A Preliminary Attempt at Qualitative Activity Recognition Using Practical Machine Learning

By James Lamb

I. Synopsis

This is a brief (in content) but verbose (in code) project prepared in partial fulfillment of the requirements for the Practical Machine Learning course offered by Johns Hopkins University via Coursera, as part of the JHU Data Science Specialization.

II. Downloading and Preprocessing the Data

A. Downloading the Data The data for this project contain qualitative activity ratings and exercise sensor data from the Human Activity Recognition project at LES. The collection process and the researchers' approach to prediction are defined in this paper.

```
## Download and Read The Data

    ## Training Data
    url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
        file <- ".\\pml_training.csv"
        if(file.exists(file)==FALSE) {
            download.file(url, destfile = file)
        }
        ## read in the data
        traindata <- read.csv(file, header = TRUE, sep=",")

    ## Test Data
    url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
        file <- ".\\pml_testing.csv"
        if(file.exists(file)==FALSE) {
            download.file(url, destfile = file)
        }
        ## read in the data
        testdata <- read.csv(file, header = TRUE, sep=",")
```

B. Preliminary Rationalization As in any data analysis, before applying any statistical techniques I took a look at the data and did some cleaning. First, I removed non-relevant or unclear information. Even after reading the supporting paper, I did not understand what the "new_window" or "num_window" variables were capturing. Because I wanted to build a generalizable algorithm that could be applied to the test set, I also removed the time stamps. I did not see any clear exploitable time-series patterns in the way they were structured, and I wanted to create an algorithm built exclusively on directly relevant concepts (i.e. exercise measurements). I also removed the observation identifier (x). I left the problem_id variable in the test data set, but in training the model I take care to exclude it from the set of candidate predictors.

```
## subset out the series we won't use as predictors
traindata <- traindata[,8:160]
testdata <- testdata[,8:159]
```

C. Diagnosing Severity of Missing Values At this point, there was still a lot of data cleaning to be done. Many of the variables were plagued by NA's or blank values. Before deciding which should be cleaned with imputation and which should be dropped entirely from the predictor set, I sought to estimate the severity of the missing data problem.

```
## Count NA values in each column of the dataframe
vars_na <- rep(0, ncol(traindata))
vars_blnk <- rep(0, ncol(traindata))

## loop through variables, calculate proportion NA or blank
for (i in 1:ncol(traindata)){vars_na[i] <- round(100*sum(is.na(traindata[,i]))/nrow(traindata),2) ##prop
                             vars_blnk[i] <- round(100*sum(traindata[,i]=="")/nrow(traindata),2) ## pr
                             }

## create data frame of variable names, and proportion of obs NA or blank
miss_sum <- data.frame(names(traindata),vars_na, vars_blnk)
names(miss_sum) <- c("variable", "NA_prop", "blank_prop")

## take a look at the missing counts (commented out here)
## miss_sum
```

After examining the data, it was clear that there were no marginal cases (just a few missing/NA values). Those variables which suffered from quality issues tended to have about 97% missing values. These were clearly not suitable candidates as predictors, so I took steps to remove them from the dataset.

```
## get list of variable names where there were lots of missings
miss_sum[is.na(miss_sum)] <- 1000 ## replace the NAs with something numeric
miss_sum$tot <- miss_sum$NA_prop + miss_sum$blank_prop
namecheck <- miss_sum[miss_sum$tot > 0,]$variable ## put variable names in a vector for looping

## get column index of these variables in the training data
indx <- 0 ## initialize the list

## get the column index
for(i in 1:length(namecheck)){indx <- c(indx,grep(paste("^", namecheck[i],"$",sep=""),names(traindata))

## remove missings-plagued series from training data
traindata <- traindata[,-indx]

## count number of remaining predictors (excluding the outcome variable classe)
numpred <- ncol(traindata)-1
```

This step eliminated a large portion of the training set, leaving 52 predictors.

