Qualitative Activity Recognition of Weight Lifting Exercises

Arthur

4/6/2020

```
url1 <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
url2 <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(url1, destfile = "./pm1-training.csv")
download.file(url2, destfile = "./pm1-testing.csv")
```

```
training <- read.csv("./pm1-training.csv", header = TRUE)
testing <- read.csv("./pm1-testing.csv", header = TRUE)
```

Data Cleaning

The data set includes “summary statistics” corresponding with variable names prefixed with “kurtosis_”, “skewness_”, “max_”, “min_”, “amplitude_”, “var_”, “avg_”, and “stddev_”.

These variables usually contain very high proportions of missing values and/or “#DIV/0!” error values.

We argue that removing these variables shall not affect the overall prediction since they are derived from the variables left (which are the raw record data).

```
# Remove the "summary statistics" calculated from the raw sensor data
# These include variable names prefixed with "kurtosis", "skewness", "max", "mi
n", "amplitude", "var", "avg", and "stddev"
# Most of them are highly enriched with missing values:
#In training data set, only 406 among 19622 observations are complete cases wit
h no missing values among these statistics.
#In testing data set, no observation has complete "summary statistics".

train_new <- training[, - grep("^kurtosis|^skewness|^max|^min|^amplitude|^var|^
avg|^stddev", colnames(training))]
test_new <- testing[, - grep("^kurtosis|^skewness|^max|^min|^amplitude|^var|^av
g|^stddev", colnames(testing))]
```

After removing these variables, there is no missing value left in either new training or new testing set.

Create the validation data set

We randomly select 20% data from the training data set to create the validation and NEW train data set.

```
library(caret)
```

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

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

```
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
```

```
set.seed(2134)
inTrain <- createDataPartition(y = train_new$classe, p = 0.8, list = FALSE)
validation <- train_new[-inTrain,]
train_sub <- train_new[inTrain,]
```

Below we will only focus on train_sub data set (called ‘training data’ below) before we perform the final validation.

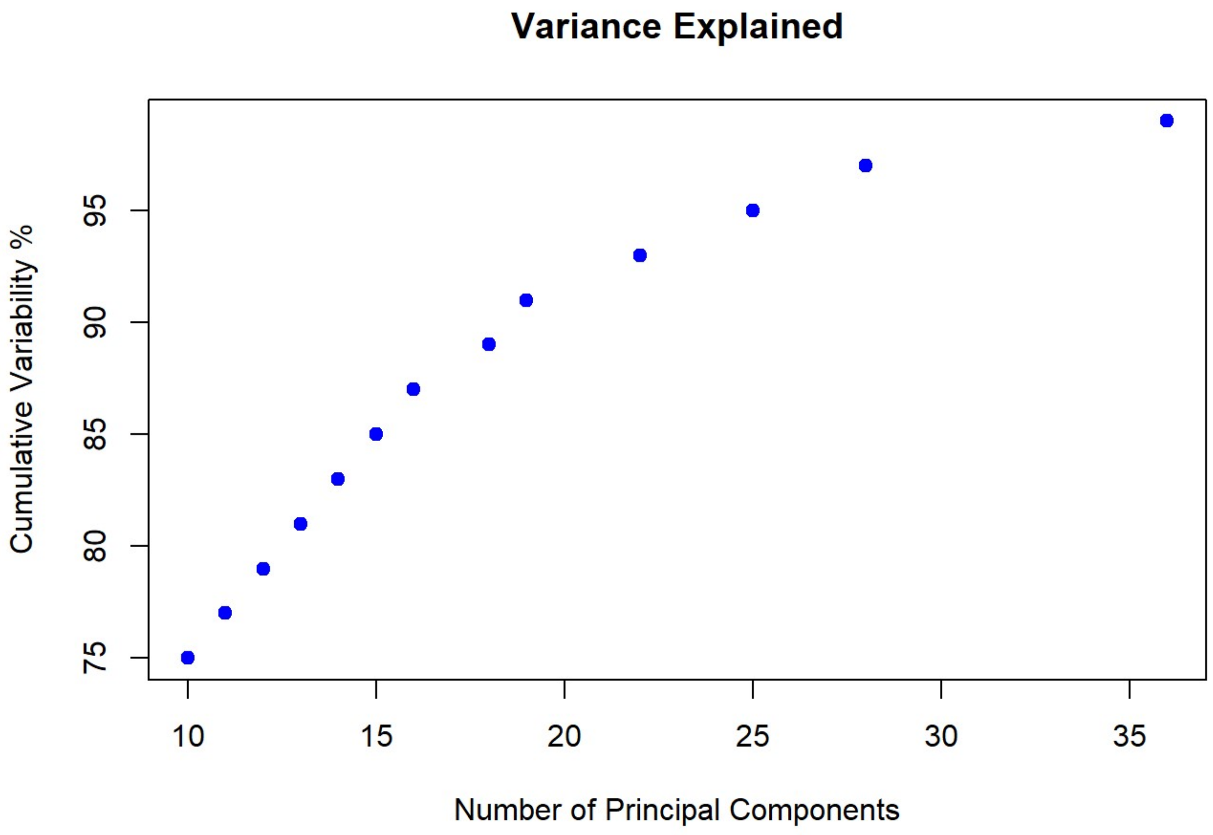
Next, we create a new dataset“train_predictors” that only contain the raw sensor data and participants’ names.

```
train_predictors <- train_sub[, -c(1:7,60)]
train_predictors <- mutate_all(train_predictors, function (x) as.numeric(as.cha
racter(x)))

#Add the classe type to the 53th column
train_predictors$classe <- train_sub$classe
#Add the participant's name to the 54th column
train_predictors$user_name <- train_sub$user_name
```

