Write up draft

Introduction and Overview

Predictive modeling is a process used to create a statistical model of future behavior. A predictive model is made up of a number of **predictors**, which are variable factors that are likely to influence future behaviors or results. Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani give an example to briefly introduce the topic:

> Suppose we are statistical consultants hired by a company to provided advice on how to improve sales of a particular product…It is not possible for our client to directly increase the sales of the prodcut. On the other hand, they can control the advertising expenditure in each of the three media [TV, radio and newspapers]. Therefore, if we determine that there is an association between advertising and sales, then we can instruct our client to adjust advertising budgets, thereby indirectly increasing sales. In other words, our goal is to develop an accurate model that can be used to predict sales on the basis of the three media budgets (James, 2013).

In predictive modeling, data is collected for the relevant predictors, a statistical model is formulated, predictions are made, and the model is revised as additional data becomes available. My research project deals with predictive models and applications of such modeling.

An example of an application of predictive modeling is activity recognition. Activity recognition is an increasingly important technology because it can be applied to many real-life problems such as, home-based proactive and preventive healthcare applications. It can also be applied in learning environments, security systems, and a variety of human-computer interfaces. The goal of activity recognition is to recognize common human activities in real-life settings, such as weight lifting.

Regular physical activity is one of the most important things that can be done for overall health. It can help control weight, lower risk for heart disease, strengthen bones and muscles, and increase chances of longer life. However, if the activity is performed incorrectly, there is a greater risk of injury, which is counterproductive. To benefit most from a fitness routine, the activity should be performed as accurately as possible. Some people can go to a gym and work with a certified trainer, but many people cannot or will not work with a personal trainer. These people may be doing the correct lift motion, but there is no way to really know unless they are taught the correct motion by a professional.

In the case of my research project, I want to see if a predictive model can be made to recognize certain weight lift motions. If an honest predictor model could be made, then the model could be integrated into the weight lift equipment and used to determine if the lift was done correctly or incorrectly. This model could be integrated with other technologies and be used to help reinforce the correct weight lift motion by commending the user for a correct movement or making a comment when the user made an incorrect movement. For example, I am trying to perform the lift motion from the study correctly, but I am actually performing a Class C error. If my predictive model is good enough (based on the measurements from the sensors, the model can accurately predict in which class my lift belongs), then my armband could beep, notifying me of my error.

Background on Data Used

The article *Qualitative Activity Recognition of Weight Lifting Exercises* describes a study presented by Eduardo Velloso, Andreas Bulling, Hans Gellersen, Wallace Ugulino, and Hugo Fuks. Among other goals, the researchers wanted to provide feedback to weight lifters using qualitative activity recognition. The study involved six male subjects, all in their twenties and with little weight lifting experience. The subjects were taught how to lift a dumb-bell correctly and were also taught how to perform the same movement in four incorrect ways. The Unilateral Dumbbell Bicep Curl was the lift that was taught to the subjects. The five categories of lift data collected were:

```
* Class A: correct lift movement
* Class B: throwing the elbows to the front
* Class C: lifting the dumbbell only halfway
* Class D: lowering the dumbbell only halfway
* Class E: throwing the hips to the front
```

The subjects repeated each lift ten times and during each lift the researchers recorded a number of inertial measurements from sensors in the users' glove, armband, lumbar belt, and dumbbell (these are pieces of equipment that are commonly used by weight lifters). The sensors recorded several data points throughout the lifting motion and the final data set includes 160 variables.

The aim of this report is to build an "honest" predictive model that can be used in realistic circumstances using the data from the Velloso et al study.

Methods

Background on Methods Used

The method of model making that will be used in this report is random forest. Random forest is a method for classification, regression, and other tasks, that operate by constructing a multitude of decision trees using a training set of data and outputting the class that is the mode of the classes for classification (Breiman, 2001). In other words, given a new observation, each tree makes a prediction of activity-type. The type that is chosen by the largest number of the trees is the type that the random forest predicts. So it's as if each tree gets to vote, and the type with the most votes wins. Statistical theory shows that although each individual tree is liable to be a poor predictor, a large number of them working by majority vote deliver much better predictions! According to Leo Breiman (who helped develop the random forests technique), random forests are grown from many classification trees. It is a statistical algorithm that is used to cluster points of data in functional groups. When the data set is large and/or there are many variables it becomes difficult to cluster the data because not all variables can be taken into account. Therefore the algorithm can also give a certain chance that a data point belongs in a certain group (Breiman, 2007).

