810 Team Project

Bo Li U24425931

2/24/2021

```r
library(data.table)
library(ggplot2)
library(ggthemes)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1
```

```r
theme_set(theme_bw())
library(MASS)
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(caret)
```

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

```r
library(e1071)
library(tree)
library(ISLR)
library(party)
```

```
## Loading required package: grid
```

```
## Loading required package: mvtnorm
```

```
## Loading required package: modeltools

## Loading required package: stats4

## Loading required package: strucchange

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich
```

```
library(tidymodels)
```

```
## Registered S3 method overwritten by 'cli':
##   method     from
##   print.tree tree

## -- Attaching packages -------------------------------------- tidymodels 0.1.2 --

## v broom     0.7.5      v recipes   0.1.15
## v dials     0.0.9      v rsample   0.0.9
## v dplyr     1.0.4      v tibble    3.0.6
## v infer     0.5.4      v tidyr     1.1.2
## v modeldata 0.1.0      v tune      0.1.2
## v parsnip   0.1.5      v workflows 0.2.1
## v purrr     0.3.4      v yardstick 0.0.7

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x dplyr::between()         masks data.table::between()
## x dplyr::combine()         masks randomForest::combine()
## x purrr::discard()         masks scales::discard()
## x tidyr::expand()          masks Matrix::expand()
## x dplyr::filter()          masks stats::filter()
## x dplyr::first()           masks data.table::first()
## x parsnip::fit()           masks party::fit(), modeltools::fit()
## x dplyr::lag()             masks stats::lag()
## x dplyr::last()            masks data.table::last()
## x purrr::lift()            masks caret::lift()
## x randomForest::margin()   masks ggplot2::margin()
## x tidyr::pack()            masks Matrix::pack()
## x tune::parameters()       masks dials::parameters(), modeltools::parameters()
## x rsample::permutations()  masks e1071::permutations()
## x yardstick::precision()   masks caret::precision()
## x dials::prune()           masks rpart::prune()
## x yardstick::recall()      masks caret::recall()
```

```
## x dplyr::select()          masks MASS::select()
## x yardstick::sensitivity() masks caret::sensitivity()
## x yardstick::specificity() masks caret::specificity()
## x recipes::step()          masks stats::step()
## x purrr::transpose()       masks data.table::transpose()
## x tune::tune()             masks e1071::tune()
## x tidyr::unpack()          masks Matrix::unpack()
## x recipes::update()        masks stats4::update(), Matrix::update(), stats::update()
```

```
library(caTools)
```

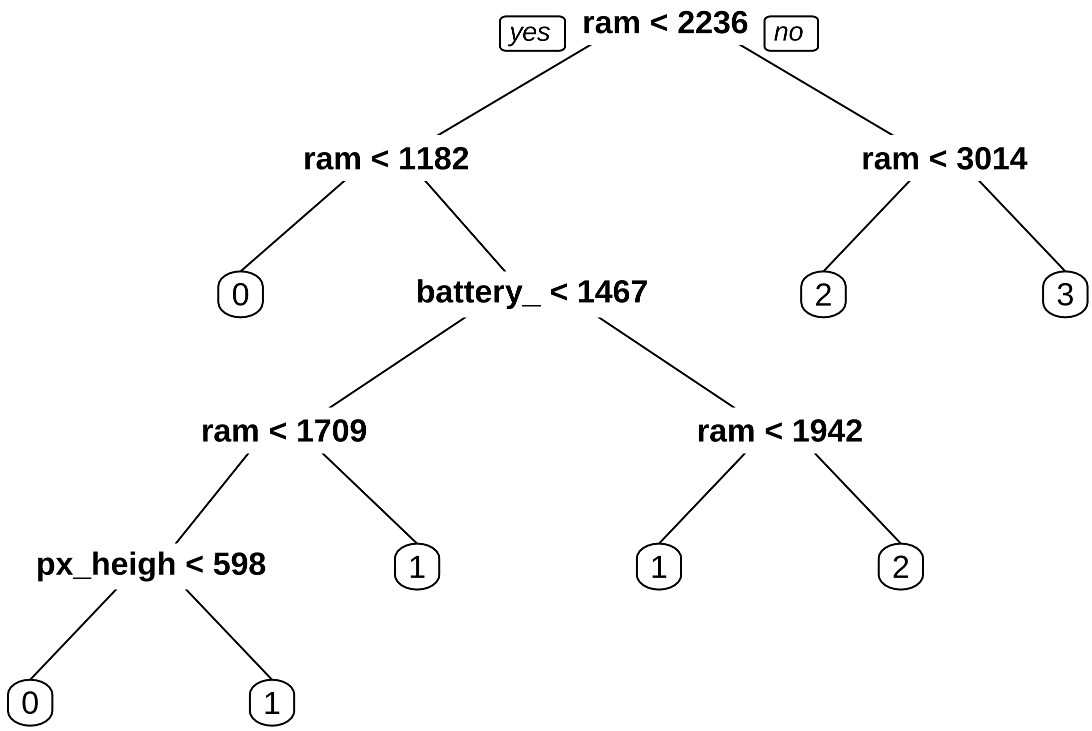
```
data <- fread("C:/Users/boli0/Downloads/train.csv")
str(data)
```

