Random Forest Algorithm

Brian Seggebruch

May 29, 2019

Executive summary

We'd like to understand if there are certain characteristics associated with breast cancer tumor measurements that can lead to accurate classification predictions of (M)alignant or (B)enign. To do this, we will explore a data set provided by the University of Wisconsin that has measurements from 579 cancer screenings. Becase we are interested in prediction and not inference, model interprability is less of a concern. We are less interested in exactly which explanatory variables and splits are used to make predictions, and instead we are most interested in maximum accuracy of predictions. A single decision tree is provided to build context. Using the decision tree, we would be able to infer which variables determine a given result, but decision trees naturally have a high variance and therefore we attempt to improve accuracy by using a random forest model. From the conclusion of our analysis, we learn that using our model we can predict correct classification of tumors with an accuracy of 97%.

Understanding our data

```
breast_cancer <- read.csv('wisconsin_breast_cancer.csv')
# commenting out for sake of space in knitted document
# names(breast_cancer)
# head(breast_cancer)
# summary(breast_cancer)
sum(is.na(breast_cancer))
```

```
## [1] 569
```

```
# decision trees and random forest are largely non-parametric and don't
require noramlity of errors or data, so we don't check for this
```

Here we read in our data set named "breast_cancer" which contains diagnostic data for 579 cancer-screenings, digitized from images of a "fine needle aspirate of mass" procedure. The data is provided by the University of Wisconsin and is intended to be used to help predict whether a mass of cells is malignant or benign. There are ten real-valued variables measured for each record. They are, (from the dataset documentation):

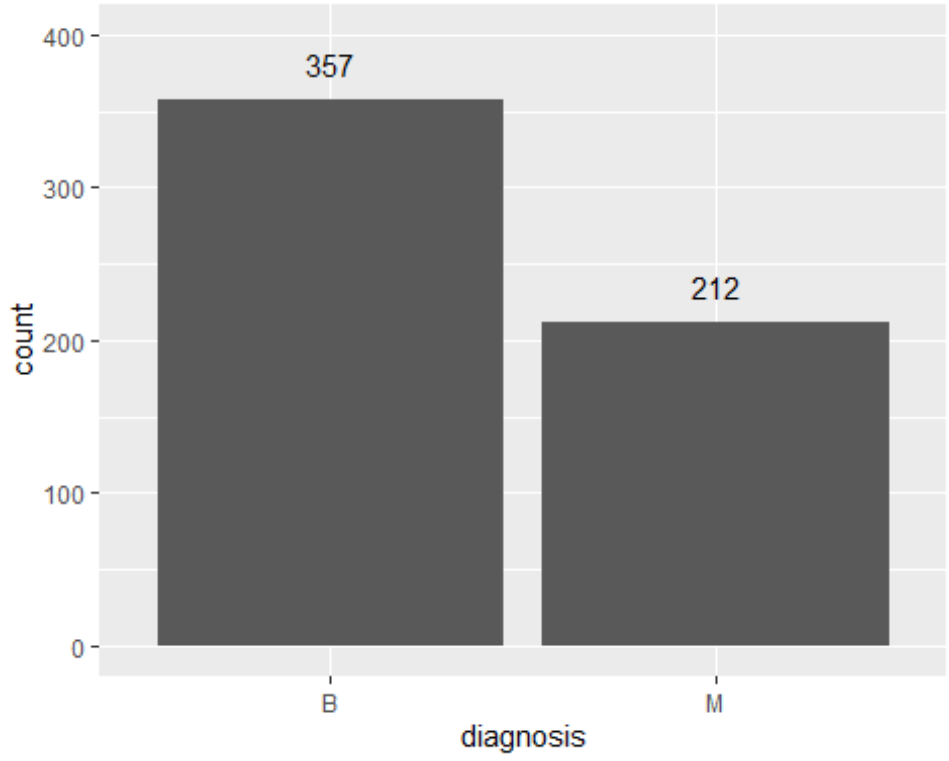
1) radius (mean of distances from center to points on the perimeter)
2) texture (standard deviation of gray-scale values)
3) perimeter
4) area
5) smoothness (local variation in radius lengths)

6) compactness (perimeter^2 / area - 1.0)
7) concavity (severity of concave portions of the contour)
8) concave points (number of concave portions of the contour)
9) symmetry
10) fractal dimension (“coastline approximation” - 1)

Each of these measures is a taken from the cell nuclei present in the image generated from the procedure. The mean, standard error, and “worst” (largest) value are calculated for each image and recorded. Therefore, we have 30 variables (10 real-value measurements * 3 statistically derived values). Each variable is recorded with four significant digits. There are 357 benign (B) classifications and 212 malignant (M) classifications.