D. Column Classes After these qualitative steps, I turned my attention to the quantitative features of the remaining predictors. To facilitate this and to prepare the traning data for prediction later on, I converted the "classe" variable into a factor and the exercise measurements to numerics.

```
## change classe to a factor
traindata$classe <- as.factor(traindata$classe)

## Change exercises to numeric
for (i in 8:ncol(traindata)){ traindata[,i] <- as.numeric(traindata[,i])}
for (i in 8:ncol(testdata)){ testdata[,i] <- as.numeric(testdata[,i])}
```

Next, I moved on to feature selection. The authors of the support paper use a correlation-based algorithm to identify key features. I decided to employ a more conservative (data-preserving) filter. I used the `nearZeroVar()` function in the `caret` package to identify the predictors in the training dataset with zero or near-zero variance. Models are fit on the basis of observed covariance, which by definition requires non-negligible variance in the predictors.

```
## load the caret package
library(caret)

## loop through training variables (just predictors, identify the ones with near-zero variance
sm_var <- nearZeroVar(traindata[,-53], saveMetrics=TRUE)

## get the subset of sm_var for which variance was zero
dels <- sm_var[sm_var$nzv==TRUE,]

## count how many features will be deleted
delete_num <- nrow(dels)
```

R did not identify any near-zero variance predictors, so this step left the training data unchanged.

E. Compatibility with Test Set As a final step, I removed all variables from the test set which were not in the newly-rationalized training set.

```
## initialize the list
indx <- 0

## get column index of the traindata variables in the testdata
namecheck <- names(traindata)
for(i in 1:length(namecheck)){indx <- c(indx,grep(paste("^", namecheck[i],"$",sep=""),names(testdata)))

## remove variables from testing data that are no longer in training data
testdata <- testdata[,indx]
```

With the preprocessing done, I moved on to the task at hand: prediction of exercise quality using sensor data.

