, ...
#> $ V26   <dbl> 0.6415, 0.3200, 0.2413, 0.5334, 0.2285, 0.5782, 0.6826, ...
#> $ V27   <dbl> 0.7104, 0.3271, 0.5070, 0.5256, 0.6995, 0.5389, 0.5713, ...
#> $ V28   <dbl> 0.8080, 0.2767, 0.8533, 0.2520, 1.0000, 0.3750, 0.5429, ...
#> $ V29   <dbl> 0.6791, 0.4423, 0.6036, 0.2090, 0.7262, 0.3411, 0.2177, ...
#> $ V30   <dbl> 0.3857, 0.2028, 0.8514, 0.3559, 0.4724, 0.5067, 0.2149, ...
#> $ V31   <dbl> 0.1307, 0.3788, 0.8512, 0.6260, 0.5103, 0.5580, 0.5811, ...
#> $ V32   <dbl> 0.2604, 0.2947, 0.5045, 0.7340, 0.5459, 0.4778, 0.6323, ...
#> $ V33   <dbl> 0.5121, 0.1984, 0.1862, 0.6120, 0.2881, 0.3299, 0.2965, ...
#> $ V34   <dbl> 0.7547, 0.2341, 0.2709, 0.3497, 0.0981, 0.2198, 0.1873, ...
#> $ V35   <dbl> 0.8537, 0.1306, 0.4232, 0.3953, 0.1951, 0.1407, 0.2969, ...
#> $ V36   <dbl> 0.8507, 0.4182, 0.3043, 0.3012, 0.4181, 0.2856, 0.5163, ...
#> $ V37   <dbl> 0.6692, 0.3835, 0.6116, 0.5408, 0.4604, 0.3807, 0.6153, ...
#> $ V38   <dbl> 0.6097, 0.1057, 0.6756, 0.8814, 0.3217, 0.4158, 0.4283, ...
#> $ V39   <dbl> 0.4943, 0.1840, 0.5375, 0.9857, 0.2828, 0.4054, 0.5479, ...
#> $ V40   <dbl> 0.2744, 0.1970, 0.4719, 0.9167, 0.2430, 0.3296, 0.6133, ...
#> $ V41   <dbl> 0.0510, 0.1674, 0.4647, 0.6121, 0.1979, 0.2707, 0.5017, ...
#> $ V42   <dbl> 0.2834, 0.0583, 0.2587, 0.5006, 0.2444, 0.2650, 0.2377, ...
#> $ V43   <dbl> 0.2825, 0.1401, 0.2129, 0.3210, 0.1847, 0.0723, 0.1957, ...
#> $ V44   <dbl> 0.4256, 0.1628, 0.2222, 0.3202, 0.0841, 0.1238, 0.1749, ...
#> $ V45   <dbl> 0.2641, 0.0621, 0.2111, 0.4295, 0.0692, 0.1192, 0.1304, ...
#> $ V46   <dbl> 0.1386, 0.0203, 0.0176, 0.3654, 0.0528, 0.1089, 0.0597, ...
#> $ V47   <dbl> 0.1051, 0.0530, 0.1348, 0.2655, 0.0357, 0.0623, 0.1124, ...
#> $ V48   <dbl> 0.1343, 0.0742, 0.0744, 0.1576, 0.0085, 0.0494, 0.1047, ...
#> $ V49   <dbl> 0.0383, 0.0409, 0.0130, 0.0681, 0.0230, 0.0264, 0.0507, ...
#> $ V50   <dbl> 0.0324, 0.0061, 0.0106, 0.0294, 0.0046, 0.0081, 0.0159, ...
#> $ V51   <dbl> 0.0232, 0.0125, 0.0033, 0.0241, 0.0156, 0.0104, 0.0195, ...
#> $ V52   <dbl> 0.0027, 0.0084, 0.0232, 0.0121, 0.0031, 0.0045, 0.0201, ...
#> $ V53   <dbl> 0.0065, 0.0089, 0.0166, 0.0036, 0.0054, 0.0014, 0.0248, ...
#> $ V54   <dbl> 0.0159, 0.0048, 0.0095, 0.0150, 0.0105, 0.0038, 0.0131, ...
#> $ V55   <dbl> 0.0072, 0.0094, 0.0180, 0.0085, 0.0110, 0.0013, 0.0070, ...
#> $ V56   <dbl> 0.0167, 0.0191, 0.0244, 0.0073, 0.0015, 0.0089, 0.0138, ...
#> $ V57   <dbl> 0.0180, 0.0140, 0.0316, 0.0050, 0.0072, 0.0057, 0.0092, ...
#> $ V58   <dbl> 0.0084, 0.0049, 0.0164, 0.0044, 0.0048, 0.0027, 0.0143, ...
#> $ V59   <dbl> 0.0090, 0.0052, 0.0095, 0.0040, 0.0107, 0.0051, 0.0036, ...
#> $ V60   <dbl> 0.0032, 0.0044, 0.0078, 0.0117, 0.0094, 0.0062, 0.0103, ...
```

```
#> $ Class <fct> R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R,...
```

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

```
#> # A tibble: 208 x 61
#>        V1     V2     V3     V4     V5     V6     V7     V8     V9    V10
#>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
#>  1 0.02   0.0371 0.0428 0.0207 0.0954 0.0986 0.154  0.160  0.311  0.211
#>  2 0.0453 0.0523 0.0843 0.0689 0.118  0.258  0.216  0.348  0.334  0.287
#>  3 0.0262 0.0582 0.110  0.108  0.0974 0.228  0.243  0.377  0.560  0.619
#>  4 0.01   0.0171 0.0623 0.0205 0.0205 0.0368 0.110  0.128  0.0598 0.126
#>  5 0.0762 0.0666 0.0481 0.0394 0.059  0.0649 0.121  0.247  0.356  0.446
#>  6 0.0286 0.0453 0.0277 0.0174 0.0384 0.099  0.120  0.183  0.210  0.304
#>  7 0.0317 0.0956 0.132  0.141  0.167  0.171  0.0731 0.140  0.208  0.351
#>  8 0.0519 0.0548 0.0842 0.0319 0.116  0.0922 0.103  0.0613 0.146  0.284
#>  9 0.0223 0.0375 0.0484 0.0475 0.0647 0.0591 0.0753 0.0098 0.0684 0.149
#> 10 0.0164 0.0173 0.0347 0.007  0.0187 0.0671 0.106  0.0697 0.0962 0.0251
#> # ... with 198 more rows, and 51 more variables: V11 <dbl>, V12 <dbl>,
#> #   V13 <dbl>, V14 <dbl>, V15 <dbl>, V16 <dbl>, V17 <dbl>, V18 <dbl>,
#> #   V19 <dbl>, V20 <dbl>, V21 <dbl>, V22 <dbl>, V23 <dbl>, V24 <dbl>,
#> #   V25 <dbl>, V26 <dbl>, V27 <dbl>, V28 <dbl>, V29 <dbl>, V30 <dbl>,
#> #   V31 <dbl>, V32 <dbl>, V33 <dbl>, V34 <dbl>, V35 <dbl>, V36 <dbl>,
#> #   V37 <dbl>, V38 <dbl>, V39 <dbl>, V40 <dbl>, V41 <dbl>, V42 <dbl>,
#> #   V43 <dbl>, V44 <dbl>, V45 <dbl>, V46 <dbl>, V47 <dbl>, V48 <dbl>,
#> #   V49 <dbl>, V50 <dbl>, V51 <dbl>, V52 <dbl>, V53 <dbl>, V54 <dbl>,
#> #   V55 <dbl>, V56 <dbl>, V57 <dbl>, V58 <dbl>, V59 <dbl>, V60 <dbl>,
#> #   Class <fct>
```

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

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

print(fit.rf)
```

```
#> Random Forest
#>
#> 167 samples
#>  60 predictor
#>   2 classes: 'M', 'R'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 3 times)
#> Summary of sample sizes: 150, 151, 150, 150, 150, 151, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy   Kappa
```

```
#>    2    0.8525572  0.7008578
#>   31    0.8184722  0.6341306
#>   60    0.7944526  0.5856805
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 2.
```

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

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

Accuracy: 85.26% at mtry=2

2.4 Apply tuning parameters for final model

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

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

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

```
#>          Neg Pred Value : 0.8750
#>              Prevalence : 0.5366
#>          Detection Rate : 0.4878
#>    Detection Prevalence : 0.6098
#>       Balanced Accuracy : 0.8230
#>
#>        'Positive' Class : M
#>
```

Accuracy: 82.93%

2.5 Save model

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

2.6 Use the saved model

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

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

2.7 Make prediction with new data

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

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  M  R
#>          M 20  5
#>          R  2 14
#>
#>                Accuracy : 0.8293
```

```
#>                 95% CI : (0.6794, 0.9285)
#>    No Information Rate : 0.5366
#>    P-Value [Acc > NIR] : 8.511e-05
#>
#>                  Kappa : 0.653
#>
#>  Mcnemar's Test P-Value : 0.4497
#>
#>             Sensitivity : 0.9091
#>             Specificity : 0.7368
#>          Pos Pred Value : 0.8000
#>          Neg Pred Value : 0.8750
#>              Prevalence : 0.5366
#>          Detection Rate : 0.4878
#>    Detection Prevalence : 0.6098
#>       Balanced Accuracy : 0.8230
#>
#>        'Positive' Class : M
#>
```

Chapter 3

Glass classification

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

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

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

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

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

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

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

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

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

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

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

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

```
#>     true
#> pred  1  2  3  5  6  7
#>    1 13  5  6  0  1  0
#>    2  6 24  2  3  2  3
#>    3  0  0  0  0  0  0
#>    5  0  0  0  1  0  0
#>    6  0  0  0  0  0  0
#>    7  0  0  0  0  0  5
```

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

```
#>     true
#> pred  1  2  3  5  6  7
#>    1 12  4  6  0  1  1
#>    2  6 23  2  0  1  1
#>    3  0  0  0  0  0  0
#>    5  0  2  0  3  1  0
#>    6  0  0  0  0  0  0
#>    7  1  0  0  1  0  6
```

3.0.1 Comparison test sets

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

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  1  2  3  5  6  7
#>          1 13  5  6  0  1  0
#>          2  6 24  2  3  2  3
#>          3  0  0  0  0  0  0
#>          5  0  0  0  1  0  0
#>          6  0  0  0  0  0  0
#>          7  0  0  0  0  0  5
#>
#> Overall Statistics
#>
#>                Accuracy : 0.6056
#>                  95% CI : (0.4825, 0.7197)
#>     No Information Rate : 0.4085
#>     P-Value [Acc > NIR] : 0.0006275
#>
#>                   Kappa : 0.4087
#>
```

```
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
#> Sensitivity            0.6842   0.8276   0.0000  0.25000  0.00000  0.62500
#> Specificity            0.7692   0.6190   1.0000  1.00000  1.00000  1.00000
#> Pos Pred Value         0.5200   0.6000      NaN  1.00000      NaN  1.00000
#> Neg Pred Value         0.8696   0.8387   0.8873  0.95714  0.95775  0.95455
#> Prevalence             0.2676   0.4085   0.1127  0.05634  0.04225  0.11268
#> Detection Rate         0.1831   0.3380   0.0000  0.01408  0.00000  0.07042
#> Detection Prevalence   0.3521   0.5634   0.0000  0.01408  0.00000  0.07042
#> Balanced Accuracy      0.7267   0.7233   0.5000  0.62500  0.50000  0.81250
```

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

```
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  1  2  3  5  6  7
#>          1 12  4  6  0  1  1
#>          2  6 23  2  0  1  1
#>          3  0  0  0  0  0  0
#>          5  0  2  0  3  1  0
#>          6  0  0  0  0  0  0
#>          7  1  0  0  1  0  6
#>
#> Overall Statistics
#>
#>                Accuracy : 0.6197
#>                  95% CI : (0.4967, 0.7324)
#>     No Information Rate : 0.4085
#>     P-Value [Acc > NIR] : 0.0002651
#>
#>                   Kappa : 0.4585
#>
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
#> Sensitivity            0.6316   0.7931   0.0000  0.75000  0.00000  0.75000
#> Specificity            0.7692   0.7619   1.0000  0.95522  1.00000  0.96825
#> Pos Pred Value         0.5000   0.6970      NaN  0.50000      NaN  0.75000
#> Neg Pred Value         0.8511   0.8421   0.8873  0.98462  0.95775  0.96825
#> Prevalence             0.2676   0.4085   0.1127  0.05634  0.04225  0.11268
#> Detection Rate         0.1690   0.3239   0.0000  0.04225  0.00000  0.08451
#> Detection Prevalence   0.3380   0.4648   0.0000  0.08451  0.00000  0.11268
#> Balanced Accuracy      0.7004   0.7775   0.5000  0.85261  0.50000  0.85913
```

3.0.2 Comparison with resamples

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

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

```
set.seed(1234567)

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

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

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

summary(results)
```

```
#>
#> Call:
#> summary.resamples(object = results)
#>
#> Models: svm, rpart
#> Number of resamples: 25
#>
#> Accuracy
#>            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> svm   0.5652174 0.6346154 0.6851852 0.6799615 0.7254902 0.7735849    0
#> rpart 0.5192308 0.6101695 0.6274510 0.6335048 0.6545455 0.7200000    0
#>
#> Kappa
#>            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
#> svm   0.4110115 0.4744681 0.5375833 0.5400856 0.6020260 0.6641550    0
#> rpart 0.3483709 0.4224193 0.4795918 0.4765636 0.5314223 0.6067416    0
```

Chapter 4

Ozone SVM

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

```
library(e1071)
library(rpart)

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

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

```
##          [,1]
## [1,] 11.89986
```

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

```
##         [,1]
## [1,] 20.2273
```

Chapter 5

A gentle introduction to support vector machines using R

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

5.1 Support vector machines in R

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

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

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

5.2 SVM on `iris` dataset

5.2.1 Training and test datasets

```
#load required library
library(e1071)

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

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

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

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

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

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

dim(trainset)
```

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

```
dim(testset)
```

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

5.2.2 Build the SVM model

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

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

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

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

5.2.3 Support Vectors

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

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

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

5.2.4 Predictions on training model

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

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

```
# [1] 0.9826087
```

5.2.5 Predictions on test model

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

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

```
# [1] 0.9142857
```

5.2.6 Confusion matrix and Accuracy

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

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

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

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

5.3 SVM with Radial Basis Function kernel. Linear

5.3.1 Training and test sets

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

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

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

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

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

5.3.2 Predictions on Training model

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

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

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

5.3.3 Predictions on test model

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

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

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

5.4 SVM with Radial Basis Function kernel. Non-linear

5.4.1 Predictions on training model

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

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

```
svm_model$gamma
```

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

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

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

5.4.2 Predictions on test model

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

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

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

5.4.3 Tuning of parameters

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

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

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

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

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

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

5.4.4 Prediction on training model with new parameters

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

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

5.4.5 Prediction on test model with new parameters

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

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

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

5.5 Wrapping up

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

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

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

Chapter 6

SMS spam. Naive Bayes. Classification

Dataset: https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/sms_spam.csv

Instructions: Machine Learning with R. Page 104.

```
library(tictoc)
```

```
sms_raw <- read.csv(file.path(data_raw_dir, "sms_spam.csv"), stringsAsFactors = FALSE)
```

```
str(sms_raw)
```

```
#> 'data.frame':    5574 obs. of  2 variables:
#>  $ type: chr  "ham" "ham" "spam" "ham" ...
#>  $ text: chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... C
```

6.0.1 convert type to a factor

```
sms_raw$type <- factor(sms_raw$type)
```

```
str(sms_raw$type)
```

```
#>  Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
```

```
table(sms_raw$type)
```

```
#>
#>  ham spam
#> 4827  747
```

```
library(tm)

sms_corpus <- VCorpus(VectorSource(sms_raw$text))
print(sms_corpus)
```

```
#> <<VCorpus>>
#> Metadata:  corpus specific: 0, document level (indexed): 0
#> Content:  documents: 5574
```

```
inspect(sms_corpus[1:2])
```

```
#> <<VCorpus>>
#> Metadata:  corpus specific: 0, document level (indexed): 0
#> Content:  documents: 2
#>
#> [[1]]
#> <<PlainTextDocument>>
#> Metadata:  7
#> Content:  chars: 111
#>
#> [[2]]
#> <<PlainTextDocument>>
#> Metadata:  7
#> Content:  chars: 29
```

```
# show some text
as.character(sms_corpus[[1]])
```

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

```
# show three documents
lapply(sms_corpus[1:3], as.character)
```

```
#> $`1`
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
#>
#> $`2`
#> [1] "Ok lar... Joking wif u oni..."
#>
#> $`3`
#> [1] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive
```

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

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

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

```
# converted to lowercase
as.character(sms_corpus_clean[[1]])
```

```
#> [1] "go until jurong point, crazy.. available only in bugis n great world la e buffet... cine there
```

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

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

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

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

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

```
library(SnowballC)
wordStem(c("learn", "learned", "learning", "learns"))
```

```
#> [1] "learn" "learn" "learn" "learn"
```

```
# stemming corpus
sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)
```

```
# remove white spaces
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)
```

```
# show what we've got so far
lapply(sms_corpus[1:3], as.character)
```

```
#> $`1`
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
#>
#> $`2`
#> [1] "Ok lar... Joking wif u oni..."
#>
#> $`3`
#> [1] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive
```

```
lapply(sms_corpus_clean[1:3], as.character)
```

```
#> $`1`
#> [1] "go jurong point crazi avail bugi n great world la e buffet cine got amor wat"
#>
#> $`2`
#> [1] "ok lar joke wif u oni"
#>
#> $`3`
#> [1] "free entri wkli comp win fa cup final tkts st may text fa receiv entri questionstd txt ratetc ap
```

6.1 Convert to Document Term Matrix

```
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
sms_dtm
```

```
#> <<DocumentTermMatrix (documents: 5574, terms: 6592)>>
#> Non-/sparse entries: 42608/36701200
#> Sparsity           : 100%
#> Maximal term length: 40
#> Weighting          : term frequency (tf)
```

6.2 split in training and test datasets

```
sms_dtm_train <- sms_dtm[1:4169, ]
sms_dtm_test  <- sms_dtm[4170:5559, ]
```

6.2.1 separate the labels

```
sms_train_labels <- sms_raw[1:4169, ]$type
sms_test_labels  <- sms_raw[4170:5559, ]$type
```

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

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

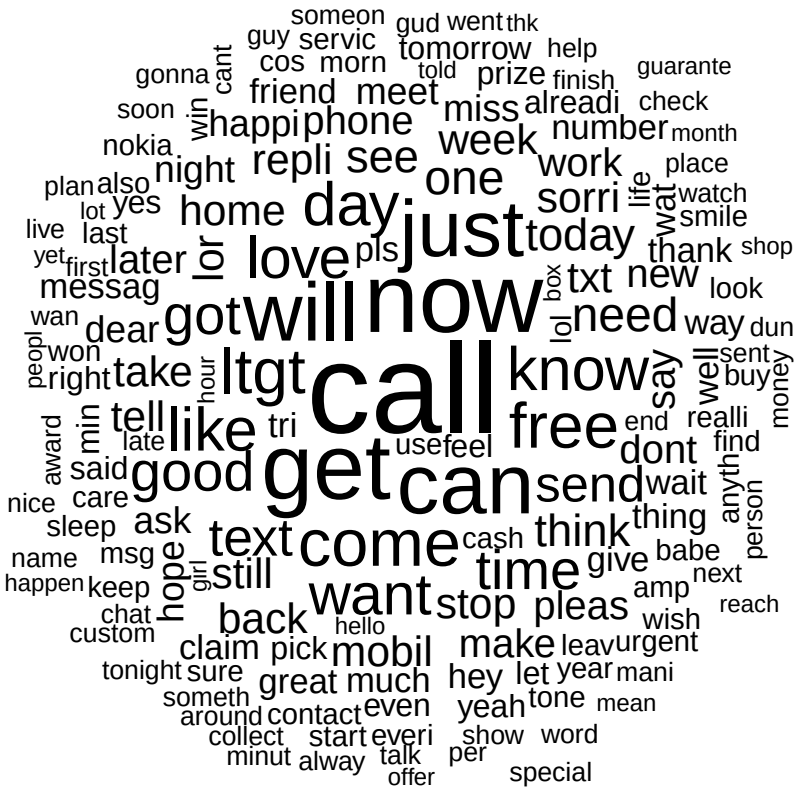
```
prop.table(table(sms_test_labels))
```

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

```
# convert dtm to matrix
sms_mat_train <- as.matrix(t(sms_dtm_train))
dtm.rs <- sort(rowSums(sms_mat_train), decreasing=TRUE)

# dataframe with word-frequency
dtm.df <- data.frame(word = names(dtm.rs), freq = as.integer(dtm.rs),
                     stringsAsFactors = FALSE)
```

6.3 plot wordcloud

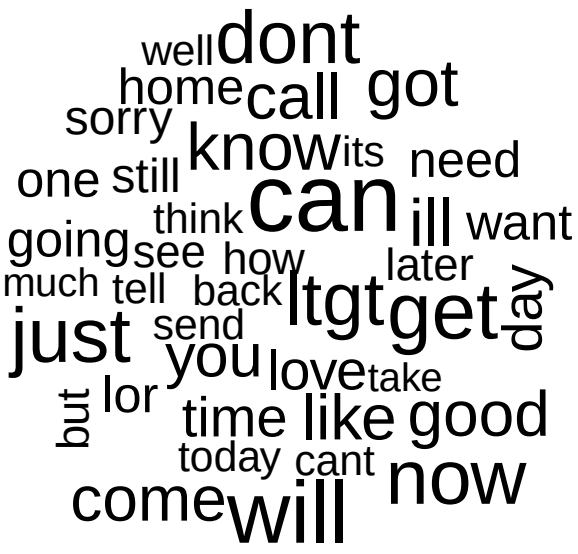
```
library(wordcloud)
wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE)
```




```
spam <- subset(sms_raw, type == "spam")
ham  <- subset(sms_raw, type == "ham")
```

```
wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
```


```
wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```



6.4 Limit Frequent words

```
# words that appear at least in 5 messages
sms_freq_words <- findFreqTerms(sms_dtm_train, 6)
```

```
str(sms_freq_words)
```

```
#>  chr [1:997] "abiola" "abl" "abt" "accept" "access" "account" "across" ...
```

6.4.1 get only frequent words

```
sms_dtm_freq_train<- sms_dtm_train[ , sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
```

6.4.2 function to change value to Yes/No

```
convert_counts <- function(x) {
    x <- ifelse(x > 0, "Yes", "No")
  }
```

```
# change from number to Yes/No
# also the result returns a matrix
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2,
                                        convert_counts)
sms_test  <- apply(sms_dtm_freq_test, MARGIN = 2,
                                       convert_counts)
```

```
# matrix of
# 4169 documents as rows
# 1159 terms as columns
dim(sms_train)
```

```
#> [1] 4169  997
```

```
length(sms_train_labels)
```

```
#> [1] 4169
```

```
# this is how the matrix looks
sms_train[1:10, 10:15]
```

```
#>     Terms
#> Docs add  address admir advanc aft  afternoon
#>   1  "No" "No"    "No"  "No"   "No" "No"
#>   2  "No" "No"    "No"  "No"   "No" "No"
#>   3  "No" "No"    "No"  "No"   "No" "No"
#>   4  "No" "No"    "No"  "No"   "No" "No"
#>   5  "No" "No"    "No"  "No"   "No" "No"
#>   6  "No" "No"    "No"  "No"   "No" "No"
#>   7  "No" "No"    "No"  "No"   "No" "No"
#>   8  "No" "No"    "No"  "No"   "No" "No"
#>   9  "No" "No"    "No"  "No"   "No" "No"
#>   10 "No" "No"    "No"  "No"   "No" "No"
```

```
library(e1071)
sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

```
sms_test_pred <- predict(sms_classifier, sms_test)
```

```
library(gmodels)
CrossTable(sms_test_pred, sms_test_labels,
    prop.chisq = FALSE, prop.t = FALSE,
    dnn = c('predicted', 'actual'))
```

```
#>
#>
#>    Cell Contents
#> |-------------------------|
#> |                       N |
#> |           N / Row Total |
#> |           N / Col Total |
#> |-------------------------|
#>
#>
#> Total Observations in Table:  1390
#>
#>
#>              | actual
#>    predicted |       ham |      spam | Row Total |
#> -------------|-----------|-----------|-----------|
#>          ham |      1202 |        21 |      1223 |
#>              |     0.983 |     0.017 |     0.880 |
#>              |     0.994 |     0.116 |           |
#> -------------|-----------|-----------|-----------|
#>         spam |         7 |       160 |       167 |
#>              |     0.042 |     0.958 |     0.120 |
#>              |     0.006 |     0.884 |           |
#> -------------|-----------|-----------|-----------|
#> Column Total |      1209 |       181 |      1390 |
```

```
#>              |     0.870 |     0.130 |           |
#> -------------|-----------|-----------|-----------|
#>
#>
```

Misclassified: 20+9 (frequency = 5) 25+7 (freq=4) 23+7 (freq=3) 25+8 (freq=2) 21+7 (freq=6)

Decreasing the minimum word frequency doesn't make the model better.

6.5 Improve model performance

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

```
tic()
sms_test_pred2 <- predict(sms_classifier2, sms_test)
toc()
```

```
#> 41.404 sec elapsed
```

```
CrossTable(sms_test_pred2, sms_test_labels,
    prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
    dnn = c('predicted', 'actual'))
```

