Name: Patiguli Mulati

Net ID :fs 6833

Using k-NN Algorithm to classify Habermans Survival Data---classification

KNN-Algorithm: The k-NN algorithm uses information about an example's k-nearest neighbors to classify unlabeled examples. The letter k is a variable term implying that any number of nearest neighbors could be used. After choosing k, the algorithm requires a training dataset made up of examples that have been classified into several categories, as labeled by a nominal variable. Then, for each unlabeled record in the test dataset, k-NN identifies k records in the training data that are the "nearest" in similarity. The unlabeled test instance is assigned the class of the majority of the k nearest neighbors.

For my project ,I used k-NN algorithm to classify Haberman's survival status as class attributes either 1 or 2 which are based on age of patients at time of operation, Patient's year of operation and Number of positive axillary nodes detected.

## Step 1: Collecting Data ----

I will utilize Haberman's Survival Dataset from the UCI Machine Learning Repository at <http://archive.ics.uci.edu/ml>. This dataset contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer.

Haberman's Survival Dataset includes 306 examples of surgery patients, each with 4 features.The survival status is coded as 1 for patient survived 5 years or longer and 2 for patients died within 5 years.The other 3 numeric features are : 1. Age of patient at time of operation (numerical) 2. Patient's year of operation (year - 1900, numerical) 3. Number of positive axillary nodes detected (numerical) 4. Survival status (class attribute) -- 1 = the patient survived 5 years or longer -- 2 = the patient died within 5 year

;

## Step 2: Exploring and preparing the data ----   
# import the CSV file  
habermans\_survival <- read.csv("C:\\Users\\Omar\\Documents\\6620\\habermans\_survival\_data.csv", stringsAsFactors = FALSE)

I will begin by importing the CSV data file to RStudio using by read .csv command .

# examine the structure of the habermans\_survival data frame  
str(habermans\_survival)

## 'data.frame': 306 obs. of 4 variables:  
## $ Age : int 30 30 30 31 31 33 33 34 34 34 ...  
## $ Operation.year : int 64 62 65 59 65 58 60 59 66 58 ...  
## $ Number.of.positive.axillary.nodes.detected: int 1 3 0 2 4 10 0 0 9 30 ...  
## $ Survival.status : int 1 1 1 1 1 1 1 2 2 1 ...

Using the str(habermans\_survival) command, I can confirm that the data is structured with 306 examples and 4 features as I expected .The several lines of output are as above .

# table of survival.status  
table(habermans\_survival$Survival.status)

##   
## 1 2   
## 225 81

The next variable, Survival.status, is of particular interest as it is the outcome I hope to predict. This feature indicates whether the example is from alive or dead patients. The table() output indicates that 225 patients are alive while 81 are dead.

# recode Survival.status as a factor  
habermans\_survival$Survival.status <- factor(habermans\_survival$Survival.status, levels = c("1", "2"),  
 labels = c("Alive", "Dead"))

Many R machine learning classifiers require that the target feature is coded as a factor, so I will need to recode the Survival.status variable. I will also take this opportunity to give the "1" and "2" values more informative labels using the labels parameter.

# table or proportions with more informative labels  
round(prop.table(table(habermans\_survival$Survival.status)) \* 100, digits = 2)

##   
## Alive Dead   
## 73.53 26.47

Now, when we look at the prop.table() output, we notice that the values have been labeled Alive and Dead with 73.53 percent and 26.47 percent of the patients, respectively; In order to examine that the data needs to be nomalized or standardized ,I used summary function .

# summarize three numeric features  
summary(habermans\_survival[c("Age", "Operation.year", "Number.of.positive.axillary.nodes.detected")])

## Age Operation.year   
## Min. :30.00 Min. :58.00   
## 1st Qu.:44.00 1st Qu.:60.00   
## Median :52.00 Median :63.00   
## Mean :52.46 Mean :62.85   
## 3rd Qu.:60.75 3rd Qu.:65.75   
## Max. :83.00 Max. :69.00   
## Number.of.positive.axillary.nodes.detected  
## Min. : 0.000   
## 1st Qu.: 0.000   
## Median : 1.000   
## Mean : 4.026   
## 3rd Qu.: 4.000   
## Max. :52.000

The remaining 3 features are all numeric,but I got some problematic issues about the values . Because the distance calculation for k-NN is heavily dependent upon the measurement scale of the input features.Since the max value of Number.of.positive.axillary.nodes.detected is much higher than the mean and it would affect the distance calculation .so I need to apply normalization to rescale the features to a standard range of values .

# create normalization function  
normalize <- function(x) {  
 return ((x - min(x)) / (max(x) - min(x)))  
}

Transformation - normalizing numeric data To normalize these features,I need to create a normalize() function in R. This function takes a vector x of numeric values, and for each value in x, subtracts the minimum value in x and divides by the range of values in x. Finally, the resulting vector is returned. The code for this function is as above ;

# normalize the habermans\_survival data  
habermans\_survival\_n <- as.data.frame(lapply(habermans\_survival[1:3], normalize))

I applied the normalize() function to the 3 numeric features in my data frame.after applying normalization function , the features are normalized with values ranging from 0 to 1 .

