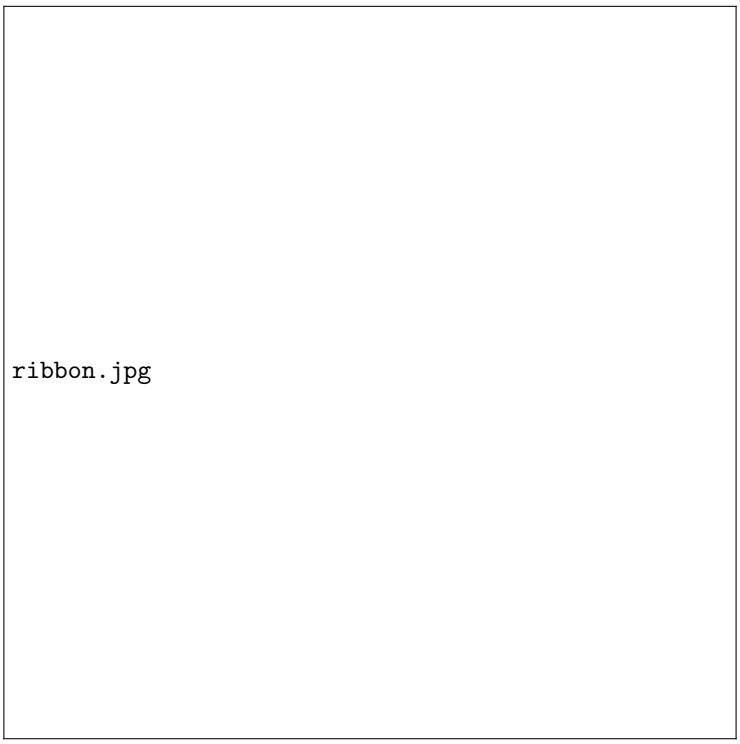

Breast Cancer Investigation

Craig Pirie

19/11/18

Contents

1 Data Explorationn

1.1 Dataset Choice

Breast Cancer Prediction Dataset
Source: https://www.kaggle.com/merishnasuwal/breast-cancer-prediction-dataset

Introduction: I chose this dataset as breast cancer is a huge problem within the world and is a vicious disease that affects so many of us. I would like to see if there is any way we can predict a positive diagnosis of thus disease to aid in the harm reduction of Breast Cancer.

1.2 Problem Statement and Data Exploration

Brief Description: This dataset contains records of people and information on their breasts such as size and shape alongside their breast cancer diagnosis.

Aim and Objectives: The main aim of this investigation is to use the Breast Cancer Prediction Dataset in order to predict those at risk of Breast Cancer.

Data Exploration: I now begin to explore the data.

The data is loaded from the 'breast cancer data.csv' file into a dataframe called 'df'. Our dataset currently holds missing values as a blank string such as "" but we replace them here with "N/A".

```
> #Import data set from CSV file
> df <- read.csv('C:/Users/Craig/Documents/3RD YEAR/Big Data/breast_cancer_data.csv', header
```

I will now check the class distribution of the dataset.
We can see here that the diagnosed class is dominant within the dataset as it has almost double the presence than un-diagnosed classes. This means we have a largely unbalanced dataset.

I want to know the names of my features so I enter the following command:-

```
> names(df) #Names of features
```

```
[1] "mean_radius"     "mean_texture"    "mean_perimeter"  "mean_area"
[5] "mean_smoothness" "diagnosis"
```

Our feature names are meanradius, meantexture, meanperimiter, meanarea, meansmoothness and diagnosis.

It will be beneficial for us to know the number of rows and columns our data set has.

```
> labelFreqs <- table(df$diagnosis)#Holds frequency of diagnosis in a table
> barplot(labelFreqs,col = gray.colors(3), #Plots a graph from the frequency of diagnosis ta
+         main="Diagnosis Frequency") #With grey bars and labelled 'Diagnosis Frequency'
```

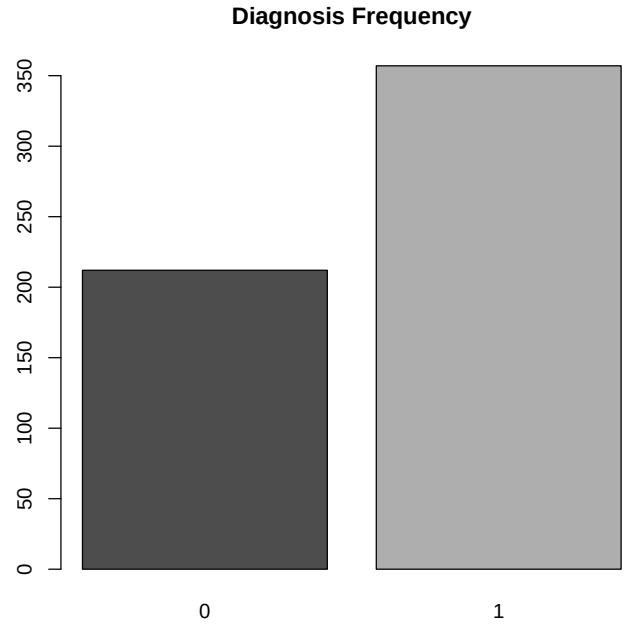

Figure 1: Dataframe Class Distribution

```
> nr <- nrow(df) #Counts the number of rows in our dataset and stores in a variable called
> nc <- ncol(df) #Counts the number of columns in our dataset and stores in a variable calle
> cat("Our data set has: ", nr, " Rows and ", nc, " Columns") #Displays number of rows and c

Our data set has:  569  Rows and  6  Columns
```

We can confirm this using the dim command...

```
> dim(df) #Dimensions of dataset (Number of rows/ Number of columns)

[1] 569   6
```