```
#>
#>
#>    Cell Contents
#> |-------------------------|
#> |                       N |
#> |           N / Col Total |
#> |-------------------------|
#>
#>
#> Total Observations in Table:  1390
#>
#>
#>              | actual
#>    predicted |       ham |      spam | Row Total |
#> -------------|-----------|-----------|-----------|
#>          ham |      1203 |        28 |      1231 |
#>              |     0.995 |     0.155 |           |
#> -------------|-----------|-----------|-----------|
#>         spam |         6 |       153 |       159 |
#>              |     0.005 |     0.845 |           |
#> -------------|-----------|-----------|-----------|
#> Column Total |      1209 |       181 |      1390 |
#>              |     0.870 |     0.130 |           |
#> -------------|-----------|-----------|-----------|
#>
#>
```

Misclassified: 28+7

Chapter 7

Classification Tree: Vehicle example

Dataset: Vehicle (mlbench) Instructions: book "Applied Predictive Modeling Techniques", Lewis, N.D.

7.1 Load packages

```
library(tree)
library(mlbench)

data(Vehicle)
str(Vehicle)
```

```
#> 'data.frame':    846 obs. of  19 variables:
#>  $ Comp        : num  95 91 104 93 85 107 97 90 86 93 ...
#>  $ Circ        : num  48 41 50 41 44 57 43 43 34 44 ...
#>  $ D.Circ      : num  83 84 106 82 70 106 73 66 62 98 ...
#>  $ Rad.Ra      : num  178 141 209 159 205 172 173 157 140 197 ...
#>  $ Pr.Axis.Ra  : num  72 57 66 63 103 50 65 65 61 62 ...
#>  $ Max.L.Ra    : num  10 9 10 9 52 6 6 9 7 11 ...
#>  $ Scat.Ra     : num  162 149 207 144 149 255 153 137 122 183 ...
#>  $ Elong       : num  42 45 32 46 45 26 42 48 54 36 ...
#>  $ Pr.Axis.Rect: num  20 19 23 19 19 28 19 18 17 22 ...
#>  $ Max.L.Rect  : num  159 143 158 143 144 169 143 146 127 146 ...
#>  $ Sc.Var.Maxis: num  176 170 223 160 241 280 176 162 141 202 ...
#>  $ Sc.Var.maxis: num  379 330 635 309 325 957 361 281 223 505 ...
#>  $ Ra.Gyr      : num  184 158 220 127 188 264 172 164 112 152 ...
#>  $ Skew.Maxis  : num  70 72 73 63 127 85 66 67 64 64 ...
#>  $ Skew.maxis  : num  6 9 14 6 9 5 13 3 2 4 ...
#>  $ Kurt.maxis  : num  16 14 9 10 11 9 1 3 14 14 ...
#>  $ Kurt.Maxis  : num  187 189 188 199 180 181 200 193 200 195 ...
#>  $ Holl.Ra     : num  197 199 196 207 183 183 204 202 208 204 ...
#>  $ Class       : Factor w/ 4 levels "bus","opel","saab",..: 4 4 3 4 1 1 1 4 4 3 ...
```

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

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

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

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

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

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

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

7.2 Prepare data

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

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

7.3 Estimate the decision tree

```
fit <- tree(Class ~., data = trainset, split ="deviance")
fit
```

```
#> node), split, n, deviance, yval, (yprob)
#>       * denotes terminal node
#>
#>   1) root 500 1386.000 saab ( 0.248000 0.254000 0.258000 0.240000 )
#>     2) Elong < 41.5 229  489.100 opel ( 0.222707 0.410480 0.366812 0.000000 )
#>       4) Max.L.Ra < 7.5 62   77.560 bus ( 0.806452 0.096774 0.096774 0.000000 )
#>         8) Comp < 95.5 20   43.560 bus ( 0.400000 0.300000 0.300000 0.000000 )
#>          16) Pr.Axis.Ra < 67 12   16.640 opel ( 0.000000 0.500000 0.500000 0.000000 ) *
#>          17) Pr.Axis.Ra > 67 8    0.000 bus ( 1.000000 0.000000 0.000000 0.000000 ) *
#>         9) Comp > 95.5 42    0.000 bus ( 1.000000 0.000000 0.000000 0.000000 ) *
#>       5) Max.L.Ra > 7.5 167  241.800 opel ( 0.005988 0.526946 0.467066 0.000000 )
#>        10) Comp < 107.5 145  205.300 opel ( 0.006897 0.600000 0.393103 0.000000 )
#>          20) Sc.Var.maxis < 720.5 131  179.400 opel ( 0.000000 0.564885 0.435115 0.000000 ) *
#>          21) Sc.Var.maxis > 720.5 14    7.205 opel ( 0.071429 0.928571 0.000000 0.000000 ) *
#>        11) Comp > 107.5 22    8.136 saab ( 0.000000 0.045455 0.954545 0.000000 ) *
```

```
#>     3) Elong > 41.5 271  687.600 van ( 0.269373 0.121771 0.166052 0.442804 )
#>       6) Max.L.Ra < 8.5 200  537.900 bus ( 0.360000 0.165000 0.210000 0.265000 )
#>        12) Sc.Var.maxis < 297.5 86  168.200 van ( 0.000000 0.186047 0.244186 0.569767 )
#>          24) Max.L.Rect < 127.5 25   48.720 saab ( 0.000000 0.280000 0.560000 0.160000 ) *
#>          25) Max.L.Rect > 127.5 61   92.130 van ( 0.000000 0.147541 0.114754 0.737705 ) *
#>        13) Sc.Var.maxis > 297.5 114  228.700 bus ( 0.631579 0.149123 0.184211 0.035088 )
#>          26) D.Circ < 76.5 92  132.300 bus ( 0.782609 0.065217 0.130435 0.021739 )
#>            52) Skew.maxis < 10.5 84   89.720 bus ( 0.857143 0.023810 0.095238 0.023810 )
#>             104) Circ < 40.5 16   36.580 saab ( 0.312500 0.125000 0.500000 0.062500 )
#>               208) Kurt.Maxis < 191 8    6.028 saab ( 0.000000 0.000000 0.875000 0.125000 ) *
#>               209) Kurt.Maxis > 191 8   14.400 bus ( 0.625000 0.250000 0.125000 0.000000 ) *
#>             105) Circ > 40.5 68   10.420 bus ( 0.985294 0.000000 0.000000 0.014706 ) *
#>            53) Skew.maxis > 10.5 8   11.090 saab ( 0.000000 0.500000 0.500000 0.000000 ) *
#>          27) D.Circ > 76.5 22   40.930 opel ( 0.000000 0.500000 0.409091 0.090909 ) *
#>       7) Max.L.Ra > 8.5 71   35.280 van ( 0.014085 0.000000 0.042254 0.943662 )
#>        14) Skew.Maxis < 64.5 7    9.561 van ( 0.000000 0.000000 0.428571 0.571429 ) *
#>        15) Skew.Maxis > 64.5 64   10.300 van ( 0.015625 0.000000 0.000000 0.984375 ) *
```

```
# fit <- tree(Class ~., data = Vehicle[train,], split ="deviance")
# fit
```

We use deviance as the splitting criteria, a common alternative is to use split="gini".

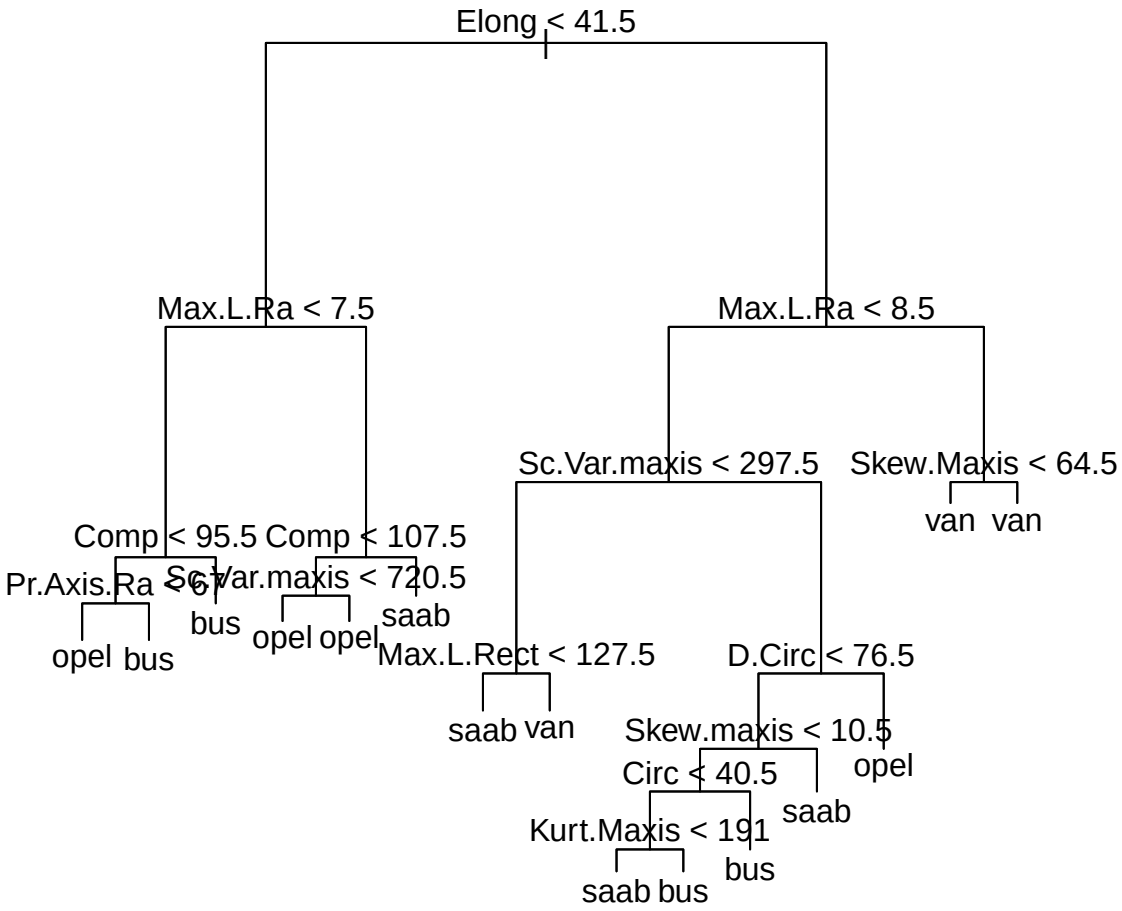
At each branch of the tree (after root) we see in order: 1. The branch number (e.g. in this case 1,2,14 and 15); 2. the split (e.g. Elong < 41.5); 3. the number of samples going along that split (e.g. 229); 4. the deviance associated with that split (e.g. 489.1); 5. the predicted class (e.g. opel); 6. the associated probabilities (e.g. (0.222707 0.410480 0.366812 0.000000); 7. and for a terminal node (or leaf), the symbol "*".

```
summary(fit)
```

```
#>
#> Classification tree:
#> tree(formula = Class ~ ., data = trainset, split = "deviance")
#> Variables actually used in tree construction:
#>  [1] "Elong"        "Max.L.Ra"     "Comp"         "Pr.Axis.Ra"
#>  [5] "Sc.Var.maxis" "Max.L.Rect"   "D.Circ"       "Skew.maxis"
#>  [9] "Circ"         "Kurt.Maxis"   "Skew.Maxis"
#> Number of terminal nodes:  15
#> Residual mean deviance:  0.9381 = 455 / 485
#> Misclassification error rate: 0.232 = 116 / 500
```

Notice that summary(fit) shows: 1. The type of tree, in this case a Classification tree; 2. the formula used to fit the tree; 3. the variables used to fit the tree; 4. the number of terminal nodes in this case 15; 5. the residual mean deviance - 0.9381; 6. the misclassification error rate 0.232 or 23.2%.

```
plot(fit); text(fit)
```



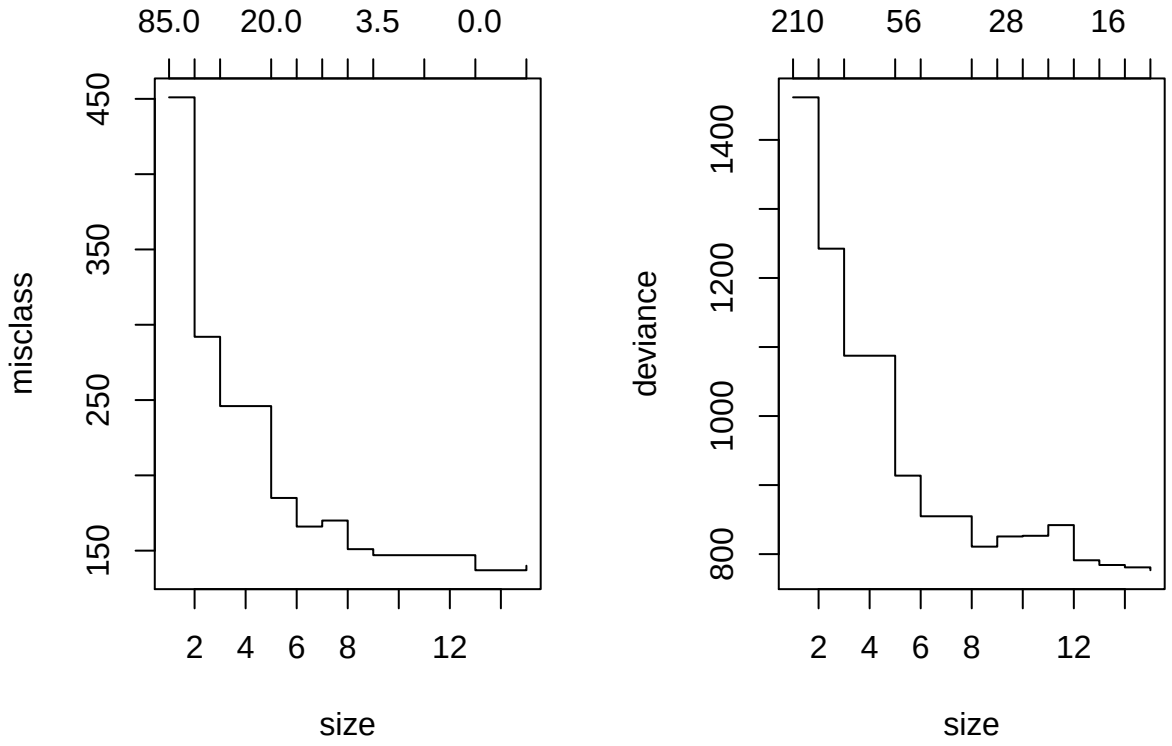
7.4 Assess model

Unfortunately, classification trees have a tendency to overfit the data. One approach to reduce this risk is to use cross-validation. For each hold out sample we fit the model and note at what level the tree gives the best results (using deviance or the misclassification rate). Then we hold out a different sample and repeat. This can be carried out using the `cv.tree()` function. We use a leave-one-out cross-validation using the misclassification rate and deviance (`FUN=prune.misclass`, followed by `FUN=prune.tree`).

```
fitM.cv <- cv.tree(fit, K=346, FUN = prune.misclass)
fitP.cv <- cv.tree(fit, K=346, FUN = prune.tree)
```

The results are plotted out side by side in Figure 1.2. The jagged lines shows where the minimum deviance / misclassification occurred with the cross-validated tree. Since the cross validated misclassification and deviance both reach their minimum close to the number of branches in the original fitted tree there is little to be gained from pruning this tree

```
par(mfrow = c(1, 2))
plot(fitM.cv)
plot(fitP.cv)
```



7.5 Make predictions

We use the validation data set and the fitted decision tree to predict vehicle classes; then we display the confusion matrix and calculate the error rate of the fitted tree. Overall, the model has an error rate of 32%.

```
testLabels <- Vehicle$Class[-train]
testLabels
```

```
#>   [1] bus  bus  saab saab bus  saab opel bus  saab bus  saab opel van  van
#>  [15] van  bus  van  saab bus  bus  van  opel opel opel bus  bus  van  bus
#>  [29] van  bus  opel van  bus  bus  opel bus  bus  van  bus  bus  opel opel
#>  [43] van  saab bus  bus  saab van  van  van  bus  saab van  saab opel opel
#>  [57] opel saab bus  van  opel opel opel saab bus  van  opel bus  van  bus
#>  [71] opel saab bus  bus  opel van  saab bus  opel opel van  van  bus  bus
#>  [85] opel van  saab van  van  saab van  saab van  bus  opel bus  saab bus
#>  [99] opel van  saab opel bus  van  saab van  bus  bus  bus  saab van  saab
#> [113] van  bus  bus  saab opel bus  saab opel van  van  bus  saab saab bus
#> [127] van  opel van  bus  saab van  bus  saab opel bus  opel opel van  opel
#> [141] van  bus  opel van  bus  bus  opel opel saab van  van  van  bus  bus
#> [155] opel bus  bus  saab opel bus  van  saab opel van  saab bus  saab opel
#> [169] van  bus  van  bus  opel opel saab van  opel opel bus  opel opel opel
#> [183] opel opel opel bus  opel van  opel opel saab opel van  opel saab saab
#> [197] van  saab saab saab opel bus  saab saab van  van  saab bus  van  saab
#> [211] opel bus  saab bus  saab opel opel saab saab saab saab van  saab saab
#> [225] opel van  bus  van  opel opel bus  van  saab van  opel bus  opel opel
#> [239] van  van  van  saab bus  saab bus  saab bus  opel bus  van  opel bus
#> [253] opel bus  saab van  opel saab bus  opel bus  bus  bus  van  bus  van
#> [267] bus  bus  opel saab saab bus  van  bus  saab opel van  opel saab opel
#> [281] bus  saab saab saab saab van  opel van  van  saab opel bus  saab bus
```

```
#> [295] saab bus  saab van  saab van  saab saab van  opel saab saab bus  saab
#> [309] opel bus  saab bus  saab van  van  saab van  bus  saab van  saab saab
#> [323] bus  opel opel opel bus  opel saab bus  saab van  saab opel saab opel
#> [337] opel opel van  bus  bus  bus  saab opel saab van
#> Levels: bus opel saab van
```

```
# Confusion Matrix
pred <- predict(fit, newdata = testset)
# find column whih has the maximum of all rows
pred.class <- colnames(pred)[max.col(pred, ties.method = c("random"))]
cm <- table(testLabels, pred.class,
      dnn = c("Observed Class", "Predicted Class"))
cm
```

```
#>               Predicted Class
#> Observed Class bus opel saab van
#>           bus   86    1    3   4
#>           opel   1   55   20   9
#>           saab   4   55   23   6
#>           van    2    2    5  70
```

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

```
#> [1] 0.6763006
```

```
# pred <- predict(fit, newdata = Vehicle[-train,])
# pred.class <- colnames(pred)[max.col(pred, ties.method = c("random"))]
# table(Vehicle$Class[-train], pred.class,
#       dnn = c("Observed Class", "Predicted Class"))
```

```
error_rate = (1 - sum(pred.class == testset) / nrow(testset))
round(error_rate, 3)
```

```
#> [1] 0.324
```

```
# error_rate = (1 - sum(pred.class == Vehicle$Class[-train])/346)
# round(error_rate,3)
```

Chapter 8

Bike sharing demand

```
#loading the required libraries
library(rpart)
library(rattle)
library(rpart.plot)
library(RColorBrewer)
library(randomForest)
library(corrplot)
library(dplyr)
library(tictoc)
```

Source: https://www.analyticsvidhya.com/blog/2015/06/solution-kaggle-competition-bike-sharing-demand/

8.1 Step 1. Hypothesis Generation

Before exploring the data to understand the relationship between variables, I'd recommend you to focus on hypothesis generation first. Now, this might sound counter-intuitive for solving a data science problem, but if there is one thing I have learnt over years, it is this. Before exploring data, you should spend some time thinking about the business problem, gaining the domain knowledge and may be gaining first hand experience of the problem (only if I could travel to North America!)

How does it help? This practice usually helps you form better features later on, which are not biased by the data available in the dataset. At this stage, you are expected to posses structured thinking i.e. a thinking process which takes into consideration all the possible aspects of a particular problem.

Here are some of the hypothesis which I thought could influence the demand of bikes:

- **Hourly trend**: There must be high demand during office timings. Early morning and late evening can have different trend (cyclist) and low demand during 10:00 pm to 4:00 am.
- **Daily Trend**: Registered users demand more bike on weekdays as compared to weekend or holiday.
- **Rain**: The demand of bikes will be lower on a rainy day as compared to a sunny day. Similarly, higher humidity will cause to lower the demand and vice versa.
- **Temperature**: Would high or low temperature encourage or disencourage bike riding?
- **Pollution**: If the pollution level in a city starts soaring, people may start using Bike (it may be influenced by government / company policies or increased awareness).

- **Time**: Total demand should have higher contribution of registered user as compared to casual because registered user base would increase over time.
- **Traffic**: It can be positively correlated with Bike demand. Higher traffic may force people to use bike as compared to other road transport medium like car, taxi etc

8.2 2. Understanding the Data Set

The dataset shows hourly rental data for two years (2011 and 2012). The training data set is for the **first 19 days of each month**. The test dataset is from **20th day to month's end**. We are required to predict the total count of bikes rented during each hour covered by the test set.

In the training data set, they have separately given bike demand by registered, casual users and sum of both is given as count.

Training data set has 12 variables (see below) and Test has 9 (excluding registered, casual and count).

8.2.0.1 Independent variables

```
datetime:   date and hour in "mm/dd/yyyy hh:mm" format
season:     Four categories-> 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday:    whether the day is a holiday or not (1/0)
workingday: whether the day is neither a weekend nor holiday (1/0)
weather:    Four Categories of weather
            1-> Clear, Few clouds, Partly cloudy, Partly cloudy
            2-> Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
            3-> Light Snow and Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
            4-> Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp:       hourly temperature in Celsius
atemp:      "feels like" temperature in Celsius
humidity:   relative humidity
windspeed:  wind speed
```

8.2.0.2 Dependent variables

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

8.3 3. Importing the dataset and Data Exploration

For this solution, I have used R (R Studio 0.99.442) in Windows Environment.

Below are the steps to import and perform data exploration. If you are new to this concept, you can refer this guide on Data Exploration in R

1. Import Train and Test Data Set

```r
# https://www.kaggle.com/c/bike-sharing-demand/data
train = read.csv(file.path(data_raw_dir, "bike_train.csv"))
test = read.csv(file.path(data_raw_dir, "bike_test.csv"))
```

```
glimpse(train)
```

```
#> Observations: 10,886
#> Variables: 12
#> $ datetime   <fct> 2011-01-01 00:00:00, 2011-01-01 01:00:00, 2011-01-0...
#> $ season     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ holiday    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ workingday <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ weather    <int> 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, ...
#> $ temp       <dbl> 9.84, 9.02, 9.02, 9.84, 9.84, 9.84, 9.02, 8.20, 9.8...
#> $ atemp      <dbl> 14.395, 13.635, 13.635, 14.395, 14.395, 12.880, 13....
#> $ humidity   <int> 81, 80, 80, 75, 75, 75, 80, 86, 75, 76, 76, 81, 77,...
#> $ windspeed  <dbl> 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 6.0032, 0.0...
#> $ casual     <int> 3, 8, 5, 3, 0, 0, 2, 1, 1, 8, 12, 26, 29, 47, 35, 4...
#> $ registered <int> 13, 32, 27, 10, 1, 1, 0, 2, 7, 6, 24, 30, 55, 47, 7...
#> $ count      <int> 16, 40, 32, 13, 1, 1, 2, 3, 8, 14, 36, 56, 84, 94, ...
```

```
glimpse(test)
```

```
#> Observations: 6,493
#> Variables: 9
#> $ datetime   <fct> 2011-01-20 00:00:00, 2011-01-20 01:00:00, 2011-01-2...
#> $ season     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ holiday    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ workingday <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ weather    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, ...
#> $ temp       <dbl> 10.66, 10.66, 10.66, 10.66, 10.66, 9.84, 9.02, 9.02...
#> $ atemp      <dbl> 11.365, 13.635, 13.635, 12.880, 12.880, 11.365, 10....
#> $ humidity   <int> 56, 56, 56, 56, 56, 60, 60, 55, 55, 52, 48, 45, 42,...
#> $ windspeed  <dbl> 26.0027, 0.0000, 0.0000, 11.0014, 11.0014, 15.0013,...
```

2. Combine both Train and Test Data set (to understand the distribution of independent variable together).

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

data = rbind(train,test)
```

3. Variable Type Identification

```
str(data)
```

```
#> 'data.frame':    17379 obs. of  12 variables:
#>  $ datetime  : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 6 7 8 9 10 ...
#>  $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
#>  $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
#>  $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
#>  $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
#>  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
#>  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
#>  $ humidity  : int  81 80 80 75 75 75 80 86 75 76 ...
#>  $ windspeed : num  0 0 0 0 0 ...
#>  $ casual    : num  3 8 5 3 0 0 2 1 1 8 ...
#>  $ registered: num  13 32 27 10 1 1 0 2 7 6 ...
```

```
#>  $ count     : num  16 40 32 13 1 1 2 3 8 14 ...
```

4. Find missing values in the dataset if any

```
table(is.na(data))
```

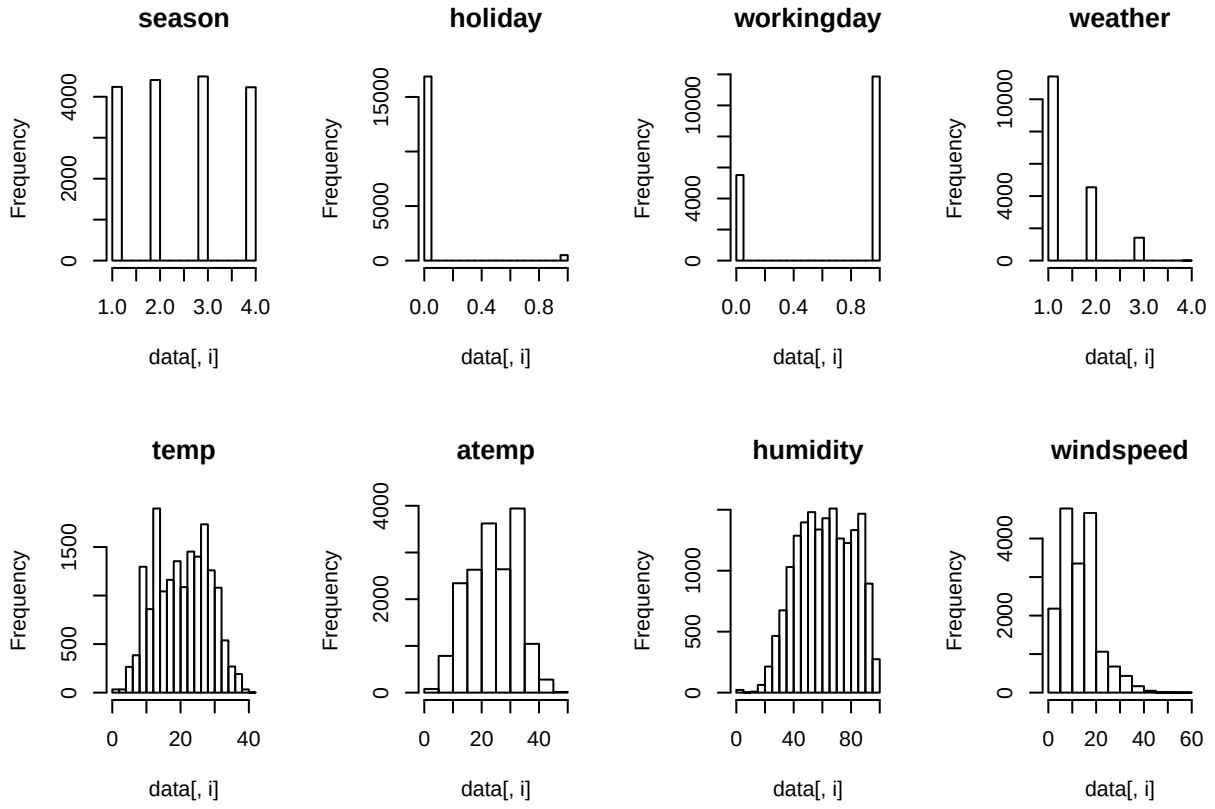
```
#>
#>  FALSE
#> 208548
```

No NAs in the dataset.