Classification Trees

Classification trees are a tree-based model and are used to predict a qualitative response. The variables that go into these classification trees can be numerical or categorical. We predict that "each observation belongs to the most commonly occuring class (or category) of training observations in the region to which it belongs" (James, 2013). They are useful because they provide predictors in situations where there are many variables that interact in complicated, non-linear ways. In interpreting these classification trees, we are often "interested in both the class prediction corresponding to a particular terminal node region, and in the class proportions among the training observations that fall into that region" (James, 2013).

So, in simpler terms, a classification tree consists of a set of true/false decision rules. It is kind of like a game of 20 questions, where we ask different questions based on the answers to previous questions, and then at the end we make a guess based on all the answers. We can visualize a decision tree as a set of nodes (corresponding to true/false questions), each of which has two branches depending on the answer to the question. Unlike real trees, we usually draw them with their "root" at the top, and the "leaves" at the bottom. In order to make predictions with the tree, we start at the top (the "root" node), and ask questions, traveling left or right in the tree based on what the answer is (left for true and right for false). At each step, we reach a new node, with a new question. Once we reach the bottom (a leaf node), we make a prediction based on the set of answers, just like 20 questions. But unlike 20 questions, the number of questions in a decision tree is not always 20, but can vary (Corso, 2013). XXXXX**This was adapted from a lecture PPT of James Corso from SUNY at Buffalo CSE 555 course.**

Shall we look at an example of a classification tree?

We are using a data set from a survey taken in the MAT 111 class (Elementary Probability and Statistics). This data set has 71 rows and 12 variables. The survey includes variables such as sex, height, GPA, sleep, and the fastest speed ever driven. The names of some of the variables may seem a little odd, such as weight_feel, love_first, and extra_life. The weight_feel variable is how the participant feels about their weight. They could have answered underweight ("a"), about right ("b"), or overweight ("c"). The love_feel variable is whether or not the participant believes in love at first sight and the extra_life variable is whether or not the participant believes in extraterrestrial life. The seat variable is where the participant sits in a classroom. The letter "a" corresponds to sitting in the front, "b" corresponds sitting in the middle rows, and "c" corresponds to sitting in the back rows.

Classification tree example:

The following packages are required:

```
library(tree)
library(tigerstats)
library(knitr)
library(randomForest)
```

