Logistic Regression

Jason Barnes

Logistic Regression is used for classification. In linear regression the response varible (Y) is quantitative, but in many situations the response variable can be qualitative or categorical. Predicting a qualitative response for an observation can be referred to as classifying that observation, since it involves assigning the observation to a category, or class. On the other hand, often the methods used for classification first predict the probability of each of the categories of a qualitative variable, as the basis for making the classification. In this sense they also behave like regression methods.

There are many possible classification techniques, or classifiers, that one might use to predict a qualitative response. Three of the most qualitative classification classifiers widely-used are: logistic regression, linear discriminant analysis, and logistic K-nearest neighbors.

Example Classification problems:

1. A person arrives at the emergency room with a set of symptoms that could possibly be attributed to one of three medical conditions. Which of the three conditions does the individual have?
2. An online banking service must be able to determine whether or not a transaction being performed on the site is fraudulent, on the basis of the user's IP address, past transaction history, and so forth.
3. On the basis of DNA sequence data for a number of patients with and without a given disease, a biologist would like to figure out which DNA mutations are deleterious (disease-causing) and which are not.

We use the logistic function to output a value ranging from 0 to 1. Based off of the probability we assign a class.

Sigmoid function: always produces results in a probability from 0 to 1 of belonging in the 1 class. That means we can set a cutoff point at 0.5, anything below it results in class 0, anything above is class 1.

After you train a logistic regression model on some training data, you will evaluate your model's performance on some test data. You can use a confusion matrix to evaluate classification models.

Terminology

True Positive (TP) cases in which we perdicted positive and in reality the the test is positive

True Negative (TN) cases in which we perdicted negative and in reality the test is negative

False Positives (FP) cases in which we predicted positive and in reality the test is negative (type 1 error) (telling a man they are pregnant)

False Negatives (FN) cases in whch we predicted negative and in reality the test is positive (type 2 error) (telling a pregnant women that is visablly showing that she is not pregnant)

Misclassification rate (error rate) measures overall how often we are wrong.

Measuring Accuracy

To calculate the accuracy of the model you would take (TP + TN) / total

To calculate the misclassifiation rate you would take (FP + FN) / total

Getting started

You will need to import the following libraries:

1. tidyverse
2. ggthemes
3. Amelia
4. kableExtra
5. knitr
6. devtools
7. knitcitations

Explore the data

```r
kable(head(df.train)) %>%
    kable_styling(bootstrap_options = "condensed")
```

PassengerId	Survived	Pclass	Name	Sex	Age
1	0	3	Braund, Mr. Owen Harris	male	22
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38
3	1	3	Heikkinen, Miss. Laina	female	26
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35
5	0	3	Allen, Mr. William Henry	male	35
6	0	3	Moran, Mr. James	male	NA

