Predicting Seminal Quality with Random Forest Classification

Dealing with Imbalanced Data

Rachel H. Oakley

```r
library(readxl)
library(dplyr)
library(caret)
library(randomForest)
library(tidyverse)
library(rpart)
library(rpart.plot)
library(smotefamily)
library(mlbench)
library(performanceEstimation)
library(rattle)
```

Introduction

Male and female infertility has been rising over the recent decades. Scientific American states that “the whole spectrum of reproductive problems in males are increasing by about 1 percent per year in Western countries,” (Swan, Colino 2021). While seemingly small, this number adds up over the decades to an extremely significant concern. Further, fertility issues can be linked to serious underlying medical troubles and should not be overlooked. Therefore, it is exceptionally important to study what factors may be impacting fertility.

Method

This project aims to contribute to the identification of male fertility problems initially explored in Gil et al. (2013) by using a random forest method for classification prediction modeling, as laid out in Liaw (2002).

Random forests are an “ensemble” learning method for classification which have been found to be useful high-dimensional datasets and can handle binary, categorical, and numerical factors with very little pre-processing. It’s a quick way to train a model and it is robust to outliers and non-linear data, (Kho 2018).

Gil et al. (2013) used this data in conjunction with multiple artificial intelligence methods (Multilayer Perceptron, Support Vector Machines, and Decision Trees) to compare their efficacy for predicting seminal quality. These methods were shown to have high accuracy but low specificity and positive predictive value. The authors posit that this may be due to the imbalanced nature of the data.

By utilizing random forests along with methods of balancing data, it may be possible to improve specificity and positive predictive value.

Data description

Using semen quality as a predictor for male fertility potential, data collected by David Gil et al. (2013) (https://archive.ics.uci.edu/ml/datasets/Fertility) will be used to train a random forest model for classifying semen quality. The data was collected from 100 volunteer sperm donors between 18 and 36 years of age.

It consists of **10** variables:

1. **season** when the analysis was performed
2. **age** at the time of analysis
3. childish **diseases** (i.e., chicken pox, measles, mumps, polio) (binary: yes, no)
4. accident or serious **trauma** (binary: yes, no)
5. **surgical** intervention (binary: yes, no)
6. recency of high **fevers**
7. frequency of **alcohol** consumption
8. **smoking habit**
9. number of hours spent **sitting** per day
10. Response: **diagnosis** (binary: normal, abnormal).

The first 9 of these variables are predictor variables while the 10th is the response variable.

Here are a few inputs to be able to quickly change our hyperparameters throughout:

```
m = 3 # Number of drawn candidate variables in each split
ntree = 1000 # Number of trees in the random forest
minsplit = 1 # The minimum number of data points required to attempt a split before it is forced
to create a terminal node. Default is 20.
```

Let’s import our data and transform the response to a binary factor.

```
fert <- read_excel("fertility_Diagnosis.xlsx")
# Turn 'N' and 'O' in diagnosis to binary response where 'N'=0 and 'O'=1
fert$diagnosis <- as.factor(fert$diagnosis)
fert$season <- as.factor(fert$season)
fert$chld_dis <- as.factor(fert$chld_dis)
fert$trauma <- as.factor(fert$trauma)
fert$surgery <- as.factor(fert$surgery)
fert$fevers <- as.factor(fert$fevers)
fert$smoking <- as.factor(fert$smoking)
fert <- fert %>%
  dplyr::mutate(diagnosis = recode(diagnosis, "N"="0", "O"="1"))
head(fert)
```

season <fct>	age <dbl>	chld_dis <fct>	trauma <fct>	surgery <fct>	fevers <fct>	alc_freq <dbl>	smoking <fct>	sitting <dbl>	diagnosis <fct>
-0.33	0.69	0	1	1	0	0.8	0	0.88	0
-0.33	0.94	1	0	1	0	0.8	1	0.31	1
-0.33	0.50	1	0	0	0	1.0	-1	0.50	0
-0.33	0.75	0	1	1	0	1.0	-1	0.38	0
-0.33	0.67	1	1	0	0	0.8	-1	0.50	1
-0.33	0.67	1	0	1	0	0.8	0	0.50	0

6 rows

Imbalanced Data

Unbalanced data can cause problems in classification. When there is a large disparity in the frequencies of the observed classes, **classification models will almost always favor the majority class** in its predictions unless extra steps are taken in the pre-processing of the data.

In our data set, when diagnosis = ‘0’ this indicates normal seminal quality, while diagnosis = ‘1’ indicates abnormal seminal quality. We will likely need to take extra steps to compensate for the imbalanced nature of the data.