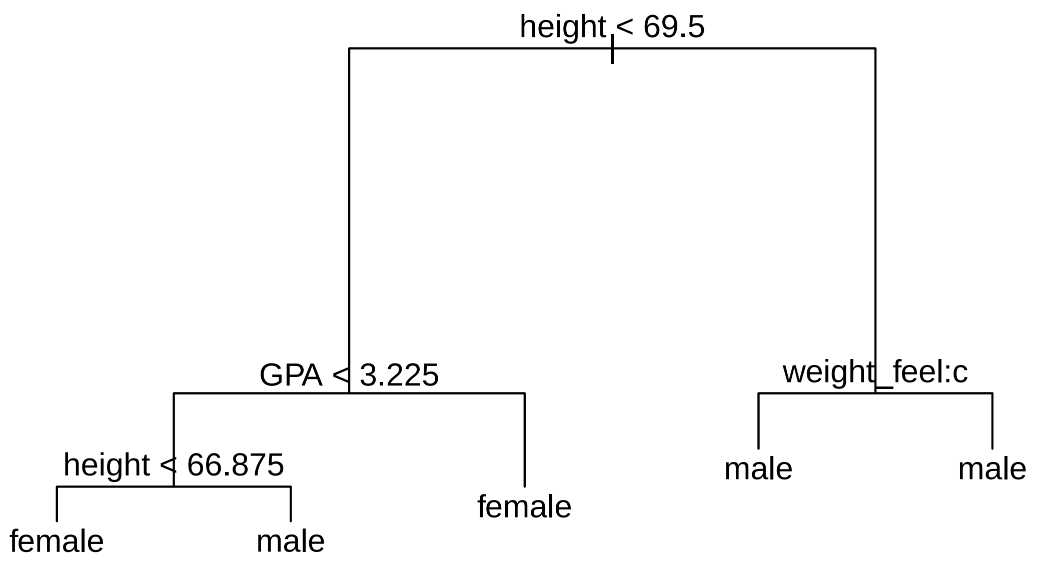
The code for a classification tree:

```
set.seed(2020)

m111s.tr <- tree(sex~fastest+GPA+height+sleep+weight_feel+love_first,
                 data=m111survey)

m111s.tr
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
## 1) root 70 96.120 female ( 0.5571 0.4429 )
##   2) height < 69.5 42 34.450 female ( 0.8571 0.1429 )
##     4) GPA < 3.225 18 22.910 female ( 0.6667 0.3333 )
##       8) height < 66.875 9  6.279 female ( 0.8889 0.1111 ) *
##       9) height > 66.875 9 12.370 male ( 0.4444 0.5556 ) *
##     5) GPA > 3.225 24  0.000 female ( 1.0000 0.0000 ) *
##   3) height > 69.5 28 19.070 male ( 0.1071 0.8929 )
##     6) weight_feel: 3_overweight 10 12.220 male ( 0.3000 0.7000 ) *
##     7) weight_feel: 1_underweight,2_about_right 18  0.000 male ( 0.0000 1.0000 ) *
```

This tree is used to predict the sex of an individual based on the variables of fastest speed ever driven, GPA, height, the amount of sleep the participant got the night before, how the participant feels about their weight, and if the participant believes in love at first sight. Below is an easy to understand schematic of a classification tree.

```
plot(m111s.tr)
text(m111s.tr)
```



Looking at this tree, we can see that the first division is set when height is less than 69.5 inches. If the height of an observation is less than 69.5 inches they are put into the left region and those with a height equal to or above 69.5 inches are put into the right region. Those in the left hand region are divided by GPA. If the GPA is greater than or equal to 3.225, the prediction is female. If the GPA is less than 3.225, then a further division by height is made. If the height is less than 66.875 inches, then female is predicted. Otherwise, the sex is predicted as male. Those in the right region are divided by how they feel about their weight. Notice that the division is by "weight_feel:c". This means that the left region feel underweight or about right and the right region feel overweight. Instead of using the full name of the variable, this tree made shorter version. The letter "a" corresponds to feeling underweight, the letter "b" corresponds to feeling about right, and the letter "c" corresponds to feeling overweight. Looking at the two terminal nodes, it appears that it doesn't matter how they feel about their weight; the prediction will still be male.

Now, it may seem odd that the classification tree made a split that ended up in two nodes with the same prediction. Why bother making another division? This split was made because it led to increased *node purity.* This means that the region could be further subdivided into 2 regions where each was more purely male or female. After the division, one node is completely male and the other is 70% male. If the height is greater than 69.5 inches and they feel overweight, then male is absolutely certain. If they feel just right or underweight, 70% of the node are male. Even though we are less certain of this classification (compared to 89% male before the division), it improves the deviance. We want the deviance as close to 0 as possible and the deviance improved from 19.070 to 12.220. The sum of the deviances from the post-division nodes is closer to 0 than the pre-division node. Let's take a closer look into how deviance is found and why it is so important.

The deviance formula used for the classification trees is:

$$-2\sum_{k} n_{ik} log_e(p_{ik})$$

where n_ik is the number of a certain type in the node andp_ik is the proportion of a certain type in the node.