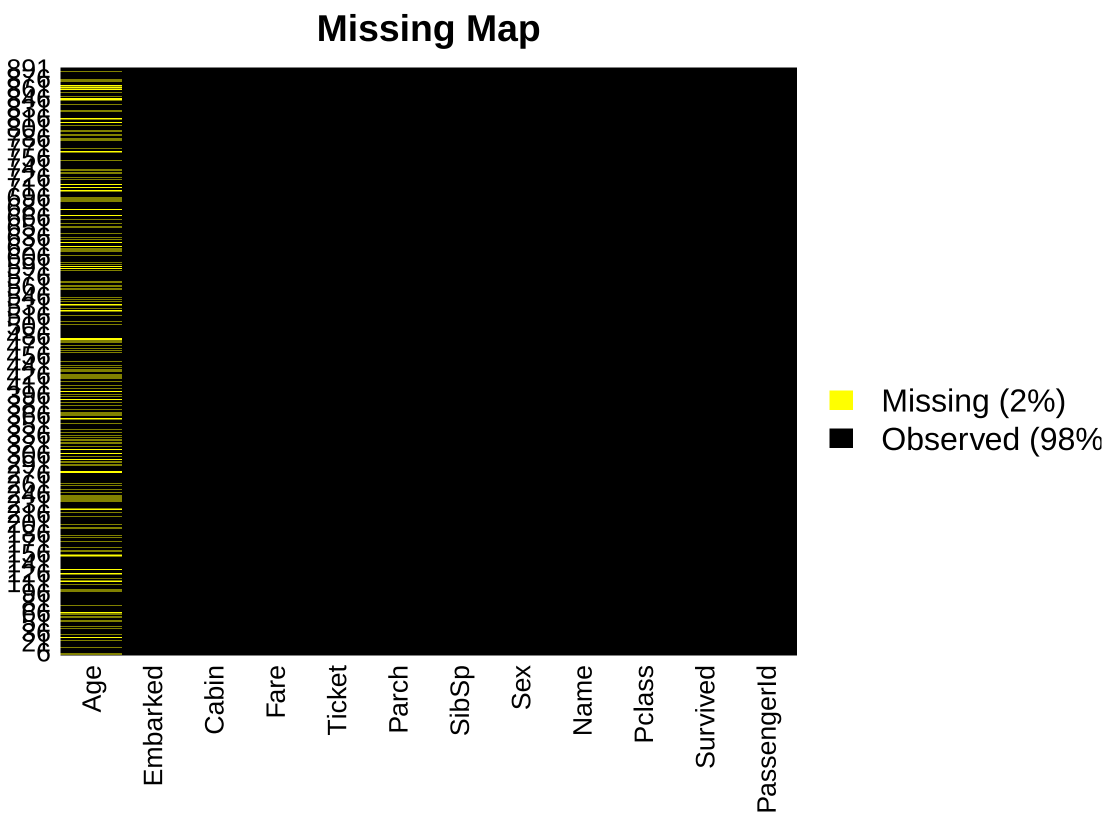
Lets take a look at the structure of the data set

```r
str(df.train)
```

```
## 'data.frame':    891 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 358 277 16 559
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
## $ SibSp       : int  1 1 0 1 0 0 0 3 0 1 ...
## $ Parch       : int  0 0 0 0 0 0 0 1 2 0 ...
## $ Ticket      : Factor w/ 681 levels "110152","110413",..: 524 597 670 50 473 276 86
## $ Fare        : num  7.25 71.28 7.92 53.1 8.05 ...
## $ Cabin       : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
## $ Embarked    : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
```

As you can see there is a total of 11 features, one thing we need to do is determine if there is an NA's or Nulls in the data set. One way to do this is by using the 'Ameila' package and creating a missing map.

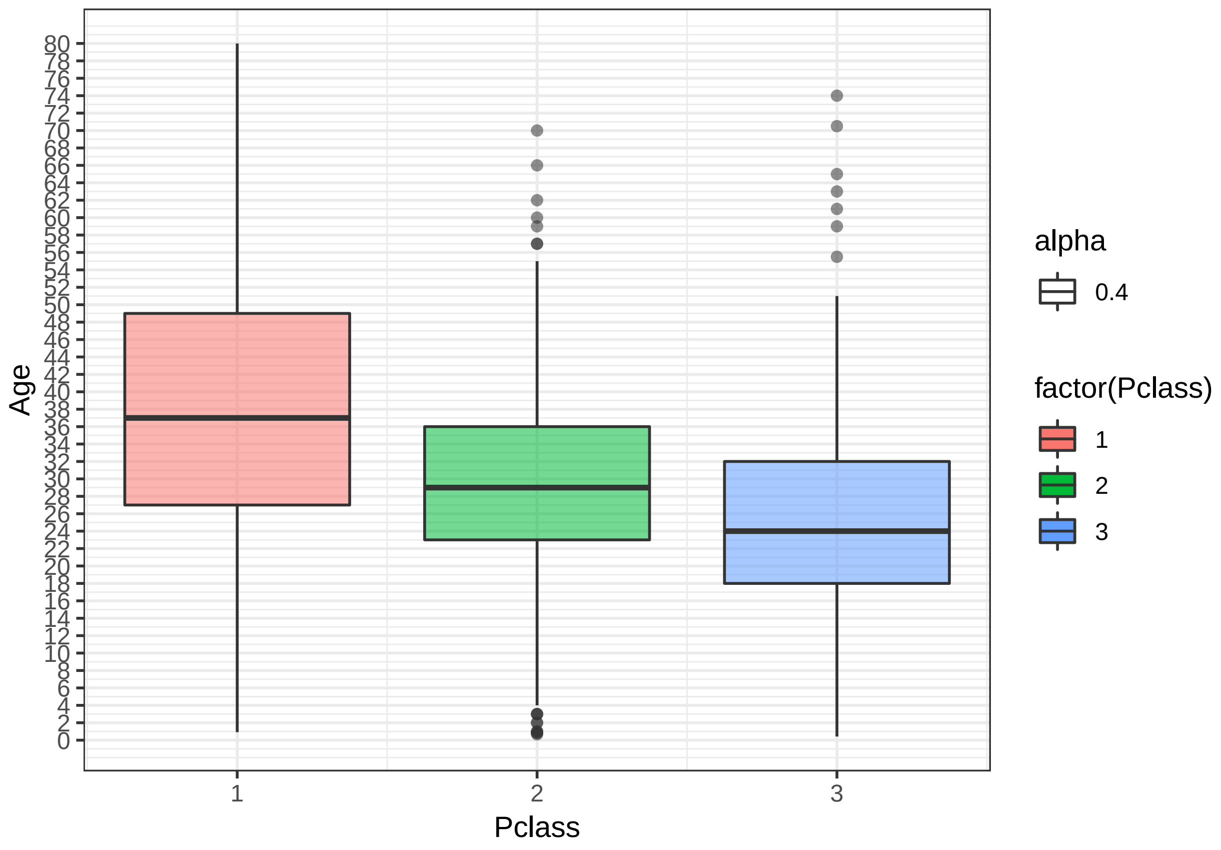
```r
missmap(df.train, main = 'Missing Map', col = c('yellow', 'black'), legend = TRUE)
```