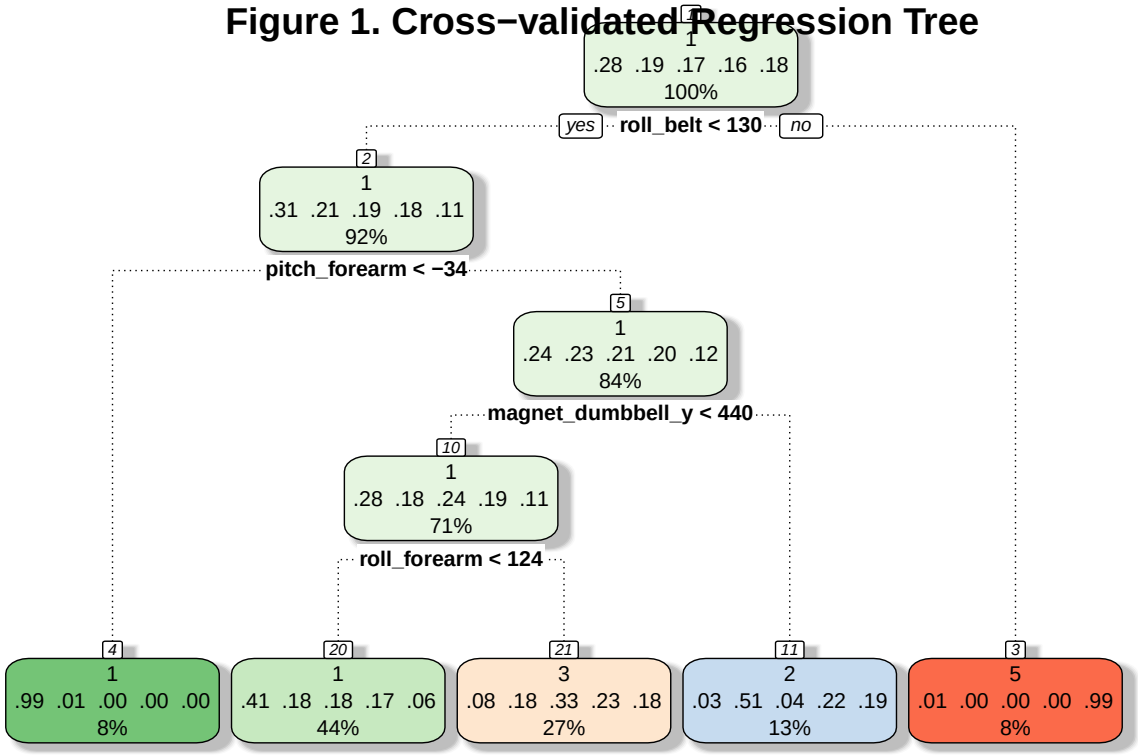
III. Training

A. Regression Trees As a first step, I estimated a CART model, using 15 bootstrapped samples to get a cross-validated estimate of out-of-sample error. The final tree structure is shown below.

```
## set cross-validation tuning parameters --> 15 bootstraped samples
set.seed(445)
ctrl1 <- trainControl(method = "boot", number = 15) ## n=50 --> 6m47s | n=15 --> 1m55s
mod1 <- train(as.factor(classe) ~., data=traindata, method="rpart", trControl = ctrl1) ## run time: 1m 5

## Prettier Tree Plot with rattle package
library(rattle)
fancyRpartPlot(mod1$finalModel, main = "Figure 1. Cross-validated Regression Tree", sub="")
```

Figure 1. Cross−validated Regression Tree



As you can see in the plot, a majority portion of the data reside in the middle three terminal nodes, which are far from homogeneous. As is implied by this picture, the cross-validated accuracy of the tree model is not particularly impressive. To see exact estimates, I printed the results of the estimation.

```
### print summary of the model
mod1
```

```
## CART
##
## 19622 samples
##    52 predictor
##     5 classes: '1', '2', '3', '4', '5'
##
## No pre-processing
## Resampling: Bootstrapped (15 reps)
##
## Summary of sample sizes: 19622, 19622, 19622, 19622, 19622, 19622, ...
##
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa       Accuracy SD  Kappa SD
##   0.03567868  0.4888818  0.32753383  0.06994785   0.11369581
##   0.05998671  0.3994215  0.18135161  0.06016672   0.09913664
##   0.11515454  0.3373230  0.08183904  0.04159063   0.05993399
##
## Accuracy was used to select the optimal model using  the largest value.
```

```
## The final value used for the model was cp = 0.03567868.
```

Cross-validated accuracy of 48.9% (error rate = 51.1%) is not horrendous. With five factors to choose from, random guessing would produce an accuracy of 20% (error rate = 80%). Nonetheless, I felt that I could do better and so moved on to alternative approaches.

B. Random Forest Looking at these preliminary results, I surmised that the simplicity of CART might be poorly suited to the rather involved classification problem at hand (5 classes, 52 predictors). In an attempt to address this issue, I next turned to a much deeper tree-based classification method: random forests.

Rather than searching for a single best tree, random forests estimate many many trees and choose the "majority vote" classification of their constituents. They are, fundamentally, combinations of bagging (upweighting misclassification to draw the algorithm's attention) and decision trees (finding thresholds of predictors at which cases switch class). By building many trees with random subsets of features, the overfitting problems from simpler tree-based models are greatly reduced.

I first attempted to use a random forest in `caret`, but this was very computationally inefficient. As a result, I instead opted for the `randomForest` package. Because bootstrapped sampling is already included in the construction of the forests, I did not need to explicitly write in any cross-validation steps. The out-of-bag (OOB) error, given in the summary table following the code, is an unbiased estimate of out-of-sample error.

On the initial run, I used the option do.trace=T and 1000 trees to watch the progression of OOB error for each class (printed to the console). For the sake of appearance, I decided to only print the final few lines of this trace when compiling this report. As a better illustration, I extracted the error estimates from the model object and plotted them below.

```
## Call randomForest from the randomForest package
library(randomForest)
##mod2 <- randomForest(y=as.factor(traindata$classe), x=traindata[,-53], ntree=350, do.trace=T) ## verbo
mod2 <- randomForest(y=as.factor(traindata$classe), x=traindata[,-53], ntree=1000)

## put the trace in a dataframe
mod2err <- data.frame(mod2$err.rate)

## add variable for number of trees estimated at each stage
mod2err$trees <- as.numeric(row.names(mod2err))

## high end of ylim, for plotting
max_temp <- max(mod2err[,1:6])

## name the columns in the data frame of error
names(mod2err) <- c("OOB", "ERROR_A", "ERROR_B", "ERROR_C", "ERROR_D", "ERROR_E", "TREES")

## print the final line of the error dataframe
tail(round(mod2err,5), n=5)
```