5. Understand the distribution of numerical variables and generate a frequency table for numeric variables. Analyze the distribution.

8.3.0.1 histograms

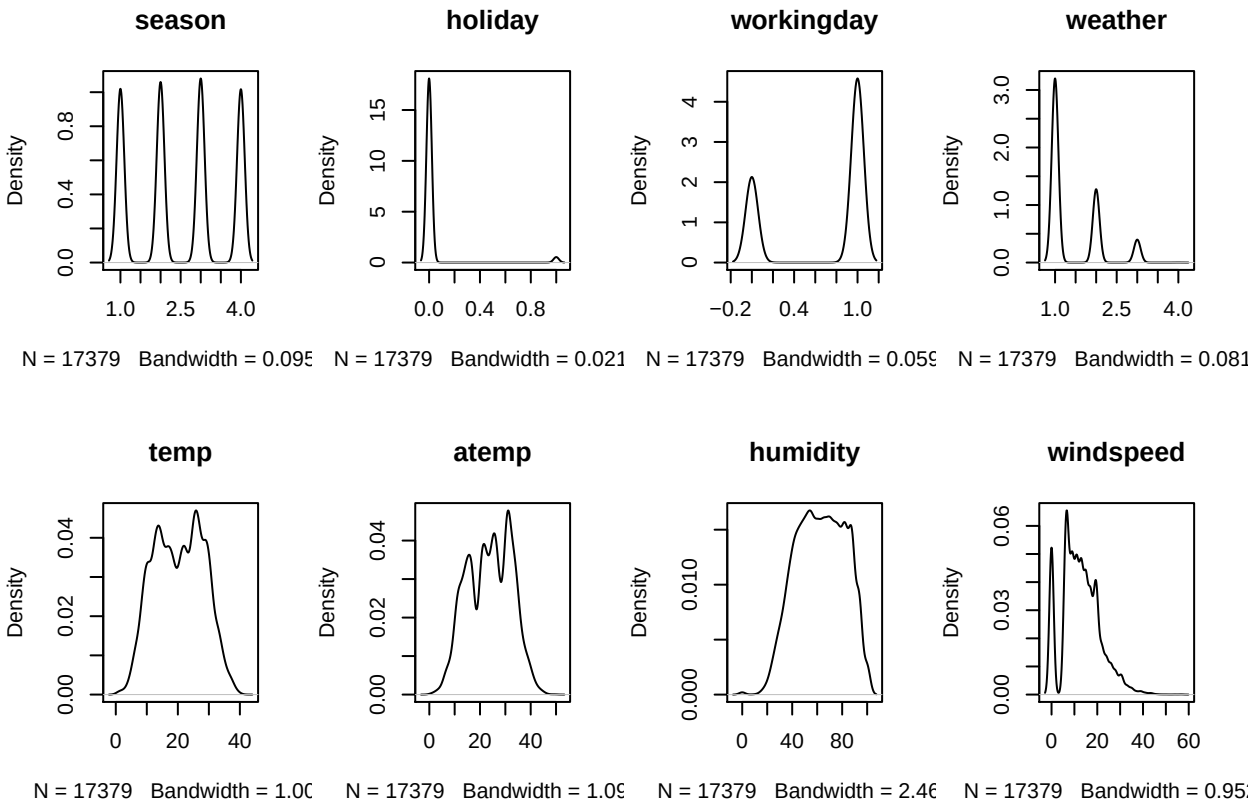
```
# histograms each attribute
par(mfrow=c(2,4))
for(i in 2:9) {
    hist(data[,i], main = names(data)[i])
}
```



8.3.0.2 density plots

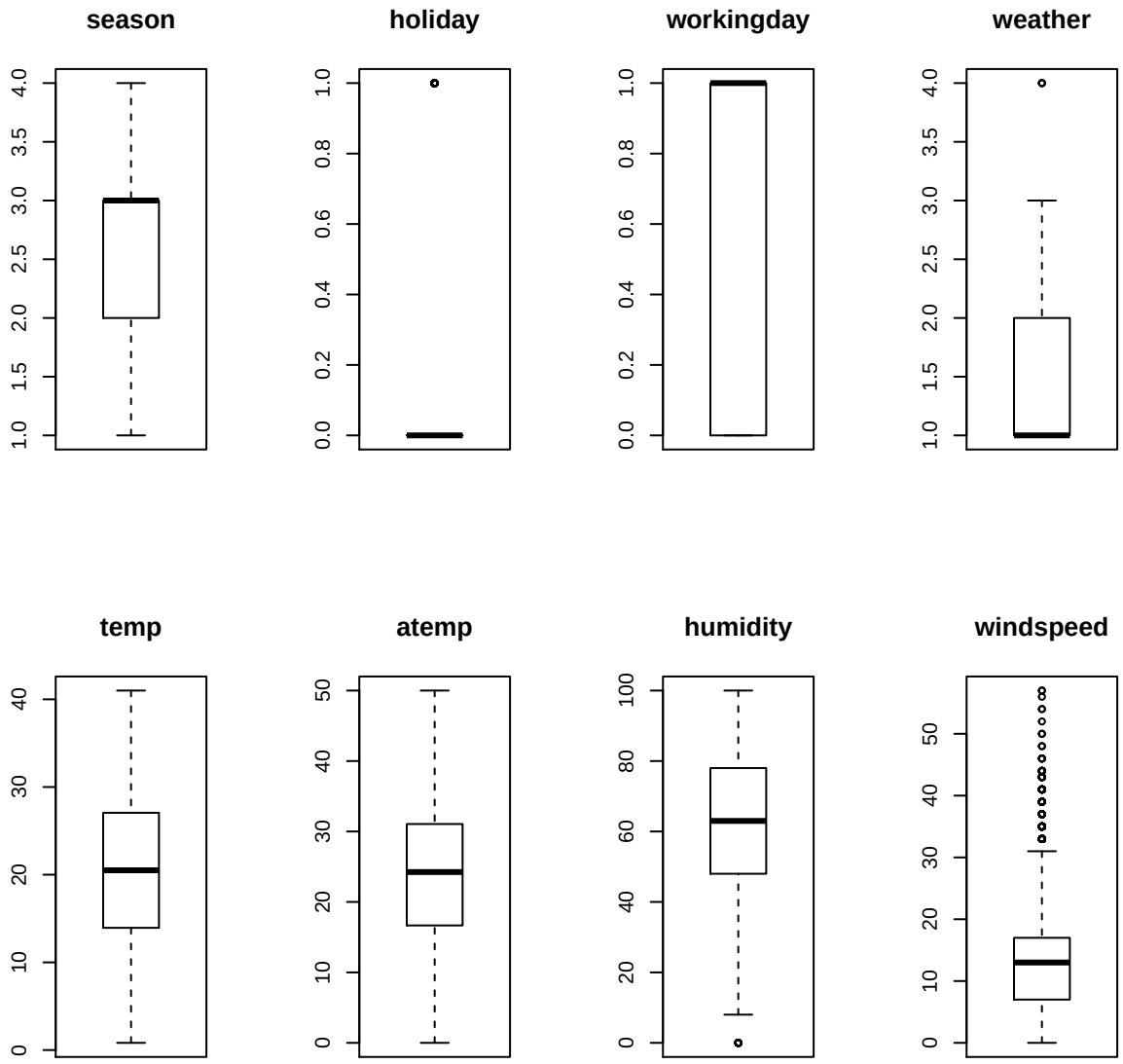
```
# density plot for each attribute
par(mfrow=c(2,4))
for(i in 2:9) {
```

```
    plot(density(data[,i]), main=names(data)[i])
}
```



8.3.0.3 boxplots

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



8.3.1 Unique values of discrete variables

```
# the discrete variables in this case are integers
ints <- unlist(lapply(data, is.integer))
names(data)[ints]
```

```
#> [1] "season"     "holiday"    "workingday" "weather"    "humidity"
```

Humidity should not be an integer or discrete variable; it is a continuous or numeric variable.

```
# convert humidity to numeric
data$humidity <- as.numeric(data$humidity)
```

```
# list unique values of integer variables
ints <- unlist(lapply(data, is.integer))
int_vars <- names(data)[ints]

sapply(int_vars, function(x) unique(data[x]))
```

```
#> $season.season
```

```
#> [1] 1 2 3 4
#>
#> $holiday.holiday
#> [1] 0 1
#>
#> $workingday.workingday
#> [1] 0 1
#>
#> $weather.weather
#> [1] 1 2 3 4
```

8.3.1.1 Inferences

1. The variables `season`, `holiday`, `workingday` and `weather` are discrete (integer).
2. Activity is even through all seasons.
3. Most of the activitity happens during non-holidays.
4. Activity doubles during the working days.
5. Activity happens mostly during clear (1) weather.
6. temp, atemp and humidity are continuous variables (numeric).

8.4 4. Hypothesis Testing (using multivariate analysis)

Till now, we have got a fair understanding of the data set. Now, let's test the hypothesis which we had generated earlier. Here I have added some additional hypothesis from the dataset. Let's test them one by one:

8.4.1 Hourly trend

> *There must be high demand during office timings. Early morning and late evening can have different trend (cyclist) and low demand during 10:00 pm to 4:00 am.*

We don't have the variable 'hour' with us. But we can extract it using the datetime column.

```
head(data$datetime)
```

```
#> [1] 2011-01-01 00:00:00 2011-01-01 01:00:00 2011-01-01 02:00:00
#> [4] 2011-01-01 03:00:00 2011-01-01 04:00:00 2011-01-01 05:00:00
#> 17379 Levels: 2011-01-01 00:00:00 2011-01-01 01:00:00 ... 2012-12-31 23:00:00
```

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

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

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

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

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

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

```
# extracting hour
data$hour = substr(data$datetime,12,13)
```

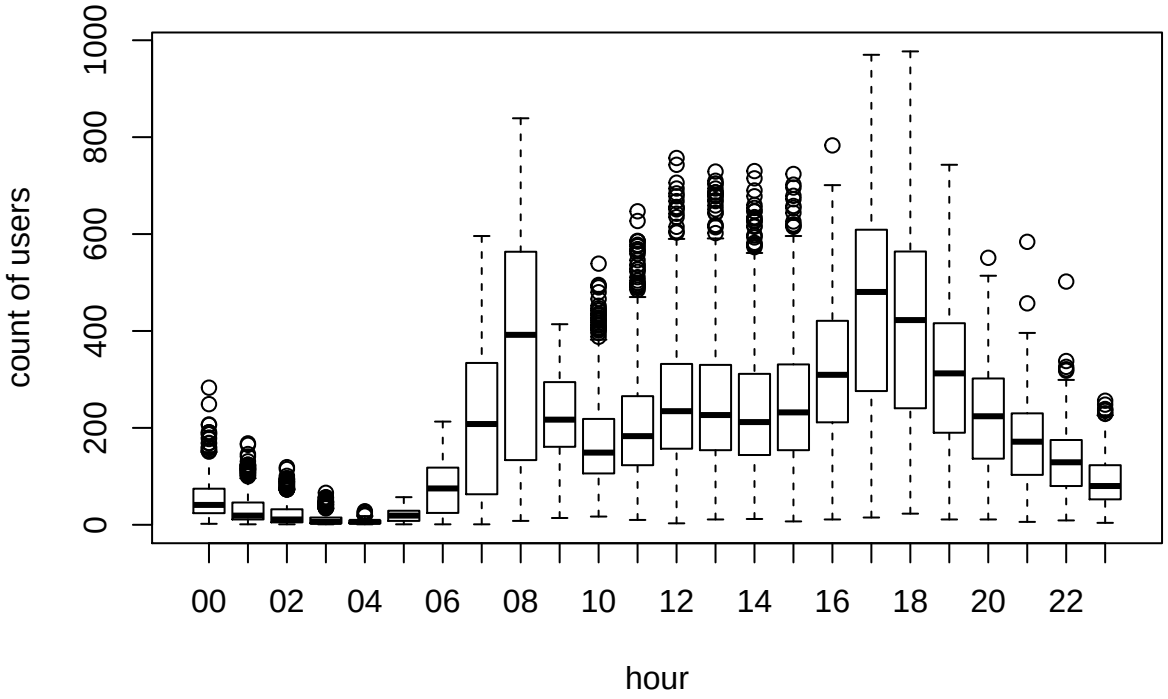
```
data$hour = as.factor(data$hour)
head(data$hour)
```

```
#> [1] 00 01 02 03 04 05
#> 24 Levels: 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 ... 23
```

```
### dividing again in train and test
# the train dataset is for the first 19 days
train = data[as.integer(substr(data$datetime, 9, 10)) < 20,]

# the test dataset is from day 20 to the end of the month
test = data[as.integer(substr(data$datetime, 9, 10)) > 19,]
```

8.4.1.1 boxplot count vs hour in training set

```
boxplot(train$count ~ train$hour, xlab="hour", ylab="count of users")
```



Rides increase from 6 am to 6pm, during office hours.

```
# casual users
casual <- data[data$casual > 0, ]
registered <- data[data$registered > 0, ]

dim(casual)
```

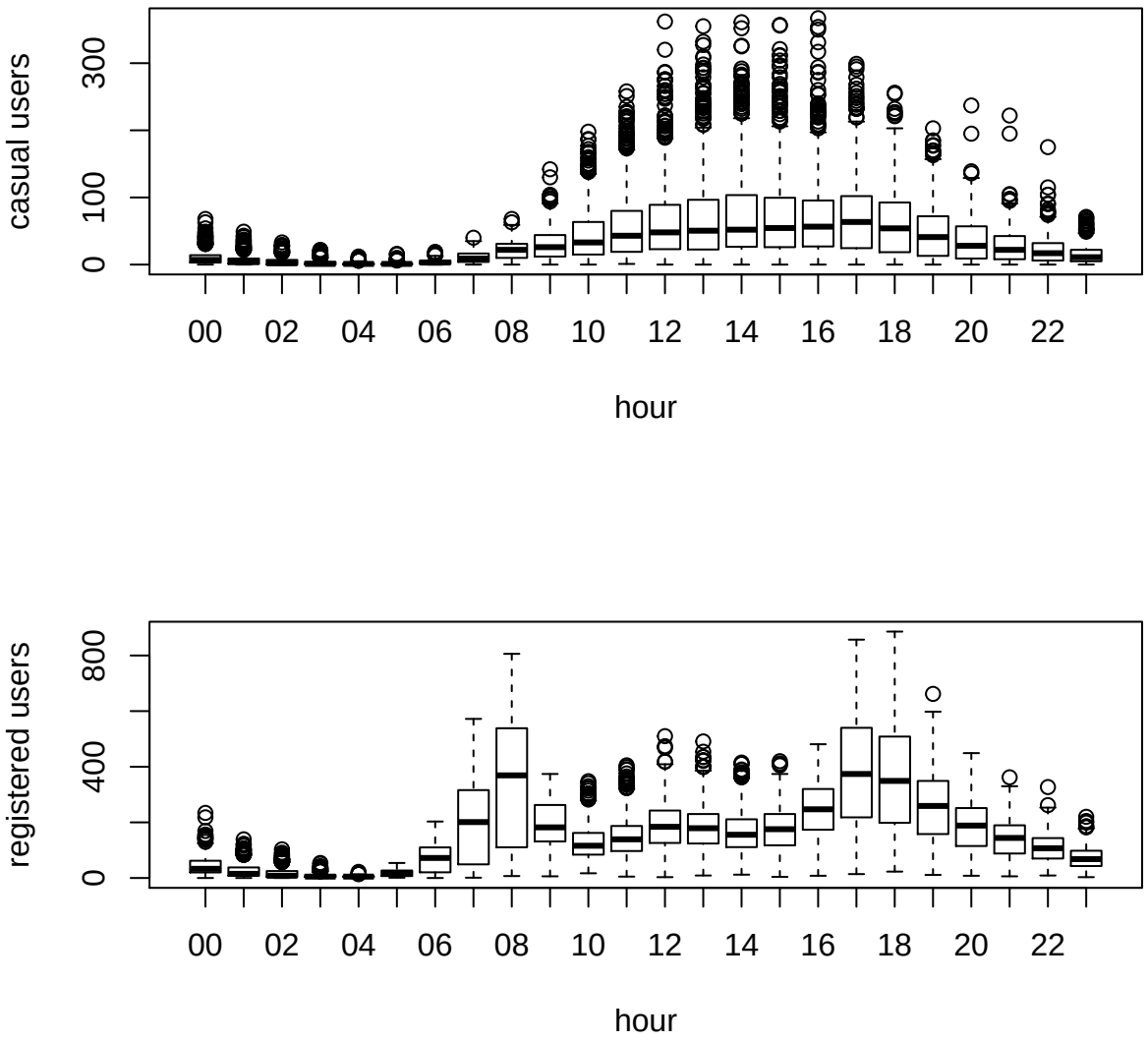
```
#> [1] 9900   13
```

```
dim(registered)
```

```
#> [1] 10871    13
```

8.4.1.2 Boxplot hourly: casual vs registered users in the training set

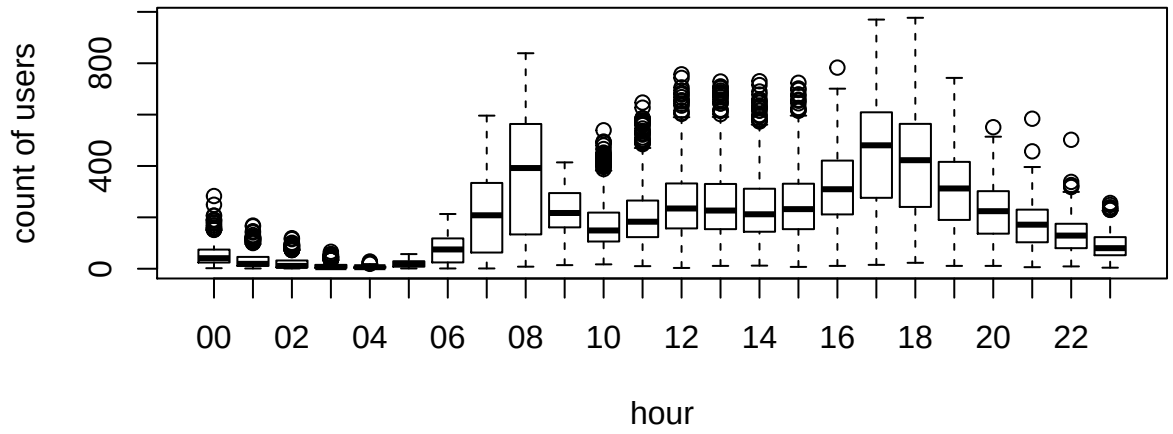
```
# by hour: casual vs registered users
par(mfrow=c(2,1))
boxplot(train$casual ~ train$hour, xlab="hour", ylab="casual users")
boxplot(train$registered ~ train$hour, xlab="hour", ylab="registered users")
```

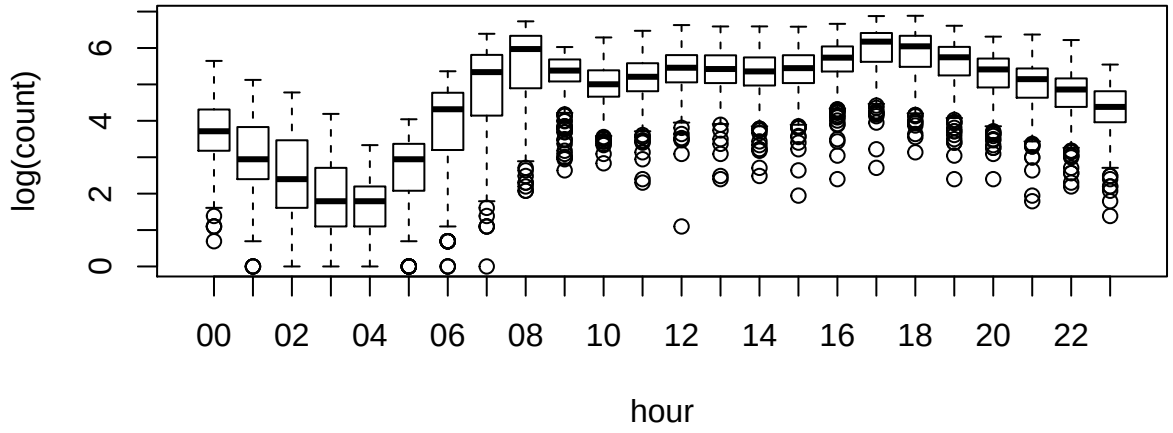


Casual and Registered users have different distributions. Casual users tend to rent more during office hours.

8.4.1.3 outliers in the training set

```
par(mfrow=c(2,1))
boxplot(train$count ~ train$hour, xlab="hour", ylab="count of users")
boxplot(log(train$count) ~ train$hour,xlab="hour",ylab="log(count)")
```





8.4.1.4 Daily trend

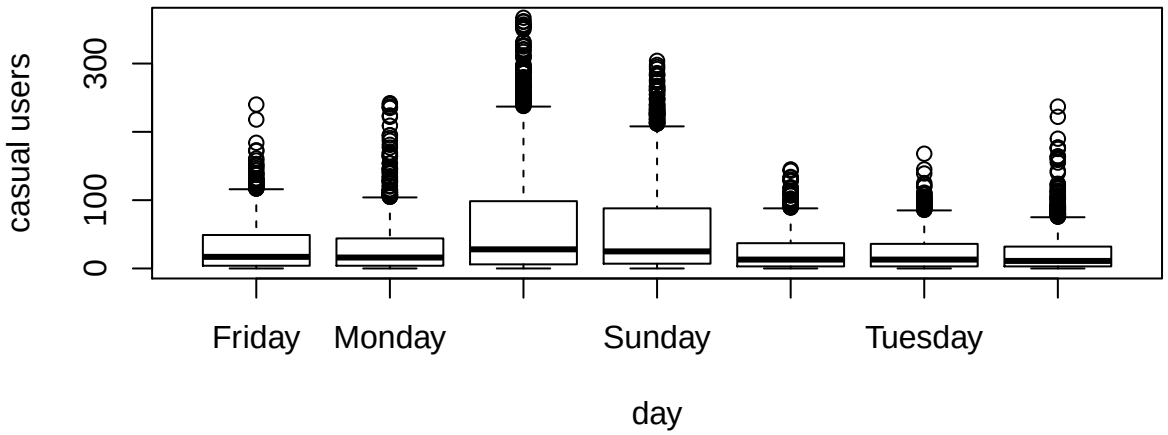
Registered users demand more bike on weekdays as compared to weekend or holiday.