Our data set contains 88 normal samples and 12 abnormal samples.

```
print(table(fert$diagnosis))
```

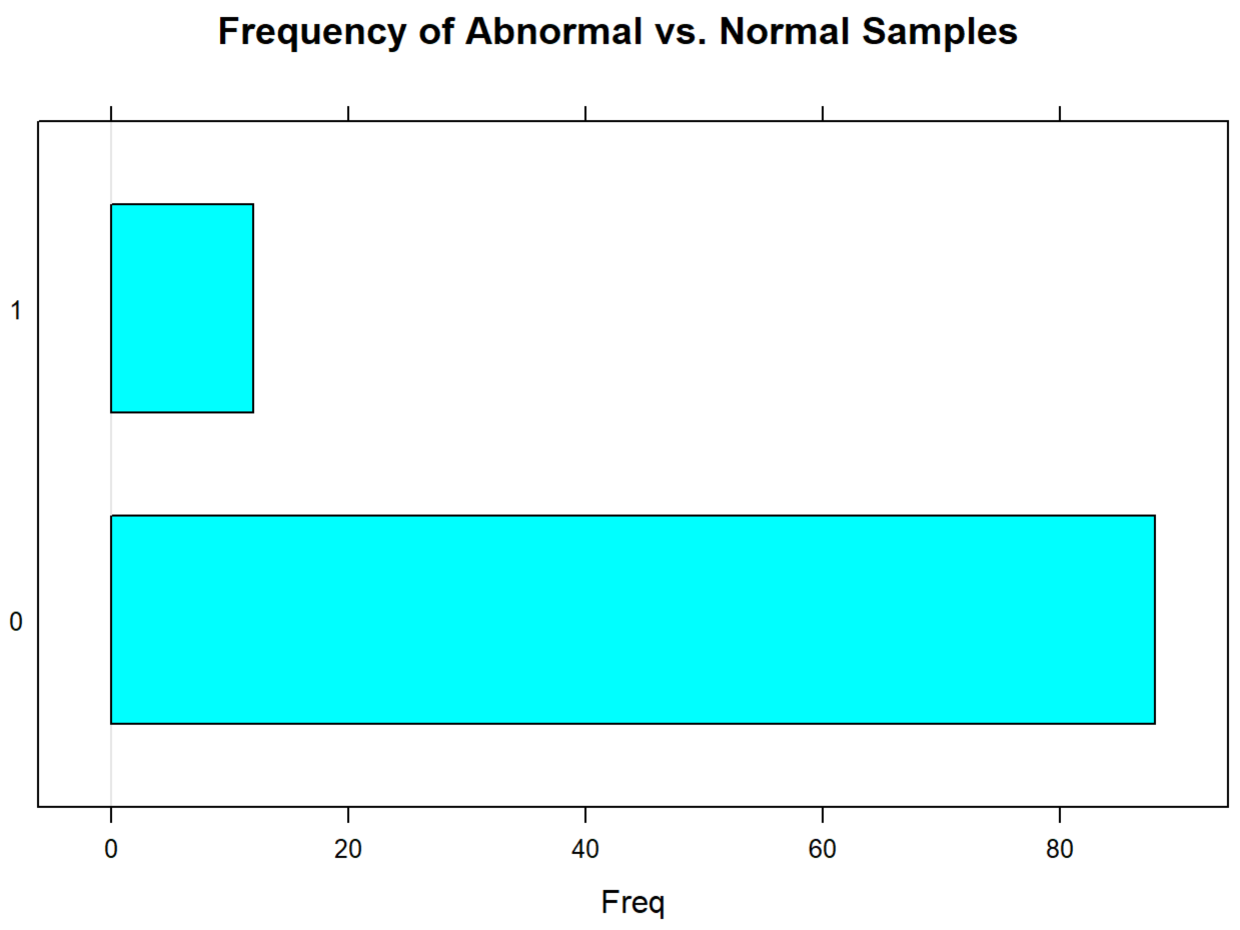
```
## 
##  0  1 
## 88 12
```

So the respective proportions of normal and abnormal classes are .88 and .12.

```
print(prop.table(table(fert$diagnosis)))
```

```
##
##    0    1
## 0.88 0.12
```

```
barchart(fert$diagnosis, main='Frequency of Abnormal vs. Normal Samples')
```



Partitioning the Data

This imbalance in the data will likely cause problems in training and testing data, as our classification model will favor the majority class.