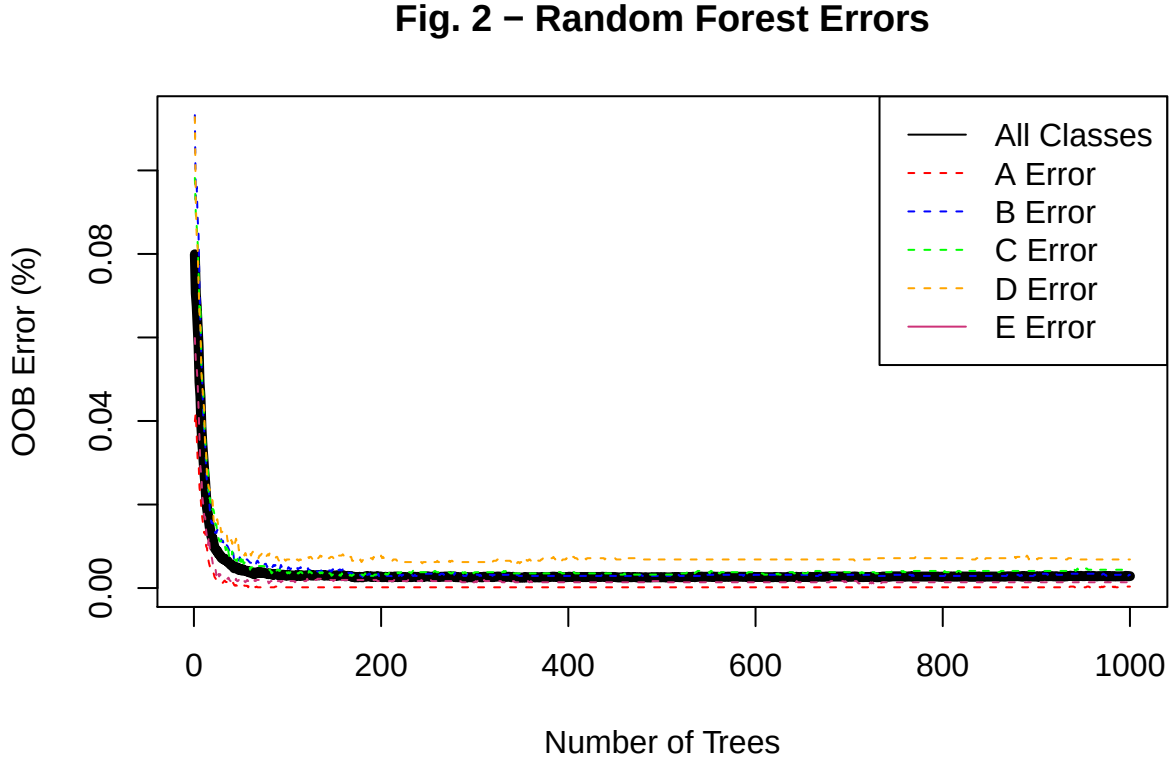
```
##          OOB ERROR_A ERROR_B ERROR_C ERROR_D ERROR_E TREES
## 996  0.00290 0.00036 0.00342 0.00438 0.00684 0.00139   996
## 997  0.00285 0.00036 0.00316 0.00438 0.00684 0.00139   997
## 998  0.00280 0.00036 0.00290 0.00438 0.00684 0.00139   998
## 999  0.00285 0.00036 0.00316 0.00438 0.00684 0.00139   999
## 1000 0.00285 0.00036 0.00316 0.00438 0.00684 0.00139  1000
```