Values match so the values are confirmed

Lets get a quick description of our dataset

```
> str(df) #Displays quick description of data set

'data.frame':        569 obs. of  6 variables:
 $ mean_radius    : num  18 20.6 19.7 11.4 20.3 ...
 $ mean_texture   : num  10.4 17.8 21.2 20.4 14.3 ...
 $ mean_perimeter : num  122.8 132.9 130 77.6 135.1 ...
 $ mean_area      : num  1001 1326 1203 386 1297 ...
 $ mean_smoothness: num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
 $ diagnosis      : int  0 0 0 0 0 0 0 0 0 0 ...

> head(df) #Displays top 6  records

  mean_radius mean_texture mean_perimeter mean_area mean_smoothness diagnosis
1       17.99        10.38         122.80    1001.0         0.11840         0
2       20.57        17.77         132.90    1326.0         0.08474         0
3       19.69        21.25         130.00    1203.0         0.10960         0
4       11.42        20.38          77.58     386.1         0.14250         0
5       20.29        14.34         135.10    1297.0         0.10030         0
6       12.45        15.70          82.57     477.1         0.12780         0

> tail(df) #Displays bottom 6 records

    mean_radius mean_texture mean_perimeter mean_area mean_smoothness diagnosis
564       20.92        25.09         143.00    1347.0         0.10990         0
565       21.56        22.39         142.00    1479.0         0.11100         0
566       20.13        28.25         131.20    1261.0         0.09780         0
567       16.60        28.08         108.30     858.1         0.08455         0
568       20.60        29.33         140.10    1265.0         0.11780         0
569        7.76        24.54          47.92     181.0         0.05263         1
```

Now we have an idea of what the data we are working with looks like

Lets start getting some values that could be useful to know before building our model

```
> min(df$mean_area) #What is the smallest breast area?

[1] 143.5

> max(df$mean_area) #What is the largest breast area?

[1] 2501

> mean(df$mean_perimeter) #Gets the mean breast perimeter

[1] 91.96903

> median(df$mean_radius) #Gets the median breast radius

[1] 13.37

> quantile(df$mean_smoothness) #Gets the quantiles for smoothness feature

     0%     25%     50%     75%    100%
0.05263 0.08637 0.09587 0.10530 0.16340

> summary(df) #Summary of the dataset

  mean_radius      mean_texture   mean_perimeter     mean_area
 Min.   : 6.981   Min.   : 9.71   Min.   : 43.79   Min.   : 143.5
 1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17   1st Qu.: 420.3
 Median :13.370   Median :18.84   Median : 86.24   Median : 551.1
 Mean   :14.127   Mean   :19.29   Mean   : 91.97   Mean   : 654.9
 3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10   3rd Qu.: 782.7
 Max.   :28.110   Max.   :39.28   Max.   :188.50   Max.   :2501.0
 mean_smoothness     diagnosis
 Min.   :0.05263   Min.   :0.0000
 1st Qu.:0.08637   1st Qu.:0.0000
 Median :0.09587   Median :1.0000
 Mean   :0.09636   Mean   :0.6274
 3rd Qu.:0.10530   3rd Qu.:1.0000
 Max.   :0.16340   Max.   :1.0000
```

The smallest breast area is 143.5 and the largest is 2501

The average breast perimeter is 91.9603

The median breast radius is 13.37

We now have the smoothness values for each quantiles

The range of values for texture feature is from 9.71 to 39.28.

The dataset summary displays a more in depth array of values for each feature such as the Min, Max and 1st + 3rd Quantiles.

I want to now see if there is a correlation between any features. We will use the 'corrplot' package to do this, as this will visualize them for us.

```
> library(corrplot) #Loads 'corrplot' package
> cor(df) #'Cor' lots a table of correlation values between features
```

```
                mean_radius mean_texture mean_perimeter  mean_area
mean_radius       1.0000000   0.32378189      0.9978553  0.9873572
mean_texture      0.3237819   1.00000000      0.3295331  0.3210857
mean_perimeter    0.9978553   0.32953306      1.0000000  0.9865068
mean_area         0.9873572   0.32108570      0.9865068  1.0000000
mean_smoothness   0.1705812  -0.02338852      0.2072782  0.1770284
diagnosis        -0.7300285  -0.41518530     -0.7426355 -0.7089838
                mean_smoothness  diagnosis
mean_radius          0.17058119 -0.7300285
mean_texture        -0.02338852 -0.4151853
mean_perimeter       0.20727816 -0.7426355
mean_area            0.17702838 -0.7089838
mean_smoothness      1.00000000 -0.3585600
diagnosis           -0.35855997  1.0000000
```

We can see there are values with a strong correlation but the table isn't quite easy to read so we will now plot a graph

The bigger the circle indicates a larger correlation value, the bluer the circle indicates a more directly proportional (i.e. a 'cor' value closer to 1. And a redder circle indicates a more inversely proportional relationship (i.e.a 'cor' value closer to -1.)

```
> correlations <- cor(df) #Store the correlation table in a variable
```

We can now visually see there are a few features that correlate with each other.

We will disregard the circles along the diagonal of the plot as these values are indicating there is a correlation with themselves which is obvious and unhelpful.

```
> corrplot(correlations, method="circle") #Plot the correlations in a graph
> #Represented by circles, using the correlation table for values.
```

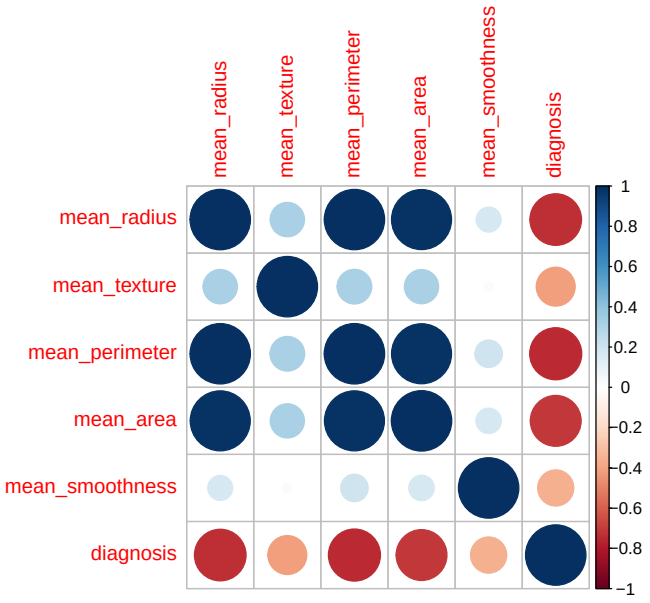

Figure 2: Correlation Plot

We want to know which features correlate with our output feature, (diagnosis), so we will look along that column to see if there are large correlation circles.

We can see that every feature has a correlation with diagnosis. The most significant correlation seems to be breast size, in particular breast perimeter, and diagnosis. There is also a smaller correlation in texture and smoothness with diagnosis.

