Predicting Technique for Barbell Lifts Using Accelerometer Data and Random Forests

Robert Wexler

2/2/2017

Summary

I used random forests to predict technique (i.e. correctly and incorrect) for barbell lifts using data collected from accelerometers located on the belt, forearm, arm, and dumbell of six participants. I cleaned the data set, performed exploratory analyses, fit a number of different models, and evaluated different preprocessing and cross validation methods. The final model, which is a random forest with 10 trees, has an out sample error of 0.9845 to 0.9896 at the 95% confidence interval as predicted by cross validation with five folds.

Exploring and Cleaning Data

First, I wanted to get a sense of the size of the training and testing sets.

```
dim(testing)[1]/dim(training)[1]*100
```

```
## [1] 0.1019264
```

```
dim(testing)[1]+dim(training)[1]
```

```
## [1] 19642
```

The first value shows that the testing set is 0.10% the size of the training set. This is much too small for evaluating model performance. The second value shows that the total number of observations is 19642. This is a medium sample size and, therefore, I will split the data 60/40 into new training and validation sets.

```
set.seed(12345)
inTrain <- createDataPartition(y = training$classe, p = 0.60, list = FALSE)
training_new <- training[inTrain,]
validation_new <- training[-inTrain,]
dim(validation_new)[1]/dim(training_new)[1]*100
```

```
## [1] 66.62704
```

The printed value shows that 67% of the original training data is in the new training set whereas the rest of the data is in the new validation set. Now I will explore the values of the data frame to ensure that there are no missing or empty entries.

```
dim(training_new)
```

```
## [1] 11776   160
```

```
colNA <- names(which(colSums(is.na(training_new)) < 1))
training_new <- training_new[names(training_new) %in% colNA]
validation_new <- validation_new[names(validation_new) %in% colNA]
colEmpty <- names(which(colSums(training_new == "") < 1))
training_new <- training_new[names(training_new) %in% colEmpty]
validation_new <- validation_new[names(validation_new) %in% colEmpty]
```

The printed value shows that there are 11776 observations and 160 predictors. Looking at the first six entries of the new training set (code not shown), it is clear that some columns have many NA values or empty entries. I calculated the number of missing entries in each column and kept only those with zero. This reduced the number of predictors from 160 to 60 with the first seven columns corresponding to the participant and date/time and the other 53 comprising the design matrix. Finally, I looked for predictors that have one unique value.

```
nsv <- nearZeroVar(training_new, saveMetrics = TRUE)
sum(nsv$nzv)
```

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

```
rownames(nsv)[which(nsv$nzv == TRUE)]
```

```
## [1] "new_window"
```

Only one predictor, new_window, had near zero variance but it is not part of the design matrix.

