Practical Machine Learning Project:

Mark Anderson

March 20, 2016

Summary

Wearable activity monitors are becoming pervasive in today's society. Fitbit, Jawbone, Nike FuelBand, etc. are recording activity levels for many people. With these devices, significant amounts of data about activity are available that could potentially be used to monitor and improve personal health. These devices, however, are only recording the amount of activity, and they do not record nor quantify the quality of the activity (e.g. if the exercise activity was conducted properly). A set of data was collected (Velloso et. al, Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.) for Six healthy participants performing a set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E). Class A is the correct execution of the exercise, and the other classes are commonly found incorrect methods which yield poorer fitness results. In this exercise, accelerometer data from wearable devices is analyzed to determine if one can predict if the dumbbell curl is being performed correctly based on data collected from a fitness monitor; and, if the exercise is not being executed correctly, what the error in the technique is to provide feedback to the subject.

The data are fit to two models, a simple tree and a random forest model using all the variables from the personal activity monitor. From these models, the Tree model had an accuracy of 49% and the Random Forest model had an accuracty of 99%. The better random forest model is then evaluated to see if reducing the number of variables used can simplify the model without sacrificing the accuracy of the prediction. From this analysis, using only 25 of the variables to fit the model lowers the accuracy slightly, but we gain computational efficiency (reducing the time of the model generation by 50%). When using the Random Forest method to model the data, an out of sample error rate of 0.41% is found.

Model Generation

The data is downloaded from the coursera course web-site to a local directory. From there, it is loaded into the R environment, and the columns with no data or NA values are deleted from the training set. The training data set is separated into a training dataset and a validation data set that are used to fit a model to the data and validate the accuracy of the model. The model can then be used to make predictions with the test data.

```
##Read in the training dataset
##
pmltraining <- read.csv("./pml-training.csv", na.strings=c("NA", ""))
##remove the columns with NA values
##
pmltraining <- pmltraining[colSums(is.na(pmltraining))==0]
##remove the columns that contain identification data
##
pmltraining <- pmltraining[,-(1:7)]
##
## Determine if any of the variables in the data are correlated by calculating the near Zero
## Variance.  If values are correlated (by having a variance that is near zero),
```

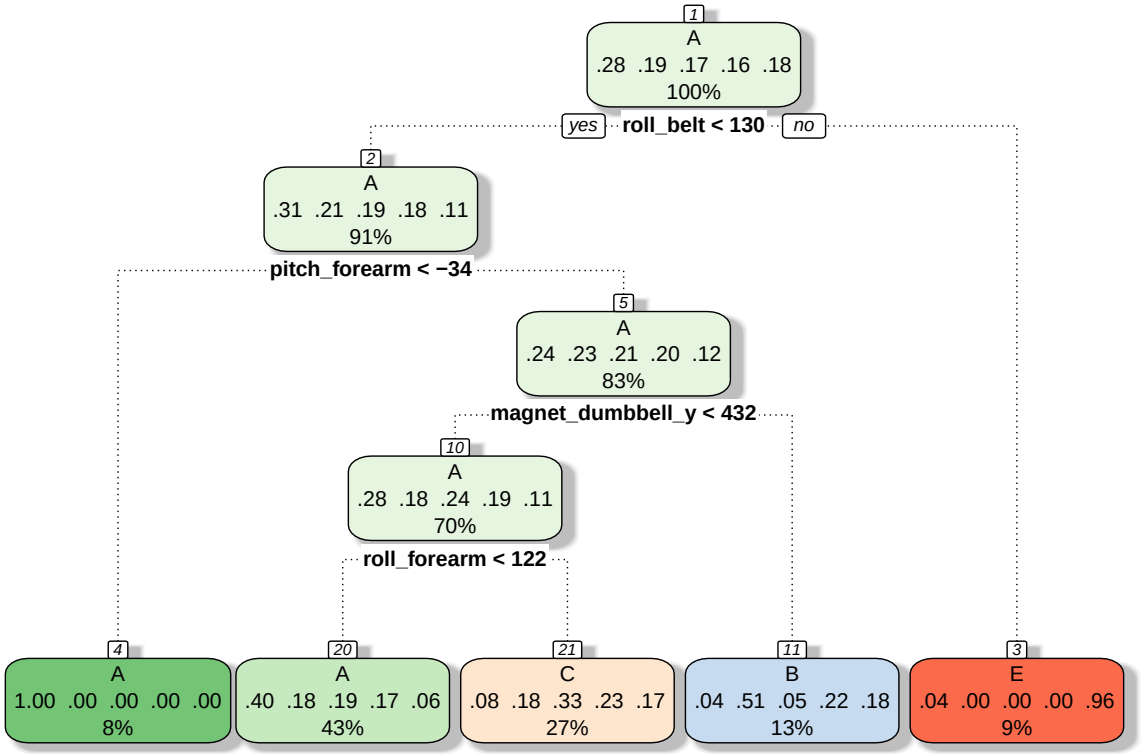
```
## that variable will be deleted from the dataset that the model will be calculated from.
##
nzv <- nearZeroVar(pmltraining[,-53])
if(length(nzv)>0) pmltraining <- pmltraining[,-nzv]
##
##Separate the pmltraining data into a training set and a test
##set for model validation
##
inTrain <- createDataPartition(pmltraining$classe, p=0.7, list=FALSE)
training <- pmltraining[inTrain,]
pmlvalid <- pmltraining[-inTrain,]
```

Because there are no variables that are strongly correlated to each other, no variables are deleted from the model.