Let's first create train and test sets of this data without making any efforts to balance the response. We partition the data in one of three commonly used fashions (https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/):

- Train: 67% Test: 33%

```
set.seed(123)
partition <- createDataPartition(fert$diagnosis, p = .67,
                                  list = FALSE,
                                  times = 1)
imbal_train <- fert[as.vector(partition),]
imbal_test <- fert[-as.vector(partition),]
```

Using the function createDataPartition partitions the data in such a way as to maintain the class proportions in the original data set when partitioning to the train and test sets.

Class proportions for training set:

```
imbal_train_prop_table <- table(imbal_train$diagnosis)
imbal_train_prop_table
```

```
##
##  0  1
## 59  9
```

Class proportions for testing set:

```
imbal_test_prop_table <- table(imbal_test$diagnosis)
imbal_test_prop_table
```

```
##
##  0  1
## 29  3
```

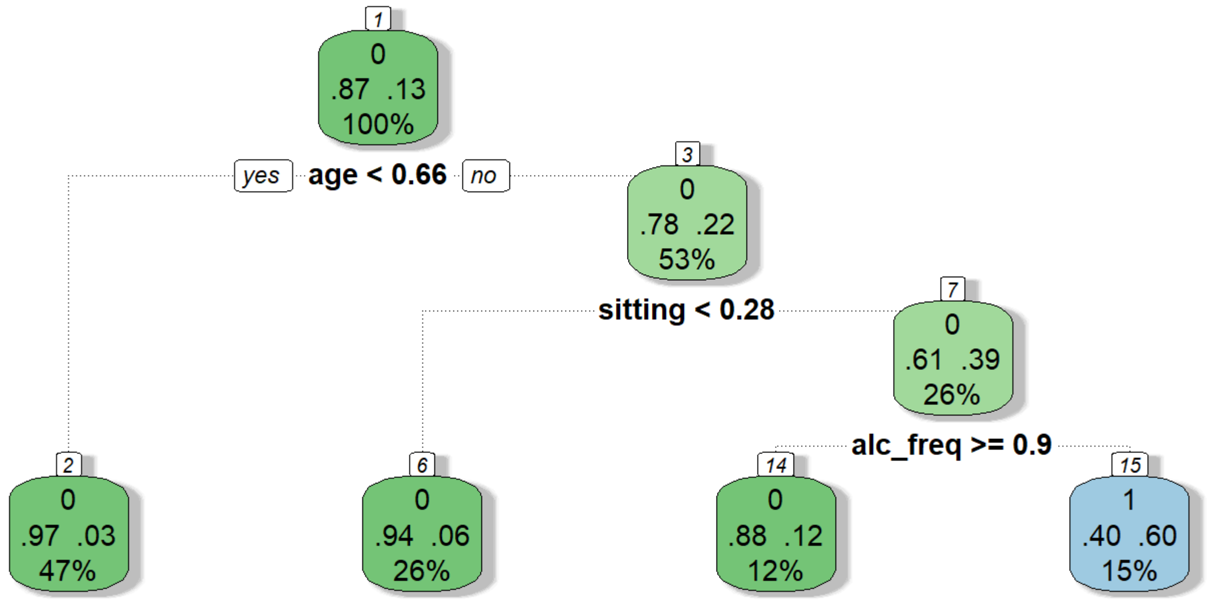
Decision Trees

Decision Trees are at the heart of random forests, so lets see what one looks like.

A decision tree uses a set of if-else conditions to visualize and classify data according to conditions.

Here is a decision tree built from our imbalanced training data.

```
set.seed(23)
dec_tree <- train(diagnosis~.,data=imbal_train, method = 'rpart',
                  control=rpart.control(minsplit =1,  minbucket = 4))
fancyRpartPlot(dec_tree$finalModel, sub= 'Decision Tree')
```



Decision Tree

The root node is age, and this tells us that from this data age was found to be the most influential variable in predicting seminal quality.

87% of the training set have normal seminal quality, while 13% have abnormal quality. The root node asks if the subject is under age .66 (=roughly 30 years old). If yes, then you go to the left “child” node. This node tells us that of the 47% of subjects who are under 30, 97% of them have normal sperm quality and 3% have abnormal.

As for the 53% of subjects older than 30, 78% have normal and 22% have abnormal. Of these subjects, the question of whether the subject sits less than .28(= about 5.2 hours) per day. If yes, travel down to the left terminal node.

26% of total subjects are over age 30 *and* sit *less* than 5.2 hours per day, and of those subjects 94% have normal seminal quality and 6% have abnormal. 26% of total subjects are over age 30 *and* sit *more* than 5.2 hours per day, and of those subjects 61% have normal seminal quality and 39% have abnormal.