```
## Classes 'data.table' and 'data.frame':   2000 obs. of  21 variables:
##  $ battery_power: int  842 1021 563 615 1821 1859 1821 1954 1445 509 ...
##  $ blue         : int  0 1 1 1 1 0 0 0 1 1 ...
##  $ clock_speed  : num  2.2 0.5 0.5 2.5 1.2 0.5 1.7 0.5 0.5 0.6 ...
##  $ dual_sim     : int  0 1 1 0 0 1 0 1 0 1 ...
##  $ fc           : int  1 0 2 0 13 3 4 0 0 2 ...
##  $ four_g       : int  0 1 1 0 1 0 1 0 0 1 ...
##  $ int_memory   : int  7 53 41 10 44 22 10 24 53 9 ...
##  $ m_dep        : num  0.6 0.7 0.9 0.8 0.6 0.7 0.8 0.8 0.7 0.1 ...
##  $ mobile_wt    : int  188 136 145 131 141 164 139 187 174 93 ...
##  $ n_cores      : int  2 3 5 6 2 1 8 4 7 5 ...
##  $ pc           : int  2 6 6 9 14 7 10 0 14 15 ...
##  $ px_height    : int  20 905 1263 1216 1208 1004 381 512 386 1137 ...
##  $ px_width     : int  756 1988 1716 1786 1212 1654 1018 1149 836 1224 ...
##  $ ram          : int  2549 2631 2603 2769 1411 1067 3220 700 1099 513 ...
##  $ sc_h         : int  9 17 11 16 8 17 13 16 17 19 ...
##  $ sc_w         : int  7 3 2 8 2 1 8 3 1 10 ...
##  $ talk_time    : int  19 7 9 11 15 10 18 5 20 12 ...
##  $ three_g      : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ touch_screen : int  0 1 1 0 1 0 0 1 0 0 ...
##  $ wifi         : int  1 0 0 0 0 0 1 1 0 0 ...
##  $ price_range  : int  1 2 2 2 1 1 3 0 0 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
data$price_range <- as.factor(data$price_range)
```

```
set.seed(810)
split = sample.split(data$price_range, SplitRatio = 0.7)
data_train = subset(data, split == TRUE)
data_test = subset(data, split == FALSE)
y_test <- data_test[,price_range]
```

```
# 1-1. Build single classification decision tree

fit = rpart(price_range ~ .,method = "class", data = data_train,control = rpart.control(minsplit = 1) ,

prp(fit)
```




```r
print(fit)
```

```
## n= 1400
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1400 1050 0 (0.250000000 0.250000000 0.250000000 0.250000000)
##    2) ram< 2235.5 727  377 0 (0.481430536 0.416781293 0.101788171 0.000000000)
##      4) ram< 1182 337   41 0 (0.878338279 0.121661721 0.000000000 0.000000000) *
##      5) ram>=1182 390  128 1 (0.138461538 0.671794872 0.189743590 0.000000000)
##       10) battery_power< 1466.5 249   68 1 (0.216867470 0.726907631 0.056224900 0.000000000)
##         20) ram< 1708.5 123   53 1 (0.430894309 0.569105691 0.000000000 0.000000000)
##           40) px_height< 598 61   17 0 (0.721311475 0.278688525 0.000000000 0.000000000) *
##           41) px_height>=598 62    9 1 (0.145161290 0.854838710 0.000000000 0.000000000) *
##         21) ram>=1708.5 126   15 1 (0.007936508 0.880952381 0.111111111 0.000000000) *
##       11) battery_power>=1466.5 141   60 1 (0.000000000 0.574468085 0.425531915 0.000000000)
##         22) ram< 1941.5 110   30 1 (0.000000000 0.727272727 0.272727273 0.000000000) *
##         23) ram>=1941.5 31    1 2 (0.000000000 0.032258065 0.967741935 0.000000000) *
##    3) ram>=2235.5 673  323 3 (0.000000000 0.069836553 0.410104012 0.520059435)
##      6) ram< 3013.5 318   98 2 (0.000000000 0.147798742 0.691823899 0.160377358) *
##      7) ram>=3013.5 355   56 3 (0.000000000 0.000000000 0.157746479 0.842253521) *
```