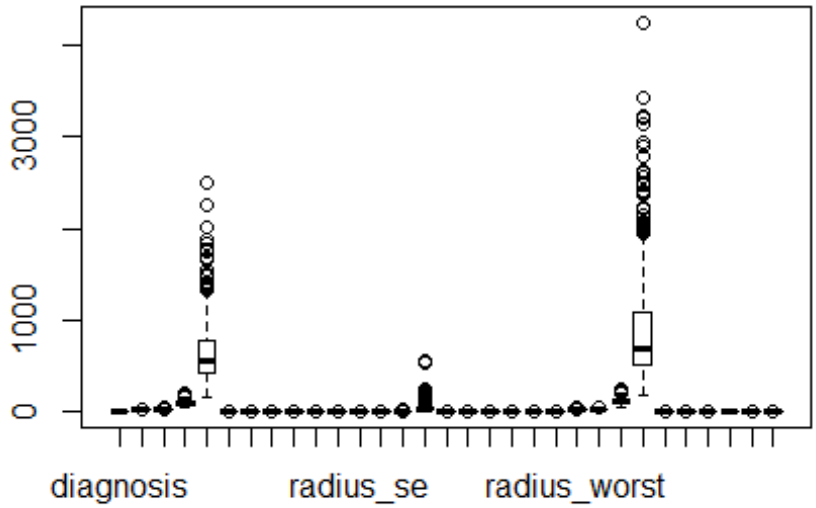
Data cleansing and preparation

```
# viewing our result distribution
ggplot(data = breast_cancer, aes(x = diagnosis)) +
  geom_bar() +
  geom_text(stat='count', aes(x = diagnosis, label = ..count..), vjust = -1)
+
  ylim(0, 400)
```

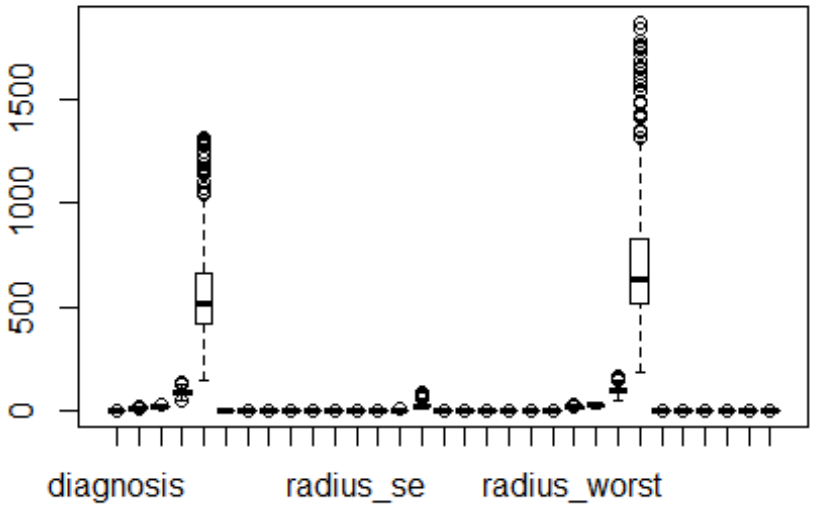



```
# removing unwanted data
breast_cancer$id <- NULL
breast_cancer$X <- NULL

# removing outliers
```

```r
breast_cancer_diagnosis <- breast_cancer[1]
breast_cancer_data <- breast_cancer[2:31]

remove_outliers <- function(x, na.rm = TRUE, ...) {
  qnt <- quantile(x, probs = c(.25, .75), na.rm = na.rm, ...)
  outlier <- 1.5 * IQR(x, na.rm = na.rm)
  y <- x
  y[x < (qnt[1] - outlier)] <- NA
  y[x > (qnt[2] + outlier)] <- NA
  y
}

boxplot(breast_cancer)
```




```r
breast_cancer <- na.omit(cbind(breast_cancer_diagnosis,
data.frame(apply(breast_cancer[2:31], 2, remove_outliers))))
boxplot(breast_cancer)
```




```r
# splitting train and test
sample_size <- floor(0.75 * nrow(breast_cancer))

set.seed(03091996)
train_index <- sample(seq_len(nrow(breast_cancer)), size = sample_size)

train <- breast_cancer[train_index, ]
test <- breast_cancer[-train_index, ]
```