Finally, the last question is asked of subjects older than 30 who sit more than 5.2 hours per day: Is their alcohol consumption greater than or equal to 0.9 (where 0.9 falls on a scale of 0 to 1 where 0 is drinking several times a day and 1 is never or hardly ever drinking.)

So, 12% of all the subjects are over age 30 *and* sit *more* than 5.2 hours per day *and* rarely drink. Of these 12%, 88% have normal quality and 12% have abnormal.

15% of all subjects are over age 30 *and* sit *more* than 5.2 hours per day *and* drink more often than rarely. Of these subjects, 40% have normal semen quality and 60% have abnormal.

Let’s use this tree to predict abnormal seminal quality in our training set:

```
dt_test_pred <- predict(dec_tree, imbal_test, type="raw")
dt_pred_cm <- confusionMatrix(dt_test_pred, imbal_test$diagnosis, positive="1")
dt_pred_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 23  1
##          1  6  2
##
##                Accuracy : 0.7812
##                  95% CI : (0.6003, 0.9072)
##     No Information Rate : 0.9062
##     P-Value [Acc > NIR] : 0.9920
##
##                   Kappa : 0.2632
##
##  Mcnemar's Test P-Value : 0.1306
##
##             Sensitivity : 0.66667
##             Specificity : 0.79310
##          Pos Pred Value : 0.25000
##          Neg Pred Value : 0.95833
##              Prevalence : 0.09375
##          Detection Rate : 0.06250
##    Detection Prevalence : 0.25000
##       Balanced Accuracy : 0.72989
##
##        'Positive' Class : 1
##
```

Using this tree on the test data set, we get a prediction **accuracy** of 78%, **sensitivity** of 66.7%, and **specificity** of 79%. The accuracy is the overall percentage of correct guesses. Sensitivity is the percentage of true positives that were guessed correctly and specificity is the number of true negatives guessed correctly.

Nodes make their splitting decisions based on their “Gini index.” This is a measure of what is known as “node purity,” which put simply is the probability that a feature was classified incorrectly.

Gini index

$$G = 1 - \sum_{k=1}^{K} (p_k)^2$$

“Purer” nodes have smaller Gini indexes.

Decision trees are useful because they are easy to interpret and provide a clear path to the classification of a subject, but they tend to perform poorly when moving from training to test data. They tend to overfit the training set, which leads to increased variance. Random forests can be used to decrease this variability.

Random Forests

A random forest basically makes decisions about classification by building many decision trees using ‘bagging’ (bootstrapping and aggregating). Here are the steps:

1. **We make a bootstrapped training dataset.** This dataset is the same size as the original training dataset, but it is built by randomly sampling from the original dataset *with replacement*.
2. **Build a decision tree using that dataset**, but instead allowing any variable to be the root node choose a random subset from which to select candidates for the root node and subsequent nodes.
3. **Repeat!** With however many trees you want in your forest.
4. The test dataset is **run through each tree in the random forest**. Votes of each tree for each datum are **aggregated** and a final vote is made based on the majority vote.

Imbalanced Random Forest

Now we will attempt to perform random forest classification using these imbalanced training and testing sets. A random forest is essentially a large collection of decision trees where at each stage of the tree construction process we randomly select which variables to consider. To do this, the parameter “mtry” is set to a number less than the number of independent variables in the data set. Typically this value is chosen as 1/3 of the available variables. Since we have 9 independent variables in our data, we will set it to mtry=3.

Here is the creation of the random forest:

```
set.seed(4564)
imbal_train_forest <- randomForest(diagnosis ~ .,
                  data=imbal_train,
                  mtry=3,
                  importance=TRUE,
                  keep.Forest=TRUE,
                  ntree= ntree,
                  nodesize=2)
imbal_train_forest
```

```
##
## Call:
##  randomForest(formula = diagnosis ~ ., data = imbal_train, mtry = 3,      importance = TRUE,
keep.Forest = TRUE, ntree = ntree, nodesize = 2)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 16.18%
## Confusion matrix:
##    0 1 class.error
## 0 57 2  0.03389831
## 1  9 0  1.00000000
```

Now we use this random forest based on the imbalanced dataset to make a prediction using the imbalanced test data.

```
imbal_test_pred <- predict(imbal_train_forest, imbal_test, type="class")
imbal_pred_cm <- confusionMatrix(imbal_test_pred, imbal_test$diagnosis, positive="1")
imbal_pred_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 29  3
##          1  0  0
##
##                Accuracy : 0.9062
##                  95% CI : (0.7498, 0.9802)
##     No Information Rate : 0.9062
##     P-Value [Acc > NIR] : 0.6474
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 0.00000
##             Specificity : 1.00000
##          Pos Pred Value :     NaN
##          Neg Pred Value : 0.90625
##              Prevalence : 0.09375
##          Detection Rate : 0.00000
##    Detection Prevalence : 0.00000
##       Balanced Accuracy : 0.50000
##
##        'Positive' Class : 1
##
```

Wow, our model is almost 90% accurate!! Not so fast. None of the values that were actually positive were predicted to be positive, which is why our sensitivity is 0. Most likely, this is due to the algorithm “preferring” the majority class.

