grid_rf

syh

June 8, 2017

```
# import grid data
grid_data <- read.csv(file = "data/grid_all.csv", stringsAsFactors = F)[,-1]
```

```
# have a look at them
str(grid_data)
```

```
## 'data.frame':    6970 obs. of  6 variables:
##  $ date : int  20160618 20160618 20160618 20160618 20160618 20160618 20160618 20160618 20160618 2016
##  $ hour : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ grid : int  190 191 193 194 195 196 197 198 216 217 ...
##  $ rrate: num  0.705 0.556 0.333 0.462 0.507 0.528 0.486 0.029 0.61 0.828 ...
##  $ speed: num  8.2 6.29 4.97 5.42 3.83 ...
##  $ demo : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
head(grid_data)
```

```
##       date hour grid rrate speed demo
## 1 20160618    0  190 0.705 8.195    0
## 2 20160618    0  191 0.556 6.292    0
## 3 20160618    0  193 0.333 4.969    0
## 4 20160618    0  194 0.462 5.424    0
## 5 20160618    0  195 0.507 3.831    0
## 6 20160618    0  196 0.528 3.294    0
```

```
summary(grid_data)
```

```
##       date                hour            grid           rrate
##  Min.   :20160618   Min.   : 0.00   Min.   :190.0   Min.   :0.000
##  1st Qu.:20161116   1st Qu.: 5.00   1st Qu.:223.0   1st Qu.:0.328
##  Median :20161126   Median :11.00   Median :274.0   Median :0.492
##  Mean   :20161045   Mean   :11.09   Mean   :277.1   Mean   :0.474
##  3rd Qu.:20161202   3rd Qu.:17.00   3rd Qu.:324.0   3rd Qu.:0.635
##  Max.   :20161203   Max.   :23.00   Max.   :380.0   Max.   :1.000
##      speed             demo
##  Min.   : 0.090   Min.   :0.00000
##  1st Qu.: 2.901   1st Qu.:0.00000
##  Median : 3.993   Median :0.00000
##  Mean   : 4.553   Mean   :0.03199
##  3rd Qu.: 5.698   3rd Qu.:0.00000
##  Max.   :17.021   Max.   :1.00000
```

```
# check if there is missing value
# sapply(grid_data, function(x){sum(is.na(x))})
```

```
# let's remove date column becasue we can't use date to predict if there is a demo
# But we can discuss if there is relations between demo and special day
# such as holiday. If so we can make a new catergocial feature by date
grid_data <- subset(x = grid_data, select = -date)