As you can see the Age column as a significant amount of missing data, one way to solve this is to create a function that imputs the median age by Pclass. Lets take a look at a boxplot and determine the medain age by Pclass.

```r
pl <- ggplot(df.train, aes(Pclass, Age))
pl <- pl + geom_boxplot(aes(group = Pclass, fill = factor(Pclass), alpha = 0.4))
pl + scale_y_continuous(breaks = seq(min(0), max(80),2)) + theme_bw()
```

```
## Warning: Removed 177 rows containing non-finite values (stat_boxplot).
```



As you can see the median age by Pclass is 37, 29 and 24 respectively. This makes sense as we would assume that the older you are the higher class you will be able to afford.

Now that we have visualized the ages lets build a function to create imputation of age based on class.

```
impute_age <- function(age, class){
    out <- age
    for (i in 1:length(age)){
        if (is.na(age[i])){
          if (class[i] == 1){
                out[i] <- 37
            }else if (class[i] == 2){
                out[i] <- 29
            }else{
                out[i] <- 24
            }
        }else{
            out[i] <-  age[i]
        }
      }
    return(out)
}
```

The above function will look through all the ages and determine if the age is blank it will place the median age based on the class of the individual. If the age is already present it will simply keep the age.

Now pass the ages to the impute_age functio and assign it to fixed.ages

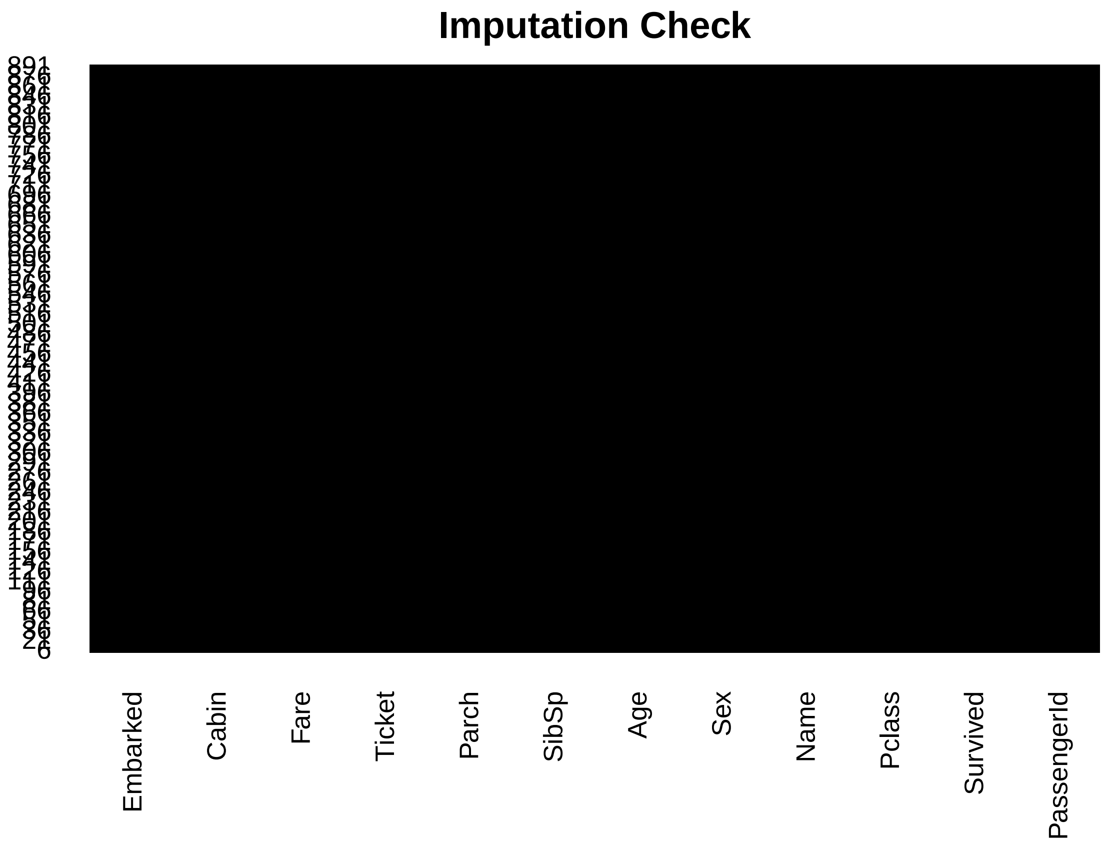
```
fixed.ages <-  impute_age(df.train$Age, df.train$Pclass)
```

