Practical Machine Learning–Predict Users' Excercises

P. Zhou

11/27/2017

1. Read in and preprocess datasets

Training data and testing data are loaded. Variables with over 10% missing values were removed from the final analysis. Only the highly relevant variables were kept. Specific user and time related varaibles were excluded.

```
testing<-read.csv("pml-testing.csv",header=TRUE)
training<-read.csv("pml-training.csv",header=TRUE)
class(training$classe)
```

```
## [1] "factor"
```

Remove features with mostly missing values based on training data

```
na_ratio<-apply(training,2,function(x) sum(is.na(x))/length(x))
summary(na_ratio)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.4101  0.9793  0.9793
```

```
training_rm<- na_ratio > 0.5
table(training_rm)
```

```
## training_rm
## FALSE  TRUE
##    93    67
```

```
training<-training[, !training_rm]
# The first variable "X" is just index. I removed it from the final data.
training<-training[,-1]
```

Only keep the highly relevant variables in the dataset, excluding the specific user name relatived or time #related variables

```
training<-training[,-grep("user|time|window",colnames(training))]
```

Because my laptop does not have low capacity to process the whole dataset, I randomly selected approximately 1000 data points to analyze.

```
sub_training<-training[sample(nrow(training),1000),]
dim(sub_training)
```

```
## [1] 1000   86
```

2. Build predictive model

I used Random Forest (RF) as the predictive method in model fitting, because Random Forest usually has high predictive performance and yield very low error rate should be very low error rate. All varaibles from the processed training data were used as predictors. I applied three times of 5-fold cross-validation to evaluate out-of-sample error. The final model is presented. The results from the confusion matrix model demonstated that the error rate is very low for all 5 classes.

```
set.seed(1234)
library(caret)
```

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

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

```
library(gbm)
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.1
```

```
library(AppliedPredictiveModeling)
library(parallel)
library(doParallel)
```

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

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

```
cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)
library(iterators)
library(foreach)
library(iterators)
library(randomForest)
```

```
## randomForest 4.6-12
```

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

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

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

```
controldata<-trainControl(method="repeatedcv", number=5,repeats=3, allowParallel = TRUE)
fit_mod<-train(classe~.,data=sub_training, method="rf",trControl=controldata)
fit_mod
```

```
## Random Forest
##
## 1000 samples
##   85 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 800, 800, 799, 799, 802, 802, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##      2  0.2730002  0.0000000
##    117  0.4010892  0.1903915
##   6856  0.8670664  0.8321870
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 6856.
```

```
print(fit_mod$finalModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6856
##
##         OOB estimate of  error rate: 11%
## Confusion matrix:
##     A   B   C   D   E class.error
## A 264   2   1   3   3  0.03296703
## B  21 145  15   5   2  0.22872340
## C   1   7 168   1   1  0.05617978
## D   3   4  16 147   2  0.14534884
## E   3   8   6   6 166  0.12169312
```

```
## 3 repeats of 5 fold cross-validation
controldata<-trainControl(method="repeatedcv", number=5,repeats=3)
fit_mod<-train(classe~.,data=sub_training, method="rf",trControl=controldata)
print(fit_mod)
```

```
## Random Forest
##
## 1000 samples
##   85 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 799, 798, 802, 800, 801, 801, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##      2  0.2730030  0.0000000
##    117  0.4131229  0.2075508
##   6856  0.8720123  0.8383928
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 6856.
```

```
print(fit_mod$finalModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6856
##
##         OOB estimate of  error rate: 12%
## Confusion matrix:
##     A   B   C   D   E class.error
## A 263   2   2   4   2  0.03663004
## B  20 146  18   3   1  0.22340426
## C   2   9 163   3   1  0.08426966
## D   4   3  18 145   2  0.15697674
## E   3   7   7   9 163  0.13756614
```