```r
summary(fit)
```

```
## Call:
## rpart(formula = price_range ~ ., data = data_train, method = "class",
##     parms = list(split = "information"), control = rpart.control(minsplit = 1))
##   n= 1400
##
##           CP nsplit rel error    xerror       xstd
## 1 0.33333333      0 1.0000000 1.0495238 0.01458632
## 2 0.19809524      1 0.6666667 0.6676190 0.01781740
## 3 0.16095238      2 0.4685714 0.4771429 0.01708226
## 4 0.01380952      3 0.3076190 0.3257143 0.01531097
## 5 0.01285714      5 0.2800000 0.3038095 0.01494703
## 6 0.01000000      7 0.2542857 0.2752381 0.01442290
##
## Variable importance
##           ram battery_power     px_height      px_width          sc_w
##            78             7             5             2             2
##    int_memory     mobile_wt            fc            pc      dual_sim
##             2             1             1             1             1
##
## Node number 1: 1400 observations,    complexity param=0.3333333
##   predicted class=0  expected loss=0.75  P(node) =1
##     class counts:   350   350   350   350
##    probabilities: 0.250 0.250 0.250 0.250
##   left son=2 (727 obs) right son=3 (673 obs)
##   Primary splits:
##       ram           < 2235.5 to the left,  improve=650.760600, (0 missing)
##       battery_power < 1332.5 to the left,  improve= 37.065280, (0 missing)
##       px_width      < 1630.5 to the left,  improve= 25.439830, (0 missing)
##       px_height     < 1212   to the left,  improve= 18.147350, (0 missing)
##       mobile_wt     < 104.5  to the left,  improve=  9.566532, (0 missing)
##   Surrogate splits:
##       px_height     < 280.5  to the right, agree=0.549, adj=0.062, (0 split)
##       battery_power < 1721.5 to the left,  agree=0.534, adj=0.031, (0 split)
##       sc_w          < 10.5   to the left,  agree=0.534, adj=0.031, (0 split)
##       fc            < 13.5   to the left,  agree=0.528, adj=0.018, (0 split)
##       int_memory    < 42.5   to the left,  agree=0.528, adj=0.018, (0 split)
##
## Node number 2: 727 observations,    complexity param=0.1980952
##   predicted class=0  expected loss=0.5185695  P(node) =0.5192857
##     class counts:   350   303    74     0
##    probabilities: 0.481 0.417 0.102 0.000
##   left son=4 (337 obs) right son=5 (390 obs)
##   Primary splits:
##       ram           < 1182   to the left,  improve=231.36100, (0 missing)
##       battery_power < 1455   to the left,  improve= 47.93258, (0 missing)
##       px_height     < 639.5  to the left,  improve= 33.61836, (0 missing)
##       px_width      < 1144.5 to the left,  improve= 29.33494, (0 missing)
##       mobile_wt     < 186.5  to the left,  improve=  4.38501, (0 missing)
##   Surrogate splits:
##       px_width   < 684.5  to the left,  agree=0.567, adj=0.065, (0 split)
##       pc         < 1.5    to the left,  agree=0.557, adj=0.045, (0 split)
##       mobile_wt  < 100.5  to the left,  agree=0.556, adj=0.042, (0 split)
##       px_height  < 286.5  to the left,  agree=0.554, adj=0.039, (0 split)
##       int_memory < 6.5    to the left,  agree=0.550, adj=0.030, (0 split)
```

```