Test the algorithm

1. First, we use PCA to explore the training data and ask if the participants' names are important for prediction (i.e. is there any person-to-person variability?)

```
numPC <- as.numeric()
for (i in seq(from =75, to = 99, by = 2)) {
               preProc <- preProcess(train_predictors[,-c(53,54)], method = "pc
a", thresh = i/100)
               numPC[i-74] <- preProc$numComp }

plot(numPC, c(75:99), xlab = "Number of Principal Components", ylab = "Cumulati
ve Variability %", main = "Variance Explained", pch = 19, col = "blue")
```

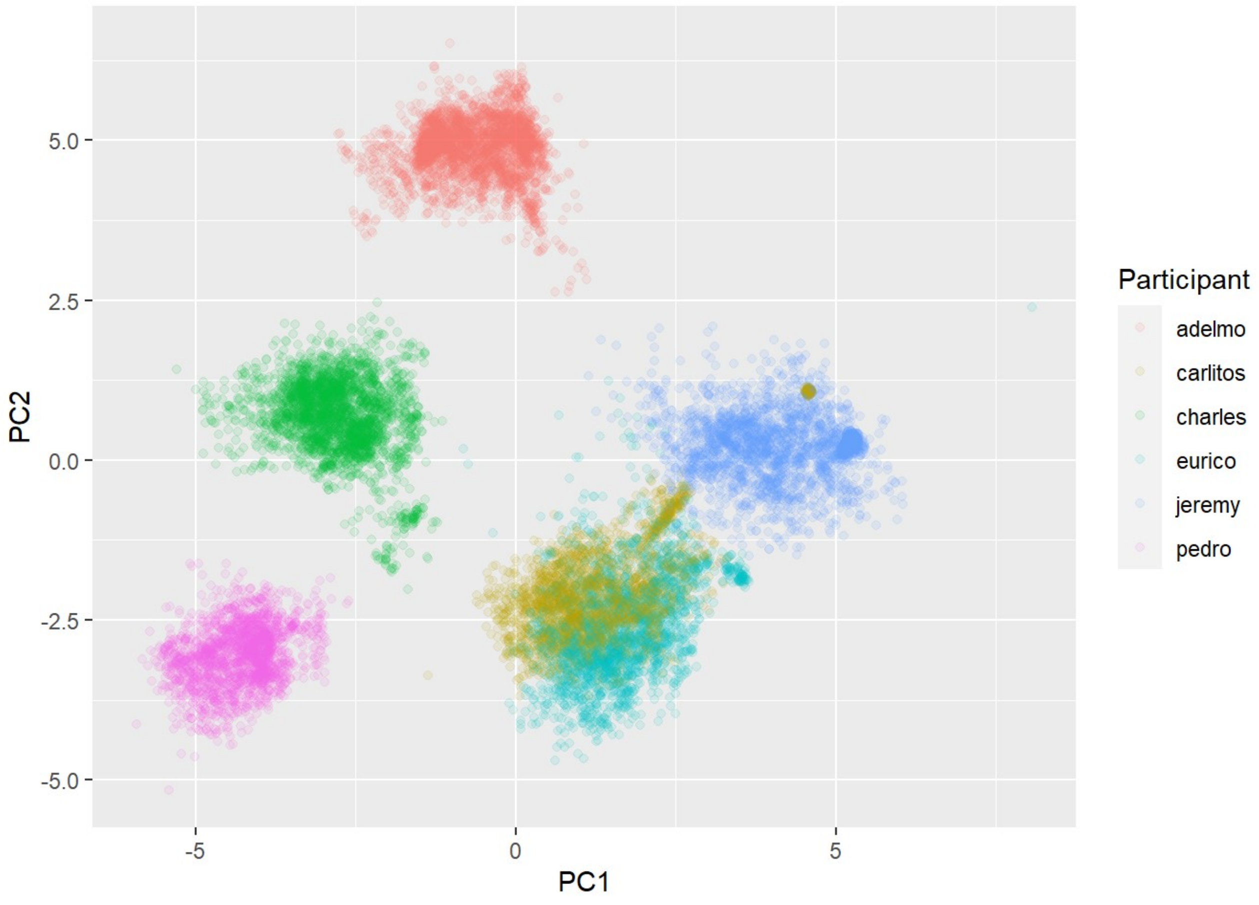


```
print(max(numPC, na.rm = TRUE))
```

```
## [1] 36
```

