MachineLearningProject

Li

3/27/2020

Get data and do explarotary analysis to clean up the data by removing all empty values and plit the training data into .75 as TrainSet for training the model and .25 as ValidSet for cross validation.

```
# get data
    url_train="https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
    url_test="https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
    download.file(url_train, "pml-training.csv", method="curl", mode="wb")
    download.file(url_test, "pml-testing.csv", method="curl", mode="wb")
    # read in data and convert all empty values to NA
    training = read.csv("pml-training.csv",na.strings=c("NA","NaN"," ",""))
    testing = read.csv("pml-testing.csv",na.strings=c("NA","NaN"," ",""))
    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
```

```
    # only keep needed variables
    colsel = grep("belt|forearm|arm|dumbell|classe",names(training))
    trainsel=select(training, names(training)[colsel])
    testsel=select(testing, names(testing)[colsel])

# remove all NA data
    trainclean=trainsel[,!colSums(is.na(trainsel)) > 0.1*nrow(trainsel)]
    testclean=testsel[,!colSums(is.na(testsel)) > 0.1*nrow(testsel)]
    testclean=testclean[,-40] # remove the last column

# split clean train data to train set and validation
    train = sample(nrow(trainclean),0.75*nrow(trainclean), replace=FALSE)
    TrainSet = trainclean[train,]
    ValidSet = trainclean[-train,]

# import libraries
    library(caret)
```

```
## Warning: package 'caret' was built under R version 3.6.3
```

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

```
## Loading required package: ggplot2
```

```
    library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.6.3
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
    library(forecast)
```

```
## Warning: package 'forecast' was built under R version 3.6.3
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
    library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.6.3
```

Modeling with Random Forest, and cross-validate with valid data set then predict the test data set. Random forest modeling took quiet some time even with parallel implementation. But accuracy is pretty higher (>99%). As expected, the in sample accuracy (which is 1 here) will be higher than the out of sample accuracy (99.23%).

```
## Parallel Implementation of Random Forest
    set.seed(1234)
    library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.3
```

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

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

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

```
## The following object is masked from 'package:rattle':
##
##     importance
```

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

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
# set up training run for x / y syntax because model format performs poorly
x = TrainSet[,-40]
y = TrainSet[,40]

# configure parallel processing
library(parallel)
library(doParallel)
```

```
## Warning: package 'doParallel' was built under R version 3.6.3
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 3.6.3
```

```
## Loading required package: iterators
```

```
## Warning: package 'iterators' was built under R version 3.6.3
```

```r
cluster = makeCluster(detectCores()-1) # convention to leave 1 core for OS
registerDoParallel(cluster) # open the cluster
modrfCtrl = trainControl(method ="cv",number=5,allowParallel=TRUE) # Configure tra
inControl object
modrf = train(x,y,method="rf",data=TrainSet,trCtrl=modrfCtrl) # develop training m
odel
stopCluster(cluster) # shut down the cluster
registerDoSEQ() # force R to return to single treaded processing

pred.train.rf = predict(modrf,TrainSet) # in sample predict
rf.in = confusionMatrix(pred.train.rf, TrainSet$classe)$overall['Accuracy'] # in s
ample accuracy
predrf = predict(modrf,ValidSet) # cross validate
rf.out = confusionMatrix(predrf, ValidSet$classe)$overall[1] # out of sample accur
acy
predrf.test = predict(modrf, testclean) # predict test data set
predrf.test # print the results for test data set
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```