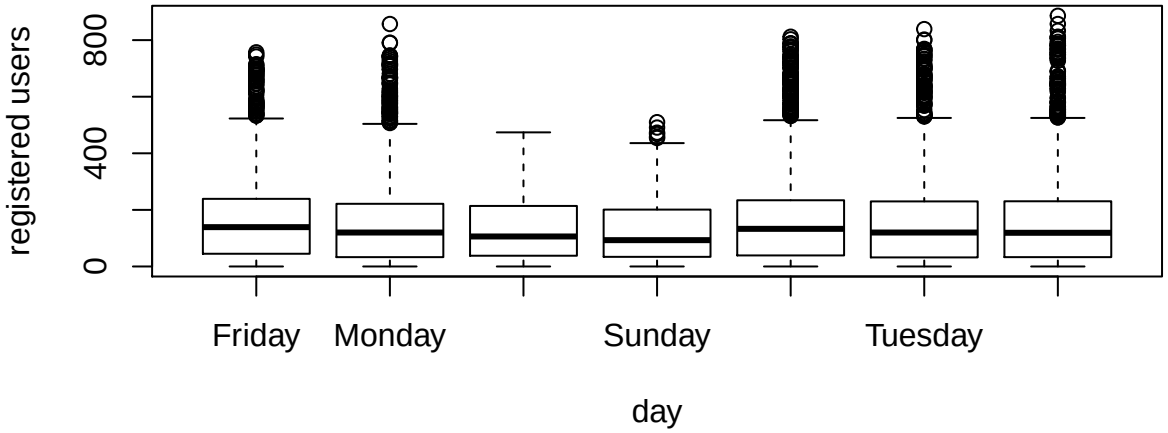
```
# extracting days of week
date <- substr(data$datetime, 1, 10)
days <- weekdays(as.Date(date))
data$day <- days
```

```
# split the dataset again at day 20 of the month, before and after
train = data[as.integer(substr(data$datetime,9,10)) < 20,]
test  = data[as.integer(substr(data$datetime,9,10)) > 19,]
```

8.4.1.5 Boxplot daily trend: casual vs registered users, training set

```
# creating boxplots for rentals with different variables to see the variation
par(mfrow=c(2,1))
boxplot(train$casual ~ train$day, xlab="day", ylab="casual users")
boxplot(train$registered ~ train$day, xlab="day", ylab="registered users")
```





Demand of casual users increases during the weekend, contrary of registered users.

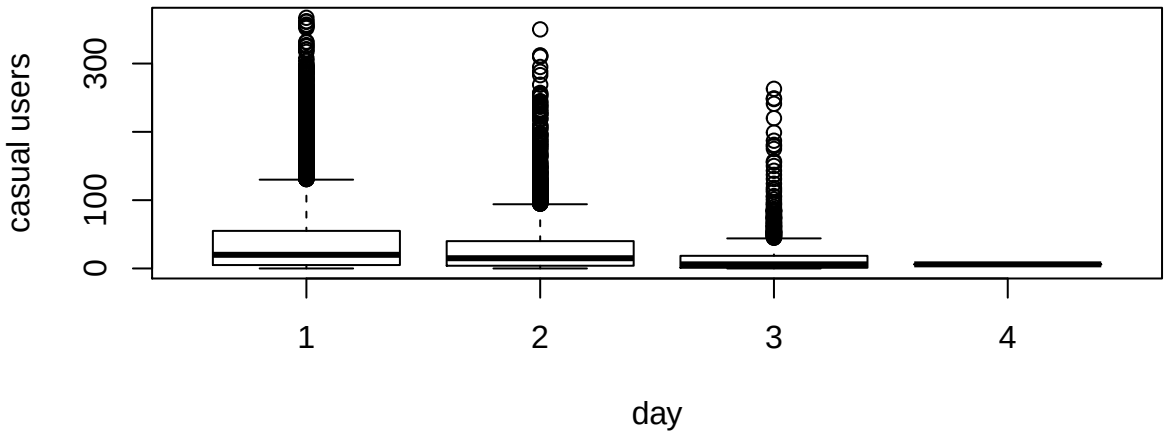
8.4.2 Rain

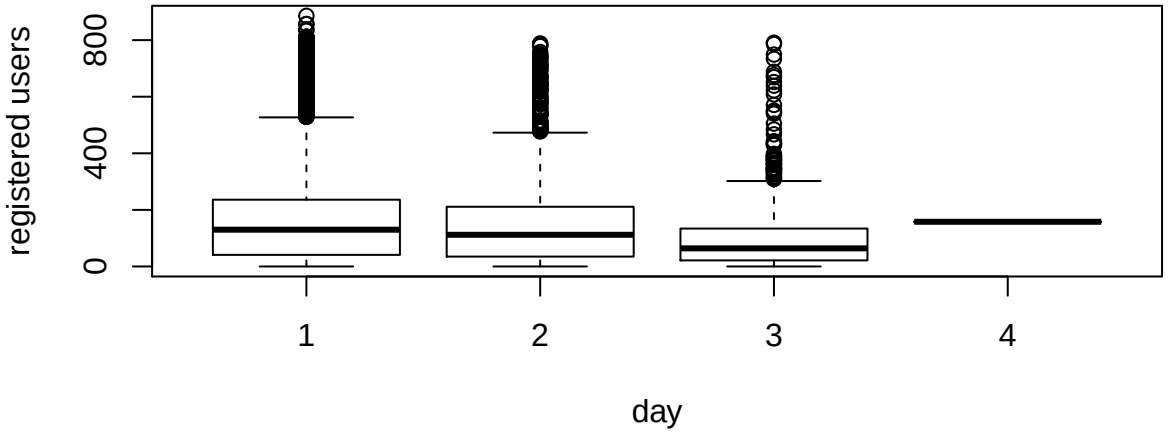
The demand of bikes will be lower on a rainy day as compared to a sunny day. Similarly, higher humidity will cause to lower the demand and vice versa.

We use the variable weather (1 to 4) to analyze riding under rain conditions.

8.4.2.1 Boxplot of rain effect on bike riding, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$weather, xlab="day", ylab="casual users")
boxplot(train$registered ~ train$weather, xlab="day", ylab="registered users")
```





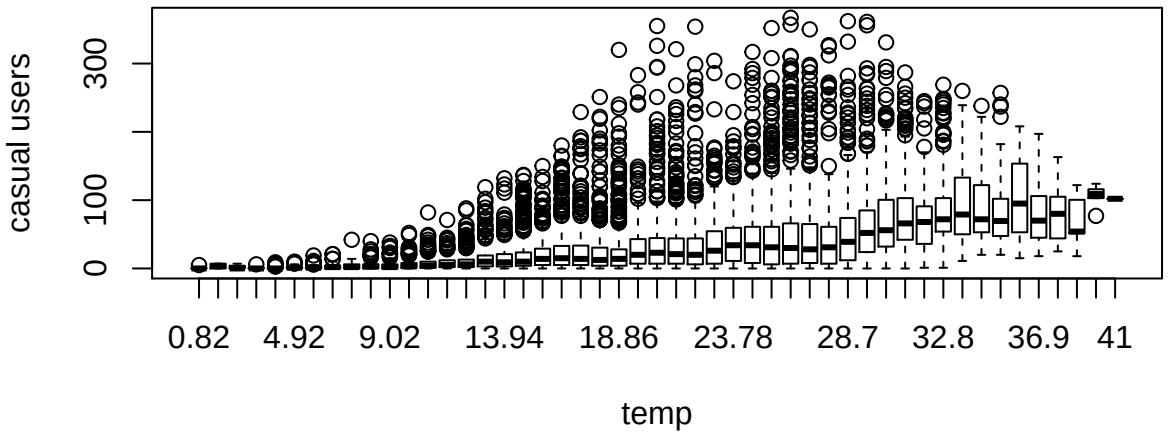
Registered used tend to ride even with rain.

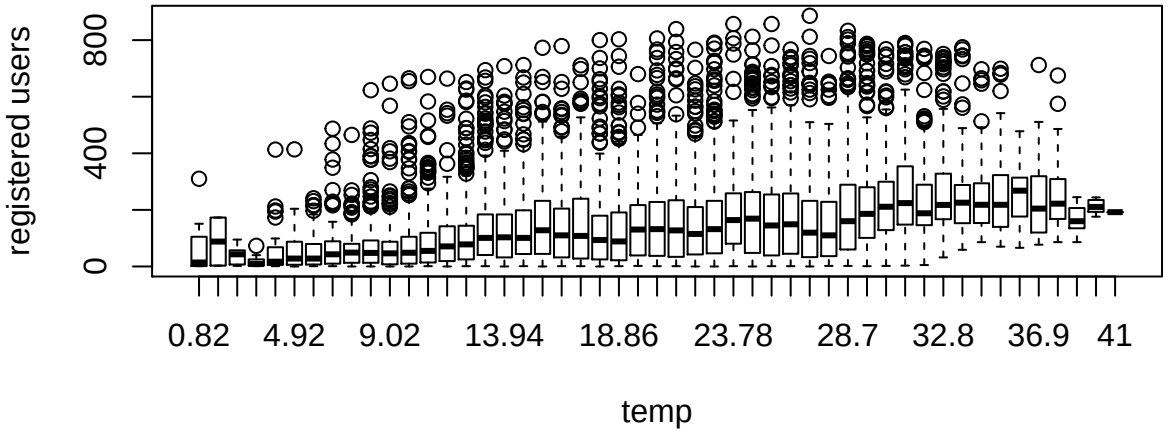
8.4.3 Temperature

Would high or low temperature encourage or disencourage bike riding?

8.4.3.1 boxplot of temperature effect, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$temp, xlab="temp", ylab="casual users")
boxplot(train$registered ~ train$temp, xlab="temp", ylab="registered users")
```





Casual users tend to ride with milder temperatures while registered users ride even at low temperatures.

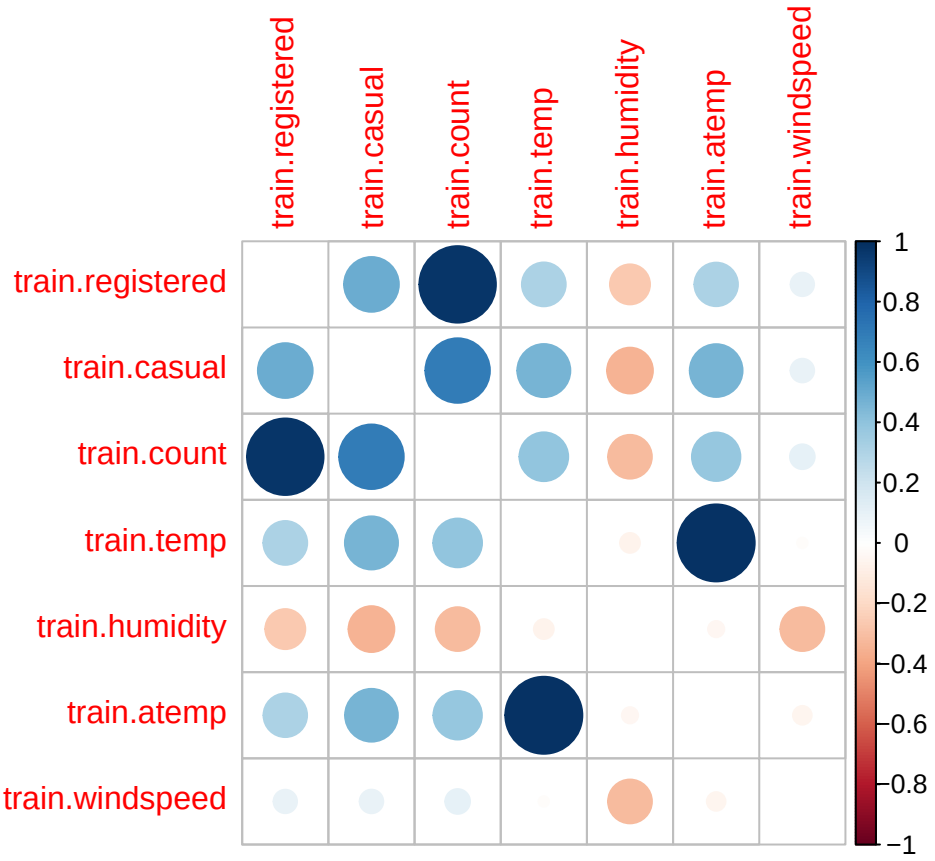
8.4.4 Correlation

```
sub = data.frame(train$registered, train$casual, train$count, train$temp,
                 train$humidity, train$atemp, train$windspeed)
cor(sub)
```

```
#>                   train.registered train.casual train.count  train.temp
#> train.registered        1.00000000   0.49724969   0.9709481  0.31857128
#> train.casual            0.49724969   1.00000000   0.6904136  0.46709706
#> train.count             0.97094811   0.69041357   1.0000000  0.39445364
#> train.temp              0.31857128   0.46709706   0.3944536  1.00000000
#> train.humidity         -0.26545787  -0.34818690  -0.3173715 -0.06494877
#> train.atemp             0.31463539   0.46206654   0.3897844  0.98494811
#> train.windspeed         0.09105166   0.09227619   0.1013695 -0.01785201
#>                   train.humidity train.atemp train.windspeed
#> train.registered    -0.26545787  0.31463539      0.09105166
```

```
#> train.casual        -0.34818690  0.46206654      0.09227619
#> train.count         -0.31737148  0.38978444      0.10136947
#> train.temp          -0.06494877  0.98494811     -0.01785201
#> train.humidity       1.00000000 -0.04353571     -0.31860699
#> train.atemp         -0.04353571  1.00000000     -0.05747300
#> train.windspeed     -0.31860699 -0.05747300      1.00000000
```

```
# do not show the diagonal
corrplot(cor(sub), diag = FALSE)
```



1. correlation between `casual` and atemp, temp.
2. Strong correlation between temp and atemp.

8.4.5 Activity by year

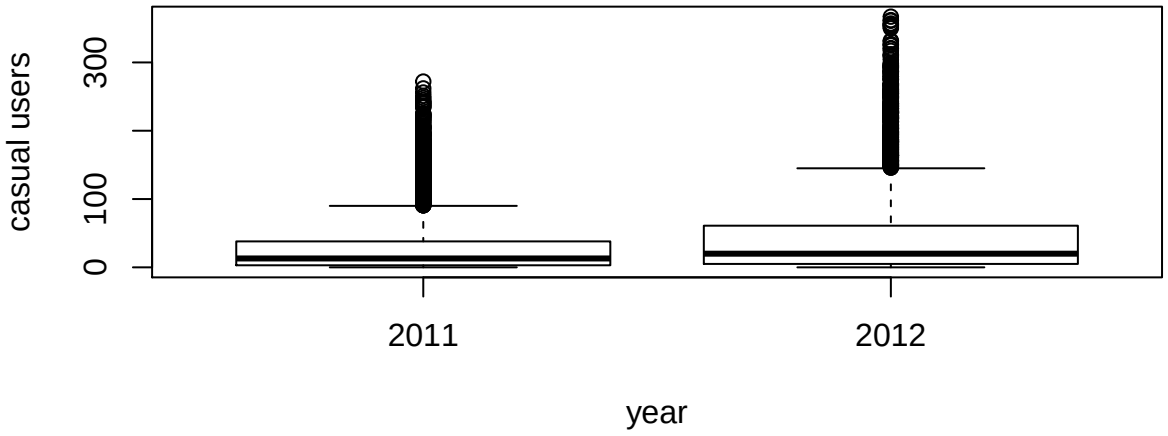
8.4.5.1 Year extraction

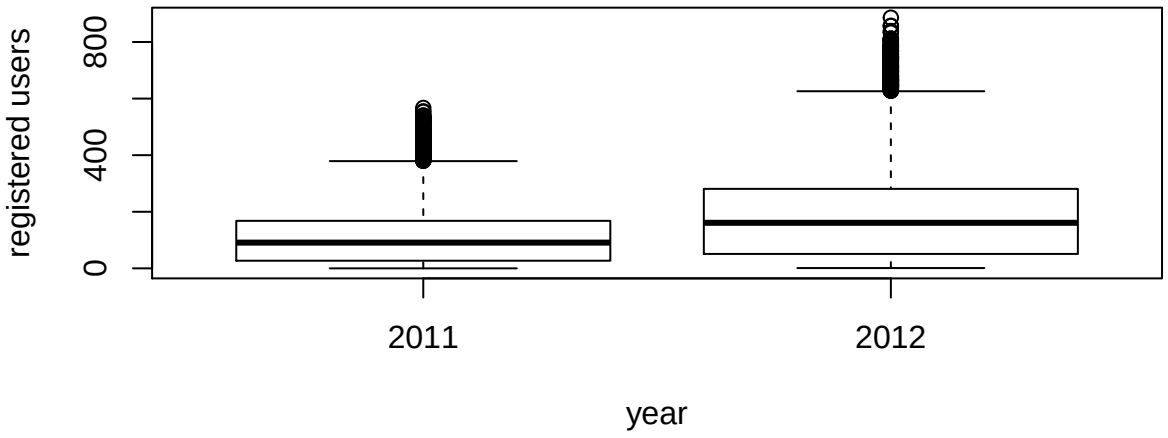
```
# extracting year
data$year = substr(data$datetime, 1, 4)
data$year = as.factor(data$year)
```

```
# ignore the division of data again and again, this could have been done together also
train = data[as.integer(substr(data$datetime,9,10)) < 20,]
test = data[as.integer(substr(data$datetime,9,10)) > 19,]
```

8.4.5.2 Trend by year, training set

```
par(mfrow=c(2,1))
# again some boxplots with different variables
# these boxplots give important information about the dependent variable with respect to the independen
boxplot(train$casual ~ train$year, xlab="year", ylab="casual users")
boxplot(train$registered ~ train$year, xlab="year", ylab="registered users")
```

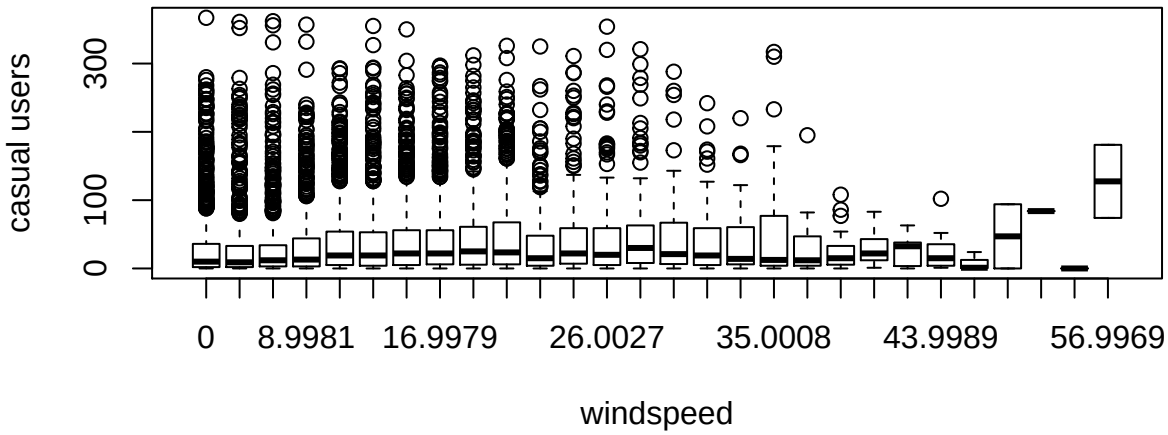


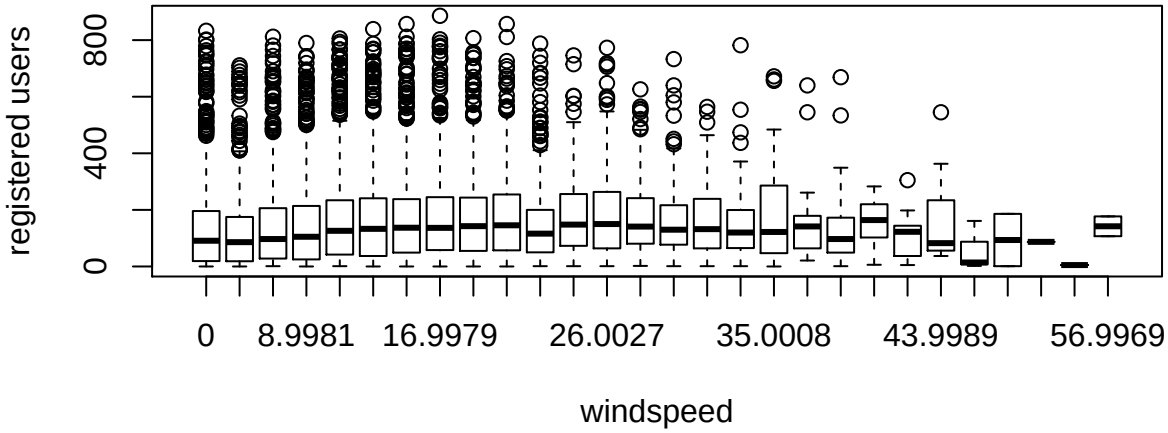


Activity increased in 2012.

8.4.5.3 trend by windspeed, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$windspeed, xlab="windspeed", ylab="casual users")
boxplot(train$registered ~ train$windspeed, xlab="windspeed", ylab="registered users")
```

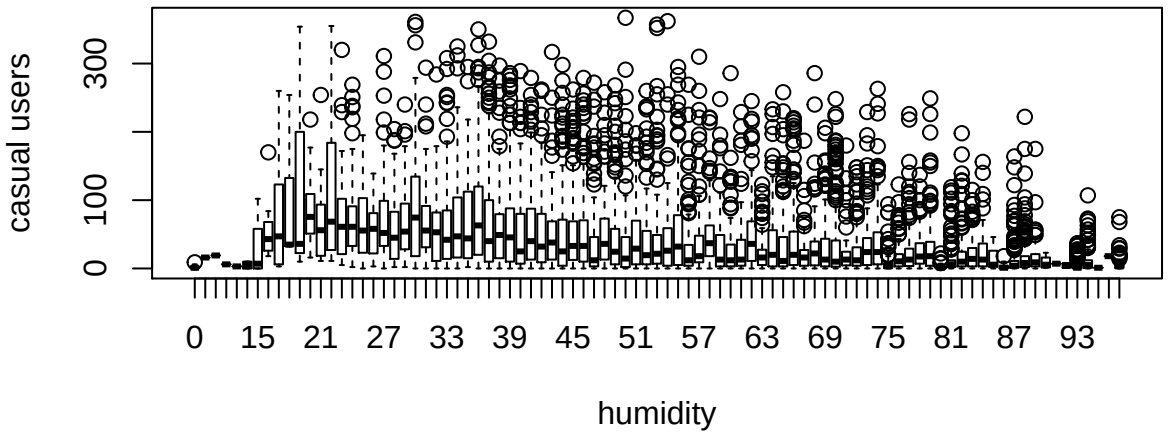


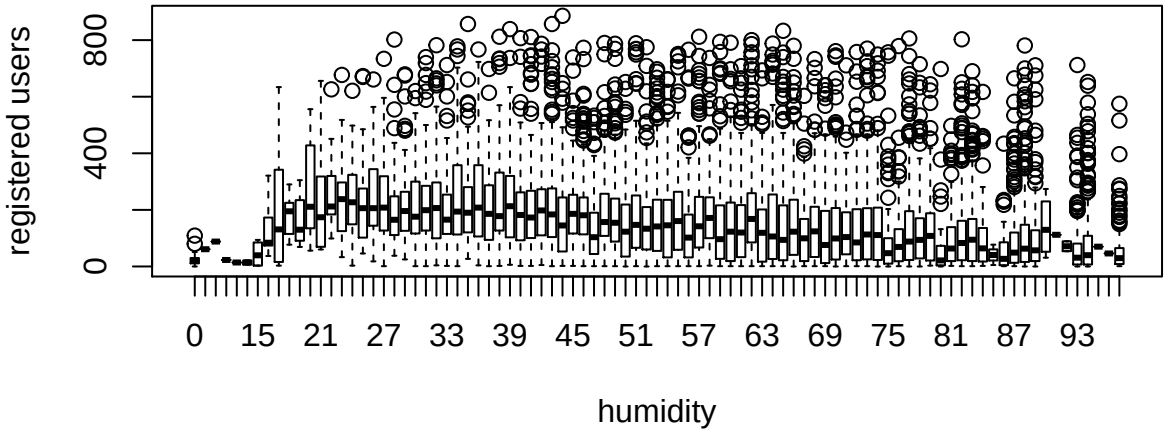


Casual users ride even with stron winds.

8.4.5.4 trend by humidity, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$humidity, xlab="humidity", ylab="casual users")
boxplot(train$registered ~ train$humidity, xlab="humidity", ylab="registered users")
```





Casual users prefer not to ride with humid weather.