When threshold is set as 99%. It results in only 36 PCs (i.e. newly created features) left.

```
preProc <- preProcess(train_predictors[,-c(53,54)], method = "pca", thresh = 0.
99)
train_subPC <- predict(preProc, newdata = train_predictors[,-c(53,54)])
ggplot(data.frame(PC1 = train_subPC[,1], PC2 = train_subPC[,2], Participant = t
rain_predictors$user_name),
       aes(x = PC1, y = PC2, col = Participant)) + geom_point(alpha = 0.1)
```



This result clearly shows that participant name is a very important predictor. So we include it as a factor predictor below.

2. We use the random forest (rf) algorithm to train the training set. We use 10-fold cross-validation method for one time to estimate the out-of-sample errors. We also set *mtry*, the number of randomly selected features at each split event, is squared root of total feature number. By default, there are in total *ntrees* = 500 trees generated when applying rf each time.

```
library(foreach)
library(parallel)
library(iterators)
library(doParallel)

cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)

control <- trainControl(method = "cv",
                        number = 10,
                        allowParallel = TRUE)
#Set the number of variables that are randomly selected at each split
mtry <- sqrt(ncol(train_predictors))

set.seed(12345)
rfFit_1 <- train(x = train_predictors[,-53], y = train_predictors$classe,
               method="rf",
               tuneGrid = expand.grid(.mtry=mtry),
               trControl = control)

stopCluster(cluster)
registerDoSEQ()

# In-sample accuracy
print(rfFit_1)
```