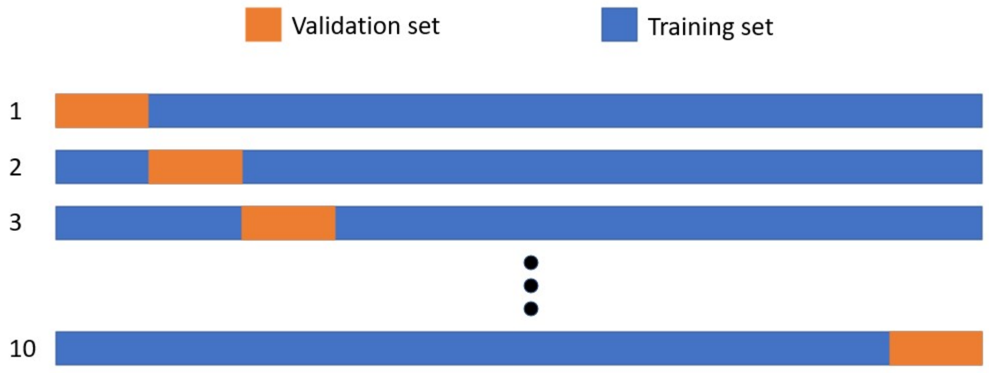
What can we do to boost this metric?

K-Fold Cross Validation

Gil et al. (2013) utilized an approach called K-Fold Cross Validation in their report on predicting seminal quality with artificial intelligence methods. The three AI approaches they covered were (1) Decision Trees, (2) Multilayer Perceptron, and (3) Support Vector Machines.

Instead of the validation set approach we used above, Gil et al. used K-Fold Cross Validation as their model evaluation method, with k=10. Instead of splitting the data into train and test sets *once*, this approach does it k times.

```
knitr::include_graphics('10foldcrossvalidation.jpg')
```



While this approach can be a good choice for small datasets like this one, it is not well suited to imbalanced data. More information on how k-fold cross validation works can be found here: [link] https://www.geeksforgeeks.org/k-fold-cross-validation-in-r-programming/ (https://www.geeksforgeeks.org/k-fold-cross-validation-in-r-programming/)

We would like to compare the results of using random forests with the methods used in Gil et al., so here we will use cross validation as our model evaluation approach. A video of the approach used can be found here: [link]https://www.youtube.com/watch?v=M7BMgHwXUoY (https://www.youtube.com/watch?v=M7BMgHwXUoY)

```
set.seed(123)
fitControl <- trainControl(method='cv',
                           number=10,
                           search='random',savePredictions=T)
modelFitrf <- train(diagnosis ~ ., data=fert,
                    method='rf',
                    trControl= fitControl, tuneLength=10, ntree=ntree,control = rpart.control(mi
nsplit =2,
              minbucket = 2))
```

From this model we can extract the optimal value of mtry, which is found to be 3.