Research question

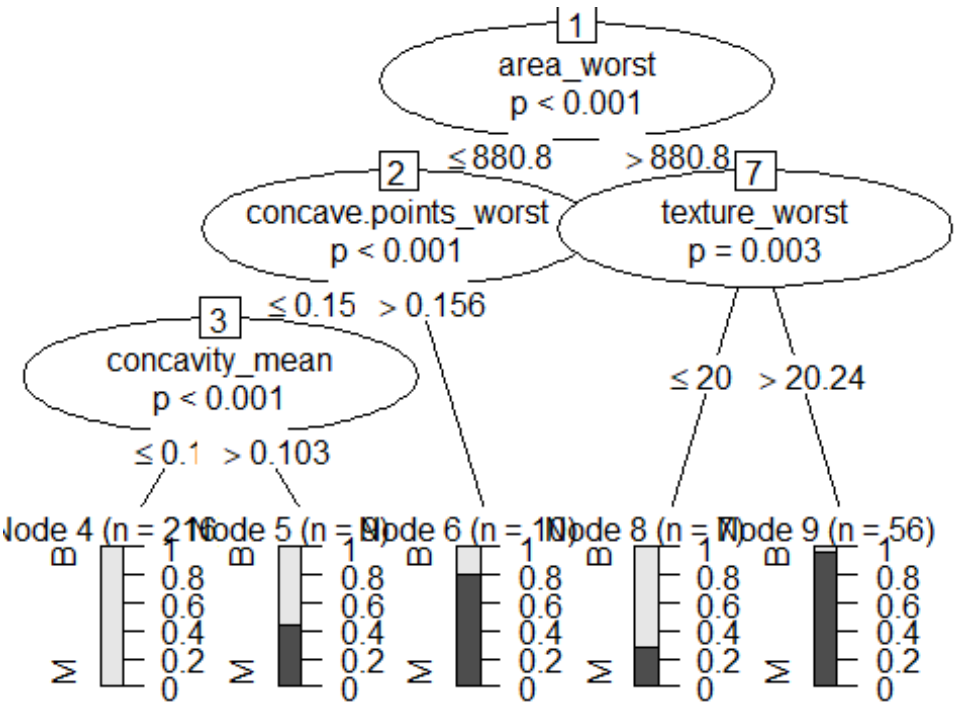
Using the measurement data provided, can we accurately predict whether a tumor is (M)alignant or (B)enign better than chance alone?

Brief overview of decision trees

Decision trees work by implementing a technique called recursive partitioning. Essentially, the algorithm analyzes each optional explanatory variable and chooses to split the data on the variable and a split value that provides the most information gain. If the explanatory variable chosen is continuous, the algorithm will find a numeric split value that minimizes the "impurity" in the resulting splitted data. Similarly, if the explanatory variable is categorical, the algorithm will choose a split value that maximizes the resulting classification probability of the resulting splitted data. The algorithm used is named Gini Impurity and its formula is $\sum_{i=1}^{J} {p_i}^2$, where p_i is the fraction of items labeled with class i and J is the number of classes we can choose from. The process will continue splitting until

no further information gain is achievable, or it meets a pre-defined stopping parameter. A brief demonstration is below.

```
set.seed(10131993)
fit <- ctree(diagnosis ~ ., data = train)
plot(fit)
```




```
(model_accuracy <- mean(as.matrix(as.character(predict(fit, newdata = test)))
== test$diagnosis))

## [1] 0.92
```

The output shows that using the decision tree model generated we have a prediction accuracy of 92%.

