Practical Machine Learning Course Project

AW88525

6/22/2019

Summary

In this report, I used the weight lifting exercise dataset from this website http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har. Specifically, given the training set, I further split it into a training and cross validation set. I trained a random forest classifier on the training set and then I applied filtering method for variable selection with the cross validation set. Lastly, I applied the trained and cross-validated model to the test model. I passed the quiz test with 100% match to the true results in the test set.

R Markdown

```
library(caret)
```

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

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

```
download.file(url = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", destfile = "
download.file(url = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", destfile = "
# Import the raw training and testing datasets
trainingraw = read.csv(file = "~/Desktop/datasciencecoursera/PracML_proj/pml-training.csv", stringsAsFa
testingraw = read.csv(file = "~/Desktop/datasciencecoursera/PracML_proj/pml-testing.csv", stringsAsFacto
# Cleaning data i.e. the NA rows
emptyratiotrain <- sapply(trainingraw, function(x){sum(is.na(x) | x=="")/length(x)})
emptyratiotest <- sapply(testingraw, function(x){sum(is.na(x) | x=="")/length(x)})
trainingraw2 <- trainingraw[, emptyratiotrain == 0]
testingraw2 <- testingraw[, emptyratiotest == 0]
# Then want to remove the first 8 rows which may not apply to the classification task
trainingraw3 <- trainingraw2[,8:length(trainingraw2)]
inTrain <- createDataPartition(y = trainingraw3$classe, p = 0.7, list = FALSE)
training <- trainingraw3[inTrain, ]
cv <- trainingraw3[-inTrain,]
testing <- testingraw2[,8:length(testingraw2)]
training$classe = as.factor(training$classe)
cv$classe =  as.factor(cv$classe)
```

Here I want to apply filtering methods to first remove some redundant features, and correlation matrix will be a good choice.

```
#correlationMatrix
correlationMatrix <- cor(training[, 1:52])
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff = 0.9)
#maybe just remove the highly correlated ones
training_removehighcorr <- subset(training, select = -highlyCorrelated)
cv_removehighcorr <- subset(cv, select = -highlyCorrelated)
testing_removehighcorr <- subset(testing, select = -highlyCorrelated)
```

Now I would like to train a random forest classifier for this task, but since there are still quite a number of features, and such features may overfit the model so I want to glean the most important variables for a finalized random forest classifier for testing set. The rationale is train a full model, and then get the variable importance with the varImp function to see which variables matter most based on the out-of-bag classifications. Notice I didn't use the more automatic recursive feature elimination (RFE) (a wrapper method) which is apparently computationally expensive and time-consuming. After getting the variable importance here, I then ranked the variables by their importance. I then applied a forward method to incorporate the most important variable to the least important one each time for training random forest classifiers which were applied to cross-validation set. Then I quantified two metrics 1) accuracy and 2) F1-score which is (recall*precision)/(recall+precision).

```
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
```

```
set.seed(1234)
rfmodel <- randomForest(classe ~ ., data = training_removehighcorr)
importance_order <- order(importance(rfmodel), decreasing = TRUE)
#install.packages("MLmetrics")
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'
```