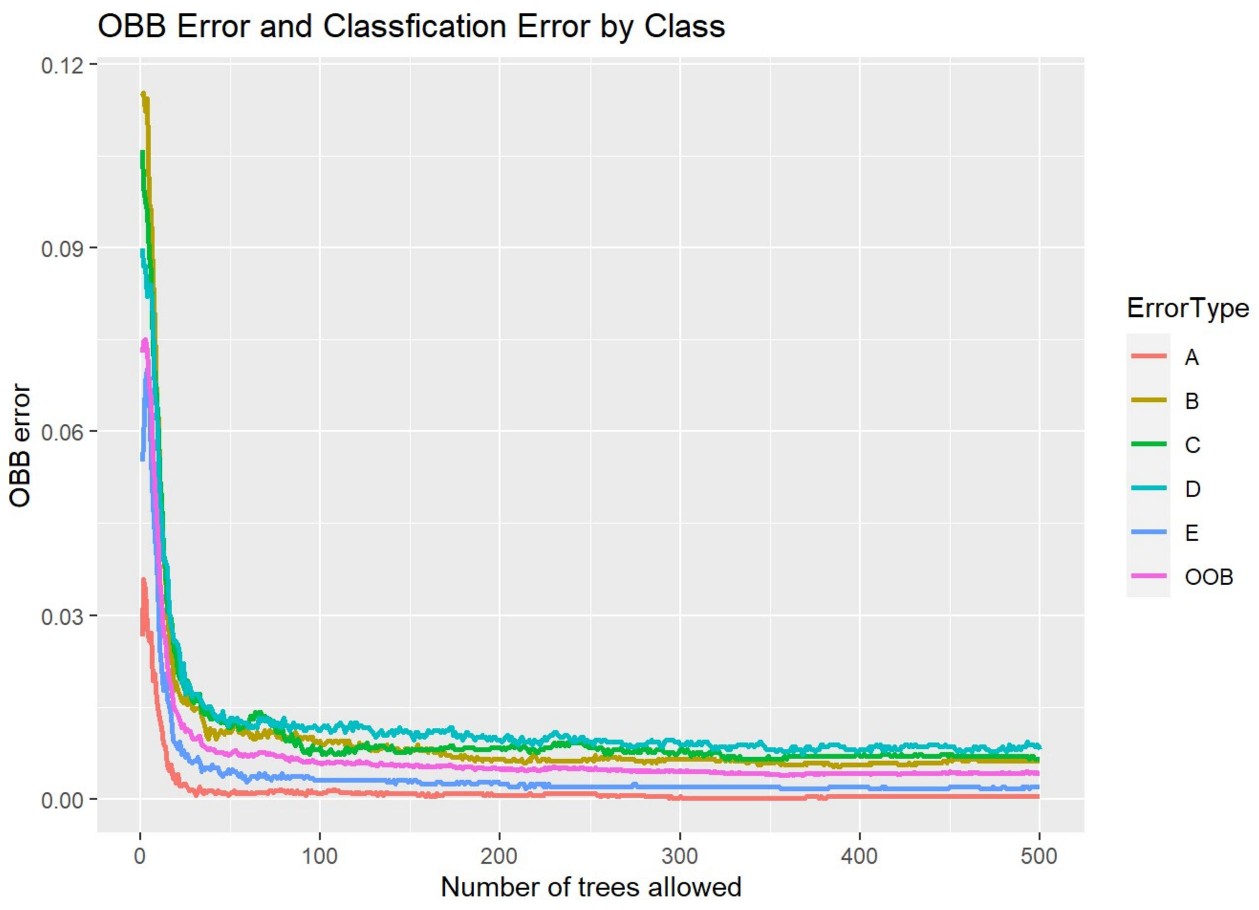
```
## Random Forest
##
## 15699 samples
##    53 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 14129, 14131, 14130, 14129, 14128, 14128, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9943947  0.9929093
##
## Tuning parameter 'mtry' was held constant at a value of 7.348469
```

```
# Out-of-bags OBB error (estimating the out-of-sample error rate)
print(rfFit_1$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: 7
##
##         OOB estimate of  error rate: 0.42%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4462    1    0    0    1 0.0004480287
## B   12 3019    7    0    0 0.0062541145
## C    0   16 2720    2    0 0.0065741417
## D    0    0   19 2552    2 0.0081616790
## E    0    0    0    6 2880 0.0020790021
```

3. Is total ntrees = 500 enough for minimizing OBB error? We plot OBB error and in-class errors by the number of trees in the final model of rfFit_1.

```
rfFit1_final <- rfFit_1$finalModel
oob.error.data <- data.frame(
  Trees=rep(1:nrow(rfFit1_final$err.rate), times=6),
  ErrorType=rep(c("OOB", "A", "B", "C", "D", "E"), each=nrow(rfFit1_final$err.r
ate)),
  Error= c(rfFit1_final$err.rate[,"OOB"],
    rfFit1_final$err.rate[,"A"],
    rfFit1_final$err.rate[,"B"],
    rfFit1_final$err.rate[,"C"],
    rfFit1_final$err.rate[,"D"],
    rfFit1_final$err.rate[,"E"]))

ggplot(data=oob.error.data, aes(x=Trees, y=Error)) + geom_line(aes(color=ErrorT
ype), size =1) + labs(title = "OBB Error and Classfication Error by Class", x
= "Number of trees allowed", y = "OBB error")
```