1.3 Pre-processing

We now want to prepare the data before building our model

First we will check for missing data, we will be using the 'mlbench' and 'Amelia' packages to do this

```
> library(Amelia)
> library(mlbench)
```

We can see from this visualization that there is no missing data, which is what we want

To confirm this we will count the amount of data in the dataframe

```
> na_counts <- sapply(df, function(x) sum(is.na(x))) #Counts how much data is missing
> na_counts #Displays counts of missing data per feature; in a table

    mean_radius    mean_texture  mean_perimeter       mean_area mean_smoothness
              0               0               0               0               0
      diagnosis
              0

> cat("Number of missing pieces of data:", sum(na_counts)) #Displays a message of total miss

Number of missing pieces of data: 0
```

The missmap was correct, there was no missing data in the dataframe. So, we do not need to go through the process of removing missing fields.

To make the investigation quicker to carry out and easier to follow we will rename the features to something more meaningful

```
> #names(df) <- c("Radius","Texture","Perimeter","Area","Smoothness", "Diagnosis") #Renames
```

```
> missmap(df, col=c("black", "yellow"), legend=TRUE) #Displays chart of missing data
> #Black represents missing data, yellow represtents whole data. We will show a legend too.
```

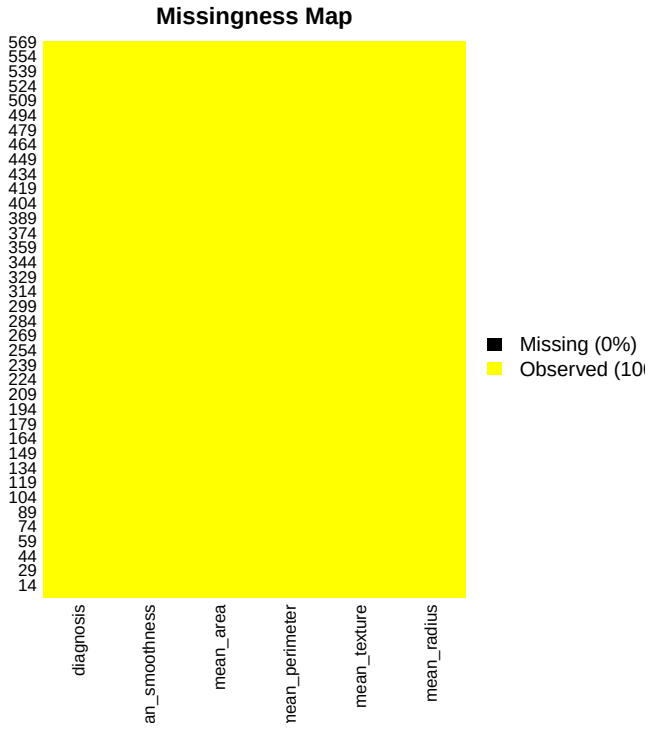

Figure 3: Missingness Map

we will now check these have been changed correctly

```
> names(df) #Checks feature names have been changed

[1] "mean_radius"     "mean_texture"    "mean_perimeter"  "mean_area"
[5] "mean_smoothness" "diagnosis"
```

Our features are now called 'Radius', 'Texture', 'Area', 'Smoothness', 'Diagnosis'.

Our dataset is now ready to fit a model with.

2 Modelling/ Classification

2.1 Subsetting The Data

We will build our modelling using training and testing data subsets. So, lets build them.

70/30 is a good starting point for a training/testing split so that's what we will use for outs. We know from exploring out data that there are 569 rows in our dataset. 0.7* 569 = 398 (rounded). So that will be the number of rows our training set will pull from our dataframe, the testing subset will take the rest of the rows (So, 569 - 398 = 171).

```
> train <- df[1:398,] #Split 70% of dataframe into training subset
> test <- df[399:nrow(df),] #Split 30% of dataframe into testing subset
```

We will now check to make sure that the subsets have been generated correctly. If they have been made correctly the total number of rows in the testing subset plus the total number of rows in the training subset will equal the number of rows in the dataframe.

So,

```
> nrow(train)+nrow(test)==nrow(df) #Checking subsets created succesfully.

[1] TRUE
```

The above code returned "TRUE" so the total rows in subsets and dataframe match, meaning they have been created successfully.

We will now check the class distribution of the subsets.

```
> barplot(table(train$diagnosis)) #Plots a bargraph of the training class distribution.
```

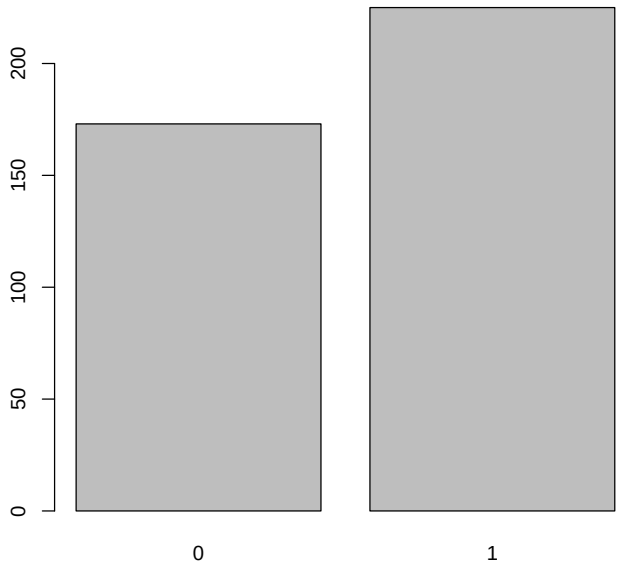

Figure 4: Training Class Distribution

```
> table(train$diagnosis) #Plotting a table of the class frequencies for training subset.

  0   1
173 225

> table(test$diagnosis) #Plotting a table of the class frequencies for testing subset.

  0   1
 39 132
```

We will visualize these tables for ease

We can see here that there is quite an uneven class distribution in our data subsets - we will improve on these and balance them out later on in the next section.

```
> barplot(table(test$diagnosis)) #Plots a bargraph of the testing class distribution.
```