Now we will have to take fixed.ages and replace the ages in the ages column. We can do this by assinging fixed.ages to the data frame column ages.

```
df.train$Age <- fixed.ages
```

Lets take another look at the Missing Map to ensure our assignments worked.

```
missmap(df.train, main = 'Imputation Check', col = c('yellow', 'black'),legend = FALSE)
```



As you can see there is no more missing ages, the function worked.

Lets take another look at the structure of the data to determine what columns we want to use in the model.

```
str(df.train)
```

```
## 'data.frame':    891 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 358 277 16 559
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ Age        : num  22 38 26 35 35 24 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : Factor w/ 681 levels "110152","110413",..: 524 597 670 50 473 276 86
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
##  $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
```

In order to ensure the model works correctly we will have to remove some columns as they would simply complicate the model because there are to many levels associated with the factor. Take names for instance, each row contains a name of an individual that was on the titanic, this information is irrelevant to whether the individual survived or not so we will remove it. We will also remove PassengerIf, Ticket, Cabin, and Parch as they are also irrelevant in determining survival.

```r
df.train <- df.train %>%
    select(-PassengerId, -Name, -Ticket, -Cabin, -Parch)
```

Now lets take a look at the head of the data frame.

```r
head(df.train)
```

```
##   Survived Pclass    Sex Age SibSp    Fare Embarked
## 1        0      3   male  22     1  7.2500        S
## 2        1      1 female  38     1 71.2833        C
## 3        1      3 female  26     0  7.9250        S
## 4        1      1 female  35     1 53.1000        S
## 5        0      3   male  35     0  8.0500        S
## 6        0      3   male  24     0  8.4583        Q
```

Next, we will convert Survived, Pclass and Sibsp to factors instead of integers.

```r
df.train$Survived <- as.factor(df.train$Survived)
df.train$Pclass <- as.factor(df.train$Pclass)
df.train$SibSp <- as.factor(df.train$SibSp)
```

Verify the new structure

```r
str(df.train)
```

```
## 'data.frame':    891 obs. of  7 variables:
##  $ Survived: Factor w/ 2 levels "0","1": 1 2 2 2 1 1 1 1 2 2 ...
##  $ Pclass  : Factor w/ 3 levels "1","2","3": 3 1 3 1 3 3 1 3 3 2 ...
##  $ Sex     : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ Age     : num  22 38 26 35 35 24 54 2 27 14 ...
##  $ SibSp   : Factor w/ 7 levels "0","1","2","3",..: 2 2 1 2 1 1 1 4 1 2 ...
##  $ Fare    : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Embarked: Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
```

Building the model

Since we are calculating a logistic regression we use the function glm (general linear model) and assign it to log.model. We have to add a few more cavets to the equation. Since we are determining whether someone survived or not we assign the Surrived feature as the first argument seperated by the tilda symbol, the "." represents 'everything' as in the rest of the features (Pclass, Age, Sex, etc.). The family argument is a description of the error distribution we choose binomial since we are trying to determine if someone survived (1) or didn't (0). Link simply tells the binomial function to use the logistic regression.

```
log.model <- glm(Survived ~ ., family = binomial(link = 'logit'), data = df.train)
```