So using the classification tree example, let's look at how the deviance was found for the first split of the classification tree:

If you recall, there are 70 participants before the first split; 39 female and 31 male. Then

$$D = -2[(39)(\ln(\frac{39}{70})) + (31)(\ln(\frac{31}{70}))]$$

$$D = 96.124$$

Now if we look at the deviance of each side of the split and add them together, the sum should be less than 96.124.

$$D\ ht < 69.5\ = -2[(36)(\ln(\frac{36}{42})) + (6)(\ln(\frac{6}{42}))] = 34.450$$

$$D\ ht > 69.5\ = -2[(3)(\ln(\frac{3}{28})) + (25)(\ln(\frac{25}{28}))] = 19.068$$

$$D\ ht < 69.5\ + D\ ht > 69.5\ = 53.518$$

At each split, the deviance is determined. The deviance must be at least 0.01 times the deviance of the root node before splitting (this is a default setting in the code used to form classification trees). For example, the deviance before the height split was 96.124. In order for the node to be divided into two new nodes, the deviance must be at least 0.96. Since the sum of the deviances of the two sides of the split is 53.518, the tree is allowed to make the split.

We can control how finely a tree will be made by the `tree.control` function. there are 4 arguments for this function, and each has a say in whether the tree can continue splitting or not. The first argument, `nobs`, is the number of observations in the training set. The next argument, `mincut`, is the minimum number of observations to include in either post-division node. The third argument, `minsize`, is the smallest allowed node size. The final argument of the function is `mindev`. The within-node deviance must be at least this times that of the root node for the node to be split. This means that for a division to be made, the deviance of the new node must be at least the value of `mindev` times the deviance of the root node. The default settings for this function are mincut = 5, minsize = 10, and mindev = 0.01.

If we set `mindev` to 0 and `minsize` to 2, a tree that fits the data perfectly will be produced. These settings produce trees that are larger and make fewer errors. Even though the terminal nodes are pure (since the deviation can be 0 the tree can have terminal nodes with sizes 2 or 3 with all the same classification), any chance error may be seen as patterns. This does not make for good predictions on new data and the model will not be useful for predicting on any other data set than the one it was made with.

Below is a summary of the classification tree example from above.

```r
summary(m111s.tr)
```

```
##
## Classification tree:
## tree(formula = sex ~ fastest + GPA + height + sleep + weight_feel +
##     love_first, data = m111survey)
## Variables actually used in tree construction:
## [1] "height"      "GPA"         "weight_feel"
## Number of terminal nodes:  5
## Residual mean deviance:  0.4748 = 30.86 / 65
## Misclassification error rate: 0.1143 = 8 / 70
```

The summary given shows the variables actually used in constructing the classification tree, the number of terminal nodes, the residual mean deviance, and the misclassification error rate.

As you can see from this example, classification trees are easy to interpret and fairly good predictions can be made from them. In this example the misclassification error rate is about 11.4%, or 8 out of 70 observations were misclassified.

Random forests

Random forests are more advanced learning models that are capable of creating more complex decision boundaries than logistic regression. The "forest" part of the name means that it is made up of multiple decision trees (in general, the more trees, the better). Usually in a random forest instead of building a tree using all the features, we use only a random subset of features and use bagging to create the trees. That means different trees will consider different features when asking questions. But in a forest, other trees would include this feature, so it would still influence the overall prediction by the random forest.

XXXXX**Step by step process of RF**

How Random Forests Work

When the training set for the current tree is drawn by sampling with replacement, about one-third of the cases are left out of the sample. This oob (out-of-bag) data is used to get a running unbiased estimate of the classification error as trees are added to the forest.

Random forests have low bias (just like individual decision trees), and by adding more trees, we reduce variance.

Here is an example of a random forest:

```
set.seed(1010)
rf.sexm111 <- randomForest(sex~fastest+GPA+height+sleep+
                             weight_feel+
                             love_first+extra_life+ideal_ht+
                             seat+enough_Sleep+diff.ideal.act.,
                             data=m111surv2)
rf.sexm111
```

```
##
## Call:
##  randomForest(formula = sex ~ fastest + GPA + height + sleep +      weight_feel + love_first + extra
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 4.41%
## Confusion matrix:
##        female male class.error
## female     37    1  0.02631579
## male        2   28  0.06666667
```

XXXXX**Talk about results of random forest**

Above are the results from the random forest. This random forest is made up of 500 classification trees.

