Activity Classification Using Machine Learning Model from Data Collected by Personal Electronic Devices

Summary

Data collected by personal electronic devices can be used perdict the activity the user is performing. Recently, the Human Activity Recognition (HAR) collected data via Inertial Measurement Units (IMU) sensors in order to understand whether how well one performs the activity can be distinguished. Utilizing this dataset collected by HAR, this project aim to build a machine learning model that catagorize different ways the user perform weight lifting exercise.

Dataset Description

Training and testing dataset were obtained from course website. The original training dataset contains 19622 observations and 160 variables. These data were collected from 6 subjects, each perfromed 5 ways of Unilateral Dumbbell Biceps Curl (hereby referred to as weight lifting). The 5 ways of weight lifting are indicated as A-E in the variable named "classe". According to HAR, class A is exactly according to the specification, class B is throwing the elbows to the front, class C is lifting the dumbbell only halfway, class D is lowering the dumbbell only halfway, and class E is throwing the hips to the front. The 160 variables of the dataset include subject, time, and data collected from IMU sensors located at different places of the body (belt, arm, forearm and dumbbell). The testing dataset (downloaded from course website, hereby referred to as quiz-testing dataset) contains only 20 observations which was used for the quiz only. A separate testing dataset used for model evaluation was created during pre-processing to better understand the performance of the model.

Refer to Appendis I for codes.

Exploratory Analysis

It is important to uderstand the distribution of outcome variable among different predictors. To understand this, feature plots were created to show the distribution of class A-E in individual variables after centering and scaling (fig.1). Only selected variables were shown, and these varaibles were selected based on the vairable importance, which will be described later with the model. The feature plots reveal that although there're some differences, the classes largely overlap with each other.

PCA analysis was also performed on the data after preprocessing (removing unuseful predictors, centering and scaling). Similar to the 1-dimentional feature plot, PCA also did not separate the classes (fig.2). This is because the majority of variation among predictors was not due to the class difference.