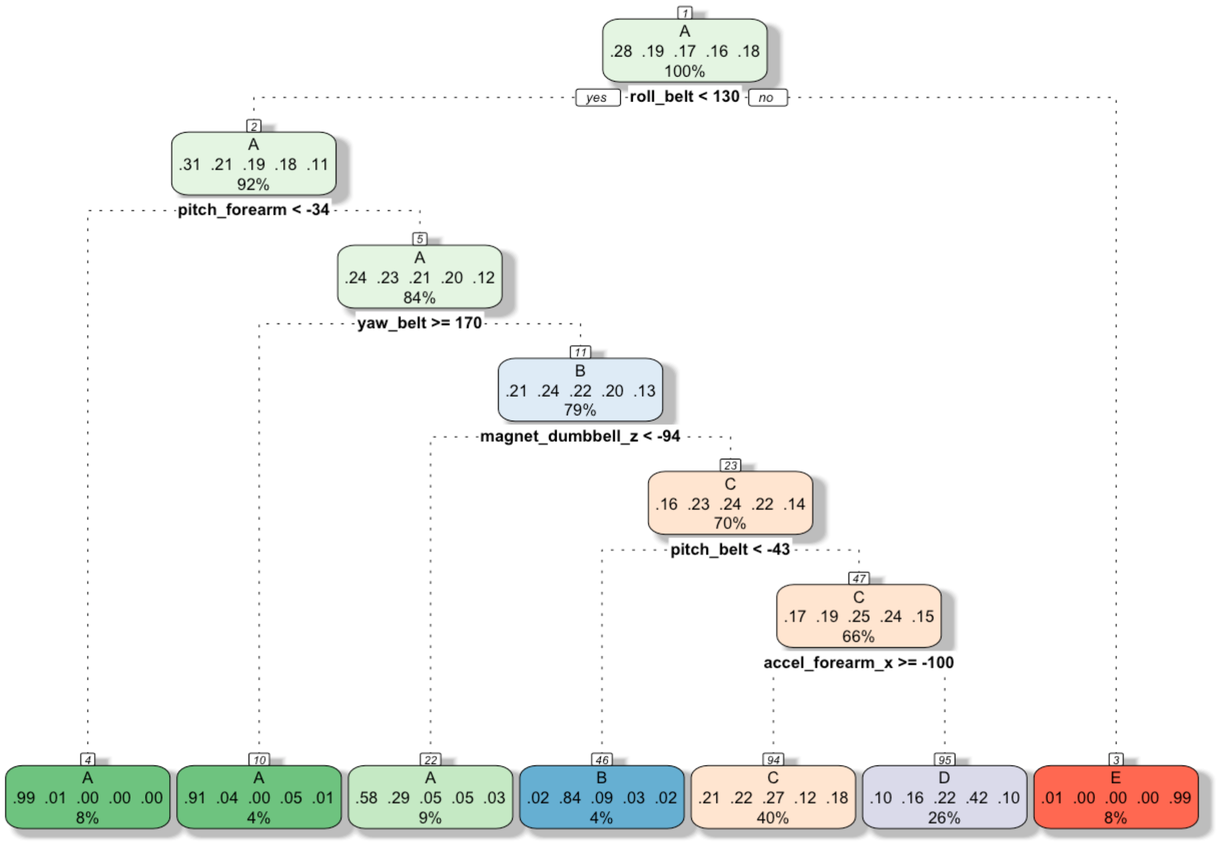
Model Selection

I tested models from three general categories: decision trees/random forests, boosting, and model based prediction (e.g. LDA and Naive Bayes).

Decision Trees and Random Forests

First, I trained a decision tree model with caret's default settings to predict classe using columns 8 to 59 of the new training set.

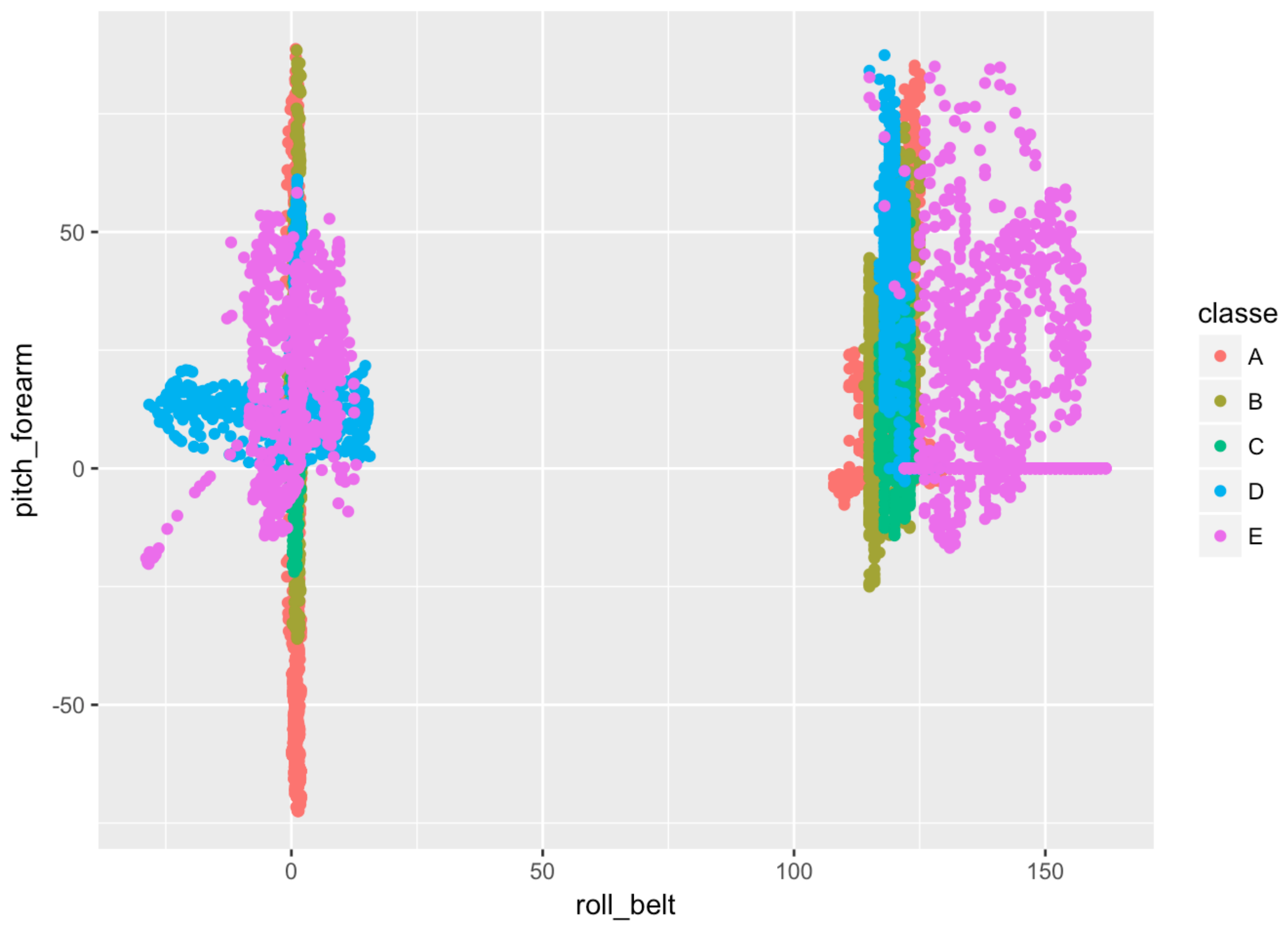
```
dcModel <- train(x = training_new[,8:59], y = training_new[,60], method = "rpart")
fancyRpartPlot(dcModel$finalModel)
```