##
## Node number 3: 673 observations,    complexity param=0.1609524
##   predicted class=3  expected loss=0.4799406  P(node) =0.4807143
##     class counts:     0    47   276   350
##    probabilities: 0.000 0.070 0.410 0.520
##   left son=6 (318 obs) right son=7 (355 obs)
##   Primary splits:
##       ram           < 3013.5 to the left,  improve=180.93670, (0 missing)
##       battery_power < 1352.5 to the left,  improve= 46.75949, (0 missing)
##       px_width      < 1283   to the left,  improve= 31.47901, (0 missing)
##       px_height     < 955    to the left,  improve= 24.86811, (0 missing)
##       int_memory    < 10.5   to the left,  improve=  7.02468, (0 missing)
##   Surrogate splits:
##       battery_power < 589    to the left,  agree=0.548, adj=0.044, (0 split)
##       sc_h          < 18.5   to the right, agree=0.544, adj=0.035, (0 split)
##       int_memory    < 4.5    to the left,  agree=0.541, adj=0.028, (0 split)
##       px_width      < 1074   to the left,  agree=0.541, adj=0.028, (0 split)
##       dual_sim      < 0.5    to the left,  agree=0.536, adj=0.019, (0 split)
##
## Node number 4: 337 observations
##   predicted class=0  expected loss=0.1216617  P(node) =0.2407143
##     class counts:   296    41     0     0
##    probabilities: 0.878 0.122 0.000 0.000
##
## Node number 5: 390 observations,    complexity param=0.01380952
##   predicted class=1  expected loss=0.3282051  P(node) =0.2785714
##     class counts:    54   262    74     0
##    probabilities: 0.138 0.672 0.190 0.000
##   left son=10 (249 obs) right son=11 (141 obs)
##   Primary splits:
##       battery_power < 1466.5 to the left,  improve=57.255060, (0 missing)
##       ram           < 1508.5 to the left,  improve=47.743900, (0 missing)
##       px_height     < 674.5  to the left,  improve=31.980610, (0 missing)
##       px_width      < 1113.5 to the left,  improve=29.405230, (0 missing)
##       n_cores       < 4.5    to the left,  improve= 3.540274, (0 missing)
##   Surrogate splits:
##       px_height < 1639.5 to the left,  agree=0.649, adj=0.028, (0 split)
##       talk_time < 3.5    to the right, agree=0.646, adj=0.021, (0 split)
##       px_width  < 530.5  to the right, agree=0.644, adj=0.014, (0 split)
##       ram       < 1203.5 to the right, agree=0.641, adj=0.007, (0 split)
##
## Node number 6: 318 observations
##   predicted class=2  expected loss=0.3081761  P(node) =0.2271429
##     class counts:     0    47   220    51
##    probabilities: 0.000 0.148 0.692 0.160
##
## Node number 7: 355 observations
##   predicted class=3  expected loss=0.1577465  P(node) =0.2535714
##     class counts:     0     0    56   299
##    probabilities: 0.000 0.000 0.158 0.842
##
## Node number 10: 249 observations,    complexity param=0.01285714
##   predicted class=1  expected loss=0.2730924  P(node) =0.1778571
##     class counts:    54   181    14     0
```

```