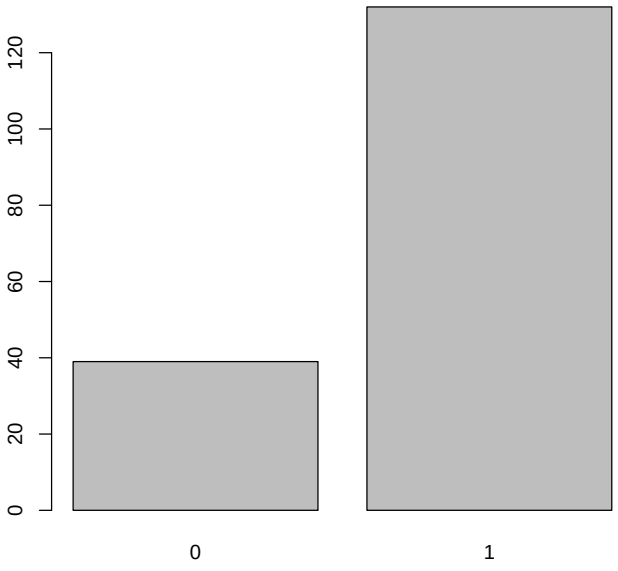

Figure 5: Testing Class Distribution

2.2 Building The Model

Now, we can start building our model...

We will be using a logistic regression model to start with.

```
> mymodel <- glm(diagnosis~.,family=binomial,data=train) #Build my model using logistic regr
```

Our model has been created.

Lets get a quick description of our model.

```
> summary(mymodel) #Getting a description of the model.

Call:
glm(formula = diagnosis ~ ., family = binomial, data = train)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.75851  -0.01903   0.02963   0.16606   2.21844

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)       16.73501    9.99846   1.674  0.09418 .
mean_radius        5.83435    2.13880   2.728  0.00637 **
mean_texture      -0.45964    0.08411  -5.465 4.64e-08 ***
mean_perimeter    -0.56019    0.21283  -2.632  0.00849 **
mean_area         -0.04036    0.01687  -2.392  0.01674 *
mean_smoothness -138.55993   27.00969  -5.130 2.90e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 544.93  on 397  degrees of freedom
Residual deviance: 112.18  on 392  degrees of freedom
AIC: 124.18

Number of Fisher Scoring iterations: 9
```

NEEDS RESEARCH

2.3 Testing And Evaluating

Now the model will use the testing subset to check that the model works and so we can evaluate it.

The code below uses our model to predict the probability of diagnosis of breast cancer of each person in the test subset. The probability of diagnosis of each person is saved in a variable called 'probs'.

```
> #Calculating predictions using my model (props will hold the probabilities)
> probs <- predict(mymodel,newdata=test,type='response')
```

Examine the probabilities

```
> head(probs) #Displays the first 5 probabilities.
```

```
         399          400          401          402          403          404
9.998847e-01 9.957763e-01 2.298069e-07 9.997984e-01 9.955025e-01 9.824550e-01
```

Now we create a variable called 'predictions' and use the 'probabilities' variable to determine the diagnosis prediction. If the prediction is less than 0.5 then we decide the person does not have breast cancer and if it is over 0.5 we decide they do have breast cancer.

```
> #Setting the predictions to 0 or 1 based on the probabilities.
> predictions <- ifelse(probs > 0.5,1,0)
```

Check this has worked.

```
> head(predictions) #Displays the first 5 predictions.
```

```
399 400 401 402 403 404
  1   1   0   1   1   1
```

This has returned 1's and 0's so predictions have been set.

Time to calculate the accuracy of our model and there are a few ways to do this.
The first method we will use is taking the average of incorrect predictions out of the total number of predictions made to give us the testing error. 1 minus the testing error gives us the testing accuracy.

Calculating and displaying the test error.

```
> #Check the prediction against the known diagnosis to determine error.
> test_err <- mean(predictions != test$diagnosis) #Gets an average of the wrong predictions
> print(paste('Testing error:',test_err*100,'%')) #Displays the test error as a percentage.
```

```
[1] "Testing error: 9.94152046783626 %"
```

The test error is quite low here 9.94% which is a good indication in itself but we still want a value for the accuracy so we can compare with other accuracies later.

Get the accuracy by subtracting the test error from one.

```
> # Accuracy is calculated by 1 - test error.
> print(paste('Testing accuracy:',(1-test_err)*100,'%')) #Displays accuracy as percentage.
```

```
[1] "Testing accuracy: 90.0584795321637 %"
```

We have quite a high testing accuracy as we can see, 90.16 %. This tells gives us a clue that our model is working quite well to predict diagnosis outcomes. But, to be sure we will cross-check the accuracy using different methods.

Lets create a new dataframe to cross-check the accuracy using a confusion matrix method.

The new data frame mimics the testing subset.

```
> dfU <- test #Creates a new dataframe called dfU using the test subset.
```

Add a feature that shows the probabilities and check it's appended correctly.

```
> dfU$Prob <- probs #Appends the probabilites to the new dataframe
> head(dfU$Prob) #Displays the first 5 probabilities so we know it has been added correctly.
```

```
[1] 9.998847e-01 9.957763e-01 2.298069e-07 9.997984e-01 9.955025e-01
[6] 9.824550e-01
```

Feature added correctly.

Add another feature to the dataframe that shows the prediction of diagnosis and check that it's added.

```
> dfU$Pred <- predictions #Appendsthe predictions to the new dataframe
> dfU$Pred #Displays the first 5 predictions to check it has been added correctly.
```

```
  [1] 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1
 [38] 0 1 1 1 1 1 0 1 1 0 0 0 1 1 0 1 0 1 1 1 0 0 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1
 [75] 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1 1 0 1 1 0 0 0 0 1 0 0 1 1 1 0
[112] 0 1 1 0 1 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 0 0 1 1
[149] 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 0 0 0 0 1
```

Prediction feature added fine.

Now we add our last feature for the new dataframe. We will check the predictions against actual diagnosis in the dfU dataframe and store the outcome in the new feature as a '1' for correct predictions and a '0' for incorrect predictions.

We will also check that it worked.

```
> #Checks the predictions against known diagnosis' so we know the number of correct predicti
> dfU$correct <- ifelse(dfU$diagnosis== dfU$Pred,1,0) #If correct store as 1, if incorrect s
> dfU$correct #Display the correct feature.

  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 0
[112] 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 1 1
[149] 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1
```

'Correct' feature added properly.