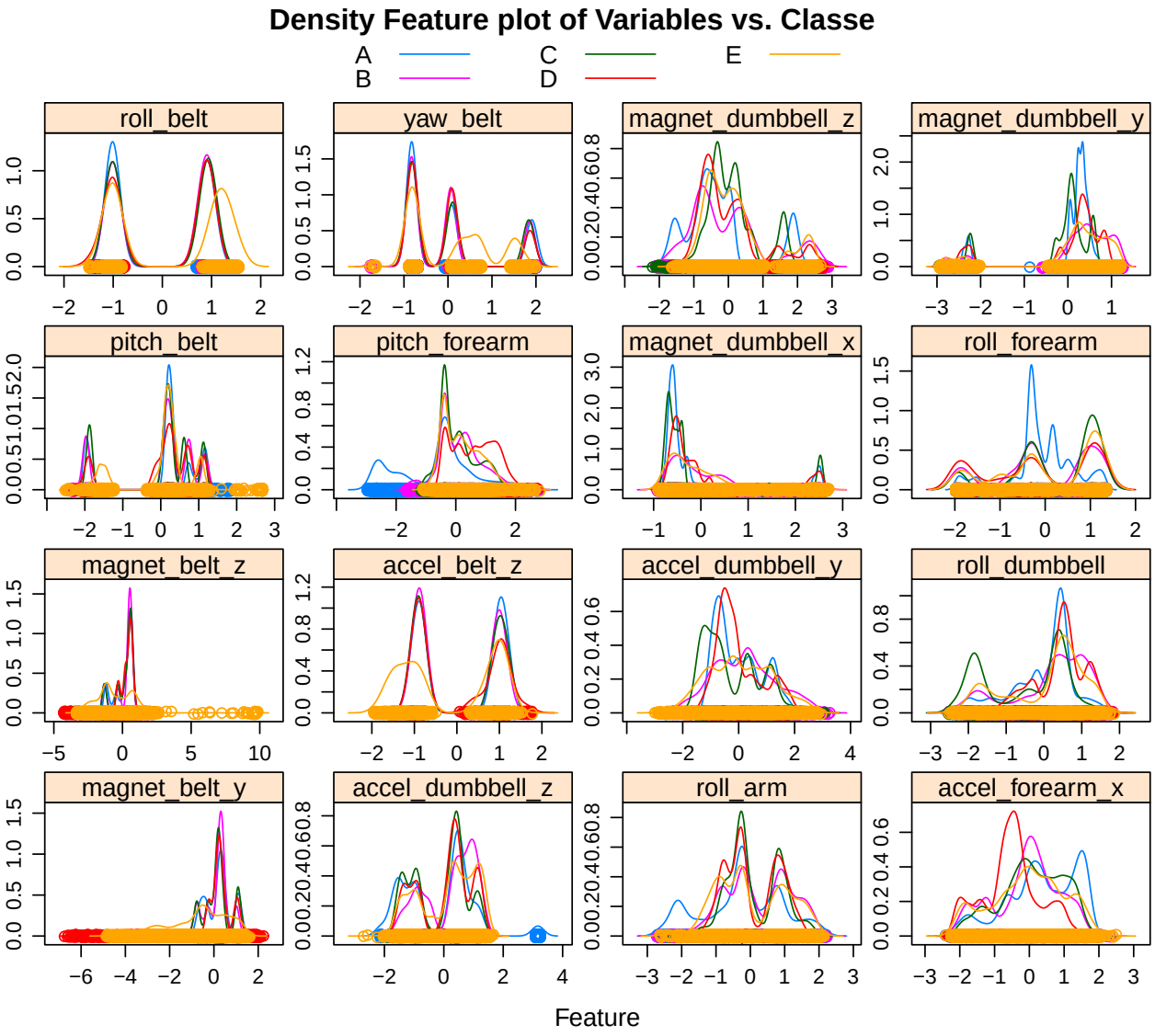

Figure 1. Distribution plot of selected variables on centered and scaled data

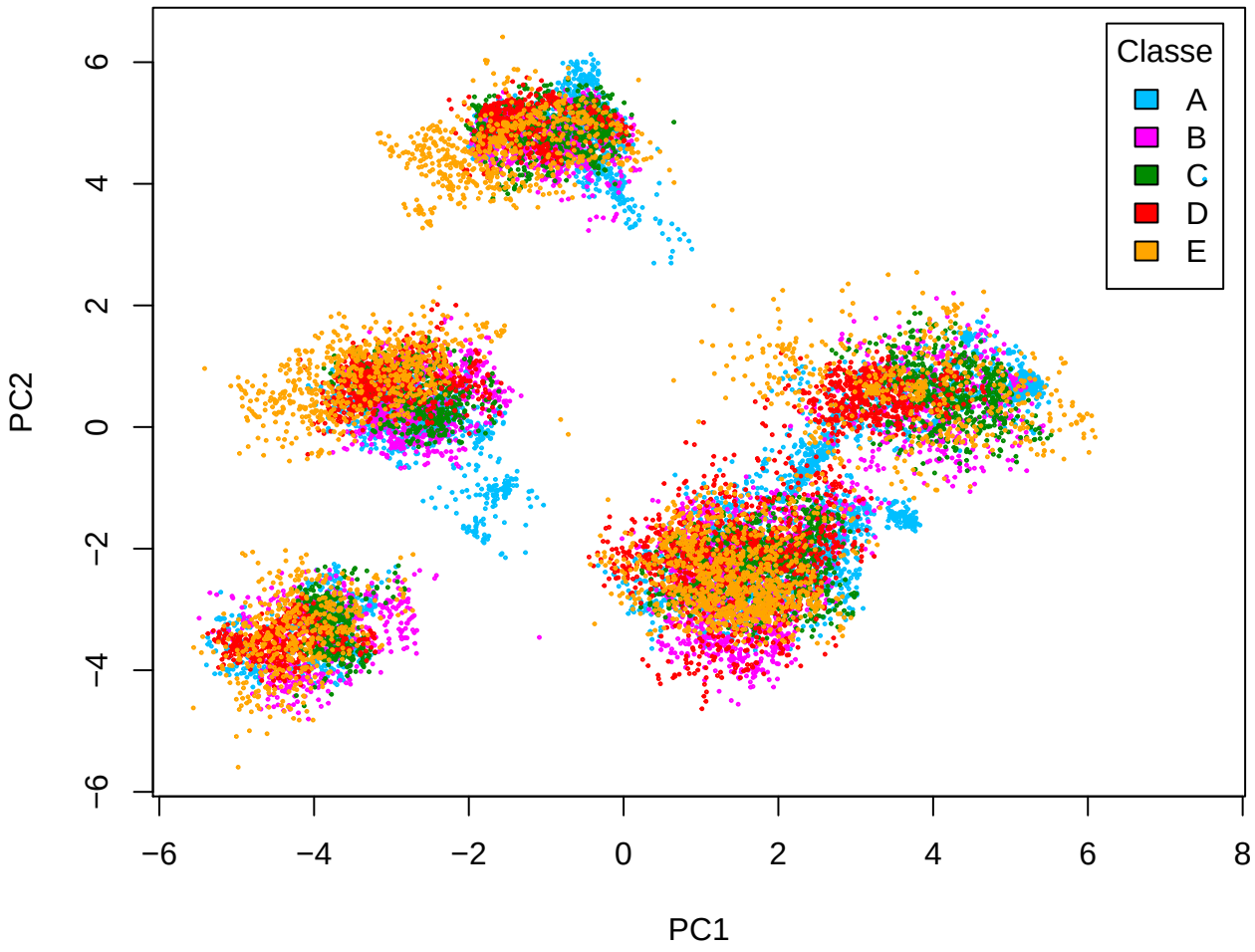

Figure 2. PCA plot

Model Selection

Multiple factors were taken into consideration for machine learning model selction. First, the dataset consists relatively large number of observations and predictors. Second, the outcome variable, classe, is a factor variable with 5 levels. Third, as revealed in exploratory analysis, the distribution of classes among individual variables were close to each other. Based on these facts, random forest was chosen as the model.

Preprocessing and Training

To preprocess the the data, user name and near-zero variables from both the training and the quiz-testing datasets were excluded from the predictors. The time variables were also removed, since it is unclear whether time measurement was consistent between different data collection processes. An outlier discovered during exploratory analysis (row 5373) was excluded. Although random forest is generally not sensitive to data distribution, centering and scaling were perfromed in order to accurately calculate variable importance.

The training set downloaded were split randomly into model-training (90% of total) and model-testing (10% of total) datasets. Considering the large size of data and running time, 3-fold cross validation was applied to model training.

Random Forest Model using All Remaining Variables

First, a random forest model was trainied with all 52 variables remained after pre-processing. An overview of the final model is displayed below. The final model has 500 trees with mtry (variables at each split) of 2. The out-of-bag (OOB) error rate of this model was 0.47%. The error rate was highest for class D and lowest for class A. Variable importance was calculated and displayed in Fig.3; this information was used later to build a simplified random forest model.

Random forest model trained with all remaining variables:

```
##
## 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: 2
##
##         OOB estimate of  error rate: 0.47%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 5020    2    0    0    0 0.0003982477
## B   14 3398    5    0    0 0.0055604331
## C    0   16 3063    1    0 0.0055194805
## D    0    0   39 2854    2 0.0141623489
## E    0    0    0    4 3243 0.0012319064
```