```

```r
# let's convert , hour, grid, demo to factors

grid_data[,"hour"] <- as.factor(grid_data[,"hour"])
grid_data[,"grid"] <- as.factor(grid_data[,"grid"])
grid_data[,"demo"] <- as.factor(grid_data[,"demo"])
```

```r
# as we know that the data is seriously imbalanced
# so we need to deal with this problem by ROSE
library(ROSE)
```

```
## Loaded ROSE 0.0-3
```

```r
grid_data <- ROSE(demo ~ ., data = grid_data, p = 0.5, seed = 1)$data
table(grid_data$demo)
```

```
##
##    0    1
## 3503 3467
```

```r
summary(grid_data)
```

```
##       hour           grid          rrate              speed        demo
##  18     :1092   222    : 207   Min.   :-0.3278   Min.   :-1.090   0:3503
##  19     : 955   242    : 191   1st Qu.: 0.3429   1st Qu.: 2.485   1:3467
##  21     : 953   190    : 186   Median : 0.5310   Median : 3.608
##  20     : 842   198    : 183   Mean   : 0.5262   Mean   : 4.030
##  3      : 181   191    : 181   3rd Qu.: 0.7164   3rd Qu.: 5.104
##  7      : 180   195    : 181   Max.   : 1.4236   Max.   :16.641
##  (Other):2767   (Other):5841
```

```r
# for hour, I think we can make it like morning, afternoon, evening
# for grid, we can aggregate them into bigger areas
```

```r
# let split data into train and test
set.seed(1)
train_ind <- sample(x = c(1:dim(grid_data)[1]), size = dim(grid_data)[1] * 0.7)

train_data <- grid_data[train_ind,]
test_data <- grid_data[-train_ind,]

# train_x <- train_data[,1:dim(train_data)[2] - 1]
# train_y <- train_data[dim(train_data)[2]]
```

```r
# let's first use rpart to train a decision tr
library(rpart)
tree <- rpart(formula = "demo ~. -demo", data = train_data,
              method = "class", control=rpart.control(cp=0))
```

```r
# look at what the tree is like
tree
```

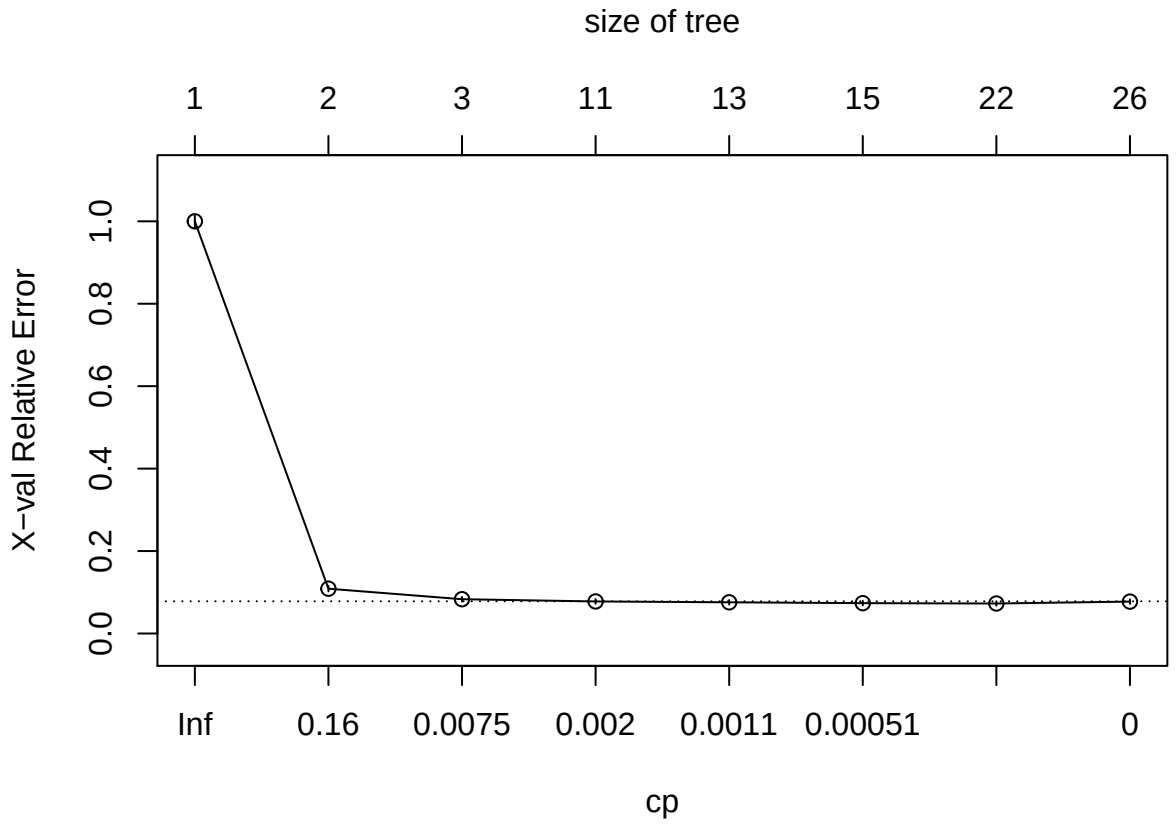
```
## n= 4879
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 4879 2392 0 (0.509735602 0.490264398)
```