Problems with decisions trees, and why we use Random Forest

Decision trees suffer from high variance due to the fact that they select a splitting variable from the entire set of explanatory variables. Splitting our training data in two and fitting a decision tree to both could potentially yield very different results. One way to account for this is to use bootstrapping and aggregation. Because averaging a set of observations reduces variance (sigma^2 / n), if we take many different high-variance decision trees and average their results, we reduce the variance and increase predictability. When we don't have multiple training sets, we bootstrap instead to simulate a similar result. Bootstrapping is essentially taking repeated randomized samples from the same set of data. Once we

create models for each bootstrapped training set, we average the results. This is known as bagging.

The name "Random Forest" comes from this idea. We are averaging many different randomized decision trees, a.k.a. a forest. Bagging and Random Forest are slightly different, however. Because we are bootstrapping, we are adding correlation into our models, and therefore we can accidentally overfit some of them. Random Forest attempts to mitigate this by adding a paramter that dictates the number of variables the model will randomly select and then choose to split on based on the result from the Gini Impurity algorithm. This factor is denoted by "mtry". Each split selects an entirely new set of "mtry" number of variables. If our "mtry" was set to p, our total number of variables, then we would simply have bagging. The standard value for "mtry" in classification models is sqrt(p). In regression models it's p/3.

Below we run a few different Random Forest models to test our output and begin to understand what we're working with. First, we use the default number of "mtry". Following that, we specify the number of variables ourselves, to help us understand how it might affect our predictive accuracy.

```
set.seed(15674)
(rf_1 <- randomForest(diagnosis ~ ., data = train, importance = TRUE))
```

```
##
## Call:
##  randomForest(formula = diagnosis ~ ., data = train, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 5.03%
## Confusion matrix:
##     B  M class.error
## B 222  5  0.02202643
## M  10 61  0.14084507
```

```
# slightly decreases OOB error
rf_2 <- randomForest(diagnosis ~ ., data = train, importance = TRUE, mtry =
7)
rf_3 <- randomForest(diagnosis ~ ., data = train, importance = TRUE, mtry =
2)
(rf_4 <- randomForest(diagnosis ~ ., data = train, importance = TRUE, mtry =
9))
```

```
##
## Call:
##  randomForest(formula = diagnosis ~ ., data = train, importance = TRUE,
mtry = 9)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 9
```

```
##
##         OOB estimate of  error rate: 3.69%
## Confusion matrix:
##     B  M class.error
## B 224  3  0.01321586
## M   8 63  0.11267606

#getTree(rf_4, k = 13, labelVar = TRUE)
```

Testing the first model we built, rf_1, we see we achieve an accuracy of ~94%. This is decent and about 2% better than the decision tree alone.

```
pred_1 <- predict(rf_1, test, type = 'class')
table(pred_1, test$diagnosis)

##
## pred_1  B  M
##      B 71  4
##      M  2 23

mean(pred_1 == test$diagnosis)

## [1] 0.94
```

Next, we want to see if adjusting "mtry" can improve this accuracy. We see below that increasing "mtry" from the default value, 5, to a new value, 9, increases the prediction accuracy. This is expected because, as seen above, changing "mtry" from 5 to 9 decreases our OOB estimate of error rate. The OOB estiamte of error rate is described in more detail below.

```
mean(predict(rf_1, test, type = 'class') == test$diagnosis)

## [1] 0.94

mean(predict(rf_4, test, type = 'class') == test$diagnosis)

## [1] 0.96
```

Because random forest is a bagging method (bootstrap and aggregation), we can use the OOB error to estimate the test error, and we don't need a validation hold-out set or cross-validation. As explained earlier, bootstrapping repeatedly samples our training data set, which is similar to cross-validation. We can use the observations that were "held out of bag" of the given randomized decision tree as a means to estimate test error. In cross-validation terms, the out-of-bag observations in our random forest model would be the ith group of observations in a cross-validation model. Because we are doing this many times in random forest, we average the prediction from each tree and use that as our prediction result to test OOB error.

