homework 04

Rastko Stojsin

March 30, 2020

```
# loading libraries
library('ElemStatLearn')
library('nnet')
library('dplyr')
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library('magrittr')
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:dplyr':
##
##     combine
```

```
library('caret')
```

```
## Loading required package: lattice

## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

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

1. Convert the response variable in the "vowel.train" data frame to a factor variable prior to training, so that "randomForest" does classification rather than regression.

```r
## read vowel training and testing data from HTF website
data(vowel.train)
data(vowel.test)

# seperating predicted and predictor vars in training set
vowel.train.y <- vowel.train[c(1)]
vowel.train.x <- vowel.train[c(-1)]

# changing predicted value as factor to allow classification
vowel.train <- vowel.train %>% mutate_if(is.integer, as.factor)
vowel.test <- vowel.test %>% mutate_if(is.integer, as.factor)
```

3. Fit the random forest model to the vowel data using all of the 11 features using the default values of the tuning parameters.

```r
# default param random forest
vowel.rf.default <- randomForest(vowel.train.x,  as.factor(vowel.train$y))
# show default random forest confusion matrix
vowel.rf.default
```

```
##
## Call:
##  randomForest(x = vowel.train.x, y = as.factor(vowel.train$y))
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 2.65%
## Confusion matrix:
##     1  2  3  4  5  6  7  8  9 10 11 class.error
## 1  48  0  0  0  0  0  0  0  0  0  0  0.00000000
## 2   1 47  0  0  0  0  0  0  0  0  0  0.02083333
## 3   0  0 47  1  0  0  0  0  0  0  0  0.02083333
## 4   0  0  0 47  0  1  0  0  0  0  0  0.02083333
## 5   0  0  0  0 46  1  0  0  0  0  1  0.04166667
## 6   0  0  0  0  0 44  0  0  0  0  4  0.08333333
## 7   0  0  0  0  1  0 45  2  0  0  0  0.06250000
## 8   0  0  0  0  0  0  0 48  0  0  0  0.00000000
## 9   0  0  0  0  0  0  1  0 47  0  0  0.02083333
## 10  0  0  0  0  0  0  1  0  0 47  0  0.02083333
## 11  0  0  0  0  0  0  0  0  0  0 48  0.00000000
```

4. Use 5-fold CV and tune the model by performing a grid search for the following tuning parameters: 1) the number of variables randomly sampled as candidates at each split; consider values 3, 4, and 5, and 2) the minimum size of terminal nodes; consider a sequence (1, 5, 10, 20, 40, and 80).

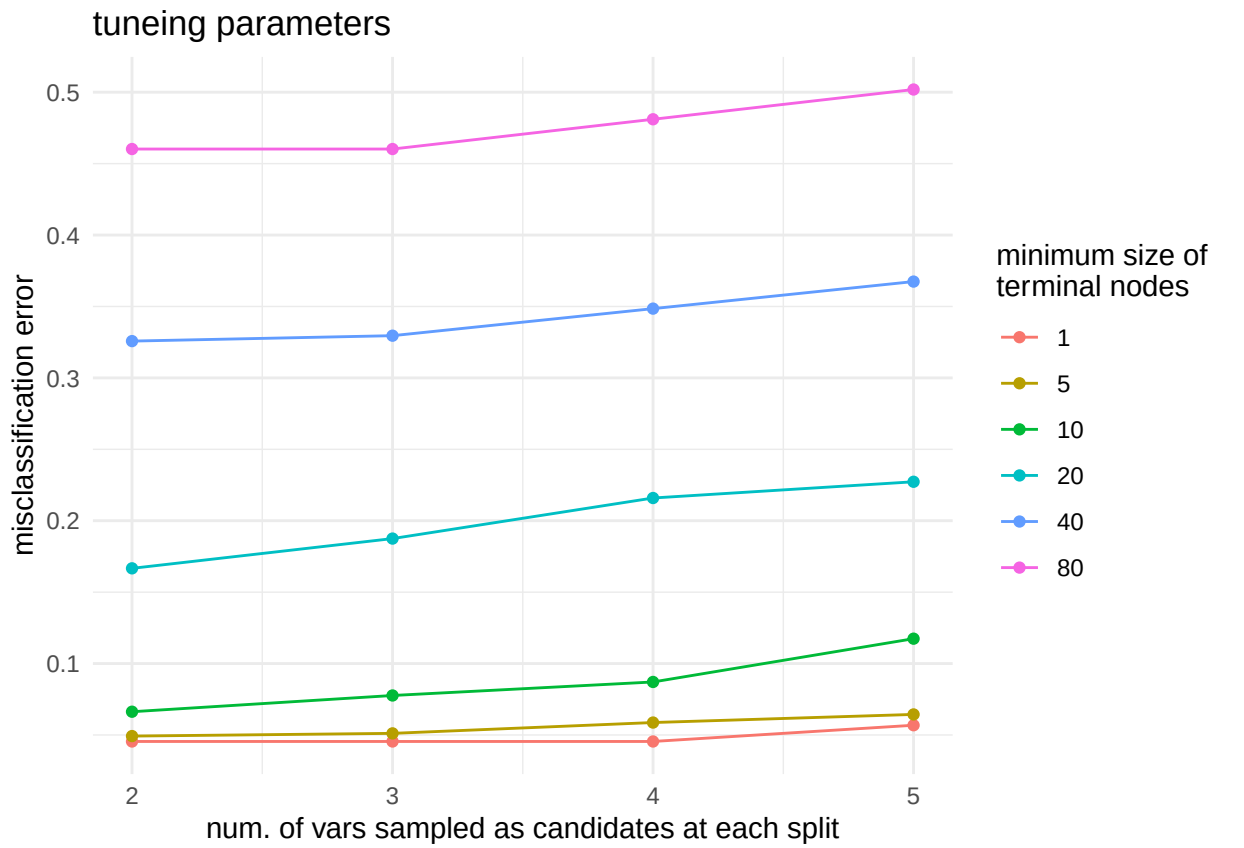
```r
# 5-fold cross-validation setup
# set seed for reproducibility
set.seed(13)
y_flds  <- createFolds(vowel.train$y, k=5)
prediction.5fld <- NA
```