The tail of the trace suggests an **expected out-of-sample error of ~0.28% across all classes**. The model struggles the most with predicting classe=D, but performs very well for all classes.

```
## set up the plot
plot(x=mod2err$TREES, ylim=c(0,max_temp),ylab="OOB Error (%)", xlab="Number of Trees", main = "Fig. 2 -

## plot the errors
lines(x=mod2err$TREES, y=mod2err$OOB, type="l", lty=1, col = "black", lwd=5)
    lines(x=mod2err$TREES, y=mod2err$ERROR_A, type="l", lty=2, col = "red", lwd=1)
    lines(x=mod2err$TREES, y=mod2err$ERROR_B, type="l", lty=2, col = "blue", lwd=1)
    lines(x=mod2err$TREES, y=mod2err$ERROR_C, type="l", lty=2, col = "green", lwd=1)
    lines(x=mod2err$TREES, y=mod2err$ERROR_D, type="l", lty=2, col = "orange", lwd=1)
    lines(x=mod2err$TREES, y=mod2err$ERROR_E, type="l", lty=2, col = "violetred3", lwd=1)

## add a legend
legend(x="topright", legend=c("All Classes", "A Error", "B Error", "C Error", "D Error", "E Error"), co
```



As you can see in the plot above, random forests give a low starting error estimate (~12%) and rapidly converge towards 0. The marginal gains in accuracy from increasing the tree count beyond 150 are minimal. Nonetheless, because estimation was still very fast (about 2 minutes on my system) and the design of random forests mitigates the risk of overfitting, I elected to keep the number of trees at 1000 for this run.

C. Algorithm Choice It was clear at this point that the random forest approach wwould be better for the classification problem at hand. The project asked us to "build a machine learning algorithm to predict activity quality from activity monitors. My final algorithm was the random forest with 1000 trees, run on the 52 features extracting in sections 2B-2D.

IV. Testing

A. Get the Predictions With the algorithm chosen, I moved on to predicting the test cases.

```
## get predictiosn on the test set
pred2 <- predict (mod2, newdata=testdata, type="class")

## convert predictions back to characters
pred2df <- data.frame(pred2)
pred2df$pred_out <- "placeholder"

## get character predictions
pred2df[as.numeric(pred2df$pred2)==1,]$pred_out <- "A"
pred2df[as.numeric(pred2df$pred2)==2,]$pred_out <- "B"
pred2df[as.numeric(pred2df$pred2)==3,]$pred_out <- "C"
pred2df[as.numeric(pred2df$pred2)==4,]$pred_out <- "D"
pred2df[as.numeric(pred2df$pred2)==5,]$pred_out <- "E"
```

B. Send Predictions to Text Files This project also required a submission of the predictions for the test set in a very specific format. Below, I used the code provided by Prof. Leek for producing text files with the predictions in them.

```
## put character predictions in a vector called 'answers'
# answers <- pred2df$pred_out
#
# ## Define function to write out the answers
# pml_write_files = 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)
#   }
# }
#
# ## write out the answers
# pml_write_files(answers)
```

V. Conclusions

In this quick analysis, I found that a random forest was far better suited to the task of qualitative activity recognition than a single regreession tree (CART) approach. The accuracy from this was very impressive, so I did not explore othermethods for classification.

Thank you for taking the time to read through and grade this report. I understand that the approaches I took were not particularly sophisticated, but they did seem to achieve very good peformance for this particular classification task. If you have additional notes/suggestions/critiques that you'd like to discuss, please contact me via LinkedIn (click my name in the title line).

References

[1] http://www.saedsayad.com/docs/gbm2.pdf [2] http://www.jstatsoft.org/v28/i05/paper [3] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/ [4] http://www.inside-r.org/packages/cran/

caret/docs/trainControl [5] https://www.quora.com/What-is-the-out-of-bag-error-in-Random-Forests [6] http://www.inside-r.org/packages/cran/caret/docs/trainControl [7] http://stackoverflow.com/questions/26828901/warning-message-missing-values-in-resampled-performance-measures-in-caret-tra [8] http://www.statmethods.net/advstats/cart.html [9] http://cran.r-project.org/web/packages/rattle/rattle.pdf [10] https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm [11] http://www.r-bloggers.com/a-brief-tour-of-the-trees-and-forests/