Tree Model

Once the training set is established, two models using different methods are fit to the data. First, a simple tree model is applied to the data.

```
set.seed(1234)
tree_mod <- train(as.factor(classe) ~ ., method="rpart", data=training)
fancyRpartPlot(tree_mod$finalModel)
```



```
##
## test the model against the pmlvalid (validation) data set
##
tree_predict <- predict(tree_mod, pmlvalid)
tree_conf <- confusionMatrix(tree_predict, pmlvalid$classe)
```

From the fit of the model to the validation dataset, we can extract the overall accuracy of this model and the confusion matrix to visually see the ability of this method for predicting the correct exercise type.

```
tree_conf[2]
```

```
## $table
##           Reference
## Prediction    A    B    C    D    E
##          A 1510  471  478  415  121
##          B   32  398   30  195  142
##          C  108  270  518  354  298
##          D    0    0    0    0    0
##          E   24    0    0    0  521
```

```
tree_conf$overall[1]
```

```
##  Accuracy
## 0.5007647
```

The simple tree model had an overall prediction accuracy of 50.08%. The out of sample error rate is estimated by comparing the prediction to the actual class (A,B,C,D, or E) in the validation dataset. The out of sample error rate is found to be 49.92%. This accuracy was not good as the model only predicts the correct exercise class 50% of the time.

Random Forest

A random forest model was next applied to the dataset to see if it would lead to an improvement in prediction accuracy. For the Random Forest model, K-fold cross-validation is utilized.

```
set.seed(5678)
control <- trainControl(method="cv", number=5)
ptm <- proc.time()
rf_mod <- randomForest(as.factor(classe) ~ ., data=training, trControl=control, ntree=1000)
tm1 <- proc.time()-ptm
##
## Measure the processing time to compare computational efficiencey of
## different models
##
tm1
```

```
##    user  system elapsed
##  114.09    1.36  115.80
```

```
rf_mod
```

```
##
## Call:
##  randomForest(formula = as.factor(classe) ~ ., data = training,      trControl = control, ntree = 10
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.5%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3901    4    0    0    1 0.001280082
## B   17 2637    4    0    0 0.007900677
## C    0   11 2383    2    0 0.005425710
## D    0    0   21 2229    2 0.010213144
## E    0    0    2    5 2518 0.002772277
```

```
##
## test the model against teh pmlvalid (validation) dataset
##
rf_predict <- predict(rf_mod, newdata=pmlvalid)
rf_conf <- confusionMatrix(rf_predict, pmlvalid$classe)
```

```
err1 <- pmlvalid$classe == rf_predict
mra <- length(err1[err1!=TRUE])/length(pmlvalid$classe)
```

From the fit of the model to the validation dataset, we can extract the overall accuracy of this model and the confusion matrix to visually see the ability of this method for predicting the correct exercise type.

```
rf_conf[2]
```

```
## $table
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    8    0    0    0
##          B    0 1124    2    0    0
##          C    0    7 1024    4    0
##          D    0    0    0  959    6
##          E    0    0    0    1 1076
```

```
rf_conf$overall[1]
```

```
##  Accuracy
## 0.9952421
```

The random forest model had an overall prediction accuracy of 99.52%. This accuracy was much higher than that found for the simple tree model. For the model calculated using the Random Forest method, the out of sample error rate is 0.4758%.

Combined tree and random forest models

If we combine the two models, will that lead to better prediction accuracy? This can easily be tested.

```
predDF <- data.frame(tree_predict, rf_predict, classe=pmlvalid$classe)
combModFit <- randomForest(as.factor(classe) ~ ., data=predDF, ntree=1000)
combPred <- predict(combModFit, newdata=predDF)
comb_conf <- confusionMatrix(combPred, predDF$classe)
comb_conf[2]
```

```
## $table
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    8    0    0    0
##          B    0 1124    2    0    0
##          C    0    7 1024    4    0
##          D    0    0    0  959    6
##          E    0    0    0    1 1076
```

```
comb_conf$overall[1]
```