We can now fit a confusion matrix below...

```
> table(dfU$diagnosis,dfU$Pred) #Confusion Matrix is essentially justa a table of prediction

     0   1
  0 38   1
  1 16 116
```

The way we will be double checking the accuracy is by using the confusion matrix and taking an average of the values in the diagonal of the matrix like below.

```
> #The sum of the diagonal of the table gives us another accuracy.
> accuracy2 <- sum(diag(table(dfU$diagnosis,
+                             dfU$Pred)))/nrow(dfU)*100
> cat("Accuracy is ", accuracy2 , "%") #Display the accuracy.

Accuracy is  90.05848 %
```

Check the accuracies are the same

```
> ((1-test_err)*100) == accuracy2 #Checks the accuracies against each other.

[1] TRUE
```

'TRUE' is returned so accuracies match and are correct.

Lets get a value for our performance of the model

Using the 'ROCR' package to plot an ROC curve we will calculate the area under the curve (auc) to give us another indication as to how our model is performing.

Use the ROCR library.

```
> library('ROCR') #Loads the 'ROCR' package.
```

Get the probabilities.

```
> probs <- predict(mymodel,newdata=test,type='response') #Gets probabilities of the testing
```

Get the predictions.

```
> pr <- prediction(probs, test$diagnosis) #Gets the predictions using the probabilities.
```

Get the True Positive Rate (TPR) and the False Positive Rate (FPR).

```
> prf <- performance(pr, measure = "tpr", x.measure = "fpr") #Plots the false positive agai
```

Now we can plot the ROC curve

We can see that performance is quite high here due to the peak of the curve being quite high.

Now we have our ROC curve it is time to calculate the area under the curve to get a percentage of performance.

```
> #Computes the area under the curve
> auc <- performance(pr, measure = "auc")
> auc <- auc@y.values[[1]]
> print(paste('Performance:',auc*100,'%')) #Displays the area under curve as a percentage.
```

```
[1] "Performance: 99.0870240870241 %"
```

Area under the curve is 99.08% which is phenomenally high, this indicates we have a highly performing discriminative model.

Examine the new dataset.

```
> plot(prf) #Plotting the ROC curve.
```

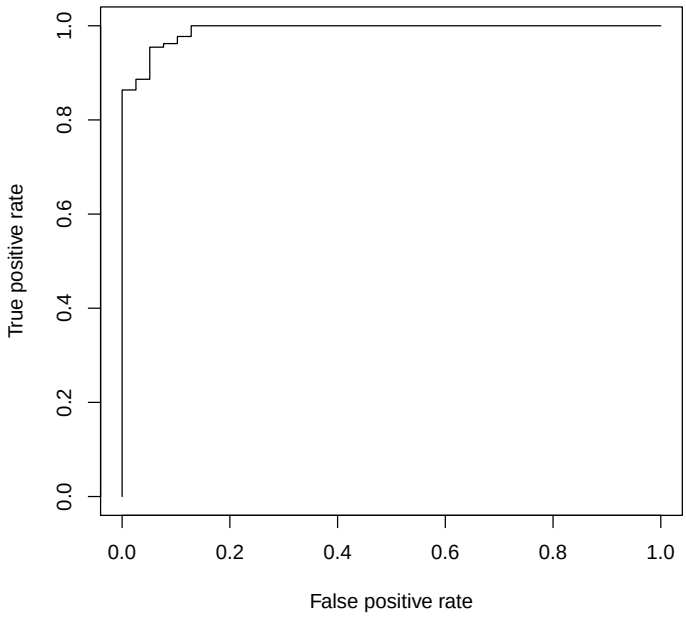

Figure 6: ROC Curve

```
> head(dfU)
```

```
    mean_radius mean_texture mean_perimeter mean_area mean_smoothness diagnosis
399       11.06        14.83          70.31     378.2         0.07741         1
400       11.80        17.26          75.26     431.9         0.09087         1
401       17.91        21.02         124.40     994.0         0.12300         0
402       11.93        10.91          76.14     442.7         0.08872         1
403       12.96        18.29          84.18     525.2         0.07351         1
404       12.94        16.17          83.18     507.6         0.09879         1
             Prob Pred correct
399 9.998847e-01    1       1
400 9.957763e-01    1       1
401 2.298069e-07    0       1
402 9.997984e-01    1       1
403 9.955025e-01    1       1
404 9.824550e-01    1       1
```

Write a copy of this new dataframe to an external CSV file for consumption.

```
> write.csv(dfU,"Breast Cancer Model.csv") #Writing dataframe to a CSV file
```

2.4 Reporting On And Discussing The Results

Accuracy within our model is very high with a testing accuracy of 90%, a testing error of 9.94% and a ROC accuracy of 99%. This tells us our model works very well for the data we fed our model. However, we previously checked from steps taken above that the class distribution isn't equal - so we cannot be certain that our accuracies are reliable. We must further improve our model to evenly distribute the classes.

3 Improving Performance

3.1 Cross-Validation

We need the 'caret' package for this Cross-Validation method

```
> #CROSS VALIDATION
> library(caret) #Loads the 'caret' package.
```

We convert the diagnosis features in the subsets to a factor.

```
> train$diagnosis = as.factor(train$diagnosis) #Converts training to factor.
> test$diagnosis = as.factor(test$diagnosis) #Converts testing to factor.

> train_control <- trainControl(method='cv', number = 5) #Our cross validation model will us
```

Building our cross-validation model.

```
> mymodel <- train(diagnosis~., data=train, trControl=train_control, method="glm", family=bi
```

Get the predictions using the cross-validation model and examine a few of the predictions

```
> predictions<- predict(mymodel,test[,-ncol(test)]) #Predict, using our model.
> head(predictions) #show the first 5 predictions.
```

```
[1] 1 1 0 1 1 1
Levels: 0 1
```