```r
# run cross validation function
cvvowelclass <- function(flds = y_flds, num_par = num_par, nd_size = nd_size) {
  # initialize helper vars
  tst.id <- NA
  tst.output <- NA
  # loop through folds
  for(tst_idx in 1:length(flds)) {
    # sent training and test (validation) sets within folds
    inc_trn <- vowel.train[-flds[[tst_idx]], ]
    inc_tst <- vowel.train[ flds[[tst_idx]], ]

    # isolate inputs in training set
    inc_trn.x <- inc_trn[c(-1)]

    # train random forest on train set
    vowel.rf <- randomForest(inc_trn.x,  as.factor(inc_trn$y),
                             mtry = num_par, nodesize = nd_size)
    # predict y groups in validation set
    pre_test <- predict(vowel.rf, inc_tst)

    # use helper functions to retain data across folds
    # there is for sure a better way to do this thats less clunky
    tst.id <- append(tst.id, as.integer(row.names(as.data.frame(pre_test))))
    tst.output <- append(tst.output, as.data.frame(pre_test)[[1]])
  }
  # put outputs in manageable dataframe and sort
  prediction.5fld <- data.frame(tst.id[-1],tst.output[-1])
  prediction.5fld <- prediction.5fld[order( prediction.5fld[,1] ),]

  # return missclassification error within training set
  return(sum(prediction.5fld[,2] == vowel.train[,1])/nrow(prediction.5fld))
}
```

```r
# seting desired param possibilites
num_vars_rand_smpld <- c(2,3,4,5)
min_size_trmnl_nodes <- c(1,5,10,20,40,80)
# creating grid of possible param combinations
tuning_grid <- expand.grid(num_vars_rand_smpld, min_size_trmnl_nodes, misclass_err = NA_real_)

# loop through rows of grid
for(i in 1:nrow(tuning_grid)){
  # calculating misclassification error for each set of params
  tuning_grid[i,c("misclass_err")] <- 1 - cvvowelclass(num_par = tuning_grid[i, "Var1"],
                                       nd_size = tuning_grid[i,"Var2"])
  # print(i)
}
```

```r
# graph tuning parameters and respective misclassification errors
ggplot(data=tuning_grid, aes(x=Var1, y=misclass_err, group=Var2)) +
  geom_line(aes(color=as.factor(Var2)))+
  geom_point(aes(color=as.factor(Var2))) +
  theme_minimal() + labs(y= "misclassification error",
                         x = "num. of vars sampled as candidates at each split",
```

```
                        colour = "minimum size of \nterminal nodes",
                        title = "tuneing parameters")
```



* the lowest missclassification error (0.04166667) is when the number of vars sampled as candidates at each split is 2 and the minimum size of terminal nodes is 1. This is the simplest model of all built.

5. With the tuned model, make predictions using the majority vote method, and compute the misclassification rate using the 'vowel.test' data.

```
# set seed for reproducibility
set.seed(13)
# random forest with identified tuning parameter values above
vowel.rf.test <- randomForest(vowel.train[-1],  as.factor(vowel.train$y),
                              mtry = 2, nodesize = 1)
# predict on test set
pred.test <- data.frame(predict(vowel.rf.test, vowel.test))
# calculate misclassifcation error
test_err <- (1 - sum(pred.test[,1] == vowel.test[,1])/nrow(pred.test))
print(paste("Test Misclassification Error: ", test_err))
```

```
## [1] "Test Misclassification Error:  0.393939393939394"
```

The test missclassifcation rate (0.393939) is pretty high compared to the validation misclassification rate. But this is expected. It is also interesting that the simplest model does the best job. This also makes sence as random forest models work better with many weak learners!