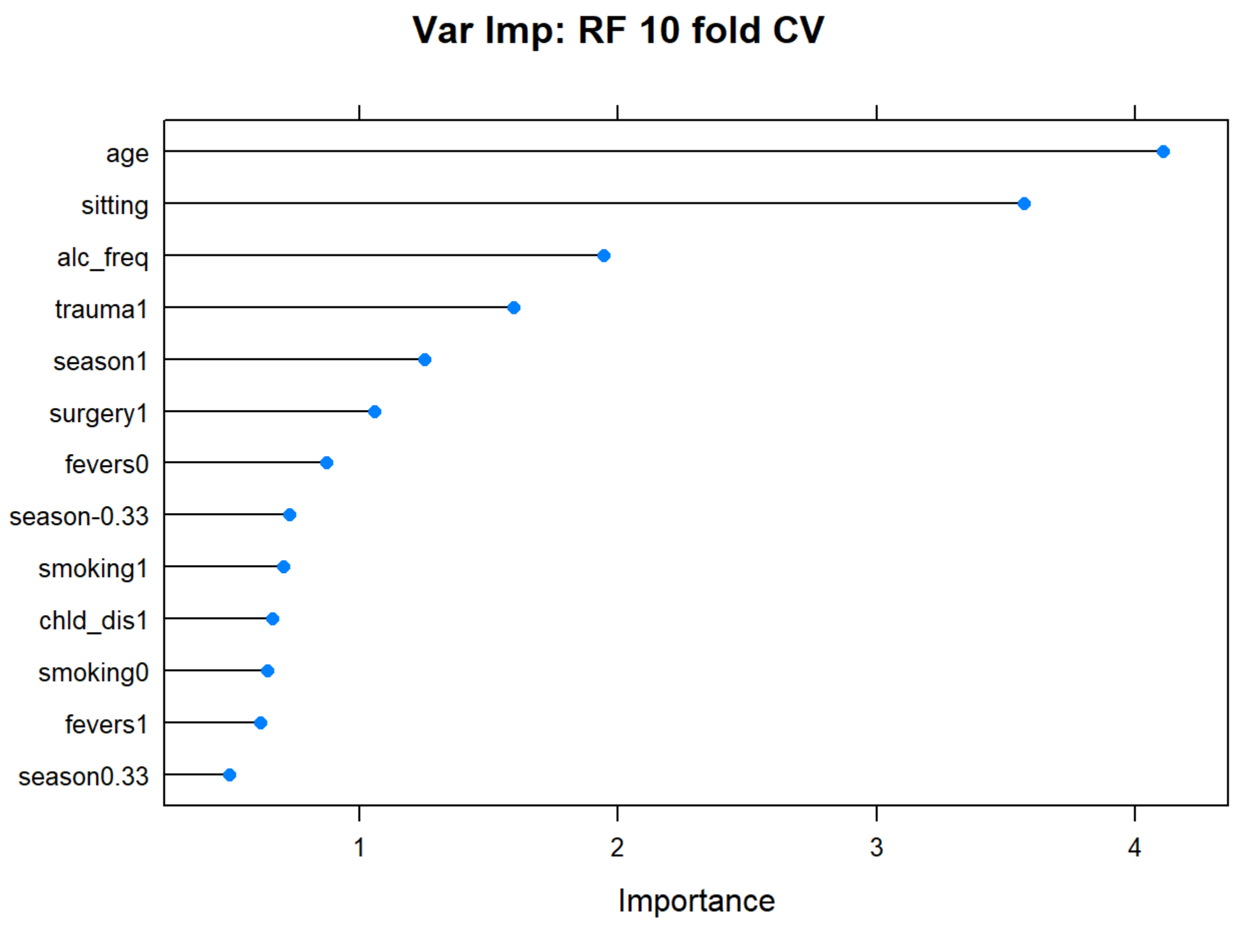
```
modelFitrf$bestTune
```

	mtry <int>
2	3

1 row

And create a plot of what are found to be the most important variables. Age and the amount of time spent sitting each days are the most impactful.

```
plot(varImp(modelFitrf, scale=F), main= "Var Imp: RF 10 fold CV")
```



The optimal value of mtry is used to predict the dataset.

```
sub_rf1= subset(modelFitrf$pred, modelFitrf$pred$mtry==modelFitrf$bestTune$mtry)
kfold_cm <- confusionMatrix(table(sub_rf1$pred,sub_rf1$obs),positive="1")
kfold_cm
```

```
## Confusion Matrix and Statistics
##
##
##      0  1
##   0 87 10
##   1  1  2
##
##                Accuracy : 0.89
##                  95% CI : (0.8117, 0.9438)
##     No Information Rate : 0.88
##     P-Value [Acc > NIR] : 0.45422
##
##                   Kappa : 0.2297
##
##  Mcnemar's Test P-Value : 0.01586
##
##             Sensitivity : 0.1667
##             Specificity : 0.9886
##          Pos Pred Value : 0.6667
##          Neg Pred Value : 0.8969
##              Prevalence : 0.1200
##          Detection Rate : 0.0200
##    Detection Prevalence : 0.0300
##       Balanced Accuracy : 0.5777
##
##        'Positive' Class : 1
##
```

Okay, this is still not great, but it's also not surprising since k-fold cross-validation isn't a good choice for imbalanced data. We will have to use another method for our approach.

Resampling

Let's try resampling our dataset. There are two main methods of resampling for even-ing out the dataset.