##    probabilities: 0.217 0.727 0.056 0.000
##   left son=20 (123 obs) right son=21 (126 obs)
##   Primary splits:
##       ram           < 1708.5 to the left,  improve=46.820460, (0 missing)
##       px_width      < 1479.5 to the left,  improve=24.350920, (0 missing)
##       px_height     < 736    to the left,  improve=20.883800, (0 missing)
##       battery_power < 1027.5 to the left,  improve=15.672300, (0 missing)
##       sc_h          < 11.5   to the right, improve= 6.621616, (0 missing)
##   Surrogate splits:
##       sc_w          < 4.5    to the left,  agree=0.574, adj=0.138, (0 split)
##       px_width      < 1779   to the right, agree=0.570, adj=0.130, (0 split)
##       battery_power < 558    to the left,  agree=0.558, adj=0.106, (0 split)
##       blue          < 0.5    to the left,  agree=0.554, adj=0.098, (0 split)
##       mobile_wt     < 161.5  to the right, agree=0.554, adj=0.098, (0 split)
##
## Node number 11: 141 observations,    complexity param=0.01380952
##   predicted class=1  expected loss=0.4255319  P(node) =0.1007143
##     class counts:     0    81    60     0
##    probabilities: 0.000 0.574 0.426 0.000
##   left son=22 (110 obs) right son=23 (31 obs)
##   Primary splits:
##       ram           < 1941.5 to the left,  improve=27.291620, (0 missing)
##       px_height     < 696    to the left,  improve=13.931310, (0 missing)
##       px_width      < 1240   to the left,  improve=12.786660, (0 missing)
##       battery_power < 1990   to the left,  improve= 2.607385, (0 missing)
##       int_memory    < 47.5   to the left,  improve= 2.351360, (0 missing)
##
## Node number 20: 123 observations,    complexity param=0.01285714
##   predicted class=1  expected loss=0.4308943  P(node) =0.08785714
##     class counts:    53    70     0     0
##    probabilities: 0.431 0.569 0.000 0.000
##   left son=40 (61 obs) right son=41 (62 obs)
##   Primary splits:
##       px_height     < 598    to the left,  improve=22.302360, (0 missing)
##       px_width      < 994.5  to the left,  improve=19.697840, (0 missing)
##       battery_power < 1027.5 to the left,  improve=19.114670, (0 missing)
##       ram           < 1515.5 to the left,  improve= 5.609121, (0 missing)
##       pc            < 3.5    to the left,  improve= 4.698548, (0 missing)
##   Surrogate splits:
##       px_width      < 1085   to the left,  agree=0.667, adj=0.328, (0 split)
##       battery_power < 919    to the left,  agree=0.626, adj=0.246, (0 split)
##       clock_speed   < 1.65   to the right, agree=0.610, adj=0.213, (0 split)
##       dual_sim      < 0.5    to the right, agree=0.593, adj=0.180, (0 split)
##       four_g        < 0.5    to the right, agree=0.585, adj=0.164, (0 split)
##
## Node number 21: 126 observations
##   predicted class=1  expected loss=0.1190476  P(node) =0.09
##     class counts:     1   111    14     0
##    probabilities: 0.008 0.881 0.111 0.000
##
## Node number 22: 110 observations
##   predicted class=1  expected loss=0.2727273  P(node) =0.07857143
##     class counts:     0    80    30     0
##    probabilities: 0.000 0.727 0.273 0.000
```

```
##
## Node number 23: 31 observations
##   predicted class=2  expected loss=0.03225806  P(node) =0.02214286
##     class counts:     0     1    30     0
##    probabilities: 0.000 0.032 0.968 0.000
##
## Node number 40: 61 observations
##   predicted class=0  expected loss=0.2786885  P(node) =0.04357143
##     class counts:    44    17     0     0
##    probabilities: 0.721 0.279 0.000 0.000
##
## Node number 41: 62 observations
##   predicted class=1  expected loss=0.1451613  P(node) =0.04428571
##     class counts:     9    53     0     0
##    probabilities: 0.145 0.855 0.000 0.000
```