Rattle 2017-Feb-03 23:13:17 robertwexler

The plot above shows that classe A is well-separated by pitch forearm < -34 and yaw belt >= 170, classe B by pitch belt < -43, and classe E by roll belt >= 130.

```
qplot(roll_belt, pitch_forearm, data = training_new, colour = classe)
```



For example, the plot of roll belt against pitch forearm shows that these two predictors isolate classe E from all others at roll belt values above 130. Next, I evaluated the performance of my decision tree on the validation data.

```
dcPrediction <- predict(dcModel, validation_new)
confusionMatrix(dcPrediction, validation_new$classe)[3][[1]][3:4]
```

```
## AccuracyLower AccuracyUpper
##     0.4882374     0.5104885
```

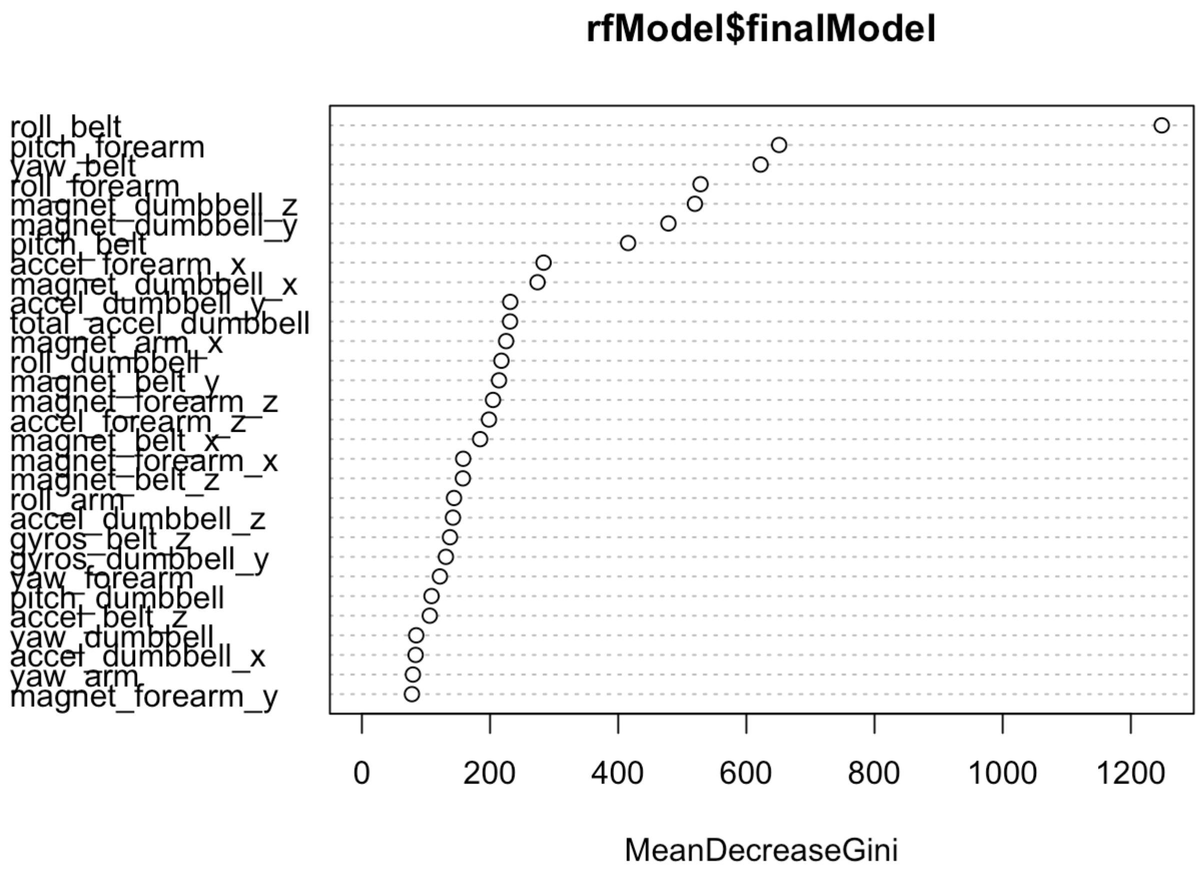
The accuracy of this model is 0.4882 to 0.5105 at the 95% confidence interval when trained on the validation set. Hereafter, this is how I will estimate out sample error.