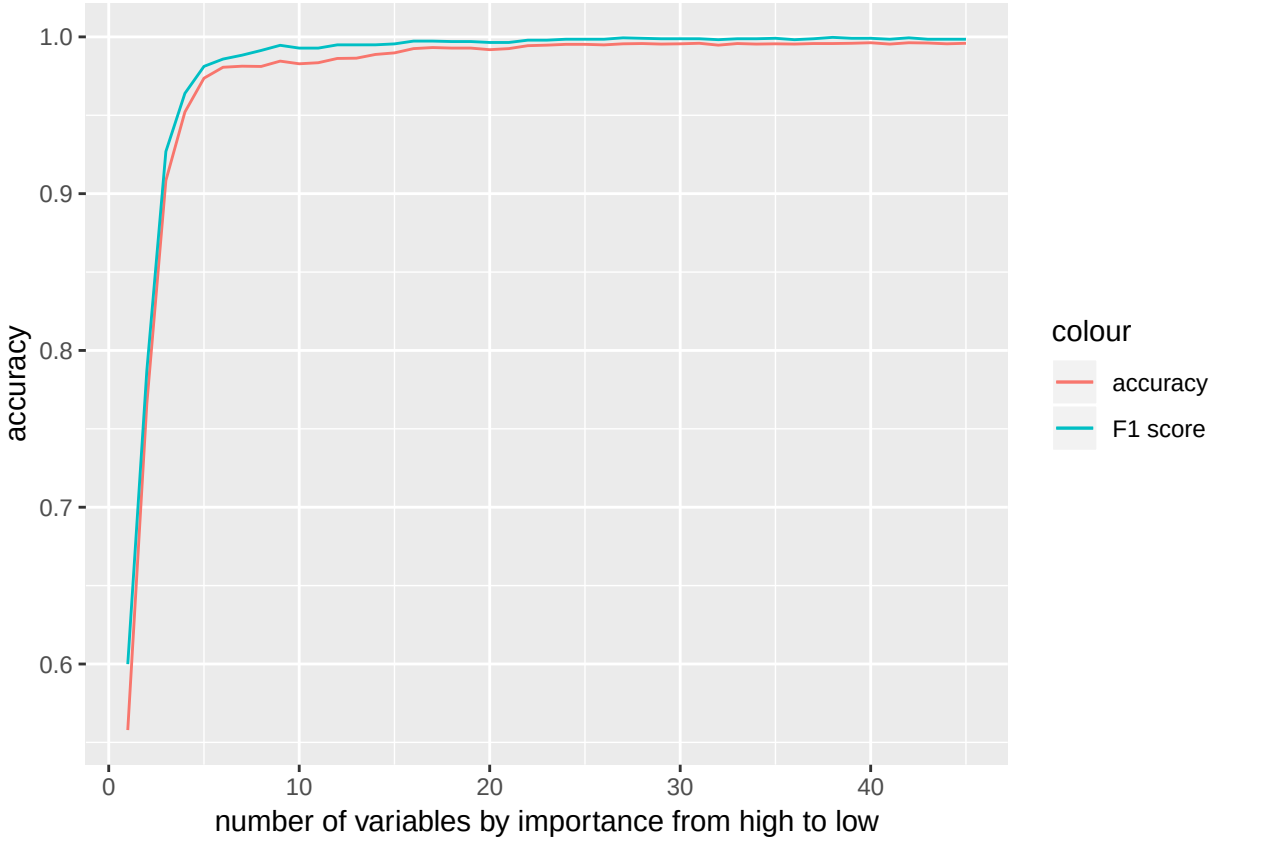
```
## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
```

```
## The following object is masked from 'package:base':
##
##     Recall
```

```
accuracy <- c()
f1score <- c()
ind <- c()
for(i in importance_order){
    ind <- c(ind,i)
    trainingsubset <- subset(training_removehighcorr, select = ind)
    trainingsubset$classe <- training_removehighcorr$classe
    cvsubset <- subset(cv_removehighcorr, select = ind)
    cvsubset$classe <- cv_removehighcorr$classe
    set.seed(1234)
    rfmodelcv <- randomForest(classe ~ ., data = trainingsubset)
    predict_cvresult <- predict(rfmodelcv, newdata = cvsubset)
    accuracy<-c(accuracy,sum(cv_removehighcorr$classe == predict_cvresult)/length(cv_removehighcorr$cla
    f1score<-c(f1score,F1_Score(cv_removehighcorr$classe,predict_cvresult))
    #print(count)
    #print(accuracy)
    #print(f1score) #long loop so you know where we are and be patient!
```

```
}

#Here I would like to plot the accuracy and f1 score vs the number of most important variables
library(ggplot2)
num <- c(1:length(importance_order))
accuracy_num <- data.frame(number = num, accuracy = accuracy)
f1score_num <- data.frame(number = num, f1score = f1score)
ggplot()+geom_line(data = accuracy_num, aes(x = num, y = accuracy, color = "accuracy"))+
  geom_line(data = f1score_num, aes(x = num, y = f1score, color = "F1 score")) +
  xlab("number of variables by importance from high to low")
```




```
#pick the variables with the highest 25 importances and train them on full datasets
training_full_removehighcorr <- rbind(training_removehighcorr, cv_removehighcorr)
training_full_bestsubset <- subset(training_full_removehighcorr, select = importance_order[1:25])
testing_bestsubset <- subset(testing_removehighcorr, select = importance_order[1:25])
testing_bestsubset$classe <- testing_removehighcorr$classe
training_full_bestsubset$classe <- training_full_removehighcorr$classe
rfmodelfull <- randomForest(classe ~ ., data = training_full_bestsubset)
```

Given the above trained random forest classifier, I now want to apply it to the testing set.

```
predict_testresult <- predict(rfmodelfull, newdata = testing_bestsubset)
print(predict_testresult)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```