8.5 5. Feature Engineering

```
# factoring some variables from integer
data$season     <- as.factor(data$season)
data$weather    <- as.factor(data$weather)
data$holiday    <- as.factor(data$holiday)
data$workingday <- as.factor(data$workingday)
```

```
# new column
data$hour <- as.integer(data$hour)
```

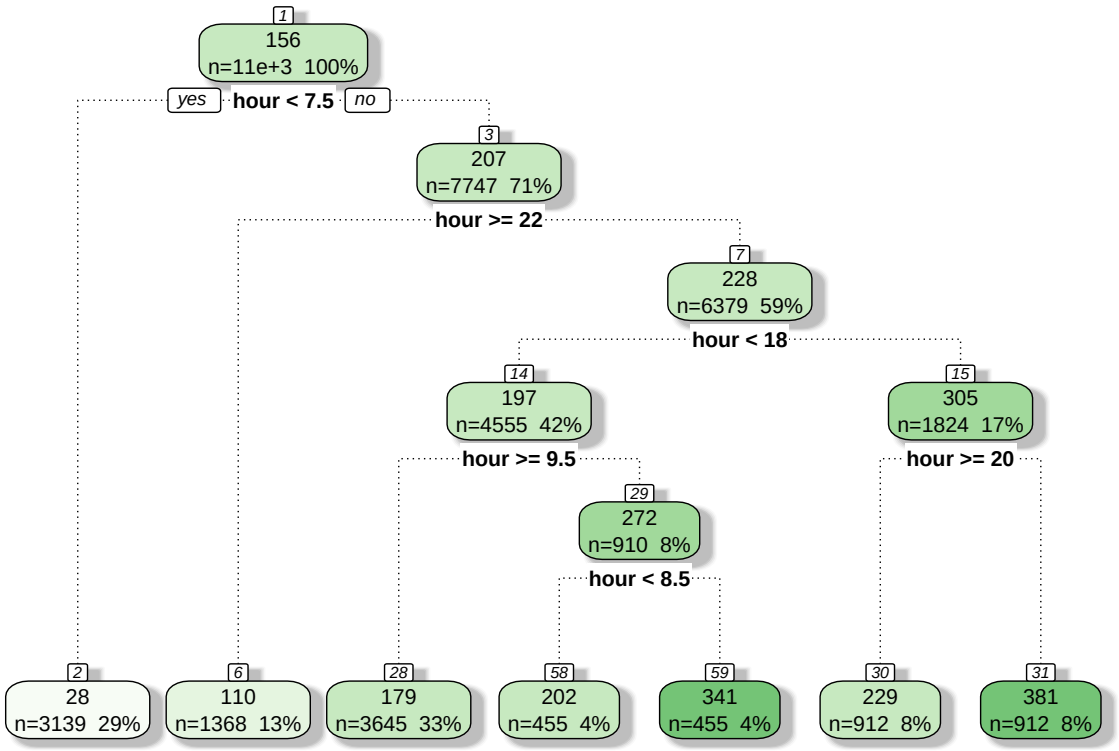
```
# created this variable to divide a day into parts, but did not finally use it
data$day_part <- 0
```

```
# split in training and test sets again
train <- data[as.integer(substr(data$datetime, 9, 10)) < 20,]
test  <- data[as.integer(substr(data$datetime, 9, 10)) > 19,]
```

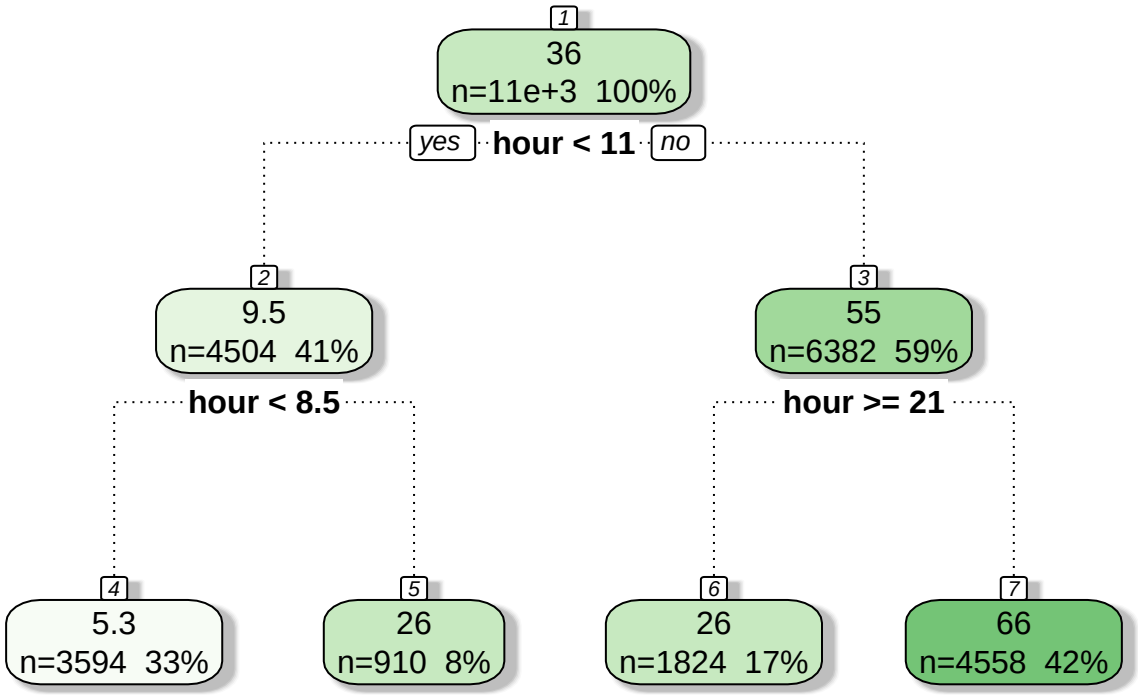
```
# combine the sets
data <- rbind(train, test)
```

8.5.1 Build hour bins

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




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



Rattle 2019−May−15 08:24:06 datascience

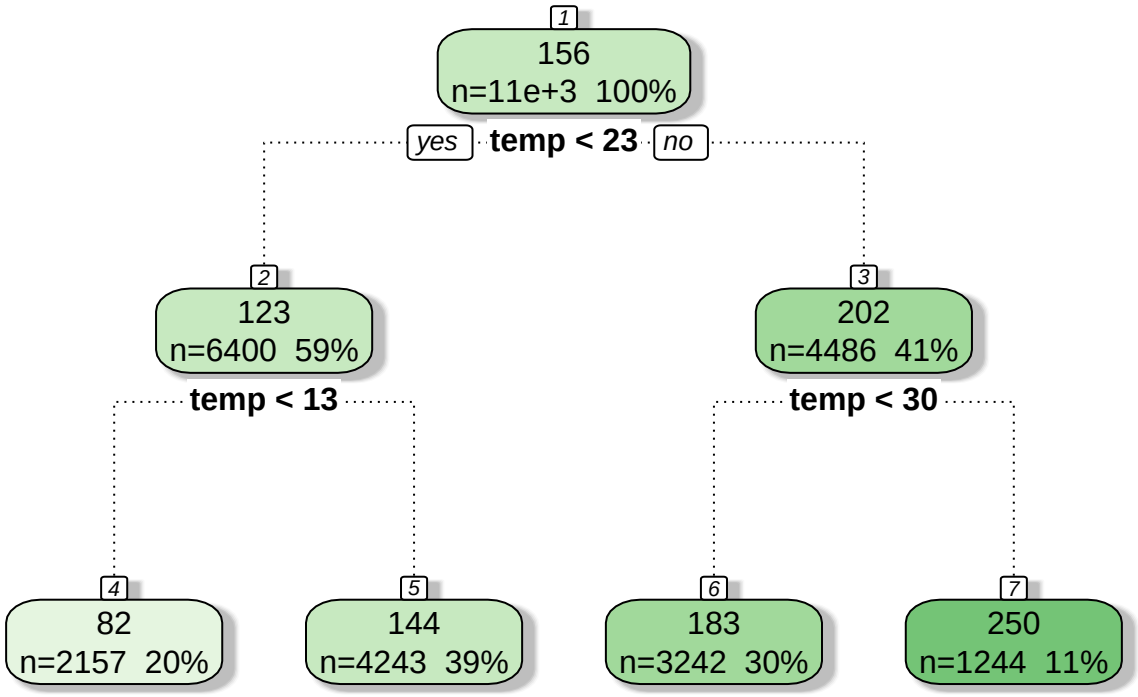
```
# Assign the timings according to tree
# fill the hour bins
data = rbind(train,test)

# create hour buckets for registered users
# 0,1,2,3,4,5,6,7 < 7.5
# 22,23,24 >=22
# 10,11,12,13,14,15,16,17: h>=9.5 & h<18
# h<9.5 & h<8.5 : 8
# h<9.5 & h>=8.5 : 9
# h>=20: 20,21
# h < 20: 18,19

data$dp_reg = 0
data$dp_reg[data$hour < 8] = 1
data$dp_reg[data$hour >= 22] = 2
data$dp_reg[data$hour > 9 & data$hour < 18] = 3
data$dp_reg[data$hour == 8] = 4
data$dp_reg[data$hour == 9] = 5
data$dp_reg[data$hour == 20 | data$hour == 21] = 6
data$dp_reg[data$hour == 19 | data$hour == 18] = 7
```

```
# casual users
# h<11, h<8.5: 0,1,2,3,4,5,6,7,8
# h>=8.5 & h<11: 9, 10
# h >=11 & h>=21: 21,22,23,24
# h >=11 & h<21: 11,12,13,14,15,16,17,18,19,20
data$dp_cas = 0
data$dp_cas[data$hour < 11 & data$hour >= 8] = 1
```

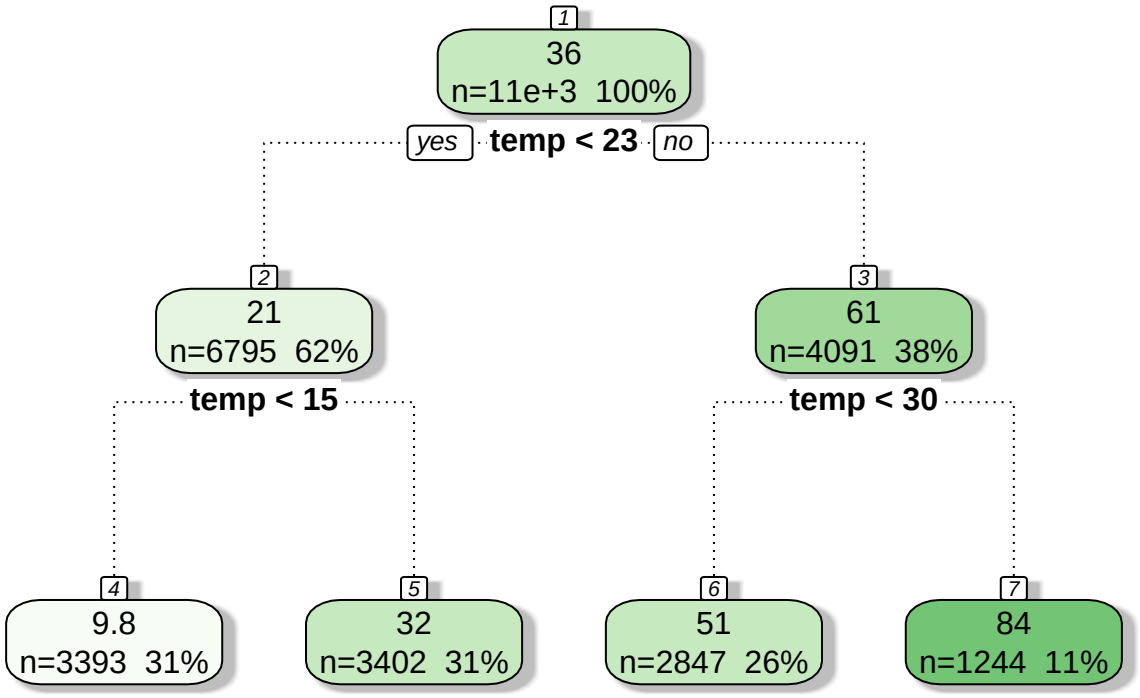
```
data$dp_cas[data$hour == 9 | data$hour == 10] = 2
data$dp_cas[data$hour >= 11 & data$hour < 21] = 3
data$dp_cas[data$hour >= 21] = 4
```

8.5.2 Temperature bins

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




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



8.5.2.1 Assign temperature ranges accoding to trees

```
data$temp_reg = 0
data$temp_reg[data$temp < 13] = 1
data$temp_reg[data$temp >= 13 & data$temp < 23] = 2
data$temp_reg[data$temp >= 23 & data$temp < 30] = 3
data$temp_reg[data$temp >= 30] = 4
```

```
data$temp_cas = 0
data$temp_cas[data$temp < 15] = 1
data$temp_cas[data$temp >= 15 & data$temp < 23] = 2
data$temp_cas[data$temp >= 23 & data$temp < 30] = 3
data$temp_cas[data$temp >= 30] = 4
```

8.5.3 Year bins by quarter

```
# add new variable with the month number
data$month <- substr(data$datetime, 6, 7)
data$month <- as.integer(data$month)
```

```
# bin by quarter manually
data$year_part[data$year=='2011']                = 1
data$year_part[data$year=='2011' & data$month>3] = 2
data$year_part[data$year=='2011' & data$month>6] = 3
data$year_part[data$year=='2011' & data$month>9] = 4
data$year_part[data$year=='2012']                = 5
```

```
data$year_part[data$year=='2012' & data$month>3] = 6
data$year_part[data$year=='2012' & data$month>6] = 7
data$year_part[data$year=='2012' & data$month>9] = 8
table(data$year_part)
```

```
#>
#>    1    2    3    4    5    6    7    8
#> 2067 2183 2192 2203 2176 2182 2208 2168
```

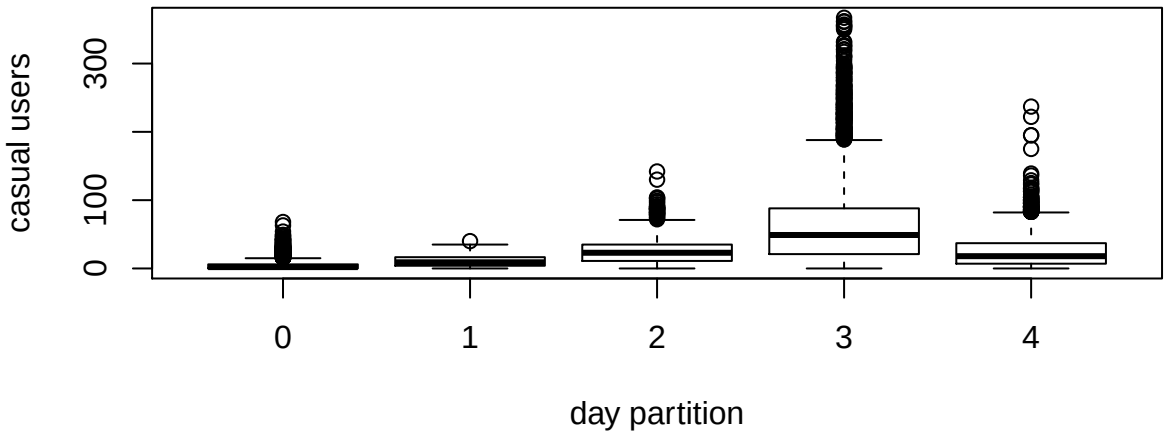
8.5.4 Day Type

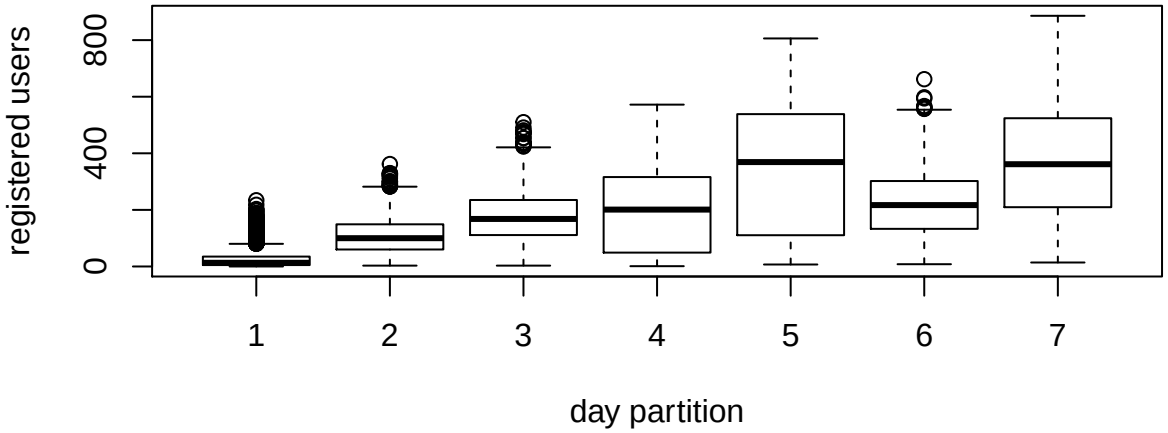
Created a variable having categories like “weekday”, “weekend” and “holiday”.

```
# creating another variable day_type which may affect our accuracy as weekends and weekdays are importa
data$day_type = 0
data$day_type[data$holiday==0 & data$workingday==0] = "weekend"
data$day_type[data$holiday==1]                      = "holiday"
data$day_type[data$holiday==0 & data$workingday==1] = "working day"
```

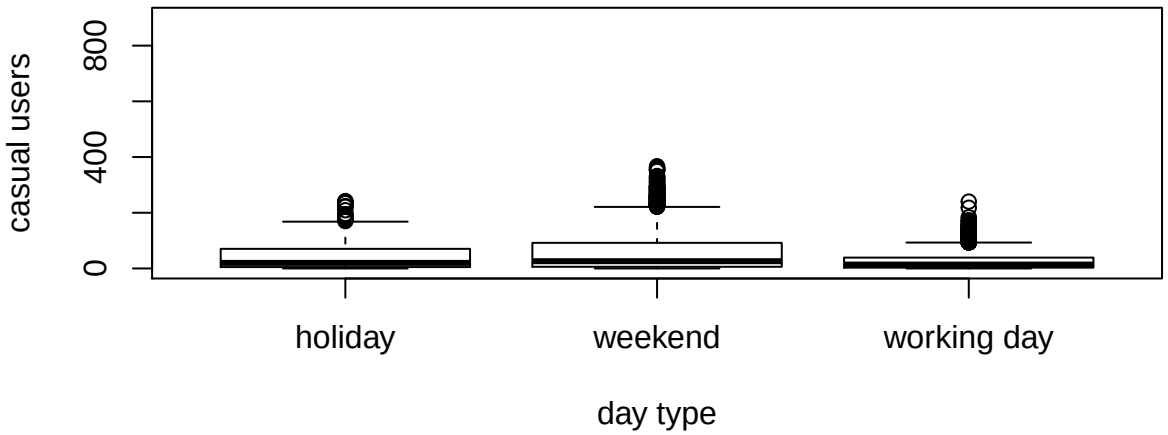
```
# split dataset again
train = data[as.integer(substr(data$datetime,9,10)) < 20,]
test = data[as.integer(substr(data$datetime,9,10)) > 19,]
```

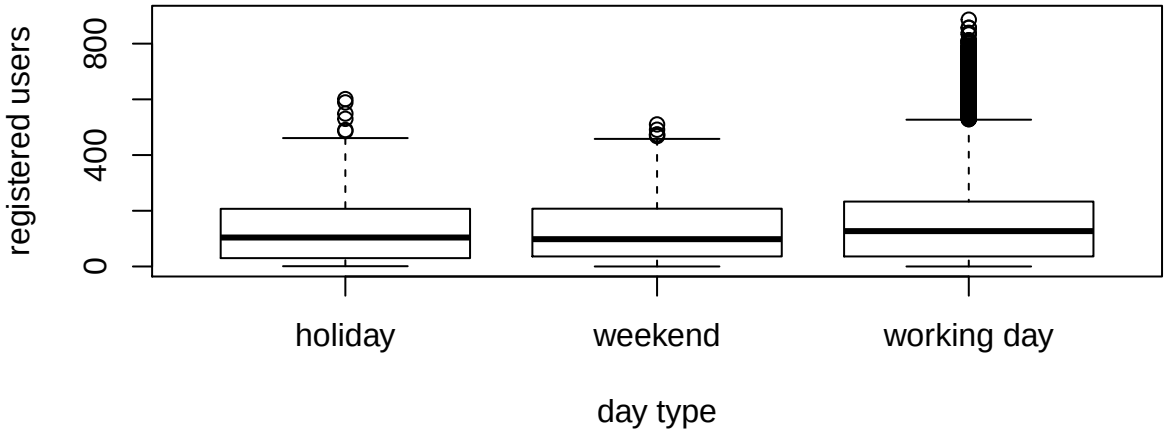
```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$dp_cas, xlab = "day partition", ylab="casual users")
boxplot(train$registered ~ train$dp_reg, xlab = "day partition", ylab="registered users")
```



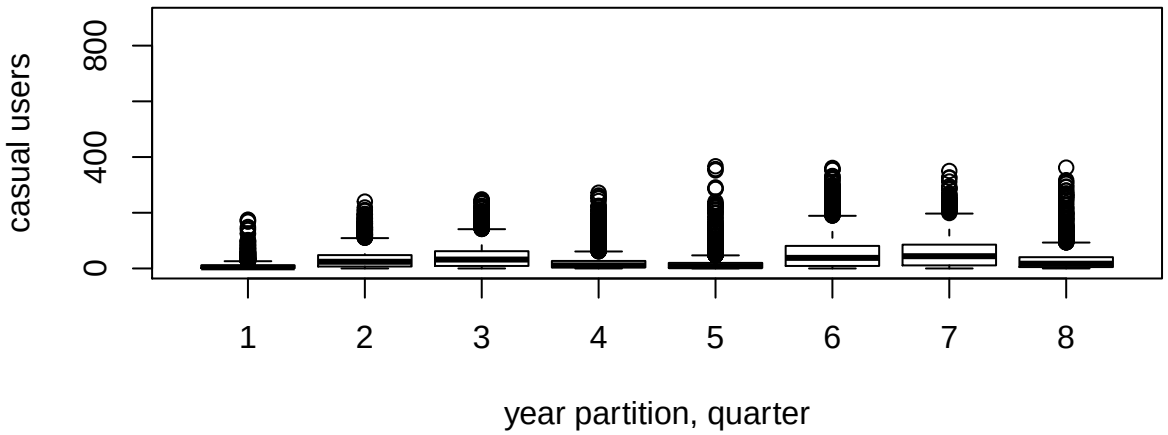


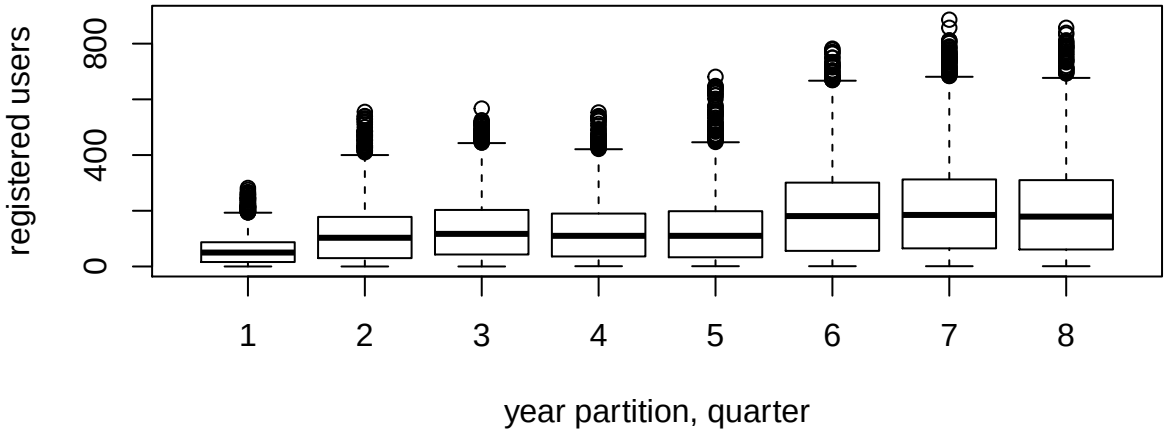

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$day_type, xlab = "day type",
        ylab="casual users", ylim = c(0,900))
boxplot(train$registered ~ train$day_type, xlab = "day type",
        ylab="registered users", ylim = c(0,900))
```





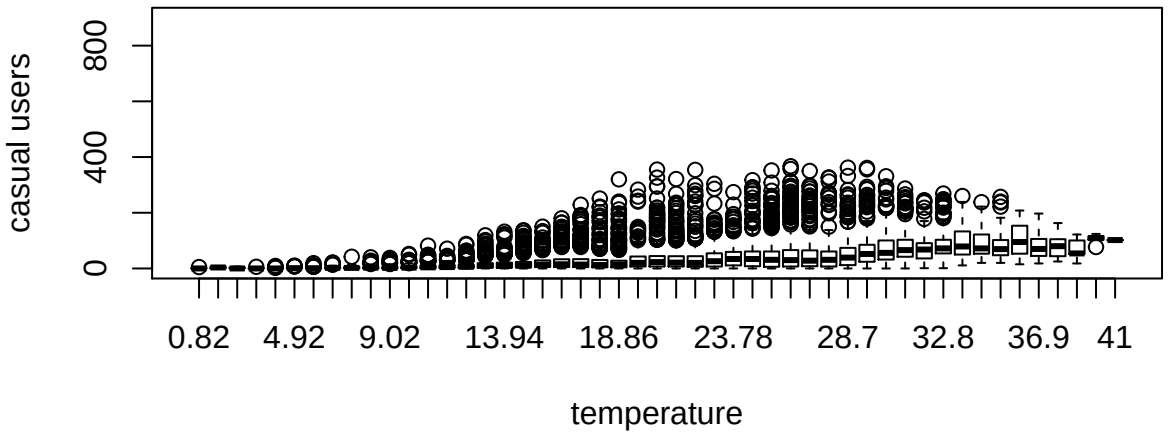

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$year_part, xlab = "year partition, quarter",
        ylab="casual users", ylim = c(0,900))
boxplot(train$registered ~ train$year_part, xlab = "year partition, quarter",
        ylab="registered users", ylim = c(0,900))
```