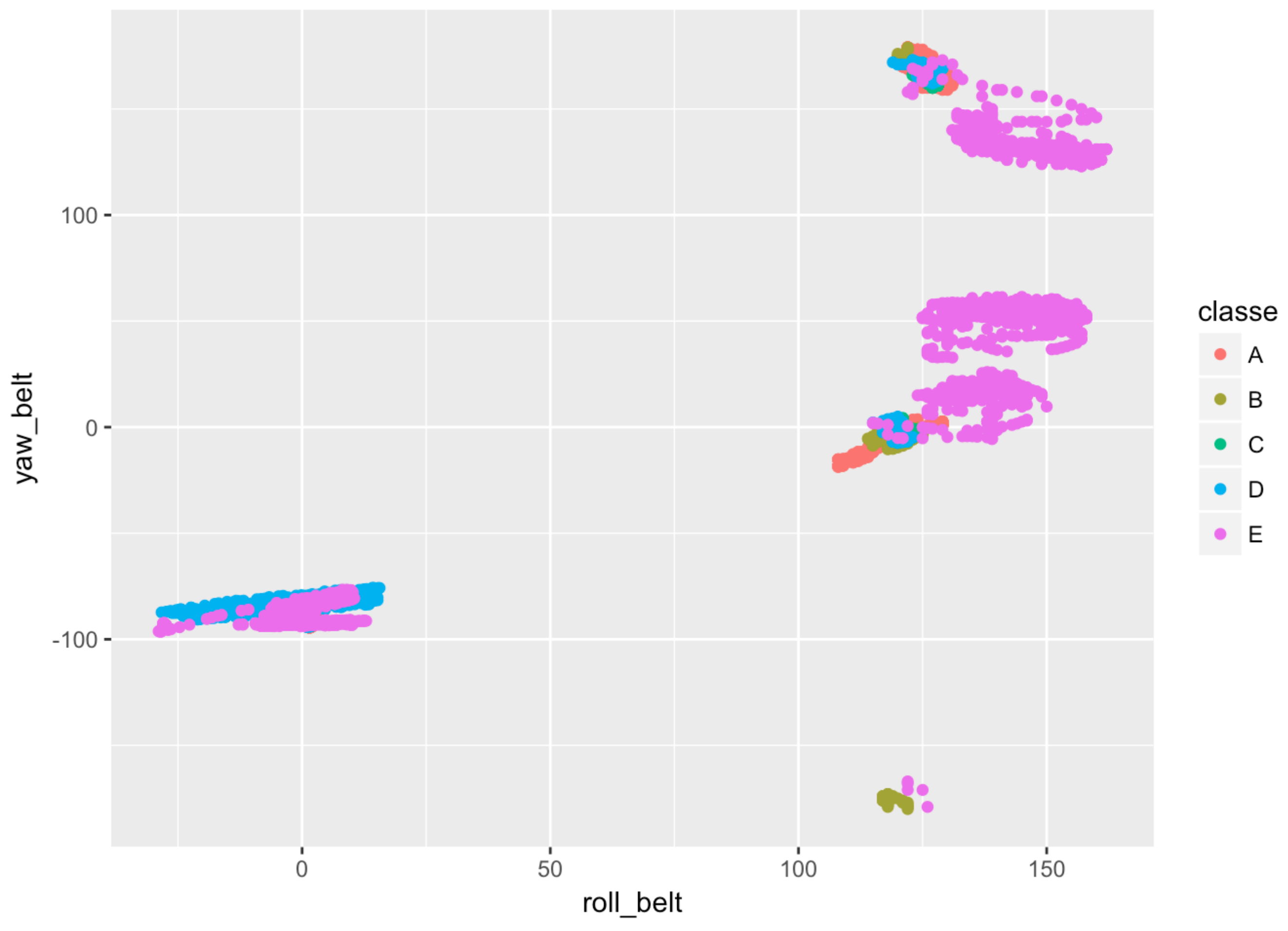
Next, I trained a random forest model with five decision trees and without resampling.

```
rfModel <- train(x = training_new[,8:59], y = training_new[,60], method = "rf", proximit
y = TRUE, ntree = 5,
                 tuneLength = 1, trControl = trainControl(method = "none"))
rfPrediction <- predict(rfModel, validation_new)
confusionMatrix(rfPrediction, validation_new$classe)[3][[1]][3:4]
```

```
## AccuracyLower AccuracyUpper
##     0.9752743     0.9817938
```

