Random Forest

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:

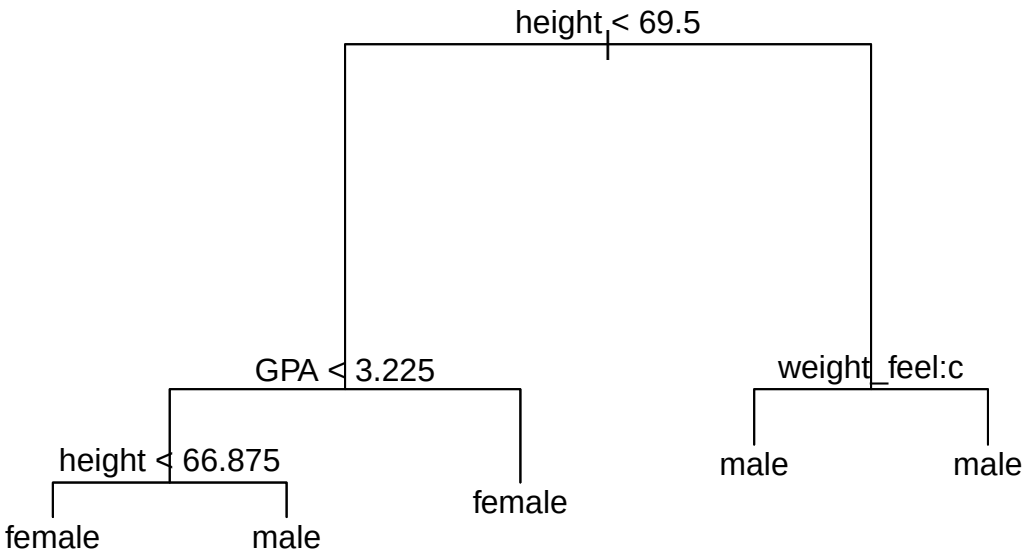
```
set.seed(2020)

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.

```
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:

```
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.

```
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

```
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.