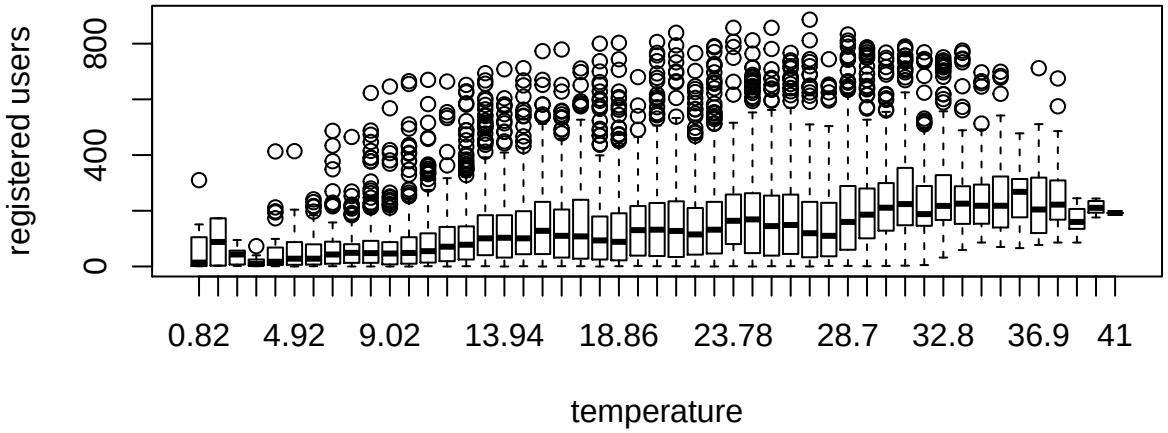




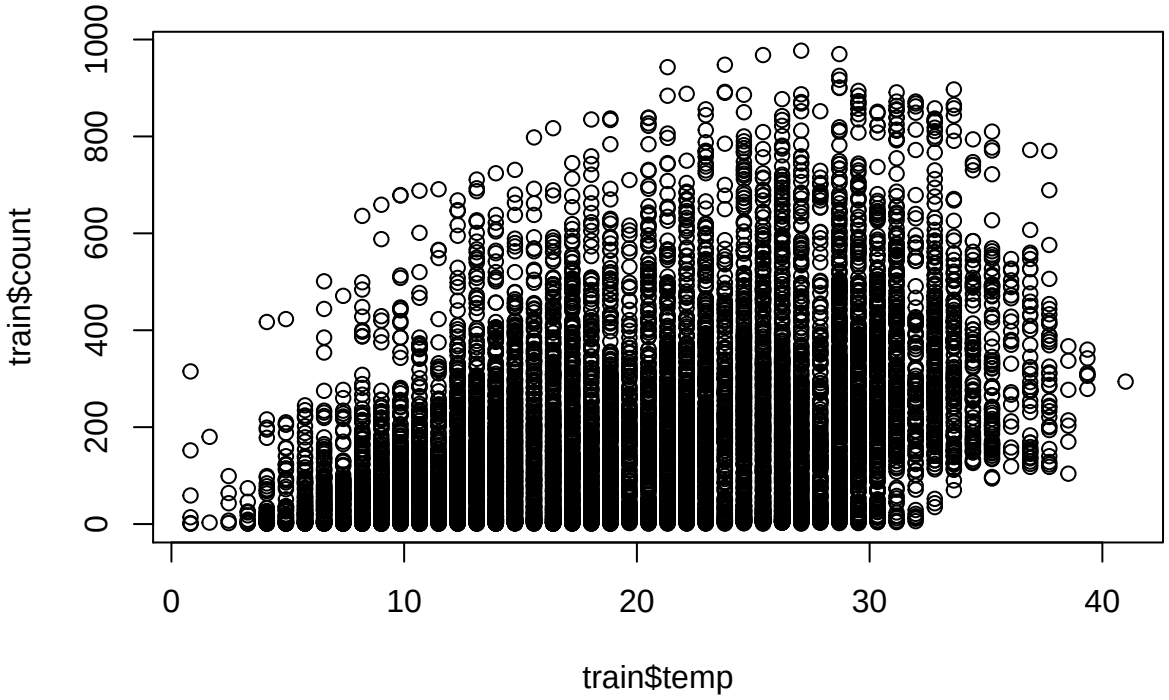
8.5.5 Temperatures

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$temp, xlab = "temperature",
        ylab="casual users", ylim = c(0,900))
boxplot(train$registered ~ train$temp, xlab = "temperature",
        ylab="registered users", ylim = c(0,900))
```






```
plot(train$temp, train$count)
```




```
data <- rbind(train, test)
# data$month <- substr(data$datetime, 6, 7)
# data$month <- as.integer(data$month)
```

8.5.6 Imputting missing data to wind speed

```
# dividing total data depending on windspeed to impute/predict the missing values
table(data$windspeed == 0)
```

```
#>
#> FALSE  TRUE
#> 15199  2180
```

```
    # FALSE  TRUE
    # 15199  2180

k = data$windspeed == 0

wind_0 = subset(data, k)     # windspeed is zero
wind_1 = subset(data, !k)    # windspeed not zero
```

```
tic()
# predicting missing values in windspeed using a random forest model
# this is a different approach to impute missing values rather than
# just using the mean or median or some other statistic for imputation

set.seed(415)
fit <- randomForest(windspeed ~ season + weather + humidity + month + temp +
                        year + atemp,
                    data = wind_1,
                    importance = TRUE,
```

```
                    ntree = 250)

pred = predict(fit, wind_0)
wind_0$windspeed = pred       # fill with wind speed predictions
toc()
```

```
#> 61.279 sec elapsed
```

```
# recompose the whole dataset
data = rbind(wind_0, wind_1)
```

```
# how many zero values now?
sum(data$windspeed == 0)
```

```
#> [1] 0
```

8.5.7 Weekend variable

Created a separate variable for weekend (0/1)

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

8.6 6. Model Building

As this was our first attempt, we applied decision tree, conditional inference tree and random forest algorithms and found that random forest is performing the best. You can also go with regression, boosted regression, neural network and find which one is working well for you.

Before executing the random forest model code, I have followed following steps:

Convert discrete variables into factor (weather, season, hour, holiday, working day, month, day)

```
str(data)
```

```
#> 'data.frame':    17379 obs. of  24 variables:
#>  $ datetime  : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 7 8 9 10 65 ...
#>  $ season    : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ holiday   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ workingday: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
#>  $ weather   : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
#>  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
#>  $ humidity  : num  81 80 80 75 75 80 86 75 76 47 ...
#>  $ windspeed : num  9.03 9.05 9.05 9.15 9.15 ...
#>  $ casual    : num  3 8 5 3 0 2 1 1 8 8 ...
#>  $ registered: num  13 32 27 10 1 0 2 7 6 102 ...
#>  $ count     : num  16 40 32 13 1 2 3 8 14 110 ...
#>  $ hour      : int  1 2 3 4 5 7 8 9 10 20 ...
#>  $ day       : chr  "Saturday" "Saturday" "Saturday" "Saturday" ...
#>  $ year      : Factor w/ 2 levels "2011","2012": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_part  : num  0 0 0 0 0 0 0 0 0 0 ...
#>  $ dp_reg    : num  1 1 1 1 1 1 4 5 3 6 ...
#>  $ dp_cas    : num  0 0 0 0 0 0 1 2 2 3 ...
#>  $ temp_reg  : num  1 1 1 1 1 1 1 1 2 1 ...
```

```
#>  $ temp_cas  : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ month     : int  1 1 1 1 1 1 1 1 1 1 ...
#>  $ year_part : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_type  : chr  "weekend" "weekend" "weekend" "weekend" ...
#>  $ weekend   : num  1 1 1 1 1 1 1 1 1 0 ...
```

8.6.1 Convert to factors

```
# converting all relevant categorical variables into factors to feed to our random forest model
data$season     = as.factor(data$season)
data$holiday    = as.factor(data$holiday)
data$workingday = as.factor(data$workingday)
data$weather    = as.factor(data$weather)
data$hour       = as.factor(data$hour)
data$month      = as.factor(data$month)
data$day_part   = as.factor(data$dp_cas)
data$day_type   = as.factor(data$dp_reg)
data$day        = as.factor(data$day)
data$temp_cas   = as.factor(data$temp_cas)
data$temp_reg   = as.factor(data$temp_reg)
```

```
str(data)
```

```
#> 'data.frame':    17379 obs. of  24 variables:
#>  $ datetime  : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 7 8 9 10 65 ...
#>  $ season    : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ holiday   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ workingday: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
#>  $ weather   : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
#>  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
#>  $ humidity  : num  81 80 80 75 75 80 86 75 76 47 ...
#>  $ windspeed : num  9.03 9.05 9.05 9.15 9.15 ...
#>  $ casual    : num  3 8 5 3 0 2 1 1 8 8 ...
#>  $ registered: num  13 32 27 10 1 0 2 7 6 102 ...
#>  $ count     : num  16 40 32 13 1 2 3 8 14 110 ...
#>  $ hour      : Factor w/ 24 levels "1","2","3","4",..: 1 2 3 4 5 7 8 9 10 20 ...
#>  $ day       : Factor w/ 7 levels "Friday","Monday",..: 3 3 3 3 3 3 3 3 3 2 ...
#>  $ year      : Factor w/ 2 levels "2011","2012": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_part  : Factor w/ 5 levels "0","1","2","3",..: 1 1 1 1 1 1 2 3 3 4 ...
#>  $ dp_reg    : num  1 1 1 1 1 1 4 5 3 6 ...
#>  $ dp_cas    : num  0 0 0 0 0 0 1 2 2 3 ...
#>  $ temp_reg  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 2 1 ...
#>  $ temp_cas  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ month     : Factor w/ 12 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
#>  $ year_part : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_type  : Factor w/ 7 levels "1","2","3","4",..: 1 1 1 1 1 1 4 5 3 6 ...
#>  $ weekend   : num  1 1 1 1 1 1 1 1 1 0 ...
```

- As we know that dependent variables have natural outliers so we will predict log of dependent variables.
- Predict bike demand registered and casual users separately. $y1 = \log(casual + 1)$ and $y2 = \log(registered + 1)$, Here we have added 1 to deal with zero values in the casual and registered columns.

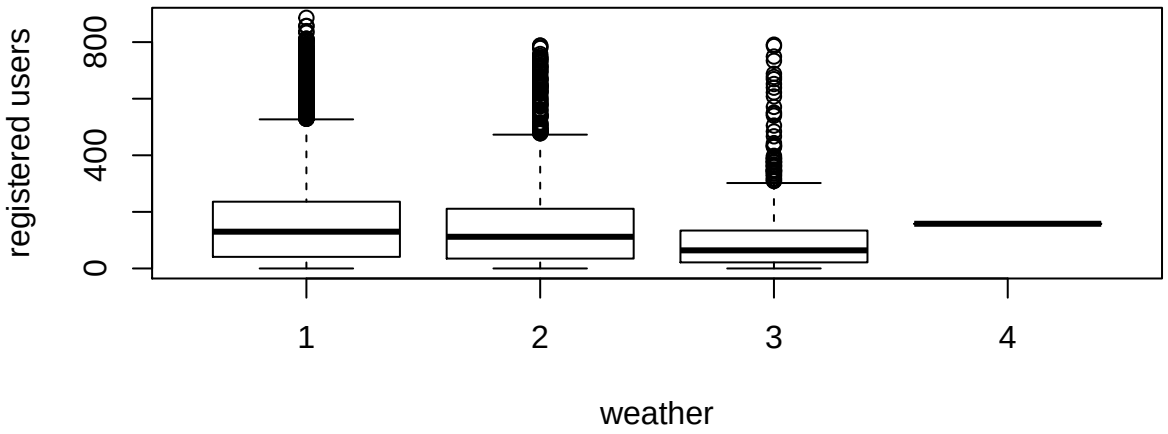
```
# separate again as train and test set
train = data[as.integer(substr(data$datetime, 9, 10)) < 20,]
test = data[as.integer(substr(data$datetime, 9, 10)) > 19,]
```

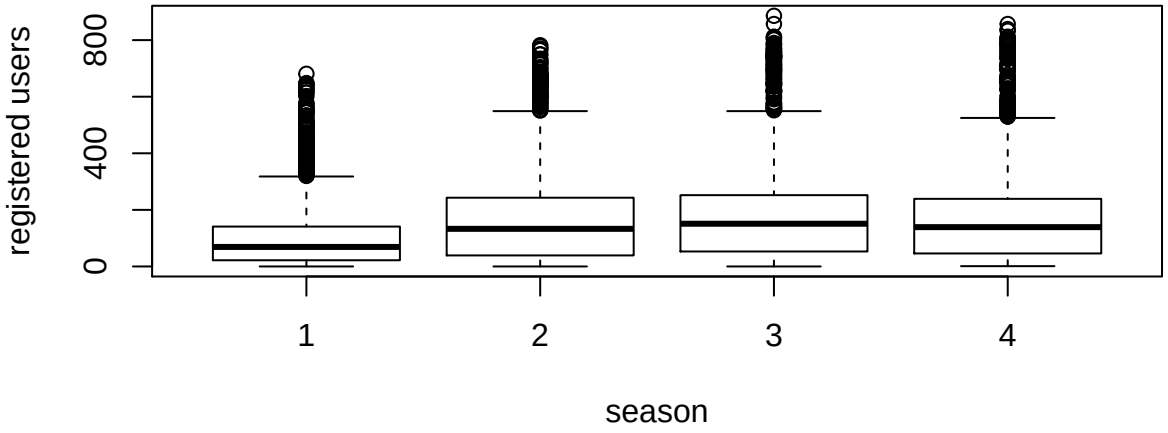
8.6.2 Log transform

```
# log transformation for some skewed variables,
# which can be seen from their distribution
train$reg1   = train$registered + 1
train$cas1   = train$casual + 1
train$logcas = log(train$cas1)
train$logreg = log(train$reg1)
test$logreg  = 0
test$logcas  = 0
```

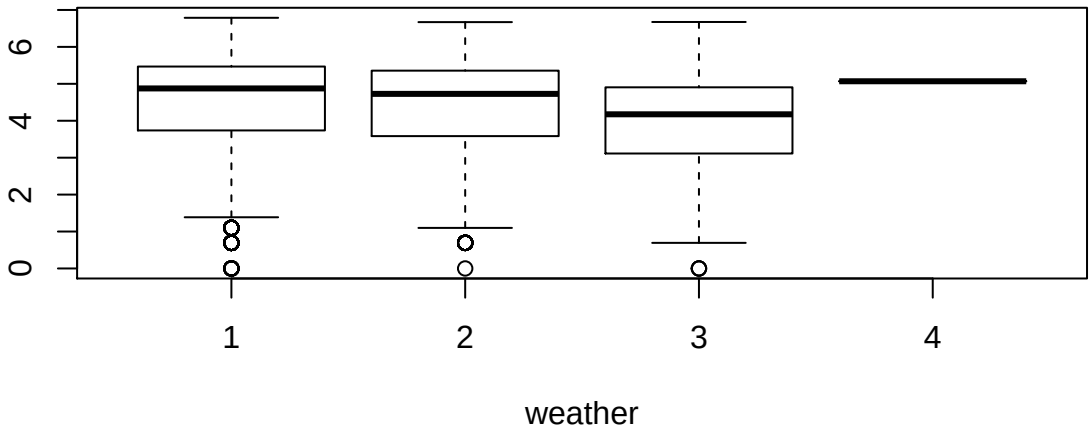
8.6.2.1 Plot by weather, by season

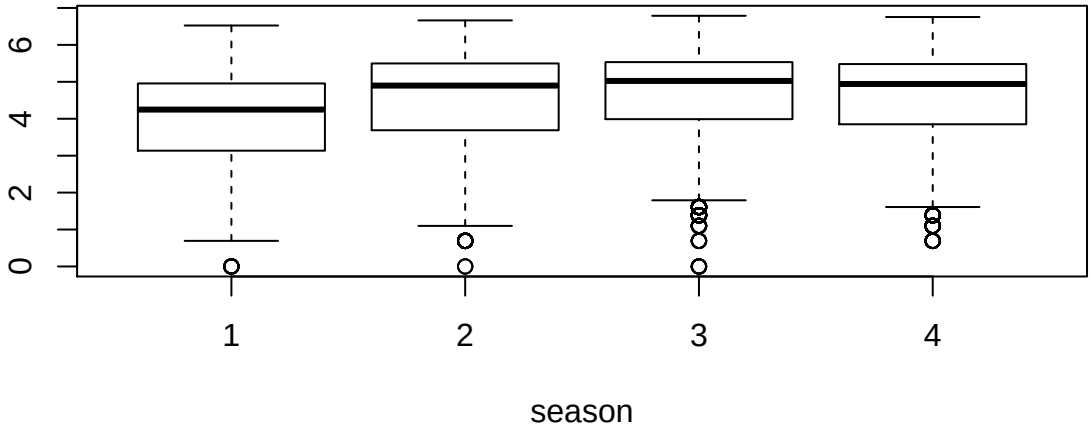
```
# cartesian plot
par(mfrow=c(2,1))
boxplot(train$registered ~ train$weather, xlab="weather", ylab="registered users")
boxplot(train$registered ~ train$season, xlab="season", ylab="registered users")
```






```
# semilog plot
par(mfrow=c(2,1))
boxplot(train$logreg ~ train$weather, xlab = "weather")
boxplot(train$logreg ~ train$season, xlab = "season")
```





8.6.3 Predicting for registered and casual users, test dataset

```
tic()
# final model building using random forest
# note that we build different models for predicting for
# registered and casual users
# this was seen as giving best result after a lot of experimentation
set.seed(415)
fit1 <- randomForest(logreg ~ hour + workingday + day + holiday + day_type +
                         temp_reg + humidity + atemp + windspeed + season +
                         weather + dp_reg + weekend + year + year_part,
                     data = train,
                     importance = TRUE,
                     ntree = 250)

pred1 = predict(fit1, test)
test$logreg = pred1
toc()
```

```
#> 144.137 sec elapsed
```

```
# casual users
set.seed(415)
fit2 <- randomForest(logcas ~ hour + day_type + day + humidity + atemp +
                         temp_cas + windspeed + season + weather + holiday +
                         workingday + dp_cas + weekend + year + year_part,
                     data = train, importance = TRUE, ntree = 250)

pred2 = predict(fit2, test)
test$logcas = pred2
```

8.6.4 Preparing and exporting results

```
# creating the final submission file
# reverse log conversion
test$registered <- exp(test$logreg) - 1
test$casual     <- exp(test$logcas) - 1
test$count      <- test$casual + test$registered

r <- data.frame(datetime = test$datetime,
                casual = test$casual,
                registered = test$registered)

print(sum(r$casual))
```

```
#> [1] 205803.7
```

```
print(sum(r$registered))
```

```
#> [1] 962834.2
```

```
s <- data.frame(datetime = test$datetime, count = test$count)
write.csv(s, file =file.path(data_out_dir, "bike-submit.csv"), row.names = FALSE)

# sum(cas+reg) = 1168638
# month number now is correct
```

After following the steps mentioned above, you can score 0.38675 on Kaggle leaderboard i.e. top 5 percentile of total participants. As you might have seen, we have not applied any extraordinary science in getting to this level. But, the real competition starts here. I would like to see, if I can improve this further by use of more features and some more advanced modeling techniques.

8.7 End Notes

In this article, we have looked at structured approach of problem solving and how this method can help you to improve performance. I would recommend you to generate hypothesis before you deep dive in the data set as this technique will not limit your thought process. You can improve your performance by applying advanced techniques (or ensemble methods) and understand your data trend better.

You can find the complete solution here : GitHub Link

```
# this is the older submission. months were incomplete
old <- read.csv(file = file.path(data_raw_dir, "bike-submit-old.csv"))
```

```
#> Error in file(file, "rt"): cannot open the connection
```

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

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

Chapter 9

Breast Cancer Wisconsin

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

9.1 Read and process the data

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

```
# assign the column names
colnames(bc_data) <- c("sample_code_number", "clump_thickness",
                       "uniformity_of_cell_size", "uniformity_of_cell_shape",
                       "marginal_adhesion", "single_epithelial_cell_size",
                       "bare_nuclei", "bland_chromatin", "normal_nucleoli",
                       "mitosis", "classes")

# change classes from numeric to character
bc_data$classes <- ifelse(bc_data$classes == "2", "benign",
                          ifelse(bc_data$classes == "4", "malignant", NA))

# if query sign make NA
bc_data[bc_data == "?"] <- NA

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

```
[1] 16
```

```
names(bc_data)
```

```
 [1] "sample_code_number"          "clump_thickness"
 [3] "uniformity_of_cell_size"     "uniformity_of_cell_shape"
 [5] "marginal_adhesion"           "single_epithelial_cell_size"
 [7] "bare_nuclei"                 "bland_chromatin"
 [9] "normal_nucleoli"             "mitosis"
[11] "classes"
```

9.1.1 Missing data

```
# impute missing data
library(mice)

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

# impute but mute
dataset_impute <- mice(bc_data[, 2:10],  print = FALSE)

# bind "classes" with the rest. skip "sample_code_number"
bc_data <- cbind(bc_data[, 11, drop = FALSE],
                 mice::complete(dataset_impute, action =1))

bc_data$classes <- as.factor(bc_data$classes)

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

```
   benign malignant
      458       241
```

```
# confirm NAs have been removed
length(which(is.na(bc_data)))
```

```
[1] 0
```

```
str(bc_data)
```

```
'data.frame':   699 obs. of  10 variables:
 $ classes                    : Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 1 1 1 1 ...
 $ clump_thickness            : num  5 5 3 6 4 8 1 2 2 4 ...
 $ uniformity_of_cell_size    : num  1 4 1 8 1 10 1 1 1 2 ...
 $ uniformity_of_cell_shape   : num  1 4 1 8 1 10 1 2 1 1 ...
 $ marginal_adhesion          : num  1 5 1 1 3 8 1 1 1 1 ...
 $ single_epithelial_cell_size: num  2 7 2 3 2 7 2 2 2 2 ...
 $ bare_nuclei                : num  1 10 2 4 1 10 10 1 1 1 ...
 $ bland_chromatin            : num  3 3 3 3 3 9 3 3 1 2 ...
 $ normal_nucleoli            : num  1 2 1 7 1 7 1 1 1 1 ...
 $ mitosis                    : num  1 1 1 1 1 1 1 1 5 1 ...
```

9.2 Principal Component Analysis (PCA)

To get an idea about the dimensionality and variance of the datasets, I am first looking at PCA plots for samples and features. The first two principal components (PCs) show the two components that explain the majority of variation in the data.

After defining my custom `ggplot2` theme, I am creating a function that performs the PCA (using the `pcaGoPromoter` package), calculates ellipses of the data points (with the `ellipse` package) and produces the plot with `ggplot2`. Some of the features in datasets 2 and 3 are not very distinct and overlap in the PCA plots, therefore I am also plotting hierarchical clustering dendrograms.

9.2.0.1 theme

```
# plotting theme

library(ggplot2)

my_theme <- function(base_size = 12, base_family = "sans"){
  theme_minimal(base_size = base_size, base_family = base_family) +
  theme(
    axis.text = element_text(size = 12),
    axis.text.x = element_text(angle = 0, vjust = 0.5, hjust = 0.5),
    axis.title = element_text(size = 14),
    panel.grid.major = element_line(color = "grey"),
    panel.grid.minor = element_blank(),
    panel.background = element_rect(fill = "aliceblue"),
    strip.background = element_rect(fill = "navy", color = "navy", size = 1),
    strip.text = element_text(face = "bold", size = 12, color = "white"),
    legend.position = "right",
    legend.justification = "top",
    legend.background = element_blank(),
    panel.border = element_rect(color = "grey", fill = NA, size = 0.5)
  )
}

theme_set(my_theme())
```

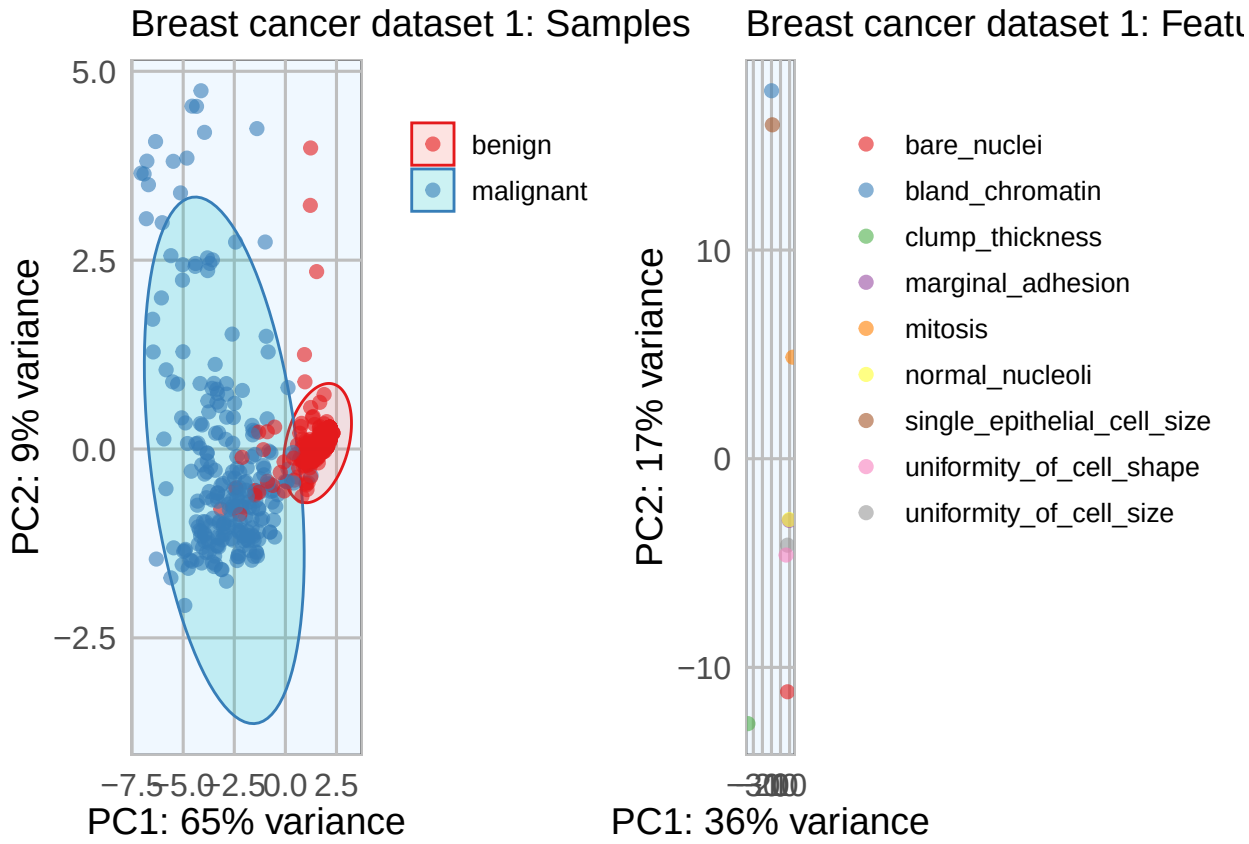
9.2.0.2 PCA function

