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 in their book *An Introduction to Statistical Learning with Applications in R* to briefly introduce the topic of predictor variables:

> 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 product. 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 this example, James, et al. use media budgets for TV, radio, and newspapers as the predictor variables and the sales of a particular product is the response variable. Using predictive modeling, the company can then use the predictor variables to predict the sales outcomes of the particular product.

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.

One real-life setting example of activity recogniton is physical activity, which 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 exercise motion, but there is no way to really know unless they are taught the correct motion by a professional. Using other physical activities as predictor variables, a predictive model could be made and used to help determine if the exercise motion is being executed properly.

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 a 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 an 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). These measurements make up the predictors that the researchers used when determing a correct or incorrect lift. The sensors recorded several data points throughout the lifting motion and the final data set includes 160 variables. Some of the variables included are: user_name, num_window, yaw_belt, pitch_belt, and total_accel_belt.

The aim of this report is to build a predictive model that can be used in realistic circumstances built on 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. A more detailed discussion of the random forest method will follow. However, there are some concepts and definitions that need to be addressed before the random forest method can be fully understood. The first important concept is a classification tree, which is used in the construction of a random forest model.

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

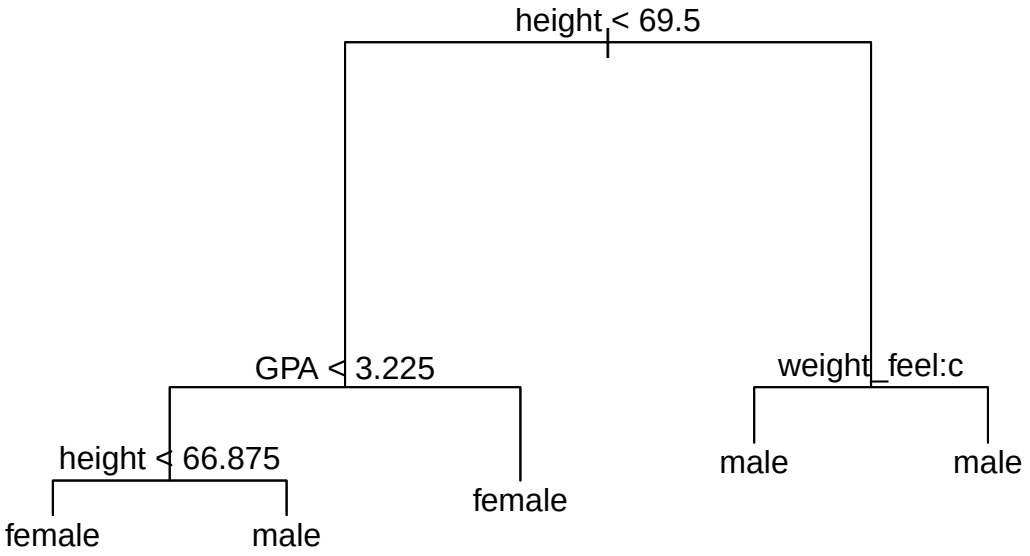
the set of answers, just like 20 questions. But unlike 20 questions, the number of questions in a decision tree is not always 20, but can vary (Corso, 2013).

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) at Georgetown College. 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:

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. R code is not only a great tool for making classification trees, it can also print a readable version of classification trees. Below is an easy to understand schematic of a classification tree.



All classification trees have nodes. The top node is referred to as the *root* node. Nodes can either split into two *daughter* nodes (or leaves) or they can stop splitting. A node that does not split any further is known as a *terminal* node. In this tree example, the majority sex in each terminal node is given under the node. This tree can be used to predict if a new individual is male or female. All we have to do is ask YES or NO questions and follow the nodes to a terminal node.

So, by looking at this tree, we can see that the first splits 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.