3. Out-of-bag error

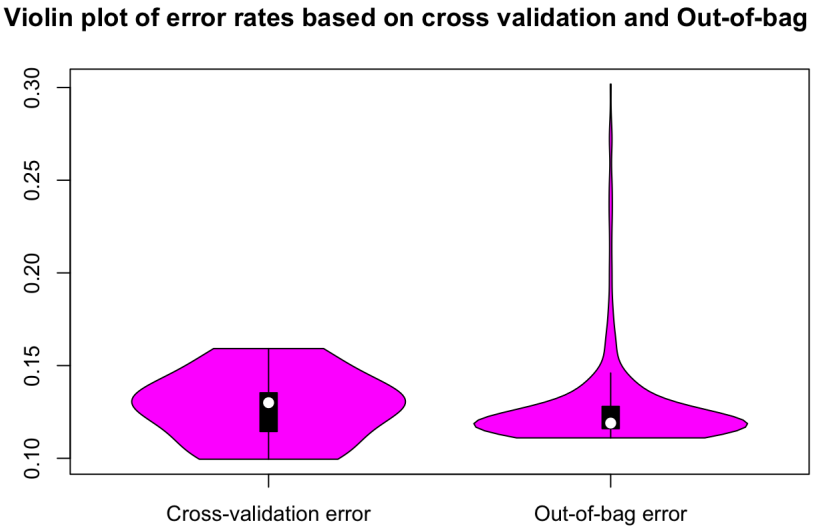
Out-of-bag error is "a method of measuring the prediction error of random forests, boosted decision trees, and other machine learning models utilizing bootstrap aggregating to sub-sample data samples used for training. OOB is the mean prediction error on each training sample x_i, using only the trees that did not have x_i in their bootstrap sample"(from WikiPedia). I used the 3 repeats of 5-fold cross-validation to evaluate out-of-sample error. When I used Random Forest, it automatically generated out-of-bag error estimates. I generated a violin plot below. From this plot, we can visually see that both the out-of-bag error and cross validation methods provide similar estimates of mean error rates. The cross validation had a mean around 0.11 while out-of-bag error showed a mean around 0.09.

```
library(vioplot)
```

```
## Loading required package: sm
```

```
## Package 'sm', version 2.2-5.4: type help(sm) for summary information
```

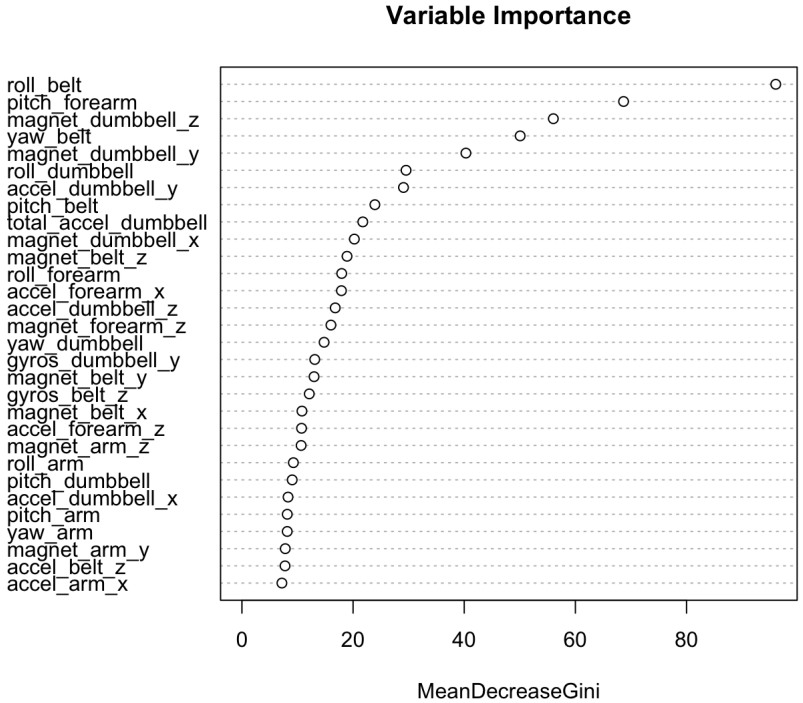
```
opt <- options(scipen = 3)
cv_error<-1-fit_mod$resample$Accuracy
outOfBag_error<-fit_mod$finalModel$err.rate[,"OOB"]
vioplot(cv_error,outOfBag_error,names=c("Cross-validation error","Out-of-bag error"))
title("Violin plot of error rates based on cross validation and Out-of-bag error")
```



4. Plot variable importance

In order to know which varaible contribute the most to the predictive performance, I create a variable importance plot. From the plot, I found that the most important varaible in terms of contributing to the predictive performance is "roll belt". The second most imporatance varaible is "yaw belt".

```
library(randomForest)
varImpPlot(fit_mod$finalModel,main="Variable Importance",pach=20)
```



5. Predict the testing data

In section5, I generated the predictions for the test data. First I need to make the testing data has the same columns as the traning dataset

{r,echo= TRUE} #testing<-testing[,!training_rm] #testing<-testing[,-1] #testing<-testing[,-grep("problem_id",colnames(testing))] #te

```
library(caret)
```

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

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

```
library(randomForest)
```

```
## randomForest 4.6-12
```

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

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

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

```
predicted<-predict(fit_mod$finalModel,data=testing)
answers<-as.character(predicted)

predict_WriteFiles = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("problem_id_",i,".txt")
    write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
  }
}
predict_WriteFiles(answers)
```