Append the predictions data to the end of the test subset and examine the new contents of the test subset.

```
> test<- cbind(test,predictions) #Adding the predictions to the testing subset.
> head(test) #Displaying the first few rows in testing subset.
```

```
    mean_radius mean_texture mean_perimeter mean_area mean_smoothness diagnosis
399       11.06        14.83          70.31     378.2         0.07741         1
400       11.80        17.26          75.26     431.9         0.09087         1
401       17.91        21.02         124.40     994.0         0.12300         0
402       11.93        10.91          76.14     442.7         0.08872         1
403       12.96        18.29          84.18     525.2         0.07351         1
404       12.94        16.17          83.18     507.6         0.09879         1
    predictions
399           1
400           1
401           0
402           1
403           1
404           1
```

```
> results <- confusionMatrix(test$predictions, test$diagnosis) #Plotting a confusing matrix
> results #Display the confusion matrix.
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  38  16
         1   1 116

               Accuracy : 0.9006
```

```
               95% CI : (0.8456, 0.941)
    No Information Rate : 0.7719
    P-Value [Acc > NIR] : 1.046e-05

                  Kappa : 0.7513
 Mcnemar's Test P-Value : 0.000685

            Sensitivity : 0.9744
            Specificity : 0.8788
         Pos Pred Value : 0.7037
         Neg Pred Value : 0.9915
             Prevalence : 0.2281
         Detection Rate : 0.2222
   Detection Prevalence : 0.3158
      Balanced Accuracy : 0.9266

       'Positive' Class : 0
```

Calculate the accuracy of our cross-validation model.

```
> cat("Accuracy=", ((sum(diag(results$table))/nrow(test)*100)), "%") #Displaying the accurac

Accuracy= 90.05848 %
```

3.2 Using Different Metrics For Evaluation

Another way to evaluate our model before we work on improving it is to use different metrics

We will create a new function to get different metrics

```
> ## GET METRICS
> getMetrics <- function(TP,FP,TN,FN){
+   TPR=TP/(TP+FN); #Calculates the True Positive Rate
+   FPR=FP/(FP+TN); #Calculates the False Positive Rate
+   TNR=TN/(TN+FP); #Calculates the True Negative Rate
+   FNR=FN/(TP+FN); #Calculates the False Negative Rate
+   ACC=(TP+TN)/((TP+FN)+(FP+TN)); #Calculates the Accuracy
+   SPC=TN/(FP+TN); #Calculates the Specificity
+   SNS=TP/(TP+FN) #Calculates the Sensitivity
+   metrics <- c('True Positive Rate','False Positive Rate','True Negative Rate','False Nega
+                'Specificity','Sensitivity') #Sets the headers.
+   values <- c(TPR,FPR,TNR,FNR,ACC,SPC,SNS) #Saves the values to a temporary dataframe.
+   values<-(values*100) #Converts the values to a percentage.
+   df <- data.frame(Metrics=metrics,Values=values) #Make a new dataframe with the metrics
```

```
+   df #Display the metrics.
+ }
```

This function get the True Positive Rate (TPR), True Negative Rate(TNR), False Positive Rate (FPR), False Negative Rate (FNR), Accuracy (ACC), Specificity (SPC) and the Sensitivity (SNS).

```
> tmpSet <- data.frame(Actual=test$diagnosis,Predicted=predictions)
> tmpSet$correct <- as.numeric(
+   tmpSet$Actual==tmpSet$Predicted)
> # compute TP, FP, TN, FN
> tp <- tmpSet[tmpSet$Actual=='1' & tmpSet$Predicted=='1',]
> fp <- tmpSet[tmpSet$Actual=='0' & tmpSet$Predicted=='1',]
> tn <- tmpSet[tmpSet$Actual=='0' & tmpSet$Predicted=='0',]
> fn <- tmpSet[tmpSet$Actual=='1' & tmpSet$Predicted=='0',]
> results <- getMetrics(nrow(tp),nrow(fp),nrow(tn),nrow(fn)) # call the function
> results #Show metrics

              Metrics    Values
1  True Positive Rate 87.878788
2 False Positive Rate  2.564103
3  True Negative Rate 97.435897
4 False Negative Rate 12.121212
5            Accuracy 90.058480
6         Specificity 97.435897
7         Sensitivity 87.878788
```

3.3 Balancing Our Class Distribution

We know from checking our class distribution while building our model above that our classes are NOT distributed evenly.

So, we need to solve this by ensuring our classes are distributed evenly in our train and test subsets.

The 'caret' and 'dplyr' packages are required for this.

```
> library('caret') #Loading the 'caret' package.
> library('dplyr') #Loading the 'dplyr' package.
```

Index is set to allow us to split the data into a 70/30 training and testing split.

```
> set.seed(101) #Set seed so report is repeatable.
> #Split the data into a 80:20 training/testing split
> splitIndex <- createDataPartition(df$diagnosis, p = .80, list = FALSE,times = 1)
```

```
> barplot(table(tSample$diagnosis)) #Plotting a bargraph of training class distribution
```

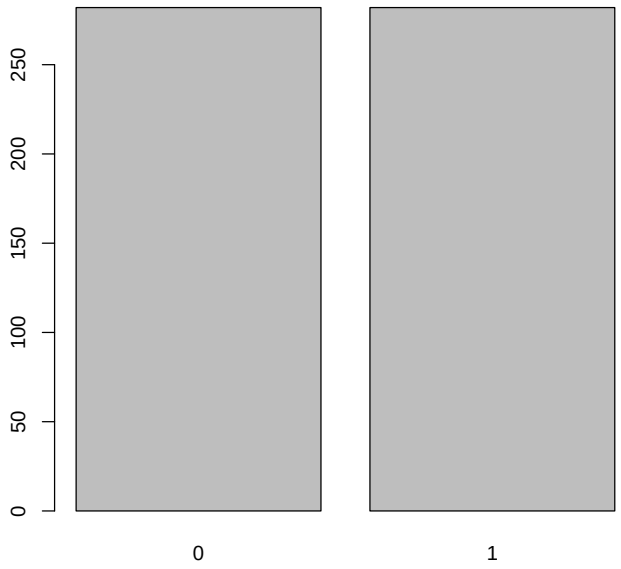

Figure 7: Even Class Distribution Graph - Training

Splitting the data into training and testing subsets.

```
> training <- df[ splitIndex,] #Set training subset to 80% of data
> testing <- df[-splitIndex,] #Set testing subset to 20% of data
```

WTF

```
> # sample negative examples equal to postive examples
> tSample <-sample_n(training[training$diagnosis==0,],
+                    nrow(training[training$diagnosis==1,]), replace = TRUE)
> # add postive examples from the training set into sample
> tSample <- rbind(tSample,training[training$diagnosis==1,])
```