Below is the same classification tree, but with more details:

```
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 printout actually gives us quite a bit of information about the classification tree. First, the node numbers are labeled. So the root node is number 1 and the daughter nodes are numbers 2 and 3. Second, the variable by which the split was made is given. For example, the root node is split based on height. With daughter node 2, any height less than 69.5 inches would be placed in that node. And for daughter node 3, any height greater than 69.5 inches would be placed in that node. Third, the number of cases that reach the node is given. For example, there are 70 observations in the root node. [As a note, classification trees do not like missing data, so one row from the original data set was removed to avoid any missing values.] After the root node is split, 42 observations are found in daughter node 2 and 28 observations are found in daughter node 3. The sum of the two observations is 70. The fourth piece of information given by the printout is deviance, which is an assessment of goodness of fit (we will go into this a litte later). The fifth piece of information in the prinout is the yval, which is the majority value for the node. This would be predicted for that node. For example, if the tree was forced to make a prediction about the sex at node 4, the tree would predict that the individual would be female. The final piece of information from the printout is the yprob, which gives the probabilities of the variables being predicted. In node 4, (0.6667 0.3333) is given. This means that 66.67% of the objects in the node are female and 33.33% of the objects are male. The yval proportions can also be used to assign a probability to the prediction of a new person. For example, if we were to try to predict the sex of a new individual (using this classification tree) and the tree had to make a guess at node 4, the new individual would be predicted to be female. Female is the majority sex in this node, which is why a female prediction would be made. In addition, the yval proportion of 0.6667 tells us that the tree is 66.67% sure that the new individual is female.

While it is wonderful to be able to recognize a classification tree and use it to make predictions, it is equally important to understand the basic mechanics of how a classification tree works. Such as, how does a tree decide to make a split? Or how does a tree decide when to stop splitting into more daughter nodes? The answer is tree control.

We can control how finely a tree will be made by the `tree.control` function. Let's look at the R code to get a better understanding of the tree construction:

```
m111s.tr <- tree(sex~fastest+GPA+height+sleep+weight_feel+love_first,
                 data=m111survey,
                 control = tree.control(
                   nobs = nrow(m111survey),
                   mincut = 5,
                   minsize = 10,
                   mindev = 0.01
                 ))
```

There are 4 arguments for this function. The first argument, `nobs`, is the number of observations that will be used to build the tree. In this tree we are using all the rows in the dataset. The other three arguments have a say in whether the tree can continue splitting or not. The second argument, `minsize`, is the smallest allowed node size. The next argument, `mincut`, is the minimum number of observations to include in either post-division node. This means that any daughter node must be at least the size of the `mincut` value. Another small note to know, the `mincut` value cannot be more than half of the `minsize` value. The final argument of the function is `mindev`. The *within-node deviance* must be at least the `mindev` valude 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 node that we are thinking of splitting must be at least the value of `mindev` times the deviance of the root node. At each split, the deviance is determined. Each node has a deviance value, and this is considered the *within-node deviance*.

The default settings for the `tree.control` function are:

- `mincut` = 5,
- `minsize` = 10,
- `mindev` = 0.01

Even though are the default values, they can be changed and this will change the final construction of the classification tree.

Now, let's take a closer look into how deviance is found and why it is so important.

The general deviance formula used for the classification trees is:

$$-2\sum_{k} n_k \ln(p_k)$$

where:

- k stands for the k^{th} possible value
- n_k is the number of a certain type in node k
- p_k is the proportion of a certain type in node k

Recall: $\ln x$ *is the natural logarithm of the value* x

For example, the deviance for any node of this classification tree would be found in the following way:

$$D = -2[(n_1 \ln(p_1)) + (n_2 \ln(p_2))]$$

where:

- $n_1 =$ `number of females in the node`
- $n_2 =$ `number of males in the node`
- $p_1 =$ `proportion of females in the node`

- $p_2 =$ `proportion of males in the node`

We can now begin looking at how this formula can work. As stated earlier, deviance is a measure of goodness of fit. In other words, the deviance is a measurement of purity. The more pure a node is (or in the case of our tree, the closer the node is to being all male or all female), the closer the deviance is to 0. Thus, when deciding on whether to make a split at a node or not, is to choose a split such that the sum of the deviance of the two daughter nodes is smaller than the deviance before the split. In fact, the tree will find a split so the sum of the two daughter nodes is as small as possible. Now, `mindev` sets a limit on how small the deviance can be. In order to split into daughter nodes, the deviance of the node which will be split must be at least

$$\texttt{mindev} \times \textbf{root deviance}$$

[Recall that the default value for `mindev` is 0.01, but this can be changed to modify the tree growth.]

To show how this would work, let's look at the split of the root node. If you look at the printout of the classification tree, you can see that the deviance before the split was 96.120 (the deviance of root node).