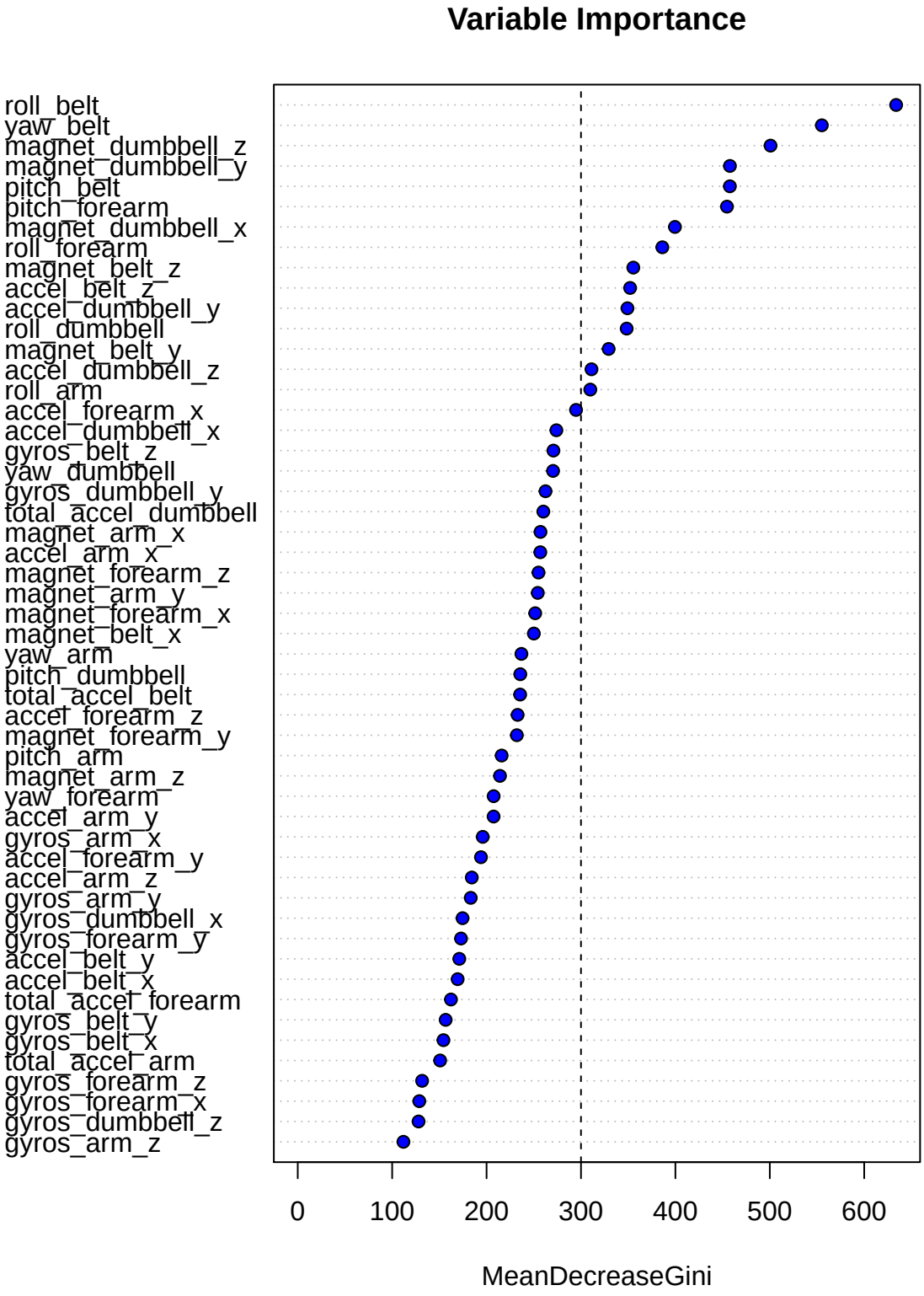

Figure 3. Variable importance ranking from the random forest model

Random Forest Model with Fewer Predictors

(OPTIONAL – to graders: skip this part if you would like)

The first model utilized 52 predictors, but some of the variables were more important than the others (fig.3). Since the model took a relatively long time to be trained, it was then explored if the running time can be shortened without loosing too much accuracy. In this approach, a second model was trained using only variables with importance score of greater than 300. This narrowed down the number of predictors from 52 to 15, and shortened model-building time from 306s to 100s, which is about 3 times faster. On the other hand, the simplified model has an OOB error rate of 0.93%, which is about twice as much as the original model. The final model also consiste of 500 trees, however, with mtry of 8.

Final model of the simplified model:

```
##
## 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: 8
##
##         OOB estimate of  error rate: 0.93%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 5005   10    1    5    1 0.003385106
## B   22 3344   38   13    0 0.021363769
## C    0   12 3053   15    0 0.008766234
## D    0    1   18 2873    3 0.007599309
## E    0    4    7   15 3221 0.008007391
```

Comparison of running times between the original and simplified model:

```
## $`Original model, building time:`
##    user  system elapsed
## 338.174   2.726 341.261
##
## $`Simplified model, building time:`
##    user  system elapsed
##  95.289   2.652  98.187
##
## $`Original model, predicting time:`
##    user  system elapsed
##   0.200   0.007   0.208
##
## $`Simplified model, predicting time:`
##    user  system elapsed
##   0.129   0.003   0.132
```

Evaluation of Models on Testing Data

When applied to the testing data, the prediction accuracy with the original model was 99.49%, whereas the simplified model was 98.83%. Both models also predicted all 20 observations from the quiz dataset correctly

(data not shown). Prediction time of the two model on model-testing data were 0.214s vs. 0.128s, which was about 2-fold difference. Therefore, the original model had higher prediction accuracy than the simplified model, however, in the case fast operation is desired, the simplified model can be used.

Testing results of the original model:

```
## $table
##           Reference
## Prediction   A   B   C   D   E
##          A 558   1   0   0   0
##          B   0 378   2   0   0
##          C   0   0 340   5   0
##          D   0   0   0 316   2
##          E   0   0   0   0 358
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9948980      0.9935456      0.9906372      0.9975507      0.2846939
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Testing results of the simplified model:

```
## $table
##           Reference
## Prediction   A   B   C   D   E
##          A 557   3   0   0   0
##          B   0 371   3   0   0
##          C   1   4 336   5   0
##          D   0   1   3 316   3
##          E   0   0   0   0 357
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9882653      0.9851556      0.9824439      0.9925471      0.2846939
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Conclusion

The random forest model(s) generated in this project successfully classified types of weight lifting exercise with approximately 98-99% accuracy. These results confirmed that data collected by IMU sensors on personal devices can be used to predict the quality of exercise a user is performing.