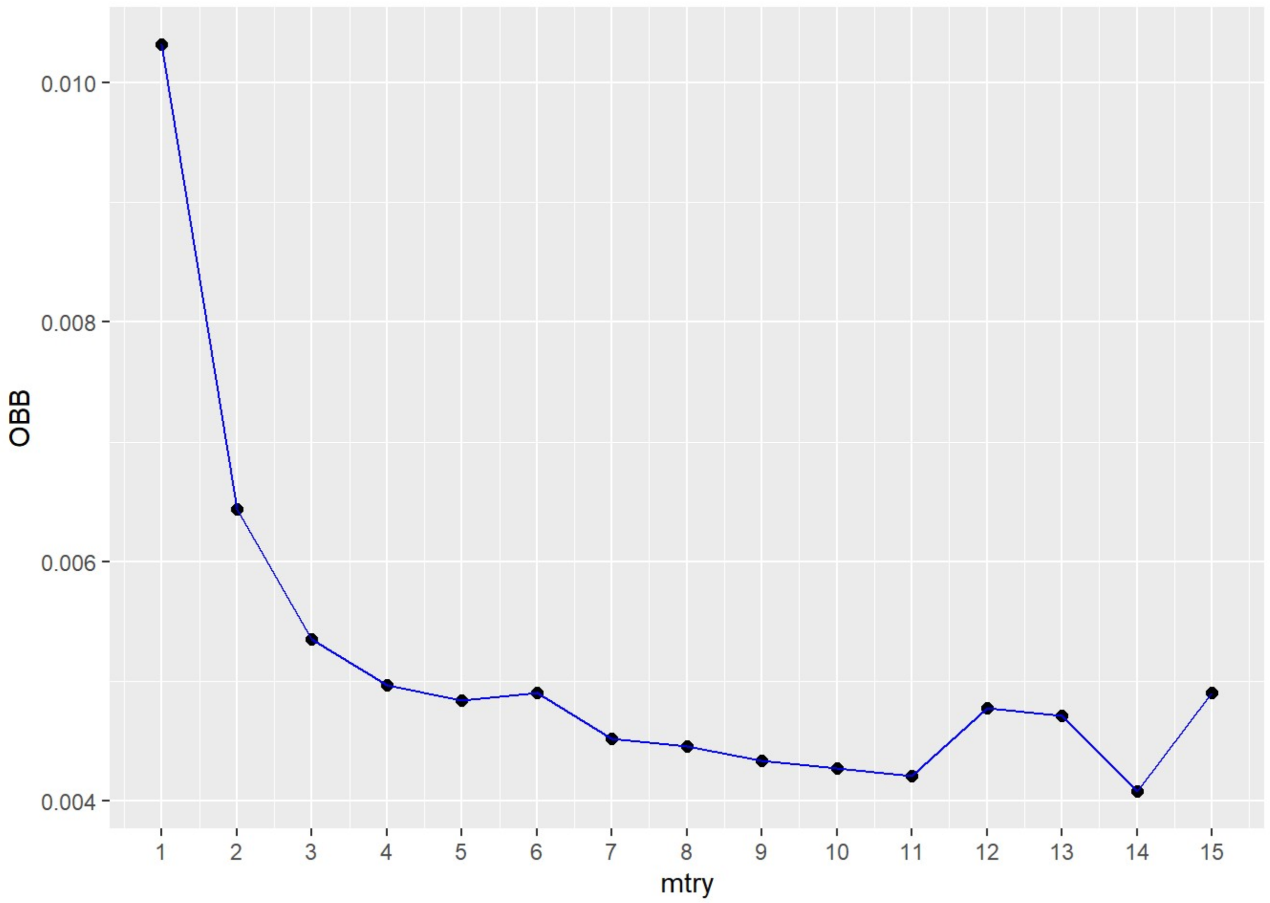


Note that OBB errors become almost flatten after ntrees > 330. So we can continue to use the default ntrees = 500.

4. Is mtry = 7 (as manifested in rfFit_1$finalModel) optimal? We test mtry from 1 to 15, and plot the OBB error and accuracy for algorithm based on every mtry parameter.