1. **Down-sampling:** This is where we delete instances of the majority class (i.e. delete rows of our dataframe where the response is "Normal" until there is balance between the two classes)
2. **Up-sampling:** This is where we add copies of instances of the minority class (i.e we add copies of rows where the response is "Abnormal" until there is balance between the two classes)

Let's try both and see which delivers the best results. The method used for this approach and more information on it can be found here: [link]https://topepo.github.io/caret/subsampling-for-class-imbalances.html (https://topepo.github.io/caret/subsampling-for-class-imbalances.html)

Down-Sampling

Starting with creating a down-sampled training set. Note that down-training and up-training are most applied to the training set after the data has been split into it's training and test sets.

```
set.seed(3240)
down_train <- downSample(x = imbal_train[, -ncol(imbal_train)],
                         y = imbal_train$diagnosis)
table(down_train$Class)
```

```
## 
## 0 1 
## 9 9
```

Our down-sampled training set has 9 normal and 9 abnormal observations.

```
set.seed(324)
down_forest <- randomForest(Class ~ .,
                  data=down_train, mtry=modelFitrf$bestTune$mtry,
                  importance=TRUE,
                  keep.Forest=TRUE,
                  ntree=1000)
```

```
down_test_pred <- predict(down_forest, imbal_test, type="class")
down_pred_cm <- confusionMatrix(down_test_pred, imbal_test$diagnosis, positive="1")
down_pred_cm
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction  0  1
##          0 19  0
##          1 10  3
##                                           
##                Accuracy : 0.6875          
##                  95% CI : (0.4999, 0.8388)
##     No Information Rate : 0.9062          
##     P-Value [Acc > NIR] : 0.999903        
##                                           
##                   Kappa : 0.2627          
##                                           
##  Mcnemar's Test P-Value : 0.004427        
##                                           
##             Sensitivity : 1.00000         
##             Specificity : 0.65517         
##          Pos Pred Value : 0.23077         
##          Neg Pred Value : 1.00000         
##              Prevalence : 0.09375         
##          Detection Rate : 0.09375         
##    Detection Prevalence : 0.40625         
##       Balanced Accuracy : 0.82759         
##                                           
##        'Positive' Class : 1               
## 
```

Using this to train a random forest and applying it to our test set results in an accuracy of 68.75%, sensitivity of 100%, and specificity of 65.5%. These are the best results we have seen so far, although it must be noted that there is a trade-off between achieving higher sensitivity and a resulting lowered specificity. In real world situations, however, it may be preferred to have a lower specificity if it means lowering the rate of false negatives.

Up-Sampling

```
set.seed(3243)
up_train <- upSample(x = imbal_train[, -ncol(imbal_train)],
                     y = imbal_train$diagnosis)
table(up_train$Class)
```

```
##
##  0  1
## 59 59
```

Our up-sampled training set has 59 normal and 59 abnormal observations.

```
set.seed(98)
up_forest <- randomForest(Class ~ .,
                  data=up_train, mtry=modelFitrf$bestTune$mtry,
                  importance=TRUE,
                  keep.Forest=TRUE,
                  ntree=1000,
                  nodesize=1)
```

```
up_test_pred <- predict(up_forest, imbal_test, type="class")
up_pred_cm <- confusionMatrix(up_test_pred, imbal_test$diagnosis, positive="1")
up_pred_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 26  3
##          1  3  0
##
##                Accuracy : 0.8125
##                  95% CI : (0.6356, 0.9279)
##     No Information Rate : 0.9062
##     P-Value [Acc > NIR] : 0.9737
##
##                   Kappa : -0.1034
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.00000
##             Specificity : 0.89655
##          Pos Pred Value : 0.00000
##          Neg Pred Value : 0.89655
##              Prevalence : 0.09375
##          Detection Rate : 0.00000
##    Detection Prevalence : 0.09375
##       Balanced Accuracy : 0.44828
##
##        'Positive' Class : 1
##
```

Applying the up-sampled trained random forest to the test set results in an accuracy of 81%, sensitivity of 0%, and specificity of 89.7%. Training the data with replications of the abnormal classifications has resulted in far worse predictions than trading the data with fewer normal classifications, as with down-sampling.

SMOTE

A hybrid approach to re-sampling is something called SMOTE (synthetic minority over-sampling technique). The general idea of this method is to artificially generate new examples of the minority class using the nearest neighbors of these cases. Furthermore, the majority class examples are also under-sampled, leading to a more balanced dataset.

```
set.seed(98960)
smote_train <- smote(diagnosis ~ ., data  = imbal_train, k=8)
table(smote_train$diagnosis)
```

```
##
##  0  1
## 36 27
```