Using the importance() function, we can see which variables give the most information gain by having the greatest "MeanDecreaseGini", which is a value telling us how much, on average, that variable reduces the Gini Impurity calculation, which we want to be as small

as possible. Clearly, we want to choose the variables that reduce the Gini Impurity value the most. Only showing the first 6 values for brevity.

```
head(importance(rf_1))
```

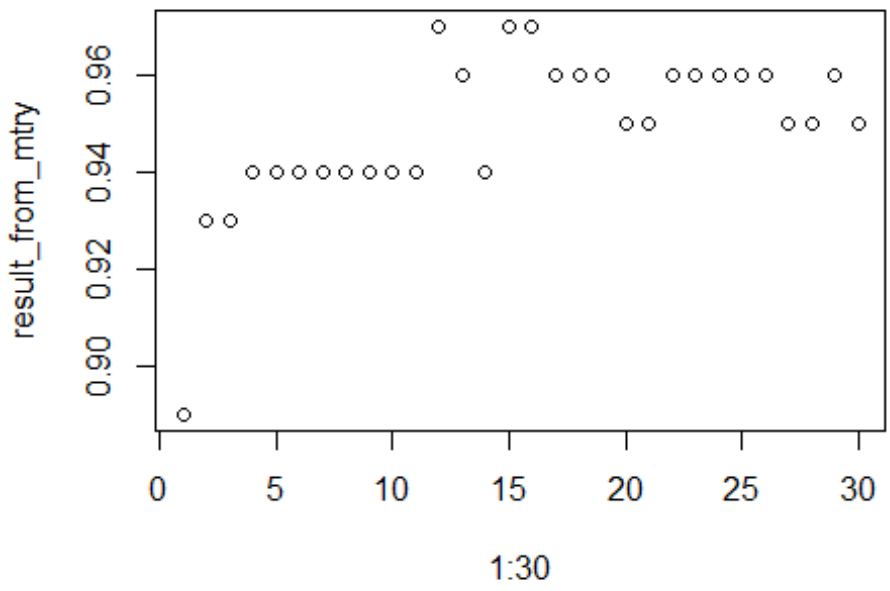
```
##                            B          M MeanDecreaseAccuracy MeanDecreaseGini
## radius_mean        8.253653  4.7202743             9.225530        4.4491263
## texture_mean       7.505818  9.6969347            11.168841        2.8465555
## perimeter_mean     6.887523  4.9832786             7.490533        4.0998432
## area_mean          7.478260  5.6040266             8.829310        4.2968850
## smoothness_mean    1.415358 -0.1303323             1.062467        0.4366171
## compactness_mean   3.280716  1.3429731             3.807789        1.0348600
```

Finally, knowing we have 30 different variables and that randomForest() tries "mtry" number of variables per split, we want to know what number supplied to "mtry" will give us the best accuracy. To do this, we loop over all optioins, 1:30, and record the results.

```
set.seed(06071990)

result_from_mtry <- c()

for (i in 1:30) {
  model3 <- randomForest(diagnosis ~ ., data = train, ntree = 500, mtry = i,
importance = TRUE)
  predTest <- predict(model3, test, type = "class")
  result_from_mtry[i] = mean(predTest == test$diagnosis)
}

results <- as.data.frame(result_from_mtry)
max(results)
```

```
## [1] 0.97
```

```
plot(1:30, result_from_mtry)
```



The output tells us the max accuracy is 0.97, which is more accurate than our single decision tree.

Testing significance of results

```
probSuccesss <- summary(test$diagnosis)[1]/sum(summary(test$diagnosis))

randomClass_B <- rbinom(10000, 89, probSuccesss)
randomClass_M <- rbinom(10000, 143-89, 1-probSuccesss)
randomClass <- randomClass_B + randomClass_M
randomClass <- as.data.frame(randomClass)
randomClass$accuracy <- randomClass$randomClass/143

length(filter(randomClass, accuracy >= mean(predict(rf_4, test, type =
'class') == test$diagnosis))[,1])

## [1] 0

mu <-  mean(randomClass$randomClass)
stdev <- sd(randomClass$randomClass)
qnorm(0.975, mean = mu, sd = stdev)

## [1] 89.89243

sum(predict(rf_4, test, type = 'class') == test$diagnosis)

## [1] 96
```

We see that when randomly generated, there is less than 2.5% probability that the number of correctly predicted classifcations our model returned in our test data (96) would be returned by chance alone. This indicates that our results are not likely due to chance and that our model is significant.

References

Friedman, J., Tibshirani, R., Hastie, T., The Elements of Statistical Learning r-bloggers: https://www.r-bloggers.com/how-to-implement-random-forests-in-r/