At 50th stage, looking at each of the items in training set, can tree vote on that item? The ones that didn't use item (individual in random resample) have a vote. Majority vote wins-either right or wrong. OOB is the percentage of time the group of 50 trees got it wrong.

Preliminary guess because OOB Females = 1 only wrong 2.6% of time, males= 2 wrong 10% of time.

Now, we can look at individual trees from this forest to see the differences:

```r
st1 <- getTree(rf.sexm111, k=1, labelVar = TRUE)
names(st1)[1:2] <- c("LD", "RD")
kable(st1, caption = "Tree 1")
```

Table 1: Tree 1

LD	RD	split var	split point	status	prediction
2	3	GPA	3.715	1	NA
4	5	ideal_ht	71.000	1	NA
0	0	NA	0.000	-1	female
6	7	height	66.875	1	NA
8	9	height	77.000	1	NA
0	0	NA	0.000	-1	female
10	11	weight_feel	1.000	1	NA
0	0	NA	0.000	-1	male
12	13	weight_feel	2.000	1	NA
0	0	NA	0.000	-1	male
14	15	seat	1.000	1	NA
0	0	NA	0.000	-1	male
0	0	NA	0.000	-1	female
0	0	NA	0.000	-1	male
0	0	NA	0.000	-1	female

The **left daughter (LD)** and the **right daughter (RD)** are the two nodes that are made by a division. The **split var** is the variable for which the division was made. The **split point** is the point where the split was made. The **status** shows if more divisions will be made (1) or if a node is terminal (-1). And then the final column shows the prediction of a terminal node. This is true for the two other tables shown below.

```r
st2 <- getTree(rf.sexm111, k=250, labelVar = TRUE)
names(st2)[1:2] <- c("LD", "RD")
kable(st2, caption = "Tree 250")
```

Table 2: Tree 250

LD	RD	split var	split point	status	prediction
2	3	height	66.875	1	NA
4	5	ideal_ht	72.500	1	NA
6	7	diff.ideal.act.	1.750	1	NA
0	0	NA	0.000	-1	female
0	0	NA	0.000	-1	male
8	9	weight_feel	1.000	1	NA
0	0	NA	0.000	-1	male
0	0	NA	0.000	-1	male
10	11	height	72.500	1	NA
0	0	NA	0.000	-1	female
0	0	NA	0.000	-1	male

```r
st3 <- getTree(rf.sexm111, k=500, labelVar = TRUE)
names(st3)[1:2] <- c("LD", "RD")
kable(st3, caption = "Tree 500")
```

Table 3: Tree 500

LD	RD	split var	split point	status	prediction
2	3	extra_life	1.00	1	NA
4	5	ideal_ht	69.00	1	NA
6	7	ideal_ht	70.00	1	NA
0	0	NA	0.00	-1	female
0	0	NA	0.00	-1	male
8	9	sleep	4.75	1	NA
0	0	NA	0.00	-1	male
0	0	NA	0.00	-1	female
0	0	NA	0.00	-1	female

Notice that each tree is different. The variables used for division differ for each tree. One tree may use variables that make the prediction off in one way, but another tree may use variables that make the prediction off in another way. Since 500 trees are used to make up this random forest, the mistakes even out to make a fairly good prediction model.

A new model should be able to be used to classify new data. Thus, it is important to have high model performance with new data. The performance of a model is measured in terms of its *error rate*: percentage of incorrectly classified instances in the data set (Witten and Eibe, 2000). The simplest way to get a handle on the ability of a predictive model to perform on future data is to try to simulate it.

The data set I am working with will have to be divided into two sets (a training set and a test set). The training set is used to build the model and the test set is new data that is used to measure the model's performance by being treated as new data. The model made with the training data will be tried out on the "new" test data. When the original data set is separated into the training and test sets, the simplest partition is a two-way random partition, careful to avoid introducing any systematic differences. The reasoning behind this type of division is that the data available for analytics fairly represents the real-world processes and that those processes are expected to remain stable over time (Steinberg, 2014). So, a well-constructed model will perform adequately on the new data.