```
# function for PCA plotting
library(pcaGoPromoter)                  # install from BioConductor
library(ellipse)

pca_func <- function(data, groups, title, print_ellipse = TRUE) {

  # perform pca and extract scores for all principal components: PC1:PC9
  pcaOutput <- pca(data, printDropped = FALSE, scale = TRUE, center = TRUE)
  pcaOutput2 <- as.data.frame(pcaOutput$scores)

  # define groups for plotting. will group the classes
  pcaOutput2$groups <- groups

  # when plotting samples calculate ellipses for plotting
  # (when plotting features, there are no replicates)
  if (print_ellipse) {
    # group and summarize by classes: benign, malignant
    # centroids w/3 columns: groups, PC1, PC2
    centroids <- aggregate(cbind(PC1, PC2) ~ groups, pcaOutput2, mean)
    # bind for the two groups (classes)
    # conf.rgn w/3 columns: groups, PC1, PC2
    conf.rgn  <- do.call(rbind, lapply(unique(pcaOutput2$groups), function(t)
      data.frame(groups = as.character(t),
                 # ellipse data for PC1 and PC2
```

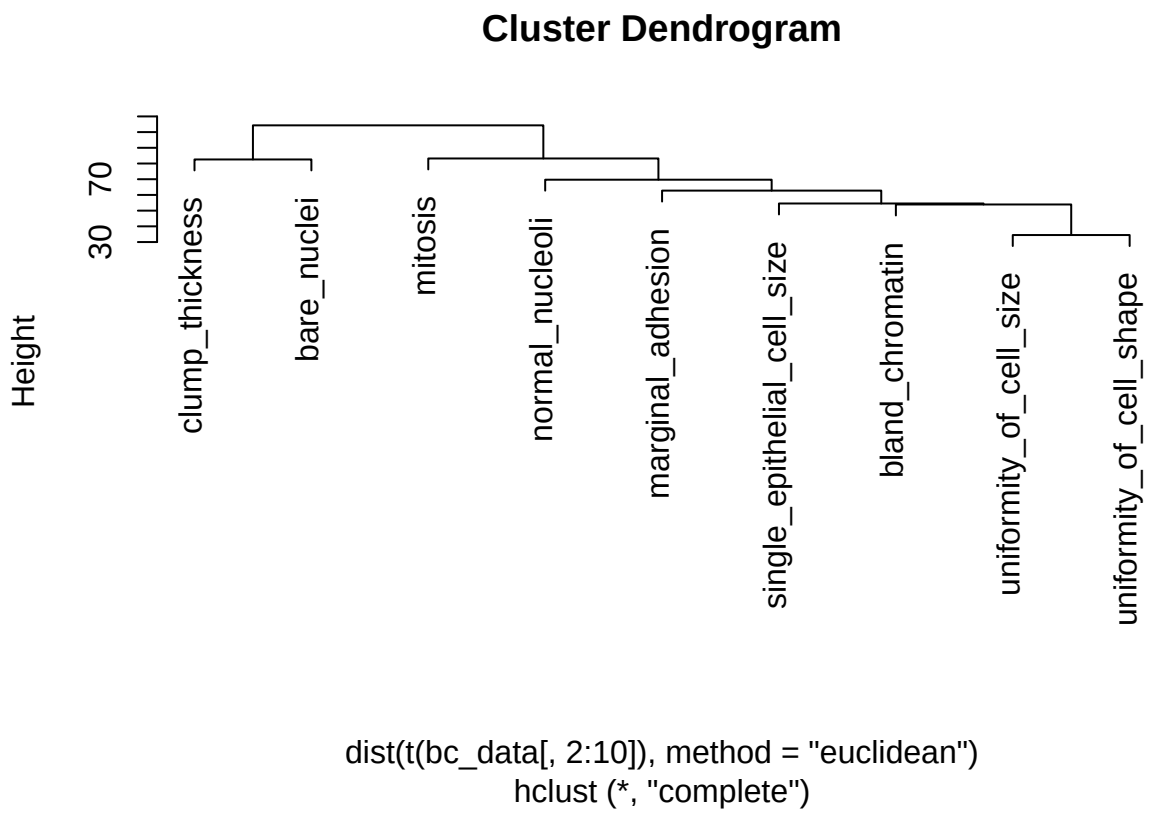
```
                ellipse(cov(pcaOutput2[pcaOutput2$groups == t, 1:2]),
                      centre = as.matrix(centroids[centroids$groups == t, 2:3]),
                      level = 0.95),
                stringsAsFactors = FALSE)))

  plot <- ggplot(data = pcaOutput2, aes(x = PC1, y = PC2,
                                        group = groups,
                                        color = groups)) +
    geom_polygon(data = conf.rgn, aes(fill = groups), alpha = 0.2) + # ellipses
    geom_point(size = 2, alpha = 0.6) +
    scale_color_brewer(palette = "Set1") +
    labs(title = title,
         color = "",
         fill = "",
         x = paste0("PC1: ", round(pcaOutput$pov[1], digits = 2) * 100,
                    "% variance"),
         y = paste0("PC2: ", round(pcaOutput$pov[2], digits = 2) * 100,
                    "% variance"))

} else {

  # if < 10 groups (e.g. the predictor classes) have colors from RColorBrewer
  if (length(unique(pcaOutput2$groups)) <= 10) {

    plot <- ggplot(data = pcaOutput2, aes(x = PC1, y = PC2,
                                          group = groups,
                                          color = groups)) +
      geom_point(size = 2, alpha = 0.6) +
      scale_color_brewer(palette = "Set1") +
      labs(title = title,
           color = "",
           fill = "",
           x = paste0("PC1: ", round(pcaOutput$pov[1], digits = 2) * 100,
                      "% variance"),
           y = paste0("PC2: ", round(pcaOutput$pov[2], digits = 2) * 100,
                      "% variance"))

  } else {
    # otherwise use the default rainbow colors
    plot <- ggplot(data = pcaOutput2, aes(x = PC1, y = PC2,
                                          group = groups, color = groups)) +
      geom_point(size = 2, alpha = 0.6) +
      labs(title = title,
           color = "",
           fill = "",
           x = paste0("PC1: ", round(pcaOutput$pov[1], digits = 2) * 100,
                      "% variance"),
           y = paste0("PC2: ", round(pcaOutput$pov[2], digits = 2) * 100,
                      "% variance"))
  }
}

return(plot)
```

```
}
```

```
library(gridExtra)
library(grid)

# plot all data. one row is a feature
p1 <- pca_func(data = t(bc_data[, 2:10]),
               groups = as.character(bc_data$classes),
               title = "Breast cancer dataset 1: Samples")

# plot features only. features as columns
p2 <- pca_func(data = bc_data[, 2:10],
               groups = as.character(colnames(bc_data[, 2:10])),
               title = "Breast cancer dataset 1: Features", print_ellipse = FALSE)

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

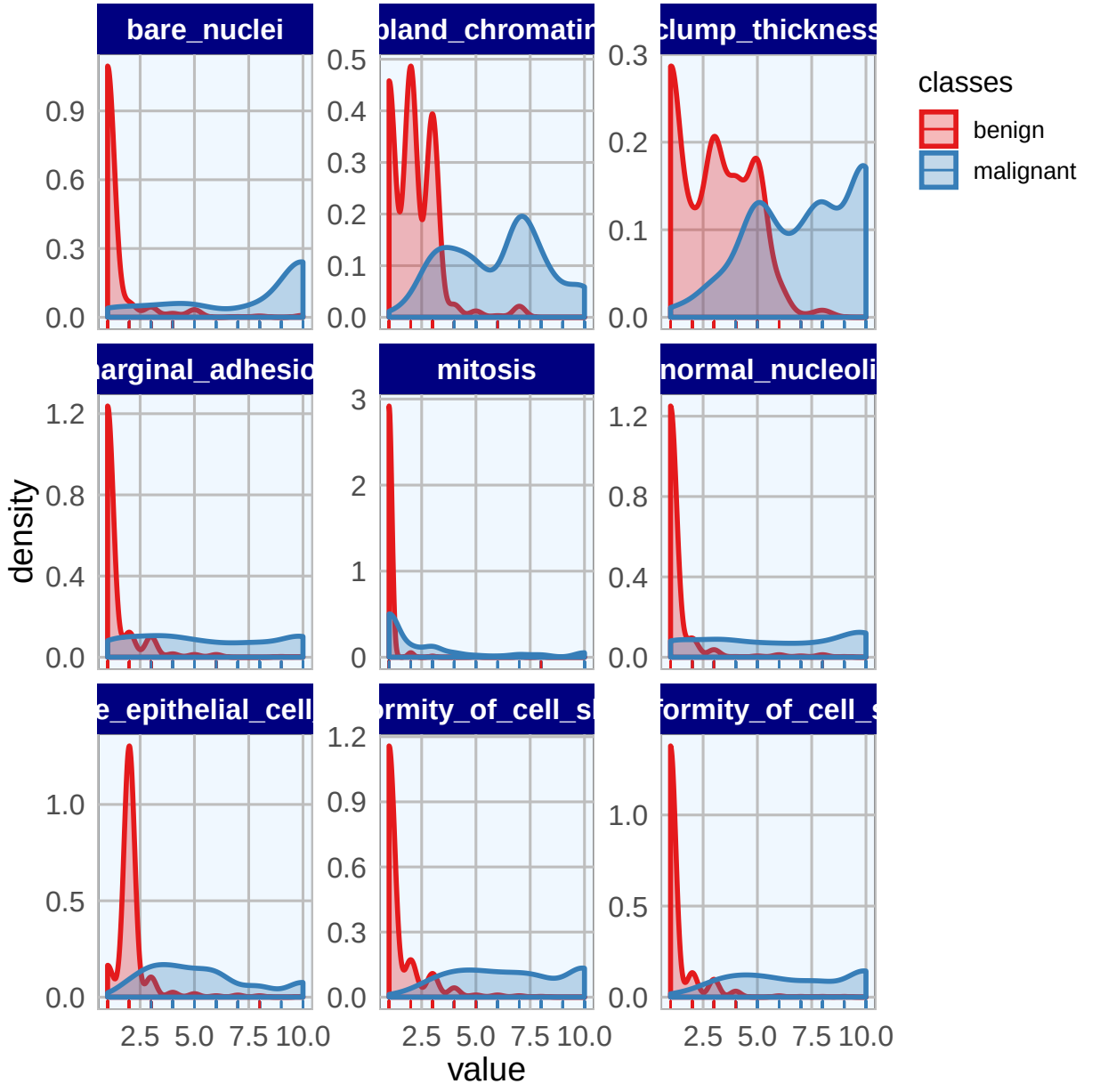



```
h_1 <- hclust(dist(t(bc_data[, 2:10]), method = "euclidean"), method = "complete")
plot(h_1)
```



9.2.1 density plots vs class

```
# density plot showing the feature vs classes
library(tidyr)

# gather data. from column clump_thickness to mitosis
bc_data_gather <- bc_data %>%
  gather(measure, value, clump_thickness:mitosis)

ggplot(data = bc_data_gather, aes(x = value, fill = classes, color = classes)) +
  geom_density(alpha = 0.3, size = 1) +
  geom_rug() +
  scale_fill_brewer(palette = "Set1") +
  scale_color_brewer(palette = "Set1") +
  facet_wrap( ~ measure, scales = "free_y", ncol = 3)
```



9.3 Feature importance

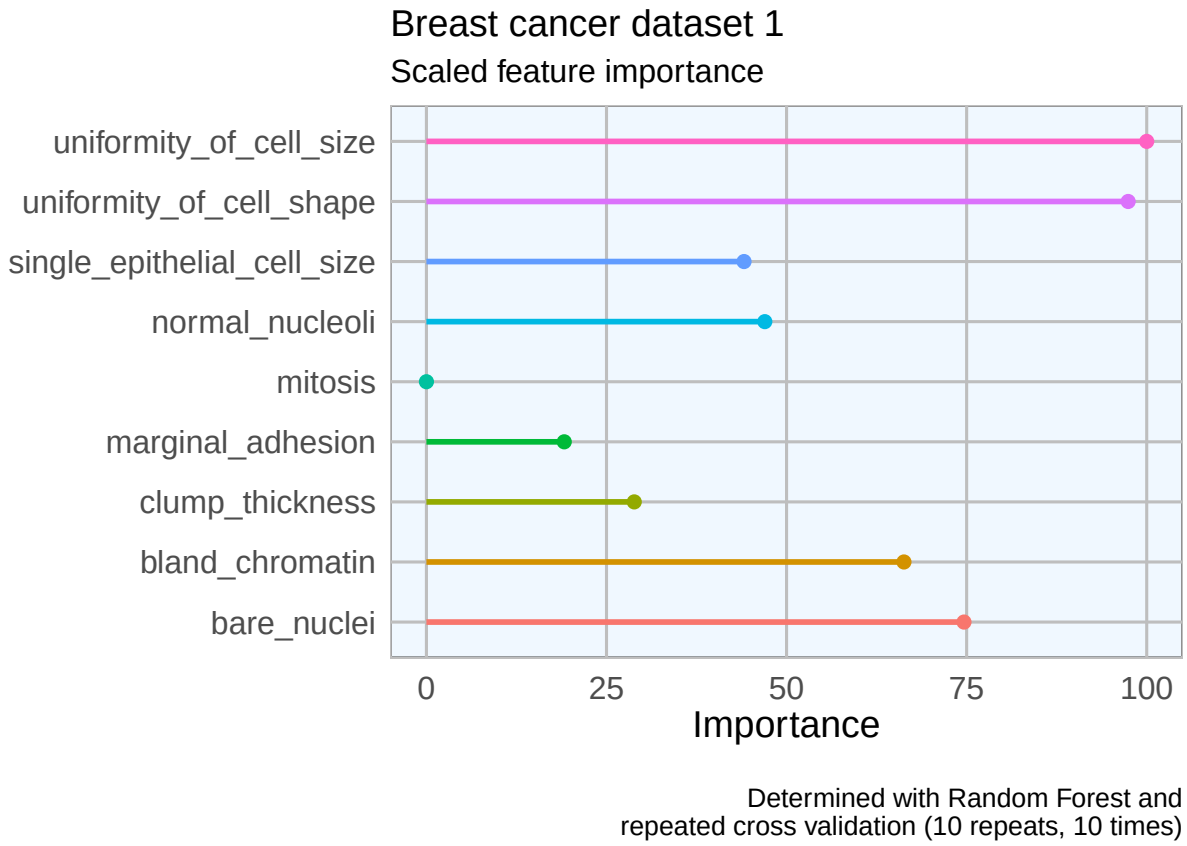
To get an idea about the feature's respective importances, I'm running Random Forest models with 10 x 10 cross validation using the `caret` package. If I wanted to use feature importance to select features for modeling, I would need to perform it on the training data instead of on the complete dataset. But here, I only want to use it to get acquainted with my data. I am again defining a function that estimates the feature importance and produces a plot.

```
library(caret)
# library(doParallel) # parallel processing
# registerDoParallel()

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

```
feature_imp <- function(model, title) {
  # estimate variable importance
  importance <- varImp(model, scale = TRUE)
  # prepare dataframes for plotting
  importance_df_1 <- importance$importance
  importance_df_1$group <- rownames(importance_df_1)

  importance_df_2 <- importance_df_1
  importance_df_2$Overall <- 0
  importance_df <- rbind(importance_df_1, importance_df_2)

  plot <- ggplot() +
    geom_point(data = importance_df_1, aes(x = Overall,
                                           y = group,
                                           color = group), size = 2) +
    geom_path(data = importance_df, aes(x = Overall,
                                        y = group,
                                        color = group,
                                        group = group), size = 1) +
    theme(legend.position = "none") +
    labs(
      x = "Importance",
      y = "",
      title = title,
      subtitle = "Scaled feature importance",
      caption = "\nDetermined with Random Forest and
      repeated cross validation (10 repeats, 10 times)"
    )
  return(plot)
}
```

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

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



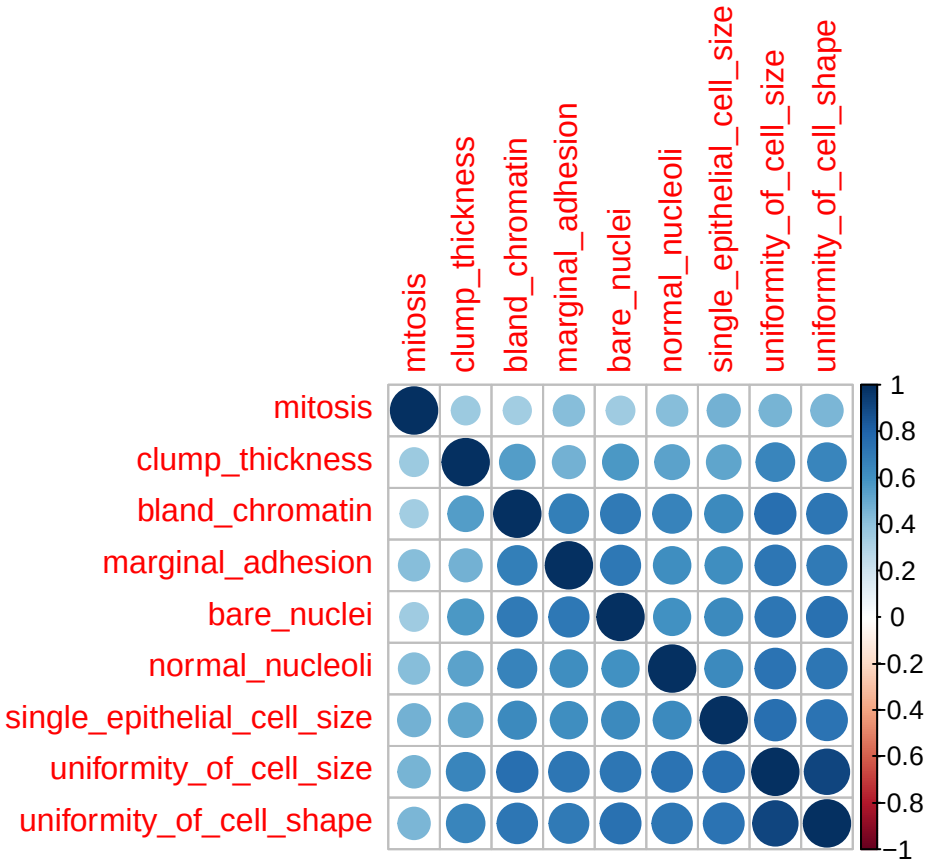
9.4 Feature Selection

1. By correlation
2. By Recursive Feature Elimination
3. By Genetic Algorithm

```
set.seed(27)
bc_data_index <- createDataPartition(bc_data$classes, p = 0.7, list = FALSE)
bc_data_train <- bc_data[bc_data_index, ]
bc_data_test  <- bc_data[-bc_data_index, ]
```

9.4.1 Correlation

```
library(corrplot)

# calculate correlation matrix
corMatMy <- cor(bc_data_train[, -1])
corrplot(corMatMy, order = "hclust")
```




```
# Apply correlation filter at 0.70,
highlyCor <- colnames(bc_data_train[, -1])[findCorrelation(corMatMy,
                                                           cutoff = 0.7,
                                                           verbose = TRUE)]
```

```
Compare row 2  and column  3 with corr  0.913
  Means:  0.715 vs 0.601 so flagging column 2
Compare row 3  and column  7 with corr  0.725
  Means:  0.677 vs 0.578 so flagging column 3
Compare row 7  and column  6 with corr  0.703
  Means:  0.6 vs 0.544 so flagging column 7
Compare row 6  and column  4 with corr  0.713
  Means:  0.576 vs 0.524 so flagging column 6
All correlations <= 0.7
```

```
# which variables are flagged for removal?
highlyCor
```

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

```
# then we remove these variables
bc_data_cor <- bc_data_train[, which(!colnames(bc_data_train) %in% highlyCor)]
names(bc_data_cor)
```

```
[1] "classes"                    "clump_thickness"
[3] "marginal_adhesion"          "single_epithelial_cell_size"
[5] "normal_nucleoli"            "mitosis"
```

```
# confirm features were removed
outersect <- function(x, y) {
  sort(c(setdiff(x, y),
         setdiff(y, x)))
}

outersect(names(bc_data_cor), names(bc_data_train))
```

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

Four features removed

9.4.2 Recursive Feature Elimination (RFE)

```
# ensure the results are repeatable
set.seed(7)

# define the control using a random forest selection function with cross validation
control <- rfeControl(functions = rfFuncs, method = "cv", number = 10)

# run the RFE algorithm
results_1 <- rfe(x = bc_data_train[, -1],
                 y = bc_data_train$classes,
                 sizes = c(1:9),
                 rfeControl = control)

# chosen features
predictors(results_1)
```

```
[1] "bare_nuclei"              "uniformity_of_cell_size"
[3] "clump_thickness"          "uniformity_of_cell_shape"
[5] "bland_chromatin"          "marginal_adhesion"
[7] "normal_nucleoli"          "mitosis"
```

```
# subset the chosen features
sel_cols <- which(colnames(bc_data_train) %in% predictors(results_1))
bc_data_rfe <- bc_data_train[, c(1, sel_cols)]
names(bc_data_rfe)
```

```
[1] "classes"                  "clump_thickness"
[3] "uniformity_of_cell_size"  "uniformity_of_cell_shape"
[5] "marginal_adhesion"        "bare_nuclei"
[7] "bland_chromatin"          "normal_nucleoli"
[9] "mitosis"
```

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

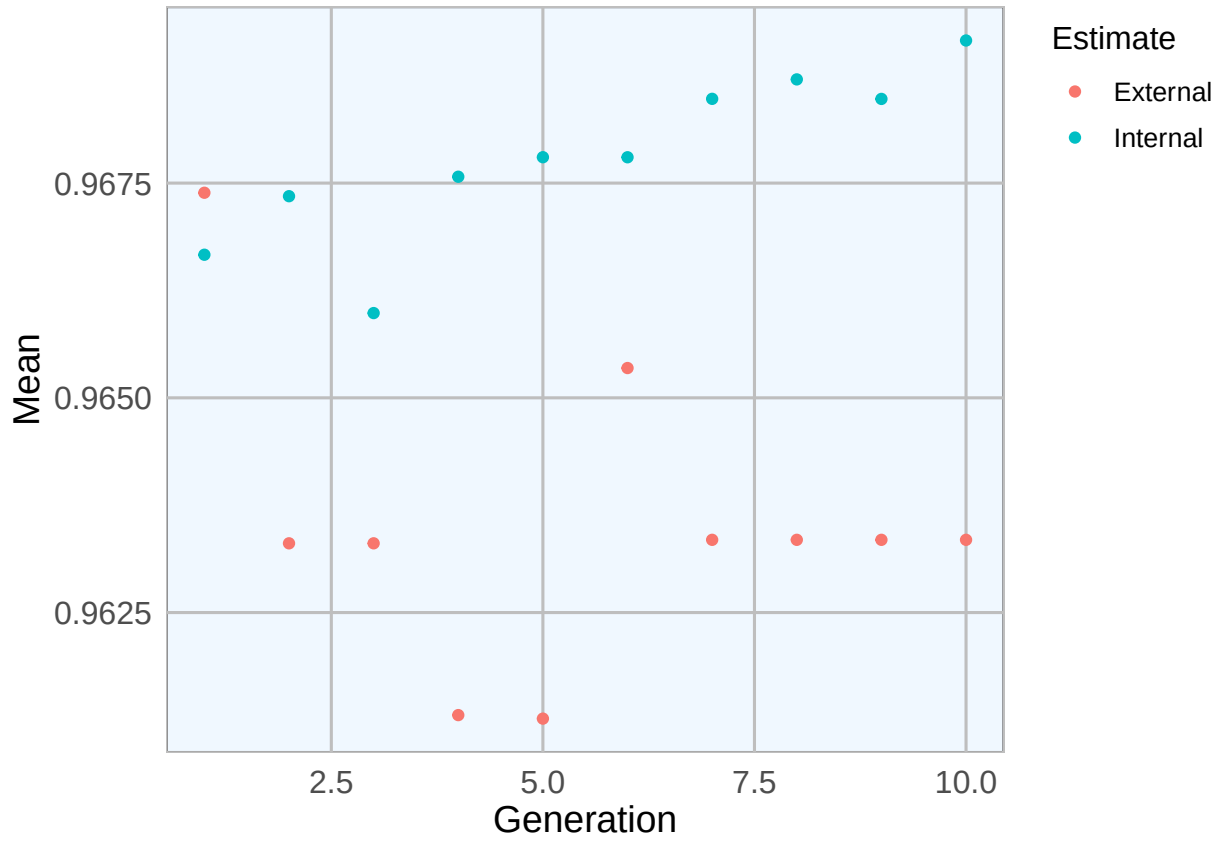
```
[1] "single_epithelial_cell_size"
```

No features removed with RFE

9.4.3 Genetic Algorithm (GA)