The accuracy of this model is 0.9758 to 0.9823 at the 95% confidence interval when trained on the validation set. I quantified the relative importance of each variable using the mean decrease in the Gini coefficient, which measures how much each variable contributes to the homogeneity of the nodes and leaves.


rfModel$finalModel
roll_belt
pitch_forearm
yaw_belt
roll_forearm
magnet_dumbbell_z
magnet_dumbbell_y
pitch_belt
accel_forearm_x
magnet_dumbbell_x
accel_dumbbell_y
total_accel_dumbbell
magnet_arm_x
roll_dumbbell
magnet_belt_y
magnet_forearm_z
accel_forearm_z
magnet_belt_x
magnet_forearm_x
magnet_belt_z
roll_arm
accel_dumbbell_z
gyros_belt_z
gyros_dumbbell_y
yaw_forearm
pitch_dumbbell
accel_belt_z
yaw_dumbbell
accel_dumbbell_x
yaw_arm
magnet_forearm_y
0
200
400
600
800
1000
1200
MeanDecreaseGini


100
0
-100
yaw_belt
0
50
100
150
roll_belt
classe
A
B
C
D
E

These plots shows that roll and yaw belt provide the most separation between observations in different classes, followed by roll forearm and magnet dumbell z and y.

Other Models

I also evaluated the performance of gradient boosting, LDA, and Naive Bayes without resampling.

```
gbmModel <- train(x = training_new[,8:59], y = training_new[,60], method = "gbm", verbos
e = FALSE,
                  tuneLength = 1, trControl = trainControl(method = "none"))
gbmPrediction <- predict(gbmModel, validation_new[,8:59])
confusionMatrix(gbmPrediction, validation_new$classe)[3][[1]][1]
```

```
##  Accuracy
## 0.7561815
```

```
ldaModel <- train(x = training_new[,8:59], y = training_new[,60], method = "lda",
                  tuneLength = 1, trControl = trainControl(method = "none"))
ldaPrediction <- predict(ldaModel, validation_new[,8:59])
confusionMatrix(ldaPrediction, validation_new$classe)[3][[1]][1]
```

```
##  Accuracy
## 0.7039256
```

```
nbModel <- naiveBayes(x = training_new[,8:59], y = training_new[,60])
nbPrediction <- predict(nbModel, validation_new[,8:59])
confusionMatrix(nbPrediction, validation_new$classe)[3][[1]][1]
```

```
##  Accuracy
## 0.4869997
```

The three printed values correspond to the accuracy of the gradient boosting, LDA, and Naive Bayes models respectively on the validation set. Since random forests without resampling provide more than 20% more accuracy when trained on the validation set than other models, I selected it as my prediction method.