```
# 1-2. Single classification tree's confusion matrix and accuracy score
# Accuracy score is 0.775.
fit.pred = predict(fit, newdata = data_test, type = "class")

cm <- table(observed = y_test, predicted = fit.pred)

cm
```

```
##         predicted
## observed   0   1   2   3
##        0 139  11   0   0
##        1  33  99  18   0
##        2   0  24  94  32
##        3   0   0  17 133
```

```
test_accuary <- mean(fit.pred == y_test)

test_accuary
```

```
## [1] 0.775
```

```
# 2-1. Build decision tree while cp = 0.01000000
# Accuracy score is 0.775.
fit_cp = rpart(price_range ~ .,method = "class", data = data_train,control = rpart.control(minsplit = 1

fit_cp.pred = predict(fit_cp, newdata = data_test, type = "class")

test_accuary_cp <- mean(fit_cp.pred == y_test)

test_accuary_cp
```

```
## [1] 0.775
```

```
# 3-1. Bagging and random forest
set.seed(810)
```

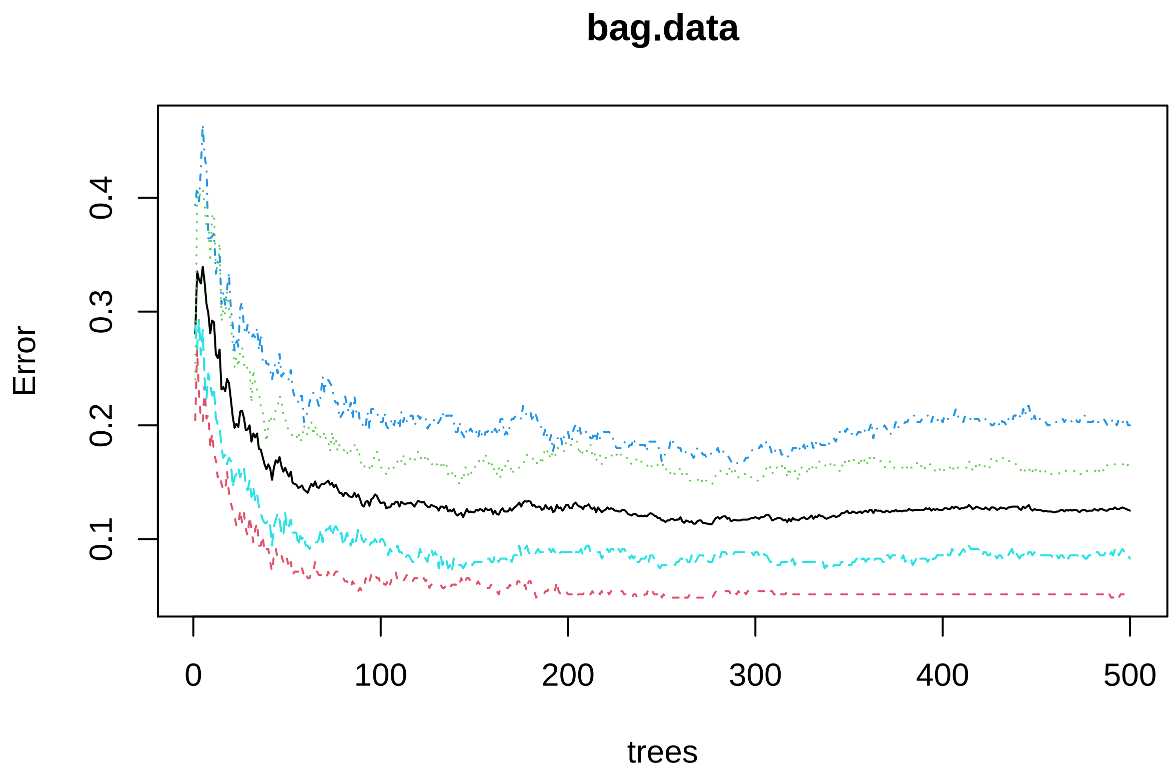
```r
bag.data <- randomForest(price_range ~., data = data_train, mytry = 20, importance = TRUE, proximity = T

print(bag.data)
```

```
##
## Call:
##  randomForest(formula = price_range ~ ., data = data_train, mytry = 20,      importance = TRUE, prox
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 12.5%
## Confusion matrix:
##     0   1   2   3 class.error
## 0 332  18   0   0  0.05142857
## 1  31 292  27   0  0.16571429
## 2   0  38 280  32  0.20000000
## 3   0   0  29 321  0.08285714
```

```r
summary(bag.data)
```

```
##                 Length  Class  Mode
## call                  6 -none- call
## type                  1 -none- character
## predicted          1400 factor numeric
## err.rate           2500 -none- numeric
## confusion            20 -none- numeric
## votes              5600 matrix numeric
## oob.times          1400 -none- numeric
## classes               4 -none- character
## importance          120 -none- numeric
## importanceSD        100 -none- numeric
## localImportance       0 -none- NULL
## proximity       1960000 -none- numeric
## ntree                 1 -none- numeric
## mtry                  1 -none- numeric
## forest               14 -none- list
## y                  1400 factor numeric
## test                  0 -none- NULL
## inbag                 0 -none- NULL
## terms                 3 terms  call
```

```r
plot(bag.data)
```




```r
bag.pred = predict(bag.data, newdata = data_test, type = "class")

test_accuary_bag <- mean(bag.pred == y_test)

test_accuary_bag
```

```
## [1] 0.8766667
```

```r
# 3-2. Random Forest using sqrt(p)
set.seed(810)

rFM.data <- randomForest(price_range ~., data = data_train, mytry = sqrt(20), importance = TRUE, proxim

print(rFM.data)
```

```
##
## Call:
##  randomForest(formula = price_range ~ ., data = data_train, mytry = sqrt(20),      importance = TRUE
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 12.5%
## Confusion matrix:
```

```
##     0   1   2   3 class.error
## 0 332  18   0   0  0.05142857
## 1  31 292  27   0  0.16571429
## 2   0  38 280  32  0.20000000
## 3   0   0  29 321  0.08285714
```

```