Summary of the model

```
summary(log.model)
```

```
##
## Call:
## glm(formula = Survived ~ ., family = binomial(link = "logit"),
##     data = df.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8567  -0.5970  -0.4081   0.6023   2.5256
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.869e+01  1.652e+03   0.011 0.990972
## Pclass2     -1.088e+00  3.050e-01  -3.567 0.000361 ***
## Pclass3     -2.291e+00  3.088e-01  -7.417 1.20e-13 ***
## Sexmale     -2.658e+00  1.986e-01 -13.385  < 2e-16 ***
## Age         -4.381e-02  8.407e-03  -5.211 1.88e-07 ***
## SibSp1       1.241e-01  2.129e-01   0.583 0.559832
## SibSp2      -2.011e-01  5.235e-01  -0.384 0.700810
## SibSp3      -2.080e+00  7.133e-01  -2.916 0.003544 **
## SibSp4      -1.592e+00  7.392e-01  -2.154 0.031271 *
## SibSp5      -1.592e+01  9.612e+02  -0.017 0.986787
## SibSp8      -1.599e+01  7.585e+02  -0.021 0.983182
## Fare         1.769e-03  2.310e-03   0.766 0.443690
## EmbarkedC   -1.459e+01  1.652e+03  -0.009 0.992955
## EmbarkedQ   -1.463e+01  1.652e+03  -0.009 0.992934
## EmbarkedS   -1.494e+01  1.652e+03  -0.009 0.992787
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1186.66  on 890  degrees of freedom
## Residual deviance:  771.26  on 876  degrees of freedom
## AIC: 801.26
##
## Number of Fisher Scoring iterations: 15
```

The p-value for each term tests the null hypothesis that the coefficient is equal to zero (no effect). A low p-value (< 0.05) indicates that you can reject the null hypothesis. In other words, a predictor that has a low p-value is likely to be a meaningful addition to your model because changes in the predictor's value are related to changes in the response variable (Survived). Conversely, a larger (insignificant) p-value suggests that changes in the predictor are not associated with changes in the response.

As you can tell by the model summary Pclass, Sex and Age are statistically significant in the survivability of the individual.

Now lets train the model to a test set so that we can determine the accuracy of the model.

Train vs Test checking the models accuracy

Lets create a test data frame by randomlly spliting the train data frame with a 70/30 split.

```
split <- sample.split(df.train$Survived, SplitRatio = 0.7)
final.train <- subset(df.train, split == TRUE)
final.test <- subset(df.train, split == FALSE)
```

The sample.split function randomlly splits the data frame by the directed split ratio in our case 70% and assigns it a 1 (TRUE) and the other 30% a 0 (FALSE) to each observation. Finally, all we need to do is assign each subset (split) to either a train or test data frame.

Now lets retrain the model on the 70% split final.train.

```
final.log.model <- glm(Survived ~ ., family = binomial(link = 'logit'), data = final.tra
```

Lets take a look at the summary of the new final log model.

```
summary(final.log.model)
```

```
##
## Call:
## glm(formula = Survived ~ ., family = binomial(link = "logit"),
##     data = final.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7785  -0.6044  -0.4189   0.6089   2.5454
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.873e+01  1.655e+03   0.011   0.9910
## Pclass2     -9.016e-01  3.735e-01  -2.414   0.0158 *
## Pclass3     -2.235e+00  3.778e-01  -5.916 3.29e-09 ***
## Sexmale     -2.563e+00  2.305e-01 -11.120  < 2e-16 ***
## Age         -4.266e-02  9.786e-03  -4.359 1.30e-05 ***
## SibSp1       3.319e-01  2.521e-01   1.317   0.1880
## SibSp2      -5.241e-01  6.632e-01  -0.790   0.4294
## SibSp3      -1.627e+00  7.279e-01  -2.235   0.0254 *
## SibSp4      -1.145e+00  7.699e-01  -1.487   0.1371
## SibSp5      -1.584e+01  1.064e+03  -0.015   0.9881
## SibSp8      -1.558e+01  8.347e+02  -0.019   0.9851
## Fare         5.098e-04  2.372e-03   0.215   0.8298
## EmbarkedC   -1.465e+01  1.655e+03  -0.009   0.9929
## EmbarkedQ   -1.478e+01  1.655e+03  -0.009   0.9929
## EmbarkedS   -1.522e+01  1.655e+03  -0.009   0.9927
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 829.60  on 622  degrees of freedom
## Residual deviance: 542.83  on 608  degrees of freedom
## AIC: 572.83
##
## Number of Fisher Scoring iterations: 15
```

Now lets predict; we will assign the prediction to fitted.probabilities

```
fitted.probabilities <-  predict(final.log.model, final.test, type = 'response')
```

Type is equal to response because we are trying to predict whether the lived or died or 0 and 1.