```
1) root 70 96.120 female ( 0.5571 0.4429 )
  2) height < 69.5 42 34.450 female ( 0.8571 0.1429 )
  3) height > 69.5 28 19.070 male ( 0.1071 0.8929 )
```

Before a node can be split into daughter nodes, a few things must be considered. First, in order for the root node to be divided into two new nodes, the deviance must be at least 0.96 (since the deviance must be at least 0.01 times the deviance of the root node before splitting). Well, the deviance of the root node will be at least 0.96, so this is a trivial calculation. Second, the node must be at least size 10 (our default `minsize` value). If these two conditions are satisfied then the tree will think about splitting the node.

So using the classification tree example, let's look at a different, nontrivial, example of how the deviance was found. How about node 4? Below is the printout of node 4 with the two daughter nodes:

```
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 ) *
```

If you look at the original print out of the classification tree, you will notice that there are 18 participants before this split is made; 12 female and 6 male. To find how many females or males are in the node, take the `nobs` value and multiply it with the `yprob` values (the proportions given). The nearest integer is the number of females or males in the node. So,

Females $= \texttt{18} \times \texttt{0.6667} = 12$

Males $= \texttt{18} \times \texttt{0.3333} = 6$

Let's verify the deviance given for this node using our formula.

$$D = -2[(12)(\ln(\frac{12}{18})) + (6)(\ln(\frac{6}{18}))]$$

$$D = 22.914$$

Since the deviance is greater than 0.96 (the deviance must be at least 0.01 times the deviance of the root node before splitting), we can think about splitting into daughter nodes. Also, since the size of the node is 18, which is greater than 10, we can think about splitting into daughter nodes.

Notice that the daughter nodes are at least equal to 5, which is the `mincut` value. Both daughter nodes had size 9. If either of the daughter nodes had been less than size 5, then the split would not have been made.

Now if we look at the deviance of each side of the split and add them together, the sum should be less than 22.914, since a split looks for the smallest total deviance.

$$D_1 = -2[(8)(\ln(\frac{8}{9})) + (1)(\ln(\frac{1}{9}))] = 6.279$$

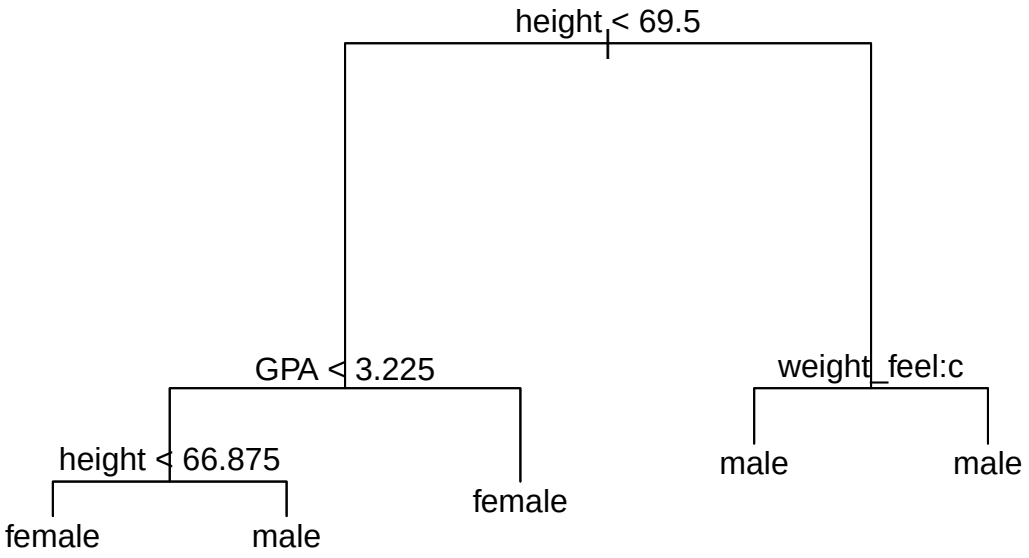
$$D_2 = -2[(4)(\ln(\frac{4}{9})) + (5)(\ln(\frac{5}{9}))] = 12.365$$

$$D_1 + D_2 = 18.644$$

If the sum of the two daughter nodes was larger than 22.914, then a split would not be made because it would not improve the total deviance of the classification tree.

Of all the possible splits, where both daughter nodes have a larger size than the `mincut` value, this was the smallest deviance found, which is why the tree made this particular split. The tree made a split because there was a way to make the total deviance smaller.

As a final example of the tree making splits, recall that the tree had two terminal nodes (from the same split) that are both male.