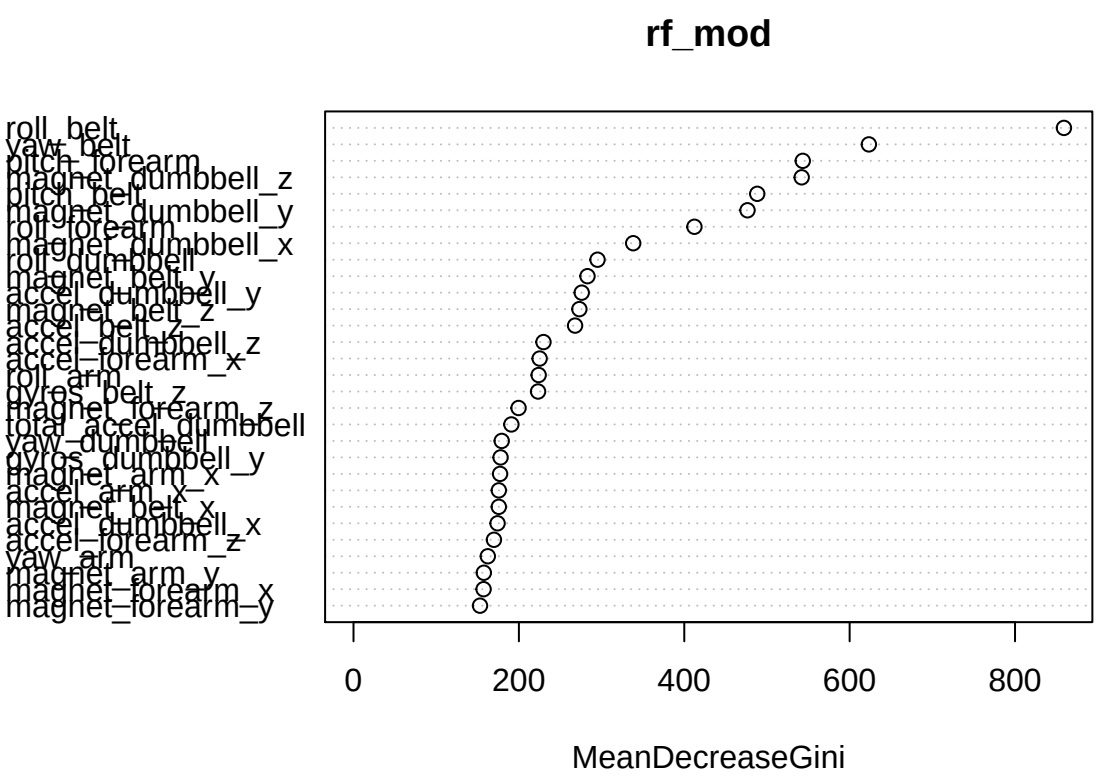
```
##  Accuracy
## 0.9952421
```

From the confusion matrix and the prediction accuracy (99.52%) of the combined model, there is no significant value in the added computational complexity for using the combined model for prediction.

Examining the Importance of Variables on the Model fit

Finally, I consider simplifying the model to prevent overfitting by eliminating covariants from the dataset that have a small impact on the model fit. Uisng the random forest fit (rf_mod) and using the impVar() function, the importance of each variable in the dataset to the model is determined. This can be shown graphically using the varImpPlot() function.

```
## Determine the relative importance of the different variables to the model
##
imp_rf_obj <- varImp(rf_mod)
varImpPlot(rf_mod, sort=TRUE)
```




```
## Remove from the training dataset those variables of low importance to the model generation
##
impThreshold <- quantile(imp_rf_obj$Overall, 0.5)
impFilter <- imp_rf_obj$Overall >= impThreshold
training1 <- training[,impFilter]
##
set.seed(1357)
ptm <- proc.time()
## create a model using the random forest
##
rf_model2 <- randomForest(as.factor(classe) ~ ., data=training1, trcontrol=control, ntree=1000)
tm2 <- proc.time() - ptm
tm2
```

```
##    user  system elapsed
##   53.13    1.14   54.61
```

```
imp_predict <- predict(rf_model2, newdata=pmlvalid)
imp_conf <- confusionMatrix(imp_predict, pmlvalid$classe)
imp_conf[2]
```

```
## $table
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    8    0    0    0
```

```
##          B    1 1127    1    1    0
##          C    0    4 1024    3    0
##          D    1    0    1  960    7
##          E    0    0    0    0 1075
```

```
imp_conf$overall[1]
```

```
##  Accuracy
## 0.9954121
```

By considering the variable importance using the varImp function, the number of variables that are fit in the model is reduced from 52 to 26. This results in a computational time difference of 115.8 seconds for the model with all the accelerometer variables included compared to 54.61 seconds when only those variables that have an importance to the model above a certain threshold value - set to the 50th percentile of all those variables in the initial model.

Conclusions

Using the model determined by the Random Forest method, predictions of the exercise class from a new set of data are made. From the model, the 20 exercises of the testing set fall in these catagories

```
## Read in the test data
##
testing <- read.csv("./pml-testing.csv", na.strings=c("NA", ""))
test_predict <- predict(rf_mod, newdata=testing)
test_predict
```

```
##  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
```

The Random Forest model works best for fitting this data. The out of sample error is estimated to be less than 0.5%.