```
##      2) hour=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,22,23 2227    0 0 (1.000000000 0.000000000)
##      3) hour=18,19,20,21 2652  260 1 (0.098039216 0.901960784)
##        6) grid=220,246,247,248,269,272,273,300,323,324,325,326,348,349,372,374,375,378 65    0 0 (1.
##        7) grid=190,191,192,193,194,195,196,197,198,216,217,218,221,222,223,224,242,243,244,245,249,25
##         14) grid=243,244,245,250,268,270,271,274,276,294,297,298,299,301,322,327,328,352,353,373 633
##           28) rrate< 0.7854899 344   84 1 (0.244186047 0.755813953)
##             56) grid=245,297,298,299,301,352,353,373 33    9 0 (0.727272727 0.272727273)
##              112) rrate< 0.7188368 24    2 0 (0.916666667 0.083333333) *
##              113) rrate>=0.7188368 9    2 1 (0.222222222 0.777777778) *
##             57) grid=243,244,250,268,270,271,274,276,294,322,327,328 311   60 1 (0.192926045 0.80707
##              114) rrate>=0.5784397 102   40 1 (0.392156863 0.607843137)
##                228) grid=244,250,276,322 16    2 0 (0.875000000 0.125000000) *
##                229) grid=243,268,294,327,328 86   26 1 (0.302325581 0.697674419)
##                  458) hour=19,21 34   17 0 (0.500000000 0.500000000)
##                    916) grid=243,268,327 9    0 0 (1.000000000 0.000000000) *
##                    917) grid=294,328 25    8 1 (0.320000000 0.680000000) *
##                  459) hour=18,20 52    9 1 (0.173076923 0.826923077)
##                    918) speed< 4.297129 24    8 1 (0.333333333 0.666666667)
##                     1836) grid=294,328 9    3 0 (0.666666667 0.333333333) *
##                     1837) grid=243,327 15    2 1 (0.133333333 0.866666667) *
##                    919) speed>=4.297129 28    1 1 (0.035714286 0.964285714) *
##              115) rrate< 0.5784397 209   20 1 (0.095693780 0.904306220)
##                230) grid=268,270,271,274,276 78   18 1 (0.230769231 0.769230769)
##                  460) hour=21 10    0 0 (1.000000000 0.000000000) *
##                  461) hour=18,19,20 68    8 1 (0.117647059 0.882352941) *
##                231) grid=243,244,250,294,322,327,328 131    2 1 (0.015267176 0.984732824) *
##           29) rrate>=0.7854899 289    6 1 (0.020761246 0.979238754) *
##          15) grid=190,191,192,193,194,195,196,197,198,216,217,218,221,222,223,224,242,249,275,295,296
##           30) rrate< 0.7405895 1470   96 1 (0.065306122 0.934693878)
##             60) rrate>=0.2987195 1107   87 1 (0.078590786 0.921409214)
##              120) grid=302,347 19    6 1 (0.315789474 0.684210526) *
##              121) grid=190,191,192,193,194,195,196,197,198,216,217,221,222,223,224,242,249,275,295,29
##                242) speed>=5.035315 116   15 1 (0.129310345 0.870689655)
##                  484) grid=216,217,321 9    2 0 (0.777777778 0.222222222) *
##                  485) grid=190,191,192,193,195,196,221,223,224,242,249,275,295,296,354,380 107    8
##                243) speed< 5.035315 972   66 1 (0.067901235 0.932098765)
##                  486) rrate>=0.7104805 61   10 1 (0.163934426 0.836065574)
##                    972) grid=190,194,195,242,295 9    3 0 (0.666666667 0.333333333) *
##                    973) grid=191,193,196,197,198,216,217,221,222,223,224,249,275,321,354 52    4 1 (
##                  487) rrate< 0.7104805 911   56 1 (0.061470911 0.938529089) *
##             