```
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 ) *
```

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? The tree still made a split because there was a way to make a smaller total deviance. The smaller the deviance of a node leads 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 the object being 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.

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

```
Classification tree:
tree(formula = sex ~ fastest + GPA + height + sleep + weight_feel +
    love_first, data = m111survey)
Variables actually used in tree construction:
"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.

In order to find the residual mean deviance, R added the deviance at all 5 terminal nodes and then divided by (number of observations - number of terminal nodes). This is a very difficult number to interpret. The main thing to understand is that the smaller the residual mean deviance, the more "pure" the nodes are on average. Thus,

$$ResidualMeanDeviance = \frac{30.869}{70-5} = 0.4749$$

The residual mean deviance (RMD) is also used to compare two different tree models and how well they predict. The smaller the value of the residual mean deviance, the better the tree is at predicting on its own data. For example, if I have one tree model used to predict the sex of individual using one dataset and another tree model used to predict if an individual is at risk for diabetes using a different dataset, the residual mean deviance could be used to compare how well the models will predict on its own data. If the RMD of the first model is 0.4749 and the RMD of the second model is 0.5812, then we know that the first model is better at making predictions.

The final output given by the classification tree summary is the misclassification error rate. The performance of a model is measured in terms of its misclassification error rate: percentage of incorrectly classified instances in the data set (Witten and Eibe, 2000). The lower the misclassification error rate, the higher the performance of the model. In other words, a lower misclassification rate means that a smaller number of objects are being misclassified and the model is making correct predictions.

As you can see from the 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%. This means that 8 out of 70 observations were misclassified. However, the tree is looking at correct answers that were given. If this classification tree were given new data, the error rate would most likely be much worse.

As an aside, unfortunately, R has some round-off error. This is why the values calculated are slightly off from the summary.

As mentioned earlier, the default settings in the `tree.control` function can be modified to change the tree construction. So, if we set `mindev` to 0 and `minsize` to 2, `mincut` to 1, a tree that fits the data perfectly

will be produced. Since the deviation can be 0 the tree can have terminal nodes with size 1 with the same classification. With these settings, a tree can be made with a large number of terminal nodes that are small in size and very near to pure. Changing the default settings to lower values produces trees that are larger and can possibly make fewer errors (on the data already given). However, even though the terminal nodes are pure, any chance variation 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.

A model should be able to be used to classify new data. Thus, it is important to have high model performance with new data. To have a model that performs well with new data it is important to divide the original data set into two new sets (training and test sets). 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 data set? Then more data will be available to make the model and the model will be more accurate, right? Actually, 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 (the data it has never "seen" before), not the training set.

Now we're ready to look more at a random forest model and what it does.

Random forests

Random forests are a way of averaging multiple classification trees, trained on different parts of the same training set. However, there are some major differences in the classification trees used in a random forest compared to a regular classification tree. First, at any point where the tree is thinking of a split, the tree does not look at all the predictor variables. Instead, a random subset (default $\sqrt{n}$) of the predictor variables is taken. If you let n be the number of predictor variables, you can take the floor of $\sqrt{n}$. This means that the classification tree may not have all the predictor variables. The regular tree examines every predictor variable. The random forest tree looks at $\sqrt{n}$ of predictor variables. Also, this means that most trees in the random forest are different.

A second important difference between random forest classification trees and regular classification trees is the size. Random forest classification trees are overgrown. Deviance is not considered when making a split nor are the node sizes. The nodes can end up being very small with one or two objects in each of the final nodes. For example, one of the arguments in 'randomForest' is `nodesize`. This is the minimum size of terminal nodes. Setting this to a larger number causes trees to be smaller because the number of objects in a terminal node cannot be less than 5. The default for the classification trees in 'randomForest' is 1. Thus, larger trees are grown because the terminal nodes can have a minimum of 1 object. This overgrowing of classification trees allows for great variability in the classification trees. Most of the classification trees will not be able to make predictions well on new data; some will end up with strange results. But putting together all the trees can actually give good results.

A third difference that needs to be considered is the fact that the classification trees used to build a random forest select at random, with replacement, from the data given and the sample size is equal to the size of the original data set. This gives a random sample where about $\frac{1}{e}$ of the items are not grabbed and gives variability that we want to take into account when making a model. The classification trees should vary like the original data. The best way to simulate population data without taking another sample is to take a random sample from the original sample.

Let me illustrate this with an example:

I have a list of 100 numbers and I put them in a bag (this is my original data set). I draw out a number from the bag and write it down. I replace the number, pick from the bag again, and write down the number I've drawn. I replace the number and continue in this way until I have a new list of 100 numbers (this is the sample data set). The new list may have duplicate numbers or some numbers may be missing entirely. This new list is the best guess for the original list of numbers.