Why not use all the data from the Weight Lifting data set? Then more data will be available to make the model and the model will be more accurate, right? However, this is incorrect. The *resubstitution error* (error rate on the training set) is a bad predictor of performance on new data because the model was built to account for the training data. The best model for predicting is the dataset itself. So, if you take a given data instance and ask for it's classification, you can look that instance up in the dataset and report the correct result every time. You are asking the model to make predictions to data that it has "seen" before- data that were used to create the model. Thus, to really know if the model would be a good predictor of the weight lift motion, it must be measured on the test data set, not the training set.

Since there are six subjects in the study, a total of 300 lifts were performed and recorded (each subject did 10 repetitions of the 5 lifts). However, during **each** lift the IMU measurements were gathered using a sliding window approach with different lengths (from 0.5 to 2.5 seconds), with a 0.5 second overlap. This resulted in a large data set (over 19,000 observations); a single observation in the data set corresponds to a specific time window for a specific subject performing one of the specified lifts. In his report, Dr. White separated the original data set into training and test sets using a random partition (a typical separation). However, in my project I will attempt to further expand on the partition by ensuring that no single lift appears in both sets. In real life, each person does a new lift and no lift is the same. While the error rate for the model will most likely be worse with new data, I want to make the model as "honest" as possible, which means none of the measurements from a single lift should be in both the training and test set. The proper separation of data is important because I not only want to see if I can make a random forest model, I wantto see how accurately it can make predictions

Results

The following sections will go through the implementation of the methods described from earlier.

Data Cleaning

Downloading

The main data, along with the examination data, can be downloaded from the web:

```r
wl <- read.csv("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
                  stringsAsFactors = FALSE)
wl_test <- read.csv("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
                    stringsAsFactors = FALSE)
save(wl, file = "data/wl.rda")
save(wl_test, file = "data/wl_test.rda")
```

Elimination of Variables

A preliminary look at the 20 examination observations indicates that for many of the variables the values are altogether missing. Although missing-ness may have predictive value, it is difficult to see how to take advantage of this fact, so we will simply exclude all such variables from our training data. We will also exclude all variables that record time-stamps, and the useless row-number variable `X`.

The spurious variables can be eliminated from our data frame. The code for this is as follows:

```r
results_test <- sapply(wl_test, FUN = function(x) !all(is.na(x)))
goodNames <- names(results_test[results_test])
keepNames <- goodNames[-c(1,3,4,5,6,7,60)]
wl2 <- wl[,keepNames]
wl2$user_name <- factor(wl$user_name)
wl2$classe <- factor(wl$classe)
wn <- wl$num_window    ### This is an addition
```

Data Separation

The data set I am working with will have to be divided into two sets (a training set and a test set). The training set is used to build the model and the test set is data that is used to measure the model's performance by being treated as "new" data. The model made with the training data will be tried out on the "new" test data. When the original data set is separated into the training and test sets, the simplest partition is a two-way random partition, careful to avoid introducing any systematic differences. The reasoning behind this type of division is that the data available for analytics fairly represents the real-world processes and that those processes are expected to remain stable over time (Steinberg, 2014). So, a well-constructed model will perform adequately on the new data.

Why not use all the data from the Weight Lifting data set? Then more data will be available to make the model and the model will be more accurate, right? However, this is incorrect. The *resubstitution error* (error rate on the training set) is a bad predictor of performance on new data because the model was built to account for the training data. The best model for predicting is the dataset itself. So, if you take a given data instance and ask for it's classification, you can look that instance up in the dataset and report the correct result every time. You are asking the model to make predictions to data that it has "seen" before- data that were used to create the model. Thus, to really know if the model would be a good predictor of the weight lift motion, it must be measured on the test data set, not the training set.