We want to assign the fitted.probabilities a 1 or a 0 so if the probability is greater than .5 then assign it a one else assign it a zero.

```
fitted.results <- ifelse(fitted.probabilities > 0.5, 1, 0)
fitted.results
```

```
##   3   4   7   9  14  24  26  27  32  34  36  37  38  41  42  47  48  49
##   1   1   0   1   0   0   1   0   1   0   0   0   0   0   1   0   1   0
##  50  51  56  61  64  67  71  73  74  77  78  79  83  93 100 101 106 114
##   1   0   0   0   0   1   0   0   0   0   0   1   1   0   0   1   0   1
## 122 123 133 135 136 137 139 140 151 152 154 155 158 163 165 168 169 177
##   0   0   0   0   0   1   0   1   0   1   0   0   0   0   0   0   0   0
## 179 182 188 197 199 201 205 211 214 219 224 225 227 228 229 231 233 238
##   0   0   0   0   1   0   0   0   0   1   0   0   0   0   0   1   0   1
## 248 249 258 264 266 267 269 270 272 274 281 285 286 288 293 296 307 309
##   1   0   1   0   0   0   1   1   0   0   0   0   0   0   0   0   1   0
## 313 314 315 318 319 321 324 326 330 331 332 333 337 338 345 347 348 353
##   1   0   0   0   1   0   1   1   1   1   0   0   1   1   0   1   1   0
## 357 368 370 383 385 386 388 390 391 395 396 397 407 408 411 415 416 419
##   1   1   1   0   0   0   1   1   0   1   0   0   0   1   0   0   1   0
## 423 433 437 438 445 448 450 453 456 460 461 462 464 468 469 470 472 476
##   0   1   0   1   0   0   0   1   0   0   0   0   0   0   0   1   0   0
## 480 481 484 492 493 500 502 507 508 510 511 520 523 525 527 529 537 539
##   1   0   0   0   0   0   1   1   0   0   0   0   0   0   1   0   0   0
## 540 543 548 549 551 554 557 560 562 563 566 569 579 581 597 601 614 621
##   1   0   0   0   1   0   1   1   0   0   0   0   1   1   1   1   0   0
## 622 629 633 636 637 638 640 642 643 644 645 649 651 652 655 658 659 660
##   0   0   1   1   0   0   0   1   0   0   1   0   0   1   1   1   0   0
## 661 662 663 672 676 677 683 690 691 697 700 703 708 712 714 715 716 717
##   0   0   0   0   0   0   0   1   0   0   0   1   0   0   0   0   0   1
## 718 720 722 725 728 734 736 737 742 748 749 751 756 757 758 759 760 765
##   1   0   0   1   1   0   0   0   0   1   1   1   1   0   0   0   1   0
## 767 770 771 774 776 779 780 782 784 789 790 801 802 819 822 827 831 834
##   0   0   0   0   0   0   1   1   0   0   0   0   1   0   0   0   1   0
## 839 843 850 857 863 864 867 873 874 876 878 881 882 884 885 887
##   0   1   1   1   1   0   1   0   0   1   0   1   0   0   0   0
```

Lets determine the error

```
missClassError <-  mean(fitted.results != final.test$Survived)
missClassError
```

```
## [1] 0.1679104
```

Finally, we will check for accuracy and assign the class error to accuracy and call accuracy to determine the result.

```
accuracy <- 1 - missClassError
accuracy
```

```
## [1] 0.8320896
```

As you can see based on the train and test data the model is 86% accurate in predicting whether someone survived or died based on input variables.

Lastly, we will create a confusion matrix

```
table(final.test$Survived, fitted.probabilities > 0.5)
```

```
##
##     FALSE TRUE
##   0   150   15
##   1    30   73
```

Breaking down the confusion matrix

0 and false (TN: True Negative): 152

0 and true (FP: False Positive): 13

1 and False (FN: False Negative): 24

1 and True (TP: True Positive): 79

Total Predicted False (Did not Survive): 176

Total Predicted True (Survived): 92

Total Actual False (Did not Survive): 165

Total Actual True (Survived): 103

Source:

Udemy - Data Science and Machine Learning Bootcamp with R (Jose Portilla)

An Intro to Statistical Learning with Applications in R (Gareth James, et al.) http://www-bcf.usc.edu/~gareth/ISL/ISLR%20Sixth%20Printing.pdf