I have made a list of 100 numbers and taken a random sample from the original list.

```
##   [1]  27  11  97  36  64   4   9   6  71  25  52  82   3  62  66  45   2
##  [18]   3  99  14  66  23  37  34  92  46  96  69  61  84  17  44  36  86
##  [35]   9  58  52  59  84  45  19  66  60  37  37  61  11  81 100  46  10
##  [52]  49  60  79  21  62  21  89  22  54   2  75  33  56  67  14   9  58
##  [69]  31  25  89  95  39  51  45  96  23  36  87  35  72   8  46  94  31
##  [86]  97   7  30  46  32  46  56  16  28  16  80  72  40  47   4
```

Notice that some of the numbers have been repeated and some numbers are missing. We can have the computer look at the numbers we are missing and the numbers that are included in the random sample. Below is a table showing the numbers that are missing from the random sample:

Below is a table showing the numbers that are part of the random sample:

Notice that 35 numbers are missing from the new list, which is 35% of the data. Something that is very important to remember is that as the size of the data set increases the number of variables that are missing from the random resample is about $\frac{1}{e}$. That means about 36.788% of the variables are not included in the resample.

As another example, suppose I have a list of numbers 1 to 10,000 instead of only 100. Then, looking at the numbers included in the new list, we find that 3,681 are missing. This is 36.81% of the original numbers, and it is very near equal to $\frac{1}{e}$.

As an added bonus, the random sample with replacement is used to make an estimation of how well the random forest will do. And that estimation will be made even when the random forest is being constructed.

Thus, the random selection of predictor variables, the overgrowing of the trees and the random sample of data allows for many different possibilities, which can help with making predictions. This makes the random forest a very effective modeling tool.

How Random Forests Work

Random forests refer to a method for classification that operates by constructing a multitude of classification trees using a training set of data and outputting the class that is the most (also referred to as 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 for that observation. 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 forest 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 (Breiman, 2007). In other words, using a large number of classification trees can help make better predictions because the trees can vary a lot and they can cancel out those variations by averaging.

Each tree in the random forest is grown as follows (Breiman and Cutler, 2007):

1. Each tree is trained on roughly $2/3^{\text{rd}}$ of the total training data. Cases are drawn at random with replacement from the original data. This sample will be the training set for growing the tree. The other $1/3^{\text{rd}}$ of the cases are left out of the sample. This out-of-bag (OOB) data is used to get a running unbiased estimate of the classification error as trees are added to the forest. It is also used to get estimates of variable importance.

2. Some predictor variables (say, m) are selected at random out of all the predictor variables and the best split on these m is used to split the node. By default, m is square root of the total number of all predictors for classification. The value of m is held constant during the forest growing. As explained earlier, in a standard classification tree, each split is created after examining every variable and picking the best split from all the variables.

3. For each tree, using the leftover data (36.8% or $\frac{1}{e}$), the misclassification rate (or OOB error rate) is calculated. The error from all trees is used to determine overall OOB error rate for the classification.

4. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes over all the trees in the forest. The vote will be YES or NO. All the YES votes are counted and this is the predicted probability for a classification.

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
```

```
randomForest(formula = sex ~ fastest + GPA + height + sleep +
weight_feel + love_first + extra_life + ideal_ht + seat +
enough_Sleep + diff.ideal.act., data = m111surv2)

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. The sex of an individual is being predicted based on the predictor 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.

For do.trace: 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:

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.

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

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.

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 want to see how accurately it can make predictions.

Results

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

Data Cleaning

Many of the following steps for downloading and cleaning the data are taken from the report "Predicting Movement-Types: Quick Model-Making with Random Forests" written by Dr. White.

Downloading

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

```
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

The main data set consists of 19622 observations on 160 variables, including:

- a spurious row-number variable `X`;
- `user_name` (the name of the subject);
- three time-stamp variables;
- two variables, `new_window` and `num_window` related to time-windows;
- 152 numerical measurements derived from the IMUs;
- the variable `classe` that records the activity-type.

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:

```
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".

I decided to separate my data by window number. Even though we cannot guarantee that a window number will not appear in both the training and the test set, this will further separate the data than a typical random partition.

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.

```
### 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:

```
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%
```

Conclusions and Further Discussion

Analysis