Visualize the class distribution of the training set

Now, the classes are evenly distributed.

```
> barplot(table(xSample$diagnosis)) #Plotting a bargraph of testing class distribution
```

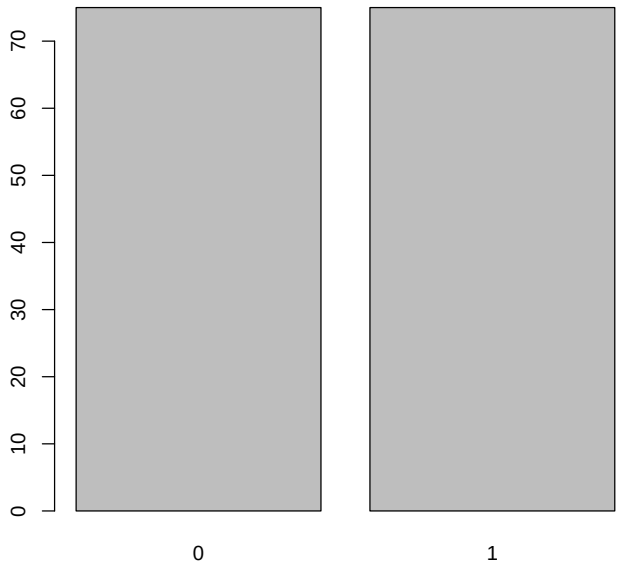

Figure 8: Even Class Distribution Graph - Testing

WTFX2

```
> xSample <-sample_n(testing[testing$diagnosis==0,],
+                    nrow(testing[testing$diagnosis==1,]), replace = TRUE)
> # add postive examples from the training set into sample
> xSample <- rbind(xSample,testing[testing$diagnosis==1,])
```

Checking the class distribution of the test set

We can see the classes are now evenly distributed.

```
> ctrl <- trainControl(method = "cv", number = 5) #Cross Validate using 5 k-folds
> tSample$diagnosis <- as.factor(tSample$diagnosis) #Convert tSample to factor
> xSample$diagnosis <- as.factor(xSample$diagnosis) #Convert xSample to factor
> mymodel <- glm(diagnosis~.,family=binomial,data=tSample) #Create a new model
> # test
> probs <- predict(mymodel,newdata=xSample,type='response') #Getting probabilities
```

```
> predictions <- ifelse(probs > 0.5,1,0) #Setting predictions using probabilities
> accuracy <- mean(predictions==xSample$diagnosis) #Calculating accuracy by taking the avera
> accuracy <- round(accuracy*100,digits = 2) #Rounding the accuracy to 2 decimal points.
> cat('Accuracy is: ',accuracy, "%") #Displaying accuracy as a percentage.

Accuracy is:  94.67 %
```

3.4 Fitting A Different Model And Comparing The Results

Random Forest is another model that could work for our investigation as it works with binary classification.

We need the 'caret' and 'randomForest' packages to do this...

```
> require(randomForest) #Loading the 'randomForest' package.
```

The seed is set so we can get the same output each time the model is run

```
> set.seed(101) #Setting the seed to repeat the random forest and get the same results.
```

The training set is created for the random forest model

```
> train=sample(1:nrow(df),398) #Using 70% of dataset data
> df.rf=randomForest(diagnosis~., data = df, subset = train) #Building our random forest mod
> df.rf #Displaying our random forest model.

Call:
 randomForest(formula = diagnosis ~ ., data = df, subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 1

          Mean of squared residuals: 0.06299725
                    % Var explained: 73.38

> oob.err=double(13)
> test.err=double(13)
> for(mtry in 1:13){
+   rf=randomForest(diagnosis~mean_area, data=df, subset = train,mtry=mtry,ntree=400)
+   oob.err[mtry] =rf$mse[400]
+
+   pred<-predict(rf,df[-train,])
+   test.err[mtry]= with(df[-train,], mean((diagnosis - pred)^2))
+ }
>
```

```
> plot(df.rf) #Displaying a graph of our random forest model performance.
```

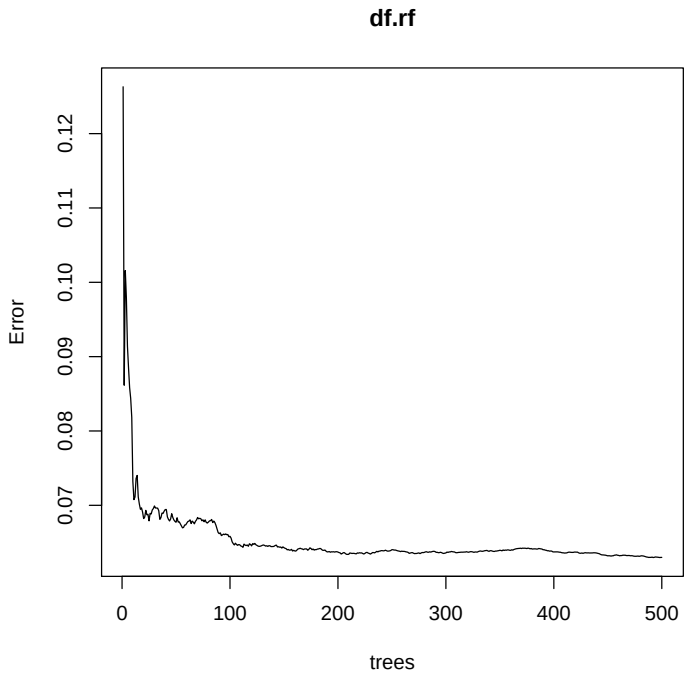

Figure 9: Random Forest Performance

Calculating the random forest model error

```
> cat("Test Error=",(sum(test.err)/10)*100,"%") #Displaying test error as a percentage.

Test Error= 16.88358 %
```

Calculating the random forest model accuracy

```
> cat("Accuracy=",(1 - sum(test.err)/10)*100,"%") #Displaying test accuracy as a percentage.

Accuracy= 83.11642 %
```

The accuracy is for the random model is quite high again however this is NOT an improvement from the accuracy given from the logistic regression model which is 90

So, we will not continue using the random forest model.

```
> oob.err

 [1] 0.1256233 0.1266926 0.1269691 0.1250540 0.1267373 0.1269150 0.1258240
 [8] 0.1263076 0.1257686 0.1256352 0.1255955 0.1255357 0.1261155

> #pred
```

3.5 Changing The Dataset Partitions

Another method that might improve our model is changing the partitions, i.e. changing the split of the dataframe into different training/testing subset sizes.