rFM.pred = predict(rFM.data, newdata = data_test, type = "class")

test_accuary_rFM <- mean(rFM.pred == y_test)

test_accuary_rFM
```

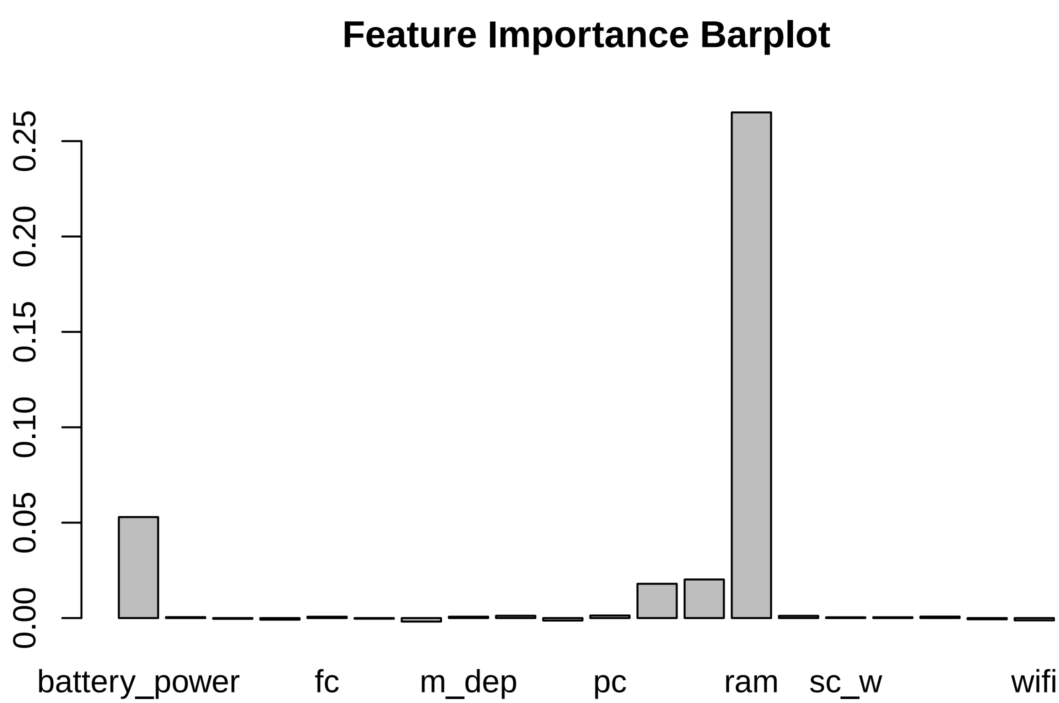
```
## [1] 0.8766667
```

```
# 3-3. Random Forest Feature Importance Chart
importance(rFM.data)
```

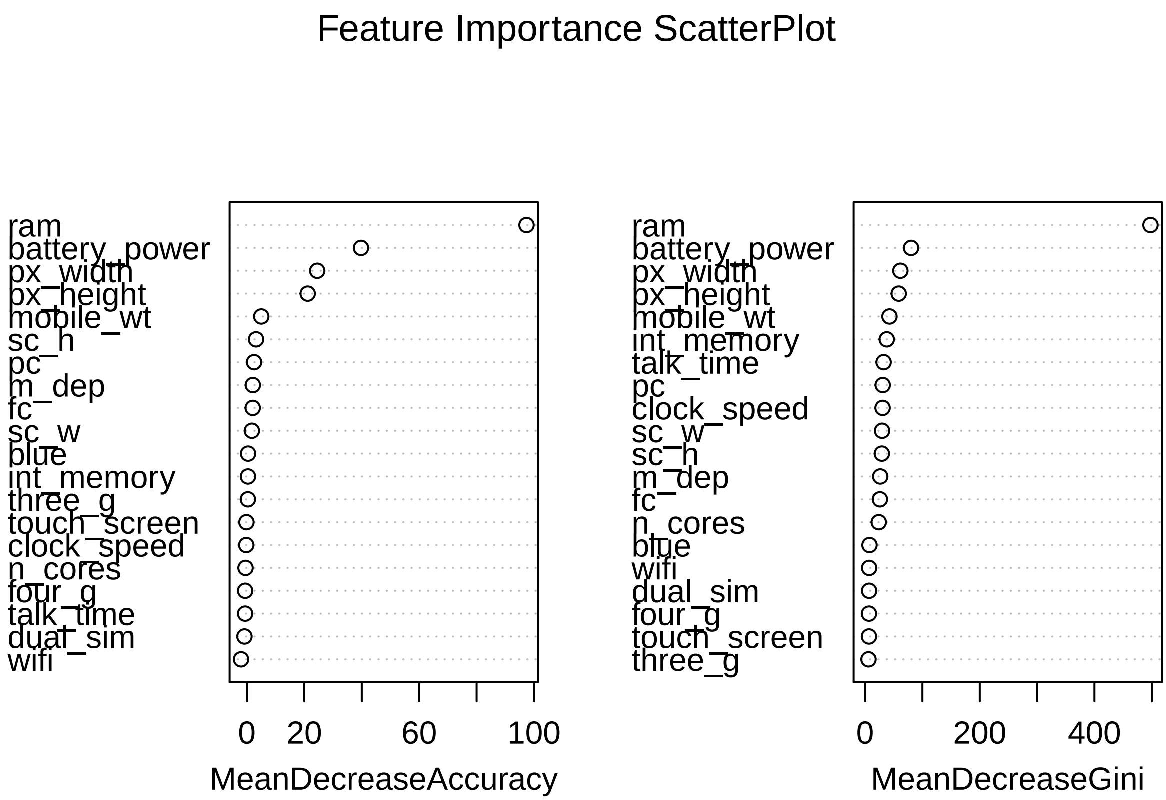
```
##                        0          1          2          3
## battery_power  20.315411783 27.2379966 26.4401697 21.5151653
## blue           -0.773113710  0.8468215 -0.7073670  1.4176344
## clock_speed     0.533121152 -0.3671219 -0.7728101  0.4665967
## dual_sim        0.595019450 -1.5371078 -0.7097531  0.1566308
## fc              2.071522629  0.7702305  0.2715627  1.3583059
## four_g          0.003579781 -0.7306292 -0.8808096  0.4838791
## int_memory      0.973175286 -1.7237171 -0.2883716  2.0023658
## m_dep          -0.336016603  0.7072096  2.5705845  0.9633760
## mobile_wt       2.920799362  1.0412852  2.8427143  3.3301351
## n_cores         0.472187817 -1.4146974  0.8289692 -0.8697330
## pc              0.502259046  1.3638927  1.0315031  2.0575466
## px_height      13.119968698 11.7476663 11.5917891  4.8043727
## px_width       13.093223962 12.5467641 12.1803913 14.2371085
## ram           102.087642597 66.7879289 66.1242760 88.9441201
## sc_h           -0.927416275  1.1550124  1.0891801  4.7737409
## sc_w            1.803773936  0.4275465  1.0901982  0.2938217
## talk_time       1.171421641  0.3951997 -1.7808425 -0.9969937
## three_g         0.596874983  1.9031053 -0.7952348 -0.7090051
## touch_screen    1.512810378 -1.2257074 -0.6652268  0.4574472
## wifi           -0.399894416 -2.3434997 -0.9525469  0.1612621
##               MeanDecreaseAccuracy MeanDecreaseGini
## battery_power           39.7358388        80.225757
## blue                     0.4227329         7.758244
## clock_speed             -0.2034133        30.559829
## dual_sim                -0.8404799         7.088138
## fc                       2.0314417        25.895901
## four_g                  -0.6001848         6.831351
## int_memory               0.3850182        37.835172
## m_dep                    2.0599851        26.501218
## mobile_wt                5.0122442        42.575186
## n_cores                 -0.4607087        23.864487
## pc                       2.5056751        30.772854
## px_height               21.1807316        58.704324
## px_width                24.4945606        61.587355
## ram                     97.3703194       497.722544
```

```
## sc_h                      3.1999096        29.263808
## sc_w                      1.7398295        29.691844
## talk_time                -0.6067494        32.351650
## three_g                   0.3531520         6.014255
## touch_screen             -0.1479079         6.822819
## wifi                     -2.0292120         7.140585
```

```r
# 3-4. Random Forest Feature Importance Barplot
barplot(rFM.data$importance[,2], main = "Feature Importance Barplot")
```




```r
# 3-5. Random Forest Feature Importance ScatterPlot
varImpPlot(rFM.data, sort = TRUE, n.var = nrow(rFM.data$importance), main = "Feature Importance Scatterl
```

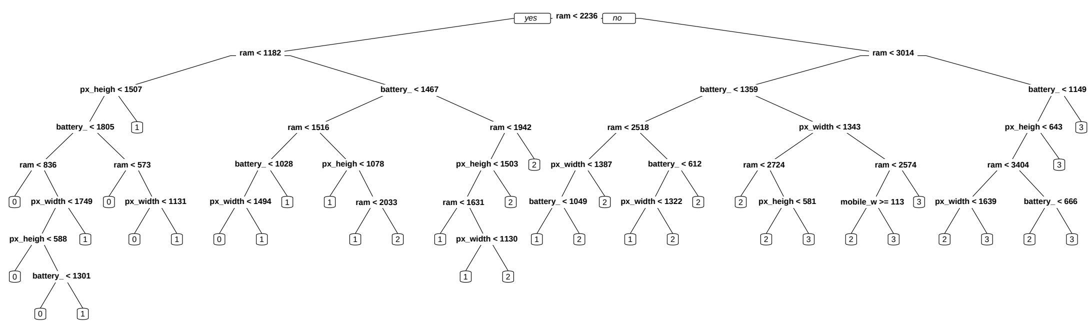



```