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

oob.values <- vector(length = 15)
accuracy.values <- vector(length = 15)
control <- trainControl(method = "cv",
                        number = 10,
                        allowParallel = TRUE)

set.seed(21345)
for(i in 1:15) {
  temp.model <- train(x = train_predictors[,-53], y = train_predictors$classe,
               method="rf",
               tuneGrid = expand.grid(.mtry=i),
               trControl = control)
  # extract the OBB error for ntree=500
  oob.values[i] <- temp.model$finalModel$err.rate[500,1]
  # extract the in-sample accuracy
  accuracy.values [i] <- temp.model$results[2]
}

stopCluster(cluster)
registerDoSEQ()
```

```
g1 <- ggplot(data.frame(mtry = c(1:15), OBB = oob.values), aes(x=mtry, y=OBB))
g1 + geom_point(size = 2) + geom_line(col = "blue") + scale_x_continuous(break
s = 1:15)
```



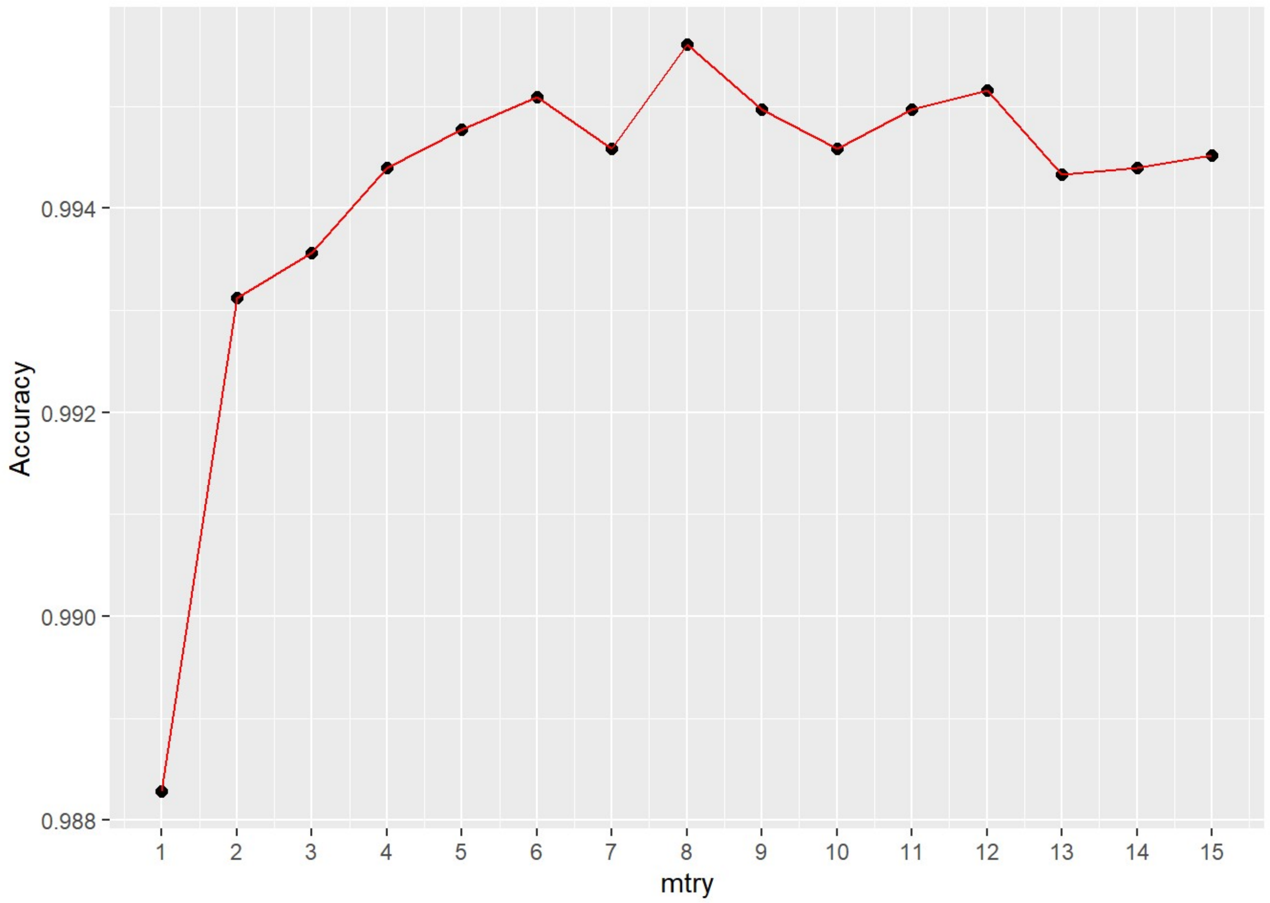
```
## find the minimum error
min(oob.values)
```