Model Optimization

I attempted to optimize my random forest model using data preprocessing techniques and cross validation.

Data Preprocessing

First, I standardized the new training set to ensure that variables measured at different scales contribute equally to the analysis. The caret package can simulataneous standardize the new training set and train a random forest with five trees. For the standarization, I centered and scaled each variable.

```
rfModelStd <- train(x = training_new[,8:59], y = training_new[,60], method = "rf", proxi
mity = TRUE, ntree = 5,
                         tuneLength = 1, trControl = trainControl(method = "none"), preP
rocess = c("center", "scale"))
rfStdPrediction <- predict(rfModelStd, validation_new)
confusionMatrix(rfStdPrediction, validation_new$classe)[3][[1]][3:4]
```

```
## AccuracyLower AccuracyUpper
##     0.9741814     0.9808485
```

The accuracy of this model is 0.9761 to 0.9825 at the 95% confidence interval when trained on the validation set. This is not an improvement on the original model without standardization. As such, I will proceed without standardizing the data.

Next, I used principal components analysis to reduce the number of the variables in my design matrix to a set of orthogonal variables that explain 95% of the variance.

```
rfModelPCA <- train(x = training_new[,8:59], y = training_new[,60], method = "rf", proxi
mity = TRUE, ntree = 5,
                    preProcess = "pca", tuneLength = 1,
                    trControl = trainControl(method = "none", preProcOptions = list(thre
sh = 0.95)))
rfPCAPrediction <- predict(rfModelPCA, validation_new)
confusionMatrix(rfPCAPrediction, validation_new$classe)[3][[1]][3:4]
```

```
## AccuracyLower AccuracyUpper
##     0.8978204     0.9109520
```

The accuracy of this model is 0.9086 to 0.9211 at the 95% confidence interval when trained on the validation set. This is not bad but worse than the original model likely because the variation left unexplained (5%) was important in the prediction. As such, I will proceed without performing principal components analysis.

Cross Validation

Finally, I used cross validation to train a random forest model with less bias. For the cross-validation, I split the data into five and ten folds.

```
rfModelCV5 <- train(x = training_new[,8:59], y = training_new[,60], method = "rf", proxi
mity = TRUE, ntree = 5,
                    tuneLength = 1, trControl = trainControl(method = "cv", number=5))
rfCV5Prediction <- predict(rfModelCV5, validation_new)
confusionMatrix(rfCV5Prediction, validation_new$classe)[3][[1]][3:4]
```

```
## AccuracyLower AccuracyUpper
##     0.9743179     0.9809668
```

```
rfModelCV10 <- train(x = training_new[,8:59], y = training_new[,60], method = "rf", prox
imity = TRUE, ntree = 5,
                     tuneLength = 1, trControl = trainControl(method = "cv", number=10))
rfCV10Prediction <- predict(rfModelCV10, validation_new)
confusionMatrix(rfCV10Prediction, validation_new$classe)[3][[1]][3:4]
```

```
## AccuracyLower AccuracyUpper
##     0.9747276     0.9813214
```

The accuracy of the model with five folds is 0.9738 and 0.9805 whereas that of the model with ten folds is 0.9705 to 0.9776 at the 95% confidence interval when trained on the validation set. This is not an improvement on the original model without cross validation. However, in order to ensure that my model is not baised towards predictions on the validation set, I included cross validation with five folds in my final model.

Final Model

For the final model, I trained a random forest with 10 decision trees and performed cross validation with five folds.

```
rfFinalModel <- train(x = training_new[,8:59], y = training_new[,60], method = "rf", pro
ximity = TRUE, ntree = 10,
                      tuneLength = 1, trControl = trainControl(method = "cv", number=5))
rfFinalPrediction <- predict(rfFinalModel, validation_new)
confusionMatrix(rfFinalPrediction, validation_new$classe)[3][[1]][3:4]
```

```
## AccuracyLower AccuracyUpper
##     0.9845197     0.9896182
```

This accuracy of my final model, i.e. the expected out of sample error, is 0.9845 to 0.9896 at the 95% confidence interval when trained on the validation set. This is an impressive improvement the decision tree model (0.4882 to 0.5105)!