61) rrate< 0.2987195 363    9 1 (0.024793388 0.975206612)
##              122) speed< -0.1777405 7    2 1 (0.285714286 0.714285714) *
##              123) speed>=-0.1777405 356    7 1 (0.019662921 0.980337079)
##                246) hour=21 77    6 1 (0.077922078 0.922077922)
##                  492) grid=295,296,320,347 23    5 1 (0.217391304 0.782608696)
##                    984) rrate>=0.136492 7    3 0 (0.571428571 0.428571429) *
##                    985) rrate< 0.136492 16    1 1 (0.062500000 0.937500000) *
##                  493) grid=190,192,193,195,196,197,198,218,221,222,223,242,346,379 54    1 1 (0.0185
##                247) hour=18,19,20 279    1 1 (0.003584229 0.996415771) *
##           31) rrate>=0.7405895 484    9 1 (0.018595041 0.981404959) *
```

```
# xerror: error in cross validation
# xstd: standard deviation of error in cross vaidation
printcp(tree)
```

```
##
## Classification tree:
## rpart(formula = "demo ~. -demo", data = train_data, method = "class",
##     control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
## [1] grid  hour  rrate speed
##
## Root node error: 2392/4879 = 0.49026
##
## n= 4879
##
##           CP nsplit rel error   xerror      xstd
## 1 0.89130435      0  1.000000 1.000000 0.0145980
## 2 0.02717391      1  0.108696 0.108696 0.0065589
## 3 0.00209030      2  0.081522 0.083194 0.0057759
## 4 0.00188127     10  0.063963 0.077759 0.0055918
## 5 0.00062709     12  0.060201 0.075669 0.0055191
## 6 0.00041806     14  0.058946 0.073579 0.0054452
## 7 0.00010452     21  0.055602 0.072742 0.0054154
## 8 0.00000000     25  0.055184 0.077341 0.0055774
```

```
# let's have a look at complexity parameter against xerror
plotcp(tree)
```




```
# let's find a appropriate cp
cptable <- as.data.frame(tree$cptable)
```

```r
opt_cp <- cptable[with(cptable,which.min(xerror)),"CP"]
```

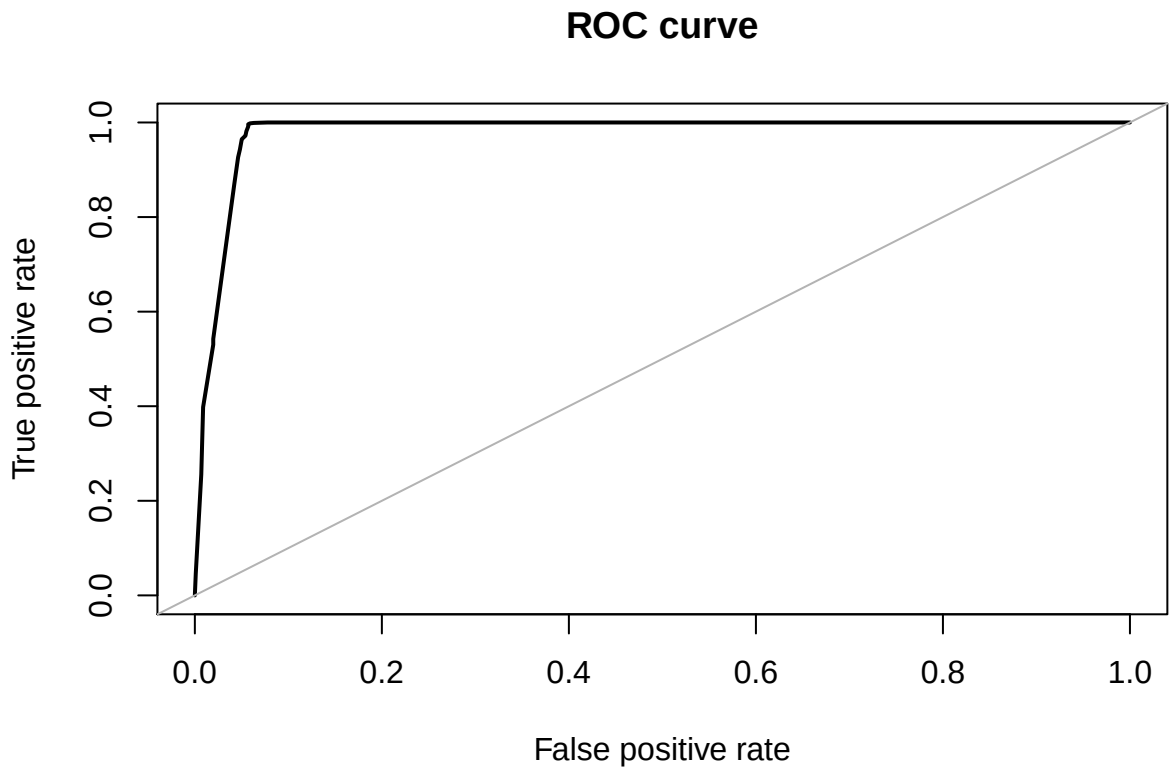
```r
# prune tree
opt_tree <- prune(tree,cp = opt_cp)
```

```r
# let's apply it on test data
est_demo <- predict(object = opt_tree, newdata = test_data)
```

```r
# let's check the performance of opt_tre
accuracy.meas(response = test_data$demo, predicted = est_demo[,2], threshold = 0.5)
```

```
##
## Call:
## accuracy.meas(response = test_data$demo, predicted = est_demo[,
##     2], threshold = 0.5)
##
## Examples are labelled as positive when predicted is greater than 0.5
##
## precision: 0.948
## recall: 0.992
## F: 0.485
```

```r
# let's look at roc, auc
roc.curve(response = test_data$demo, predicted = est_demo[,2], plotit = T)
```




```
## Area under the curve (AUC): 0.979
```

```
# let's plot ROC curve
# library(ROCR)
# pre <- prediction(predictions = est_demo[,2], labels = test_data$demo)

# roc_per <- performance(prediction.obj = pre, measure = "tpr", x.measure = "fpr")

# plot(roc_per, col = "blue")
# abline(a = 0, b = 1)
```

```
# AUC
# auc_perf = performance(pre, measure = "auc")

# paste("AUC is ", auc_perf@y.values)
```

```
# we would like to find the largest sensitivity + specificity with cutoff
# opt.cut = function(perf, pred){
    # cut.ind = mapply(FUN=function(x, y, p){
        # d = (x - 0)^2 + (y-1)^2
        # ind = which(d == min(d))
        # c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            # cutoff = p[[ind]])
    # }, perf@x.values, perf@y.values, pred@cutoffs)
# }
# print(opt.cut(roc_per, pre))
```