Since there are six subjects in the study, a total of 300 lifts were performed and recorded (each subject did 10 repetitions of the 5 lifts). However, during **each** lift the IMU measurements were gathered using a sliding window approach with different lengths (from 0.5 to 2.5 seconds), with a 0.5 second overlap. This resulted in a large data set (over 19,000 observations); a single observation in the data set corresponds to a specific time window for a specific subject performing one of the specified lifts. While the simplest division is to separate the original data set into training and test sets using a random partition (a typical separation), an expanded separation will be done to make the model more "honest".

The data will be separated by the window number. Even though it cannot be guaranteed that no single lift appears in both sets, by separating by new window numbers **XXXXX**

Below is a function that has been written for easy separation. The arguments of this function are the data set to be separated and the variable by which the separation will be done.

```r
### Function for separation by variable

partition_var <- function(data, var){
  vals <- unique(var)
  n <- length(vals)
  m <- floor(2/3*n)
  bools <- c(rep(TRUE,m), rep(FALSE,n-m))
  inTrain <- sample(bools, size = n, replace = FALSE)
  trvals <- vals[inTrain]
  vartr <- var %in% trvals ##vector of trues and falses
  tr <- data[vartr,]
  tst <- data[!vartr,]
  return(list(tr, tst))
}
```

Now the data can be separated into the training and test sets and used for a random forest procedure:

```r
set.seed(2020)
results <- partition_var(wl2, wn)
wlTrain <- results[[1]]
wlTest <- results[[2]]

(rf <- randomForest(x = wlTrain[,1:53], y = wlTrain$classe,
                   xtest = wlTest[,1:53], ytest = wlTest$classe,
                   do.trace = 50))
```

```
## ntree      OOB      1      2      3      4      5|    Test      1      2      3      4      5
##    50:   0.52%  0.22%  0.68%  0.56%  0.96%  0.39%|   7.04%  3.16%  6.63% 15.12%  7.84%  3.90%
##   100:   0.31%  0.14%  0.34%  0.33%  0.70%  0.16%|   6.42%  3.06%  6.08% 13.35%  7.30%  3.61%
##   150:   0.28%  0.11%  0.21%  0.38%  0.70%  0.16%|   6.40%  2.85%  6.01% 14.12%  7.30%  3.04%
##   200:   0.28%  0.14%  0.26%  0.33%  0.61%  0.20%|   6.43%  3.01%  6.22% 13.50%  7.52%  3.23%
##   250:   0.25%  0.14%  0.21%  0.28%  0.57%  0.16%|   6.42%  2.85%  6.36% 13.35%  7.52%  3.42%
##   300:   0.29%  0.14%  0.17%  0.47%  0.66%  0.16%|   6.49%  2.95%  6.57% 13.43%  7.52%  3.33%
##   350:   0.27%  0.14%  0.17%  0.38%  0.66%  0.12%|   6.48%  2.95%  6.50% 13.50%  7.41%  3.33%
##   400:   0.26%  0.14%  0.17%  0.33%  0.66%  0.12%|   6.45%  2.90%  6.63% 13.43%  7.20%  3.33%
##   450:   0.26%  0.14%  0.13%  0.33%  0.70%  0.12%|   6.42%  2.95%  6.63% 13.27%  7.20%  3.23%
##   500:   0.26%  0.14%  0.17%  0.33%  0.66%  0.12%|   6.43%  2.90%  6.63% 13.50%  7.09%  3.23%
```

```
##
## Call:
##  randomForest(x = wlTrain[, 1:53], y = wlTrain$classe, xtest = wlTest[,      1:53], ytest = wlTest$cl
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.26%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3679    3    0    1    1 0.001357220
## B    2 2346    2    0    0 0.001702128
## C    0    5 2119    2    0 0.003292568
## D    0    0   14 2270    1 0.006564551
## E    0    0    1    2 2552 0.001174168
##                 Test set error rate: 6.43%
## Confusion matrix:
##      A    B    C   D    E class.error
## A 1841   45    2   3    5  0.02900844
## B   40 1351   53   3    0  0.06634416
## C   12  140 1121  23    0  0.13503086
## D    0    0   52 865   14  0.07089151
## E    0   11    1  22 1018  0.03231939
```

Conclusions and Further Discussion

Analysis