```
## [1] 0.004076693
```

```
## find the optimal value for mtry...
which(oob.values == min(oob.values))
```

```
## [1] 14
```

```
g2 <- ggplot(data.frame(mtry = c(1:15), Accuracy = as.numeric(accuracy.value
s)), aes(x=mtry, y=Accuracy))
g2 + geom_point(size = 2) + geom_line(col = "red") + scale_x_continuous(breaks
= 1:15)
```



```
## find the maximum accuracy
max(as.numeric(accuracy.values))
```

```
## [1] 0.9956045
```

```
## find the optimal value for mtry...
which(as.numeric(accuracy.values) == max(as.numeric(accuracy.values)))
```

```
## [1] 8
```

We choose to use mtry = 8 which gives the highest accuracy and very low OBB error.

Finalize the model

Final model is following: 10-fold CV; random forest with mtry = 8, ntrees = 500

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

control <- trainControl(method = "cv",
                        number = 10,
                        allowParallel = TRUE)

set.seed(233)
rfFit_2 <- train(x = train_predictors[,-53], y = train_predictors$classe,
               method="rf",
               tuneGrid = expand.grid(.mtry=8),
               trControl = control)

stopCluster(cluster)
registerDoSEQ()

# In-sample accuracy
print(rfFit_2)
```

```
## Random Forest
##
## 15699 samples
##    53 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 14129, 14127, 14130, 14128, 14130, 14128, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9949674  0.993634
##
## Tuning parameter 'mtry' was held constant at a value of 8
```

```
# Out-of-bags OBB error (estimating the out-of-sample error rate)
print(rfFit_2$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: 8
##
##         OOB estimate of  error rate: 0.42%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4463    1    0    0    0 0.0002240143
## B    9 3022    7    0    0 0.0052666228
## C    0   18 2719    1    0 0.0069393718
## D    0    0   22 2548    3 0.0097162845
## E    0    0    1    4 2881 0.0017325017
```

Validation

We test the accuracy of our final model in the validation data set.

```
validation_predictors <- validation[, -c(1:7,60)]
train_predictors <- mutate_all(validation_predictors, function (x) as.numeric(a
s.character(x)))

#Add the classe type to the 53th column
validation_predictors$classe <- validation$classe
#Add the participant's name to the 54th column
validation_predictors$user_name <- validation$user_name
```

```
confusionMatrix(predict(rfFit_2, newdata = validation_predictors), validation_p
redictors$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    3    0    0    0
##          B    0  754    2    0    0
##          C    0    2  682    5    0
##          D    0    0    0  638    1
##          E    0    0    0    0  720
##
## Overall Statistics
##
##                Accuracy : 0.9967
##                  95% CI : (0.9943, 0.9982)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9958
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9934   0.9971   0.9922   0.9986
## Specificity            0.9989   0.9994   0.9978   0.9997   1.0000
## Pos Pred Value         0.9973   0.9974   0.9898   0.9984   1.0000
## Neg Pred Value         1.0000   0.9984   0.9994   0.9985   0.9997
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1922   0.1738   0.1626   0.1835
## Detection Prevalence   0.2852   0.1927   0.1756   0.1629   0.1835
## Balanced Accuracy      0.9995   0.9964   0.9975   0.9960   0.9993
```

Testing

Apply the model to the test data set.

```
testing_predictors <- test_new[, -c(1:7,60)]
testing_predictors <- mutate_all(testing_predictors, function (x) as.numeric(a
s.character(x)))

#Add the participant's name to the 53th column
testing_predictors$user_name <- test_new$user_name

#predict(rfFit_2, newdata = testing_predictors)  Results not disclosed.
```