```
library(dplyr)

ga_ctrl <- gafsControl(functions = rfGA, # Assess fitness with RF
                       method = "cv",    # 10 fold cross validation
                       genParallel = TRUE, # Use parallel programming
                       allowParallel = TRUE)

lev <- c("malignant", "benign")     # Set the levels

set.seed(27)
model_1 <- gafs(x = bc_data_train[, -1], y = bc_data_train$classes,
                  iters = 10, # generations of algorithm
                  popSize = 5, # population size for each generation
                  levels = lev,
                  gafsControl = ga_ctrl)
```

```
plot(model_1) # Plot mean fitness (AUC) by generation
```




```
# features
model_1$ga$final
```

```
[1] "clump_thickness"             "uniformity_of_cell_size"
[3] "uniformity_of_cell_shape"    "marginal_adhesion"
[5] "single_epithelial_cell_size" "bare_nuclei"
[7] "bland_chromatin"             "normal_nucleoli"
```

```
[9] "mitosis"
```

```
# select features
sel_cols_ga <- which(colnames(bc_data_train) %in% model_1$ga$final)
bc_data_ga <- bc_data_train[, c(1, sel_cols_ga)]
names(bc_data_ga)
```

```
 [1] "classes"                    "clump_thickness"
 [3] "uniformity_of_cell_size"    "uniformity_of_cell_shape"
 [5] "marginal_adhesion"          "single_epithelial_cell_size"
 [7] "bare_nuclei"                "bland_chromatin"
 [9] "normal_nucleoli"            "mitosis"
```

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

```
character(0)
```

Two features removed with GA.

9.5 Model comparison

9.5.1 Using all features

```
set.seed(27)
model_bc_data_all <- train(classes ~ .,
                           data = bc_data_train,
                           method = "rf",
                           preProcess = c("scale", "center"),
                           trControl = trainControl(method = "repeatedcv",
                                                    number = 5, repeats = 10,
                                                    verboseIter = FALSE))
```

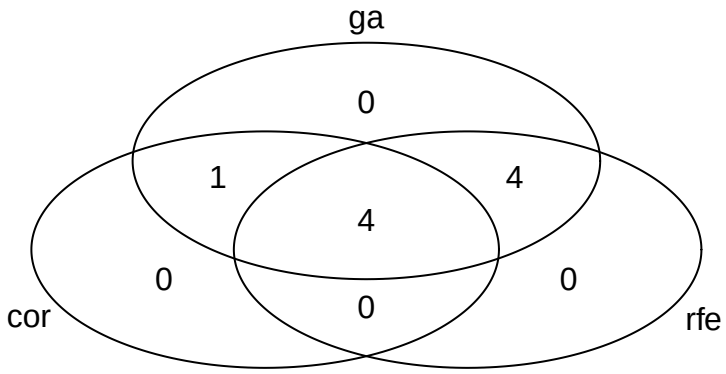
```
# confusion matrix
cm_all_1 <- confusionMatrix(predict(model_bc_data_all, bc_data_test[, -1]), bc_data_test$classes)
cm_all_1
```

```
Confusion Matrix and Statistics

           Reference
Prediction  benign malignant
  benign       134         5
  malignant      3        67

               Accuracy : 0.9617
                 95% CI : (0.926, 0.9833)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9147

 Mcnemar's Test P-Value : 0.7237

            Sensitivity : 0.9781
```

```
            Specificity : 0.9306
         Pos Pred Value : 0.9640
         Neg Pred Value : 0.9571
             Prevalence : 0.6555
         Detection Rate : 0.6411
   Detection Prevalence : 0.6651
      Balanced Accuracy : 0.9543

       'Positive' Class : benign
```

9.5.2 Compare selection methods

```
# compare features selected by the three methods
library(gplots)

venn_list <- list(cor = colnames(bc_data_cor)[-1],
                  rfe = colnames(bc_data_rfe)[-1],
                  ga  = colnames(bc_data_ga)[-1])

venn <- venn(venn_list)
```




```
venn
```

```
    num cor rfe ga
000   0   0   0  0
001   0   0   0  1
010   0   0   1  0
011   4   0   1  1
100   0   1   0  0
101   1   1   0  1
110   0   1   1  0
111   4   1   1  1
attr(,"intersections")
attr(,"intersections")$`cor:rfe:ga`
[1] "clump_thickness"   "marginal_adhesion" "normal_nucleoli"
[4] "mitosis"

attr(,"intersections")$`cor:ga`
[1] "single_epithelial_cell_size"

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

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

attr(,"class")
[1] "venn"
```

4 out of 10 features were chosen by all three methods; the biggest overlap is seen between GA and RFE with 7 features. RFE and GA both retained 8 features for modeling, compared to only 5 based on the correlation method.

9.5.3 Correlation

```
# correlation
set.seed(127)
model_bc_data_cor <- train(classes ~ .,
                 data = bc_data_cor,
                 method = "rf",
                 preProcess = c("scale", "center"),
                 trControl = trainControl(method = "repeatedcv", number = 5, repeats = 10, verboseIter =
```

```
cm_cor_1 <- confusionMatrix(predict(model_bc_data_cor, bc_data_test[, -1]), bc_data_test$classes)
cm_cor_1
```

```
Confusion Matrix and Statistics

           Reference
Prediction  benign malignant
  benign       132         6
  malignant      5        66

               Accuracy : 0.9474
                 95% CI : (0.9078, 0.9734)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8831

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9635
            Specificity : 0.9167
         Pos Pred Value : 0.9565
         Neg Pred Value : 0.9296
             Prevalence : 0.6555
         Detection Rate : 0.6316
   Detection Prevalence : 0.6603
      Balanced Accuracy : 0.9401

       'Positive' Class : benign
```

9.5.4 Recursive Feature Elimination

```
set.seed(127)
model_bc_data_rfe <- train(classes ~ .,
                           data = bc_data_rfe,
                           method = "rf",
                           preProcess = c("scale", "center"),
                           trControl = trainControl(method = "repeatedcv",
                                                    number = 5, repeats = 10,
                                                    verboseIter = FALSE))
```

```
cm_rfe_1 <- confusionMatrix(predict(model_bc_data_rfe, bc_data_test[, -1]), bc_data_test$classes)
cm_rfe_1
```

```
Confusion Matrix and Statistics

           Reference
Prediction  benign malignant
  benign       134         4
  malignant      3        68

               Accuracy : 0.9665
                 95% CI : (0.9322, 0.9864)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9256

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9781
            Specificity : 0.9444
         Pos Pred Value : 0.9710
         Neg Pred Value : 0.9577
             Prevalence : 0.6555
         Detection Rate : 0.6411
   Detection Prevalence : 0.6603
      Balanced Accuracy : 0.9613

       'Positive' Class : benign
```

9.5.5 GA

```
set.seed(127)
model_bc_data_ga <- train(classes ~ .,
                          data = bc_data_ga,
                          method = "rf",
                          preProcess = c("scale", "center"),
                          trControl = trainControl(method = "repeatedcv",
                                                   number = 5, repeats = 10,
                                                   verboseIter = FALSE))
```

```
cm_ga_1 <- confusionMatrix(predict(model_bc_data_ga, bc_data_test[, -1]), bc_data_test$classes)
cm_ga_1
```

```
Confusion Matrix and Statistics

           Reference
Prediction  benign malignant
  benign       134         5
  malignant      3        67

               Accuracy : 0.9617
                 95% CI : (0.926, 0.9833)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9147

 Mcnemar's Test P-Value : 0.7237

            Sensitivity : 0.9781
            Specificity : 0.9306
         Pos Pred Value : 0.9640
         Neg Pred Value : 0.9571
             Prevalence : 0.6555
         Detection Rate : 0.6411
   Detection Prevalence : 0.6651
      Balanced Accuracy : 0.9543

       'Positive' Class : benign
```

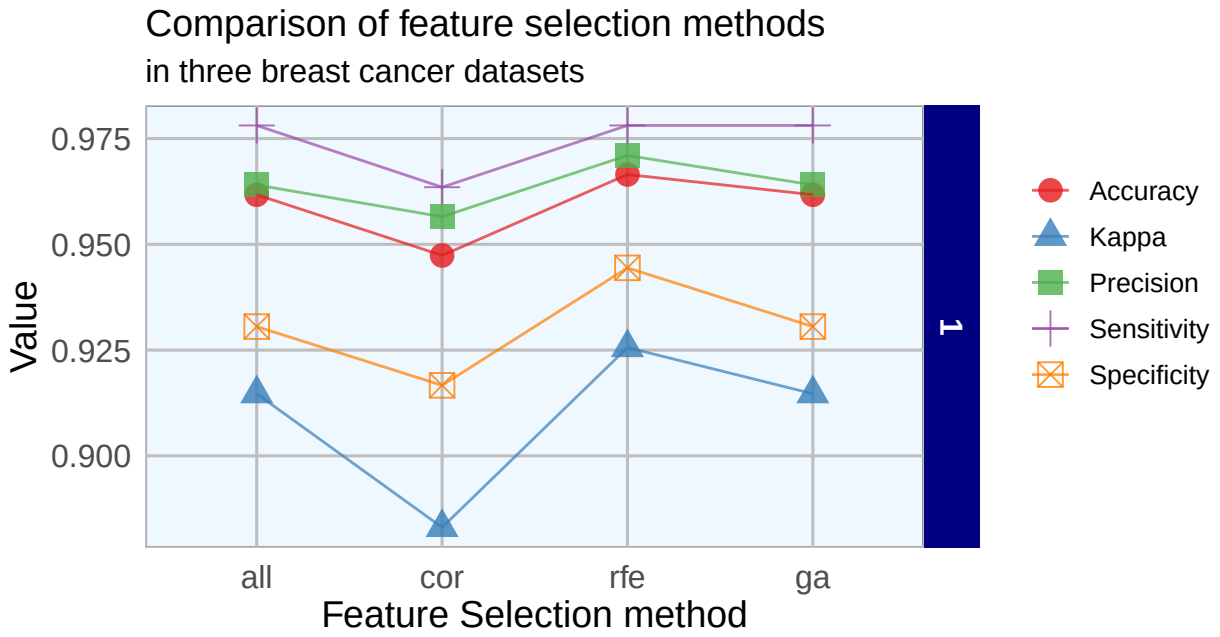
9.6 Create comparison tables

```
# take "overall" variable only from Confusion Matrix
overall <- data.frame(dataset = 1,
           model = rep(c("all", "cor", "rfe", "ga"), 1),
           rbind(cm_all_1$overall,
                 cm_cor_1$overall,
                 cm_rfe_1$overall,
                 cm_ga_1$overall)
)

# convert to tidy data
library(tidyr)
overall_gather <- overall[, 1:4] %>%     # take the first columns:
  gather(measure, value, Accuracy:Kappa) # dataset, model, Accuracy and Kappa
```

```
# take "byClass" variable only from Confusion Matrix
byClass <- data.frame(dataset = 1,
                      model = rep(c("all", "cor", "rfe", "ga"), 1),
                      rbind(cm_all_1$byClass,
                      cm_cor_1$byClass,
```

```
                      cm_rfe_1$byClass,
                      cm_ga_1$byClass)
)

# convert to tidy data
byClass_gather <- byClass[, c(1:4, 7)] %>%      # select columns: dataset, model
  gather(measure, value, Sensitivity:Precision) # Sensitiv, Specific, Precis
```

```
# join the two tables
overall_byClass_gather <- rbind(overall_gather, byClass_gather)
overall_byClass_gather <- within(
  overall_byClass_gather, model <- factor(model,
                                          levels = c("all", "cor", "rfe", "ga")))
                                          # convert to factor

ggplot(overall_byClass_gather, aes(x = model, y = value, color = measure,
                                   shape = measure, group = measure)) +
  geom_point(size = 4, alpha = 0.8) +
  geom_path(alpha = 0.7) +
  scale_colour_brewer(palette = "Set1") +
  facet_grid(dataset ~ ., scales = "free_y") +
  labs(
    x = "Feature Selection method",
    y = "Value",
    color = "",
    shape = "",
    title = "Comparison of feature selection methods",
    subtitle = "in three breast cancer datasets",
    caption = "\nBreast Cancer Wisconsin (Diagnostic) Data Sets: 1, 2 & 3
    Street et al., 1993;
    all: no feature selection
    cor: features with correlation > 0.7 removed
    rfe: Recursive Feature Elimination
    ga: Genetic Algorithm"
  )
```



Breast Cancer Wisconsin (Diagnostic) Data Sets: 1, 2 & 3
Street et al., 1993;
all: no feature selection
cor: features with correlation > 0.7 removed
rfe: Recursive Feature Elimination
ga: Genetic Algorithm

1. Less accurate: selection of features by correlation
2. More accurate: genetic algorithm
3. Including all features is more accurate to removing features by correlation.

9.7 Notes

`pcaGoPromoter` is a BioConductor package. Its dependencies are `BioGenerics`, `AnnotationDbi` and `BioStrings`, which at their turn require `DBI` and `RSQLite` packages from CRAN. Install first those from CRAN, and then move to install `pcaGoPromoter`.

Chapter 10

Titanic with Naive-Bayes Classifier

The Titanic dataset in R is a table for about 2200 passengers summarised according to four factors – economic status ranging from 1st class, 2nd class, 3rd class and crew; gender which is either male or female; Age category which is either Child or Adult and whether the type of passenger survived. For each combination of Age, Gender, Class and Survived status, the table gives the number of passengers who fall into the combination. We will use the Naive Bayes Technique to classify such passengers and check how well it performs.

```
#Getting started with Naive Bayes
#Install the package
#install.packages("e1071")
#Loading the library
library(e1071)

#Next load the Titanic dataset
data("Titanic")
#Save into a data frame and view it
Titanic_df = as.data.frame(Titanic)
```

We see that there are 32 observations which represent all possible combinations of Class, Sex, Age and Survived with their frequency. Since it is summarised, this table is not suitable for modelling purposes. We need to expand the table into individual rows. Let's create a repeating sequence of rows based on the frequencies in the table

```
#Creating data from table
repeating_sequence=rep.int(seq_len(nrow(Titanic_df)), Titanic_df$Freq) #This will repeat each combinati

# Create the dataset by row repetition created
Titanic_dataset=Titanic_df[repeating_sequence,]

# We no longer need the frequency, drop the feature
Titanic_dataset$Freq=NULL
```

The data is now ready for Naive Bayes to process. Let's fit the model

```
# Fitting the Naive Bayes model
Naive_Bayes_Model=naiveBayes(Survived ~., data=Titanic_dataset)

# What does the model say? Print the model summary
Naive_Bayes_Model
```

```
:>
:> Naive Bayes Classifier for Discrete Predictors
:>
:> Call:
:> naiveBayes.default(x = X, y = Y, laplace = laplace)
:>
:> A-priori probabilities:
:> Y
:>       No      Yes
:> 0.676965 0.323035
:>
:> Conditional probabilities:
:>      Class
:> Y            1st        2nd        3rd       Crew
:>   No  0.08187919 0.11208054 0.35436242 0.45167785
:>   Yes 0.28551336 0.16596343 0.25035162 0.29817159
:>
:>      Sex
:> Y           Male     Female
:>   No  0.91543624 0.08456376
:>   Yes 0.51617440 0.48382560
:>
:>      Age
:> Y          Child      Adult
:>   No  0.03489933 0.96510067
:>   Yes 0.08016878 0.91983122
```

The model creates the conditional probability for each feature separately. We also have the a-priori probabilities which indicates the distribution of our data. Let's calculate how we perform on the data.

```
# Prediction on the dataset
NB_Predictions=predict(Naive_Bayes_Model,Titanic_dataset)
# Confusion matrix to check accuracy
table(NB_Predictions,Titanic_dataset$Survived)
```

```
:>
:> NB_Predictions   No  Yes
:>            No  1364  362
:>            Yes  126  349
```

We have the results! We are able to classify 1364 out of 1490 "No" cases correctly and 349 out of 711 "Yes" cases correctly. This means the ability of Naive Bayes algorithm to predict "No" cases is about 91.5% but it falls down to only 49% of the "Yes" cases resulting in an overall accuracy of 77.8%

Chapter 11

Can we Do any Better?

Naive Bayes is a parametric algorithm which implies that you cannot perform differently in different runs as long as the data remains the same. We will, however, learn another implementation of Naive Bayes algorithm using the 'mlr' package. Assuming the same session is going on for the readers, I will install and load the package and start fitting a model

```
# Getting started with Naive Bayes in mlr
# install.packages("mlr")
# Loading the library
library(mlr)
```

The mlr package consists of a lot of models and works by creating tasks and learners which are then trained. Let's create a classification task using the titanic dataset and fit a model with the naive bayes algorithm.

```
# Create a classification task for learning on Titanic Dataset and specify the target feature
task = makeClassifTask(data = Titanic_dataset, target = "Survived")

# Initialize the Naive Bayes classifier
selected_model = makeLearner("classif.naiveBayes")

# Train the model
NB_mlr = train(selected_model, task)
```

The summary of the model which was printed in e3071 package is stored in learner model. Let's print it and compare

```
# Read the model learned
NB_mlr$learner.model
```

```
:>
:> Naive Bayes Classifier for Discrete Predictors
:>
:> Call:
:> naiveBayes.default(x = X, y = Y, laplace = laplace)
:>
:> A-priori probabilities:
:> Y
:>       No      Yes
:> 0.676965 0.323035
:>
:> Conditional probabilities:
```

```
:>      Class
:> Y            1st        2nd        3rd       Crew
:>   No  0.08187919 0.11208054 0.35436242 0.45167785
:>   Yes 0.28551336 0.16596343 0.25035162 0.29817159
:>
:>      Sex
:> Y           Male     Female
:>   No  0.91543624 0.08456376
:>   Yes 0.51617440 0.48382560
:>
:>      Age
:> Y          Child      Adult
:>   No  0.03489933 0.96510067
:>   Yes 0.08016878 0.91983122
```

The a-priori probabilities and the conditional probabilities for the model are similar to the one calculated by e3071 package as was expected. This means that our predictions will also be the same.

```
# Predict on the dataset without passing the target feature
predictions_mlr = as.data.frame(predict(NB_mlr, newdata = Titanic_dataset[,1:3]))

## Confusion matrix to check accuracy
table(predictions_mlr[,1],Titanic_dataset$Survived)
```

```
:>
:>         No  Yes
:>   No  1364  362
:>   Yes  126  349
```

As we see, the predictions are exactly same. The only way to improve is to have more features or more data. Perhaps, if we have more features such as the exact age, size of family, number of parents in the ship and siblings then we may arrive at a better model using Naive Bayes. In essence, Naive Bayes has an advantage of a strong foundation build and is very robust. I know of the 'caret' package which also consists of Naive Bayes function but it will also give us the same predictions and probability.

Chapter 12

Building a Naive Bayes Classifier in R

https://www.machinelearningplus.com/predictive-modeling/how-naive-bayes-algorithm-works-with-example-and-full-code/

12.1 8. Building a Naive Bayes Classifier in R

Understanding Naive Bayes was the (slightly) tricky part. Implementing it is fairly straightforward.

In R, Naive Bayes classifier is implemented in packages such as `e1071`, `klaR` and `bnlearn`. In Python, it is implemented in `scikit-learn`.

For sake of demonstration, let's use the standard iris dataset to predict the Species of flower using 4 different features: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width

```
# Import Data
training <- read.csv('https://raw.githubusercontent.com/selva86/datasets/master/iris_train.csv')
test <- read.csv('https://raw.githubusercontent.com/selva86/datasets/master/iris_test.csv')
```

The training data is now contained in training and test data in test dataframe. Lets load the klaR package and build the naive bayes model.

```
# Using klaR for Naive Bayes
library(klaR)
nb_mod <- NaiveBayes(Species ~ ., data=training)
pred <- predict(nb_mod, test)
```

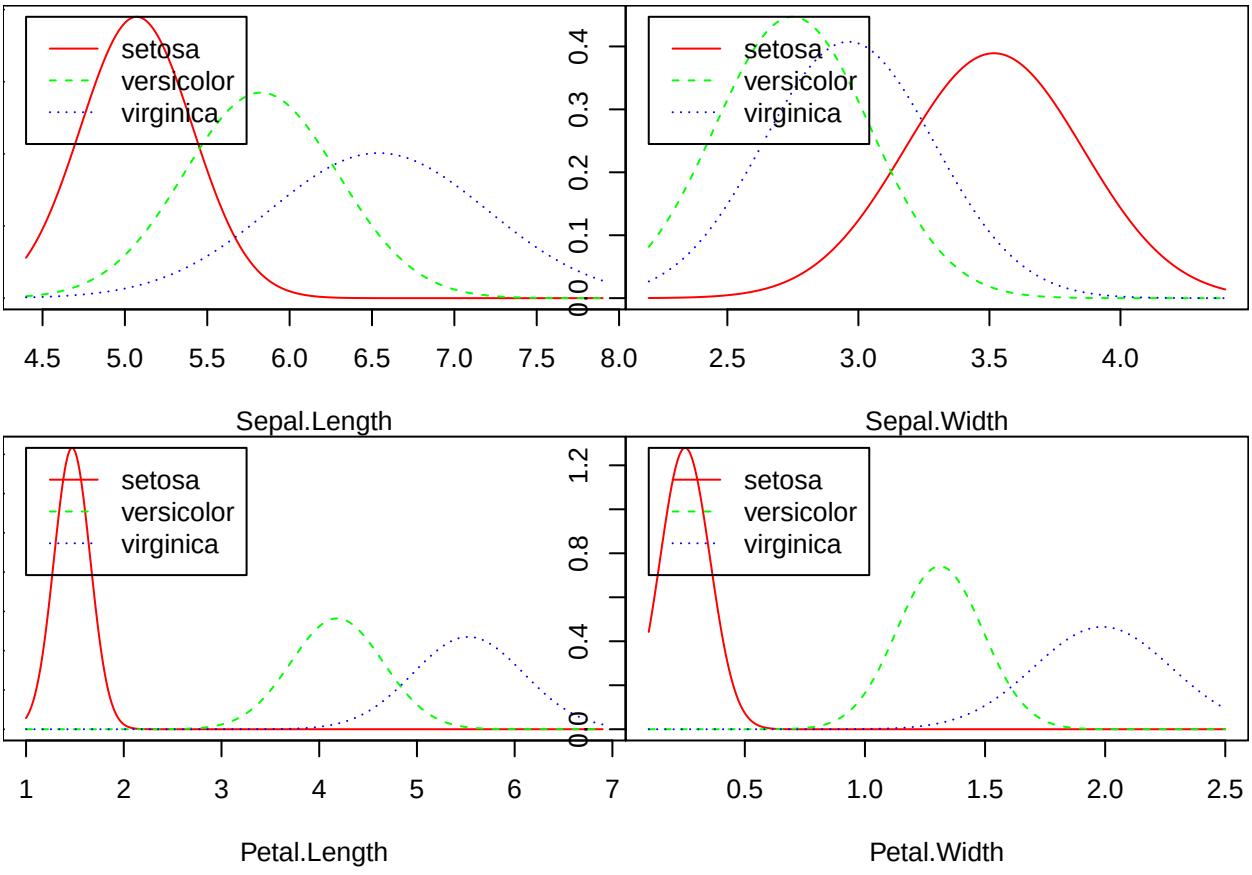
Lets see the confusion matrix.

```
# Confusion Matrix
tab <- table(pred$class, test$Species)
caret::confusionMatrix(tab)
```

```
:> Confusion Matrix and Statistics
:>
:>
:>              setosa versicolor virginica
:>   setosa         15          0         0
:>   versicolor      0         11         0
:>   virginica       0          4        15
:>
:> Overall Statistics
```

```
:>
:>                Accuracy : 0.9111
:>                  95% CI : (0.7878, 0.9752)
:>     No Information Rate : 0.3333
:>     P-Value [Acc > NIR] : 8.467e-16
:>
:>                   Kappa : 0.8667
:>
:>  Mcnemar's Test P-Value : NA
:>
:> Statistics by Class:
:>
:>                      Class: setosa Class: versicolor Class: virginica
:> Sensitivity                 1.0000            0.7333           1.0000
:> Specificity                 1.0000            1.0000           0.8667
:> Pos Pred Value              1.0000            1.0000           0.7895
:> Neg Pred Value              1.0000            0.8824           1.0000
:> Prevalence                  0.3333            0.3333           0.3333
:> Detection Rate              0.3333            0.2444           0.3333
:> Detection Prevalence        0.3333            0.2444           0.4222
:> Balanced Accuracy           1.0000            0.8667           0.9333
```

```
# Plot density of each feature using nb_mod
opar = par(mfrow=c(2, 2), mar=c(4,0,0,0))
plot(nb_mod, main="")
```



```
par(opar)
```

```
# Plot the Confusion Matrix
library(ggplot2)
test$pred <- pred$class
ggplot(test, aes(Species, pred, color = Species)) +
  geom_jitter(width = 0.2, height = 0.1, size=2) +
  labs(title="Confusion Matrix",
       subtitle="Predicted vs. Observed from Iris dataset",
       y="Predicted",
       x="Truth",
       caption="machinelearningplus.com")
```