First we will try a 90 training and a 10 testing split.

```
> train <- df[1:511,] #Setting training subset to 90% of dataset
> test <- df[512:nrow(df),] #Setting testing subset to 10% of dataset
```

511, is 90 of dataframe, testing takes the rest of the rows.

Now we will check, the same way as before, that they were split correctly.

```
> nrow(train)+nrow(test)==nrow(df) #Checking subsets are formed correctly.

[1] TRUE
```

The code returns TRUE so number of rows in splits is the same as that of in the dataframe, so data has been split correctly.

Now we shall check the class distribution of the data
.

```
> table(train$diagnosis) #Stores training class distribution in a table.

  0   1
198 313

> table(test$diagnosis) #Stores testing class distribution in a table.

 0  1
14 44
```

...and in a visual form.

Again, we can see that the class distribution is still unbalanced so we need to consider this when interpreting our results at the end of this test.

Fitting a new logistic regression model using our new data split.

```
> mymodel <- glm(diagnosis~mean_area,family=binomial,data=train) #Building a new logistic re
```

Our new model is now ready.

Have a look at the new model.

```
> summary(mymodel) #Displays a description of our new model.

Call:
glm(formula = diagnosis ~ mean_area, family = binomial, data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6886  -0.1395   0.2144   0.4804   2.7142

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  7.74187    0.68783   11.26   <2e-16 ***
mean_area   -0.01149    0.00110  -10.45   <2e-16 ***
---
```

```
> barplot(table(train$diagnosis)) #Plots a bargraph of testing class distribution
```

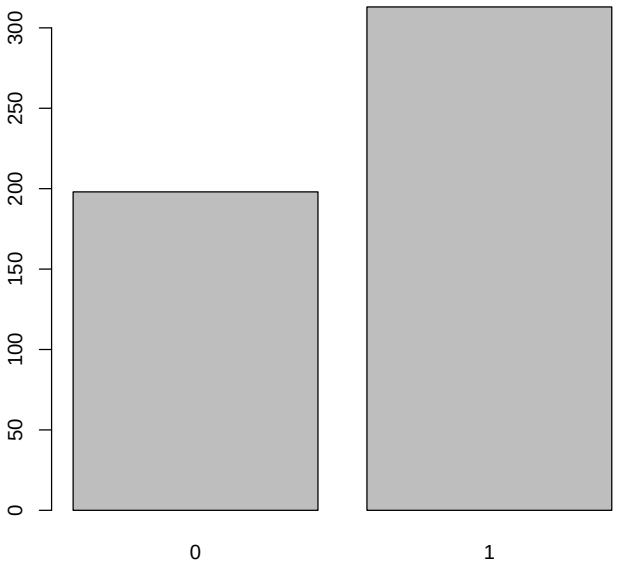

Figure 10: Final Model Training Class Distribution

```
> barplot(table(test$diagnosis)) #Plots a bargraph of training class distribution
```

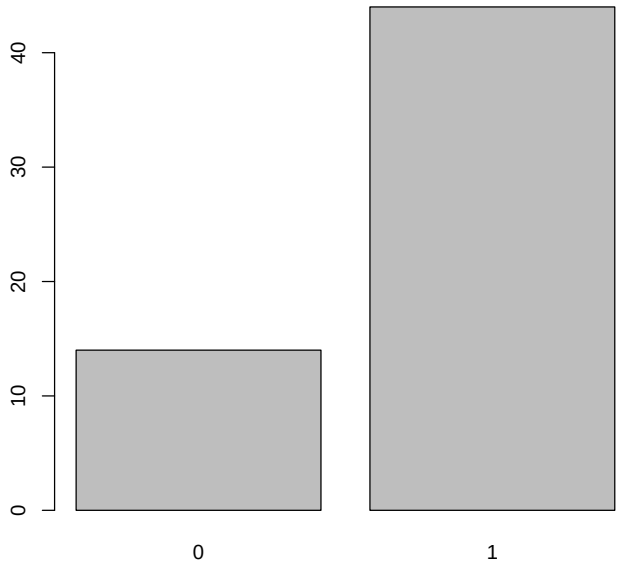

Figure 11: Final Model Testing Class Distribution

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 682.29  on 510  degrees of freedom
Residual deviance: 305.05  on 509  degrees of freedom
AIC: 309.05

Number of Fisher Scoring iterations: 7
```

MORE RESEARCH NEEDED

Calculating the probabilities and having a look at a sample of them.

```
> probabilities <- predict(mymodel,newdata=test,type='response') #Perform predictions using
> head(probabilities) #Displaying first 5 probabilities.

      512       513       514       515       516       517
0.4800741 0.7933245 0.5428688 0.4198606 0.9625568 0.0127904
```

Determining the predictions using the probabilities again.

```
> predictions <- ifelse(probabilities > 0.5,1,0) #Setting predictions to 0 or 1 depending or
```

Calculating the test error and accuracy again.

```
> test_err <- mean(predictions != test$diagnosis) #Computing test error by checking average
> #Displaying testing error and testing accuracy
> print(paste('Testing Error:', (test_err*100), '% ,', 'Testing Accuracy:',(1-test_err)*100,

[1] "Testing Error: 5.17241379310345 % , Testing Accuracy: 94.8275862068966 %"
```

We can see here splitting the dataframe up so that we have more training data improves our performance by 4.8, which is quite a significant increase. We will continue using this model with this data split.

3.6 Report On Improvements And Final Model

From our experiments we can see that using balanced data and more training data increases our accuracy. We attempted to fit a random forest model but the results showed a decline in accuracy so we will stick with our logistic regression model with a training : testing ratio of 80:20 using data with balanced class distribution, such as our model in section 3.3. This is now a highly performing model.

4 Conclusion

Overall, our model was a success. The aim to predict breast cancer diagnosis in patients given data on their breasts has been achieved. We started with a high accuracy of and managed to improve on the model to gain a improvement of on our model. I believe that the report written on the investigation is clear, reproducible and easy to follow and I am satisfied with the outcome of this investigation and have confidence in my model. It was an enjoyable and interesting topic and I hope you gain as much pleasure reading it as I did producing it!

5 Appendix

List of Figures