Our SMOTEd training set has 36 normal and 27 abnormal observations.

```
set.seed(26633)
smote_forest <- randomForest(diagnosis ~ .,
                  data=smote_train, mtry=modelFitrf$bestTune$mtry,
                  importance=TRUE,
                  keep.Forest=TRUE,
                  ntree=1000,
                  nodesize=1)
```

```
smote_test_pred <- predict(smote_forest, imbal_test, type="class")
smote_pred_cm <- confusionMatrix(smote_test_pred, imbal_test$diagnosis, positive="1")
smote_pred_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 23  2
##          1  6  1
##
##                Accuracy : 0.75
##                  95% CI : (0.566, 0.8854)
##     No Information Rate : 0.9062
##     P-Value [Acc > NIR] : 0.9979
##
##                   Kappa : 0.0791
##
##  Mcnemar's Test P-Value : 0.2888
##
##             Sensitivity : 0.33333
##             Specificity : 0.79310
##          Pos Pred Value : 0.14286
##          Neg Pred Value : 0.92000
##              Prevalence : 0.09375
##          Detection Rate : 0.03125
##    Detection Prevalence : 0.21875
##       Balanced Accuracy : 0.56322
##
##        'Positive' Class : 1
##
```

The results of this approach are accuracy of 75%, sensitivity of 33%, and specificity of 79%.

Balanced Bootstrap Sampling

The last technique we used was balanced bootstrap sampling, where each random forest tree is built from a bootstrap sample containing the same number of normal and abnormal classifications (Scott 2017). Cases are drawn with replacement from each class, and we set this value equal to the size of the minority class. This is implemented using the sampsize argument from the randomForest package.

```
set.seed(34324)
ssweights <- c(imbal_train_prop_table[['1']],imbal_train_prop_table[['1']])
weighted_forest <- randomForest(diagnosis ~ .,
                  data=imbal_train, mtry=modelFitrf$bestTune$mtry,
                  importance=TRUE,
                  keep.Forest=TRUE,
                  ntree=1000,
                  nodesize=1,
                  sampsize=ssweights)
weighted_forest
```

```
##
## Call:
##  randomForest(formula = diagnosis ~ ., data = imbal_train, mtry = modelFitrf$bestTune$mtry,
importance = TRUE, keep.Forest = TRUE, ntree = 1000, nodesize = 1,      sampsize = ssweights)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 22.06%
## Confusion matrix:
##    0  1 class.error
## 0 49 10   0.1694915
## 1  5  4   0.5555556
```

```
weighted_test_pred <- predict(weighted_forest, imbal_test, type="class")
weighted_pred_cm <- confusionMatrix(weighted_test_pred, imbal_test$diagnosis, positive="1")
weighted_pred_cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 24  1
##          1  5  2
##
##                Accuracy : 0.8125
##                  95% CI : (0.6356, 0.9279)
##     No Information Rate : 0.9062
##     P-Value [Acc > NIR] : 0.9737
##
##                   Kappa : 0.3094
##
##  Mcnemar's Test P-Value : 0.2207
##
##             Sensitivity : 0.66667
##             Specificity : 0.82759
##          Pos Pred Value : 0.28571
##          Neg Pred Value : 0.96000
##              Prevalence : 0.09375
##          Detection Rate : 0.06250
##    Detection Prevalence : 0.21875
##       Balanced Accuracy : 0.74713
##
##        'Positive' Class : 1
##
```

Results of this technique are accuracy of 81%, sensitivity of 66.7%, and specificity of 82.8%. Figure 9. This appears to achieve a happy medium between accuracy, sensitivity, and specificity.

Conclusion

The overall comparison of these methods can be seen below. From this table we can see that there are a lot of challenges in finding the right balance between Sensitivity and Specificity when dealing with a small, imbalanced dataset like this. The approaches with the best results appear to be the down-sampled random forest and the balanced bootstrap sampling random forest. The down-sampled random forest may be most appropriate for real world situations if the goal is to flag possible abnormal sperm quality for further examination and testing, as the rate of false negatives is lowest using this approach.

```
examples <- c('Imbalanced RF', 'K-Fold RF', 'Down-Sampled RF', 'Up-Sampled RF', 'Balanced Bootst
rap RF', 'SMOTE RF')
metrics <- c('Accuracy', 'Sensitivity', 'Specificity')



comparison_df <- data.frame(Method = examples, Accuracy = accuracy, Sensitivity = round(sensitiv
ity,4), Specificity = round(specificity,4))
comparison_df
```

Method	Accuracy	Sensitivity	Specificity
<chr>	<dbl>	<dbl>	<dbl>
Imbalanced RF	0.90625	0.0000	1.0000
K-Fold RF	0.89000	0.1667	0.9886
Down-Sampled RF	0.68750	1.0000	0.6552
Up-Sampled RF	0.81250	0.0000	0.8966
Balanced Bootstrap RF	0.81250	0.6667	0.8276
SMOTE RF	0.75000	0.3333	0.7931

6 rows