Appendix I

Codes:

```r
# Load data and packages
urlTest <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
urlTrain <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
download.file(urlTest,"./pmlTest.csv",mode = "wb")
download.file(urlTrain,"./pmlTrain.csv",mode = "wb")
quizTest <- read.csv("./pmlTest.csv", header = TRUE)
train <- read.csv("./pmlTrain.csv", header = TRUE)

library(caret)
library(ggplot2)
library(tidyr)
library(randomForest)
```

```r
# Data cleaning
# Removenear zero variables (using training set):
nzvCheck <- nearZeroVar(train, saveMetrics = TRUE)
train2 <- train[, nzvCheck$nzv == FALSE]
quizTest2 <- quizTest[, nzvCheck$nzv == FALSE]

# Remove near-zero variables (using quiz-test set)
nzvCheck2 <- nearZeroVar(quizTest2, saveMetrics = TRUE)
train3 <- train2[, -which(nzvCheck2$nzv == TRUE)]
quizTest3 <- quizTest2[, -which(nzvCheck2$nzv == TRUE)]

# Remove time and other non-predictor variables
train4 <- train3[, -c(1:6)]
quizTest4 <- quizTest3[, -c(1:6)]
```

```r
# Create model training and testing sets from the dowloaded train set
set.seed(100)
inTrain <- createDataPartition(y = train4$classe, p = 0.9, list = FALSE)
training <- train4[inTrain,]
testing <- train4[-inTrain,]

training <- training[-5373,] # Exclude an outlier
```

```r
# Preprocess data-- centering and scaling
preProcValues <- preProcess(training, method = c("center", "scale"))

training_cs <- predict(preProcValues, training)
testing_cs <- predict(preProcValues, testing)
quizTest_cs <- predict(preProcValues, quizTest4)
```

```r
# Feature plot
x <- training_cs[-8347, order[c(13:16, 9:12, 5:8, 1:4)]]
y <- training_cs[-8347, 53] # note: row 8347 is an outlier

scales <- list(x=list(relation="free"), y=list(relation="free"))
featurePlot(x=x, y=y, plot="density", scales = scales, layout = c(4,4),
            main = "Density Feature plot of Variables vs. Classe",
            auto.key = list(space = "top", columns = 3, cex = 1))
```

```r
# PCA analysis
procPCA <- preProcess(training_cs, method = "pca")
training_pca <- predict(procPCA, training_cs)

cols <- c("deepskyblue", "magenta", "green4", "red", "orange" )
plot(PC2 ~ PC1, data = training_pca, cex = 0.2, col = cols[classe])
legend("topright", inset=.02, title="Classe", c("A","B","C","D","E"),
       fill=cols, horiz=FALSE, cex=1)
```

```r
set.seed(123)
modRF <- train(classe ~., method = "rf", data = training_cs,
                trControl = trainControl(method = "cv", number = 3))

modRF$finalModel
```

```r
# Sort variables by importance from the above model
variableImp <- varImp(modRF$finalModel)
order <- order(variableImp$Overall, decreasing = T)
variableImp$var <- rownames(variableImp)

# Build a rf model with variables of importance > 300
var300 <- which(variableImp$Overall>300)
training_cs_small <- training_cs[, c(var300, 53)]

set.seed(123)
modRF_small <- train(classe ~., method = "rf", data = training_cs_small,
                trControl = trainControl(method = "cv", number = 3))

modRF_small$finalModel
```

```r
# time comparison
names<- c("Original model, building time:", "Simplified model, building time:",
          "Original model, predicting time:", "Simplified model, predicting time:")
times <- list()
times[[1]] <- modRF$times[1][[1]]
times[[2]] <- modRF_small$times[1][[1]]
times[[3]] <- system.time(predict(modRF, testing_cs))
times[[4]] <- system.time(predict(modRF_small, testing_cs_s))
names(times) <- as.list(names)
times
```

```r
# Evaluate model performance using testing data and perform prediction on quiz-test data:

# modRF
pred_modRF <- predict(modRF, testing_cs) # 99.44% accuracy
confusionMatrix(pred_modRF, reference = testing_cs$classe) # 99.49% accuracy

pred_quiz1 <- predict(modRF, quizTest_cs)

# modRF_small
testing_cs_s <- testing_cs[, c(var300, 53)]
pred_modRF_s <- predict(modRF_small, testing_cs_s)
confusionMatrix(pred_modRF_s, reference = testing_cs_s$classe) # 99.23% accuracy
```

```
quizTest_cs_s <- quizTest_cs[, c(var300)]
pred_quiz2 <- predict(modRF_small, quizTest_cs_s)

# Comparing quiz answers
pred_quiz1 == pred_quiz2 # All answers match.
```