# 4-1. Classification trees cross-validation
fit = rpart(price_range ~ .,method = "class", data = data_train,control = rpart.control(minsplit = 1) ,

tr.control = trainControl(method = "cv", number = 100)
cp.grid = expand.grid(.cp = (0:10)*0.01)
tr = train(price_range ~., data = data_train, method = "rpart", trControl = tr.control, tuneGrid = cp.g
tr
```

```
## CART
##
## 1400 samples
##   20 predictor
##    4 classes: '0', '1', '2', '3'
##
## No pre-processing
## Resampling: Cross-Validated (100 fold)
## Summary of sample sizes: 1385, 1386, 1384, 1387, 1388, 1386, ...
## Resampling results across tuning parameters:
##
##   cp    Accuracy   Kappa
##   0.00  0.8583663  0.8105304
##   0.01  0.7794895  0.7048383
##   0.02  0.7628736  0.6828909
##   0.03  0.7606758  0.6798834
##   0.04  0.7606758  0.6798834
##   0.05  0.7606758  0.6798834
```

```
##    0.06  0.7606758  0.6798834
##    0.07  0.7606758  0.6798834
##    0.08  0.7606758  0.6798834
##    0.09  0.7606758  0.6798834
##    0.10  0.7606758  0.6798834
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.
```

```r
# 4-2. Plot best tree
best.tree = tr$finalModel
prp(best.tree)
```

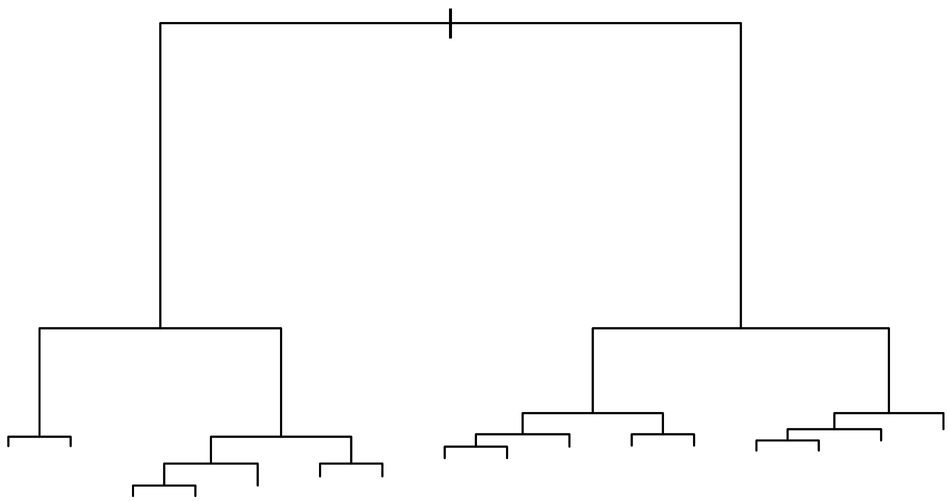



```r
# 4-3. Best tree accuracy score is
best.tree.pred = predict(best.tree, newdata = data_test)
test_accuary_cv <- mean(best.tree.pred == y_test)

test_accuary_cv
```

```
## [1] 0.1408333
```

```r
# 5. Classification tree, textbook method
tree.data = tree(price_range ~., data = data_train)
```

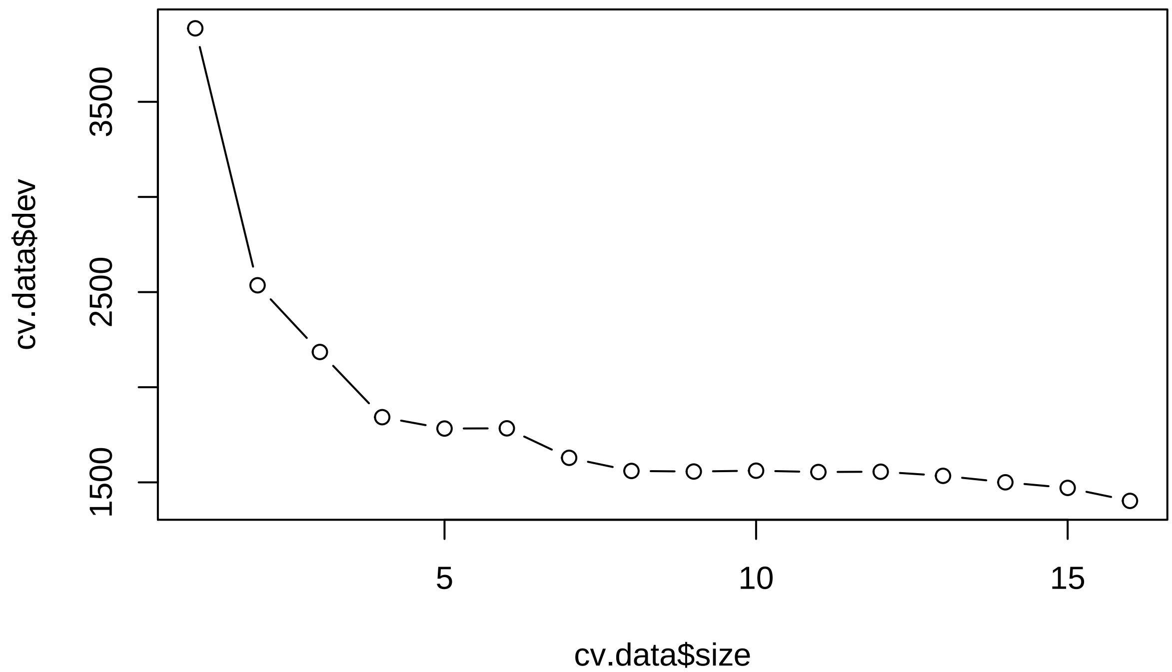
```r
plot(tree.data)
```



```r
tree.pred = predict(tree.data, data_test, type = "class")

table(tree.pred, data_test$price_range)
```

```
##
## tree.pred   0   1   2   3
##         0 139  33   0   0
##         1  11 111  41   0
##         2   0   6  76  11
##         3   0   0  33 139
```

```r
set.seed(810)

cv.data = cv.tree(tree.data)

plot(cv.data$size, cv.data$dev, type = "b")
```



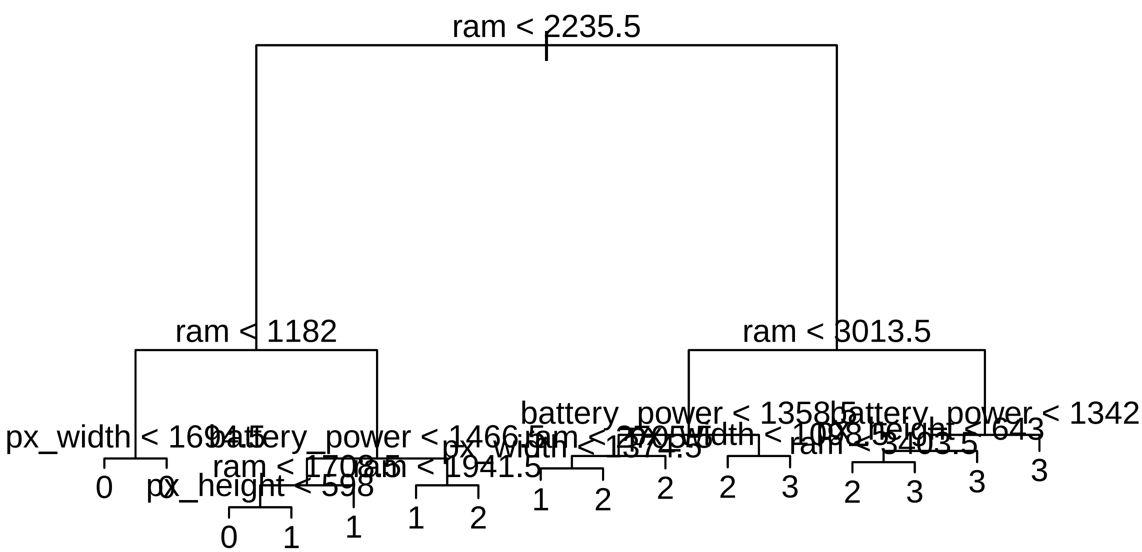

```r
cv.data
```

```
## $size
##  [1] 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
##
## $dev
##  [1] 1402.683 1470.788 1500.265 1534.336 1555.897 1554.345 1561.259 1556.844
##  [9] 1559.950 1628.919 1783.879 1782.755 1842.543 2185.217 2535.742 3886.251
##
## $k
##  [1]       -Inf   40.06300   43.23335   44.60473   48.00002   48.44745
##  [7]   49.03944   52.90116   54.58324   68.65486   90.19778   93.64093
## [13]  114.51011  361.87340  462.72196 1301.52118
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
prune.data = prune.misclass(tree.data, best = 16)

tree.pred = predict(prune.data, data_test, type = "class")

table(tree.pred, data_test$price_range)
```

```
##
## tree.pred   0   1   2   3
##         0 139  33   0   0
##         1  11 111  41   0
##         2   0   6  76  11
##         3   0   0  33 139
```

```
plot(prune.data);text(prune.data, pretty = 0)
```




```
# 6. Boosting
boost_spec <- boost_tree(
mode = "classification",
tree_depth = 4,
trees = 5000,
learn_rate = 0.01,
)%>%
set_engine("xgboost")

boost_spec
```

```
## Boosted Tree Model Specification (classification)
##
## Main Arguments:
##   trees = 5000
##   tree_depth = 4
```

```
##   learn_rate = 0.01
##
## Computational engine: xgboost
```