Modeling with Extreme Gradient Boosting (xgboost), which has bulit-in cross-validation then predict the test data set. XGBoost modeling is very fast compared with rf modeling, and provide very high accuracy (~100%), the best modeling. xgb.cv provides the out of sample errors.

```
## XGBoost Multinomial Classification
    library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:rattle':
##
##     xgboost
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
  # convert the response factor to an integer class starting at 0
    trainlabel = as.integer(trainclean$classe)-1
  # convert data frame to matrix
    TrainMatrix=xgb.DMatrix(data=as.matrix(trainclean[,1:39]),label=as.matrix(trainlab
el))
    TestMatrix=xgb.DMatrix(data=as.matrix(testclean))
  # Set parameters(default)
    params = list(booster ="gbtree",objective="multi:softprob",num_class=5,eval_metric
="merror")
  # train model using full training set
    modxgb = xgb.train(params =params,data=TrainMatrix,nrounds=1000)
  # Predict outcomes with the test data
    predxgb = as.data.frame(predict(modxgb,newdata=TestMatrix,reshape=T))
    colnames(predxgb) = levels(TrainSet$classe)
  # label the highest probability with classe levels
    predxgb$predict = apply(predxgb,1,function(x) colnames(predxgb)[which.max(x)])
    predxgb$predict # print the predition result for test data set
```

```
##  [1] "B" "A" "B" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B"
## [20] "B"
```

```
  # modeling with build-in cross validation and checking for out of sample error
    modxgbcv = xgb.cv(params=params,data=TrainMatrix,nrounds=1000,nfold=10,showsd=TRU
E,
            stratified=TRUE,print_every_n=100,early_stop_round=20,maximize=FALSE,predi
ction=TRUE)
```

```
## [1]  train-merror:0.215778+0.013885  test-merror:0.232033+0.015178
## [101]    train-merror:0.000000+0.000000  test-merror:0.004230+0.001406
## [201]    train-merror:0.000000+0.000000  test-merror:0.003006+0.001127
## [301]    train-merror:0.000000+0.000000  test-merror:0.002905+0.001020
## [401]    train-merror:0.000000+0.000000  test-merror:0.002803+0.001145
## [501]    train-merror:0.000000+0.000000  test-merror:0.002803+0.001025
## [601]    train-merror:0.000000+0.000000  test-merror:0.002803+0.001025
## [701]    train-merror:0.000000+0.000000  test-merror:0.002803+0.001075
## [801]    train-merror:0.000000+0.000000  test-merror:0.002752+0.001121
## [901]    train-merror:0.000000+0.000000  test-merror:0.002853+0.001188
## [1000]   train-merror:0.000000+0.000000  test-merror:0.002853+0.001188
```

```
    xgb.in = 1-modxgbcv$evaluation_log$train_merror_mean[1000] # print in sample accur
acy
    xgb.out = 1- modxgbcv$evaluation_log$train_merror_mean[1000] # print out of sampl
e accuracy
```

Other modelings: Decision tree modeling (rpart) is fast but accuracy is only 53%; the basic gradient boosting took sometime with accuracy 93%; Linear Discriminant Analysis (lda) is fast, but accuracy is only 56%

```
## other modelings
    # decision tree modeling
    modrpart=train(classe~.,method="rpart",data=TrainSet) # decision tree
    pred.in.rpart <- predict(modrpart, TrainSet) # predict in sample
    rpart.in = confusionMatrix(pred.in.rpart, TrainSet$classe)$overall[1] # in sample 
accuracy
    predrpart <- predict(modrpart, ValidSet) # predict with ValidSet
    rpart.out = confusionMatrix(predrpart, ValidSet$classe)$overall[1] # Accuracy chec
k

    # gradient boosting modeling
    modgbm =train(classe~.,method="gbm",data=TrainSet,verbose=FALSE) # gradient boosti
ng
    pred.in.gbm <- predict(modgbm, TrainSet) # predict in sample
    gbm.in = confusionMatrix(pred.in.gbm, TrainSet$classe)$overall[1] # in sample accu
racy
    predgbm <- predict(modgbm, ValidSet) # predict with ValidSet
    gbm.out = confusionMatrix(predgbm, ValidSet$classe)$overall[1] # Accuracy check

    #  Linear Discriminant Analysis
    modlda=train(classe~.,method="lda",data=TrainSet)
    pred.in.lda <- predict(modlda, TrainSet) # predict in sample
    lda.in = confusionMatrix(pred.in.lda, TrainSet$classe)$overall[1] # in sample accu
racy
    predlda <- predict(modlda, ValidSet) # predict with ValidSet
    lda.out = confusionMatrix(predlda, ValidSet$classe)$overall[1]
```

Cross validation

```
InAccuracy = c(xgb.in,rf.in,rpart.in,gbm.in,lda.in)
OutAccuracy =c(xgb.out,rf.out,rpart.out,gbm.out,lda.out)
knitr::kable(data.frame(Model=c("XGBoost","Random Forest","Decision Tree","Boost","LD
A"), InSampleAccuracy=paste0(round(InAccuracy*100,2),"%"), OutofSampleAccuracy=paste0
(round(OutAccuracy*100,2),"%"), OutofSampleError=paste0(round((1-OutAccuracy)*100,
2),"%")))
```

Model	InSampleAccuracy	OutofSampleAccuracy	OutofSampleError
XGBoost	100%	100%	0%
Random Forest	100%	99.18%	0.82%
Decision Tree	47.02%	47.33%	52.67%
Boost	95.1%	93.8%	6.2%
LDA	58.15%	57.95%	42.05%

Inclusion, both XGBoost and Random Forest modelig provide the highest in sample and out of sample accuracy than other modeling (decision tree, basic gradient boosting, Linear Discriminant Analysis, etc.), but XGBoost modeling is way faster. So I'll choose XBGoost modeling to predict the test data set. The prediction result is listed below, which passed the final quiz with 100% score.

Predicted results of test data set: "B" "A" "B" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B" "B"