# confirm that normalization worked  
summary(habermans\_survival\_n$Age )

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0000 0.2642 0.4151 0.4237 0.5802 1.0000

To confirm that the transformation was applied correctly, I used summary ()function to look at Age variable's summary statistics .

summary(habermans\_survival\_n$Operation.year)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0000 0.1818 0.4545 0.4412 0.7045 1.0000

Here's the Operation.year's summary statistics .Ranges from min as 0.0000 and max as 1.0000 .

summary(habermans\_survival\_n$Number.of.positive.axillary.nodes.detected)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.00000 0.00000 0.01923 0.07743 0.07692 1.00000

Number.of.positive.axillary.nodes.detected is also listed above . Mean as 0.01923 and median as 0.07692 .

# create training and test data  
habermans\_survival\_train <- habermans\_survival\_n[1:206, ]  
habermans\_survival\_test <- habermans\_survival\_n[207:306, ]

Data preparation - creating training and test datasets I divided my habermans\_survival dataset into two portions: a training dataset that will be used to build the k-NN model and a test dataset that will be used to estimate the predictive accuracy of the model. I will use the first 206 records for the training dataset and the remaining 100 to simulate new patients.

# create labels for training and test data  
  
habermans\_survival\_train\_labels <- habermans\_survival[1:206, 4]  
habermans\_survival\_test\_labels <- habermans\_survival[207:306, 4]

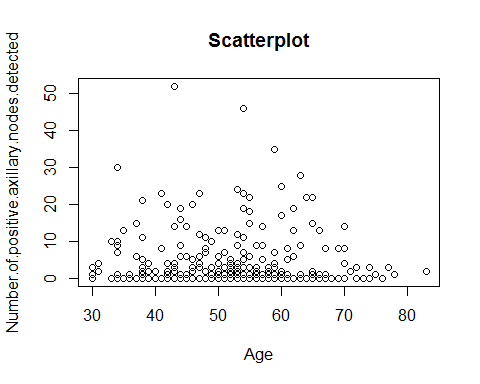
For training the k-NN model, we will need to store the survival status as a factor vectors, split between the training and test datasets.

summary(habermans\_survival\_train\_labels)

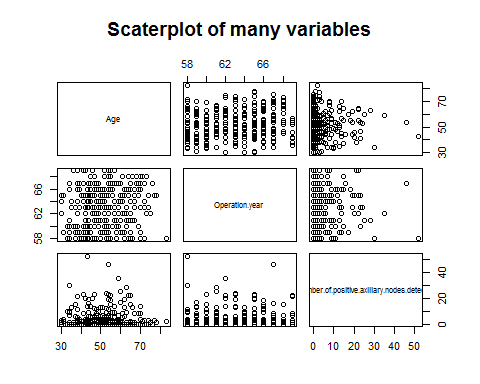
## Alive Dead   
## 150 56

Using summary () function ,I checked my train dataset whether labeled correctly or not . I set 206 as train dataset previously .By getting 150 Alive and 56 Dead patients , I confirmed my creating lables was correct .

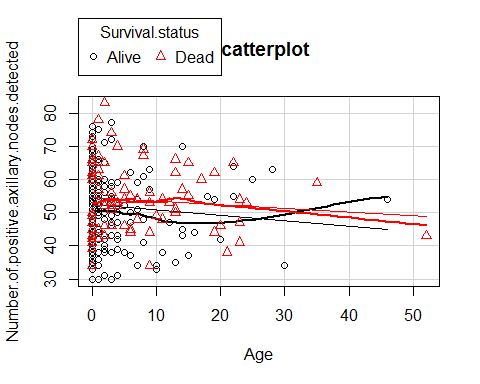
# visualize the data using labels  
  
plot(habermans\_survival$Age,habermans\_survival$Number.of.positive.axillary.nodes.detected,   
 main = 'Scatterplot',  
 xlab = 'Age',  
 ylab = 'Number.of.positive.axillary.nodes.detected')



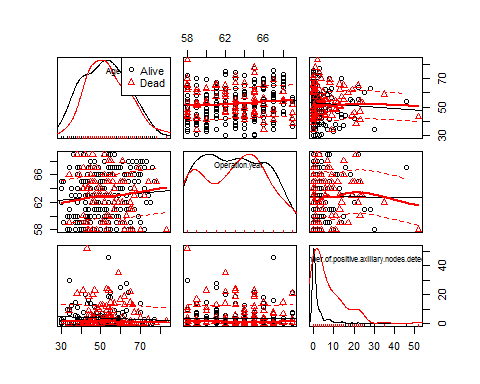
pairs(~Age+Operation.year+Number.of.positive.axillary.nodes.detected,   
 data = habermans\_survival,  
 main = 'Scaterplot of many variables')



library(car)  
  
scatterplot(Age ~ Number.of.positive.axillary.nodes.detected |Survival.status, data = habermans\_survival,  
 main = 'Scatterplot',  
 xlab = 'Age',  
 ylab = 'Number.of.positive.axillary.nodes.detected')



scatterplotMatrix(~Age+Operation.year+Number.of.positive.axillary.nodes.detected| Survival.status, data = habermans\_survival)



## Step 3: Training a model on the data ----  
  
# load the "class" library  
library(class)  
  
habermans\_survival\_test\_pred <- knn(train =habermans\_survival\_train, test = habermans\_survival\_test,  
 cl =habermans\_survival\_train\_labels, k =13)  
  
head(habermans\_survival\_test)

## Age Operation.year Number.of.positive.axillary.nodes.detected  
## 207 0.5094340 0.54545455 0.00000000  
## 208 0.5094340 0.81818182 0.00000000  
## 209 0.5283019 0.09090909 0.00000000  
## 210 0.5283019 0.18181818 0.05769231  
## 211 0.5283019 0.27272727 0.01923077  
## 212 0.5283019 0.81818182 0.00000000

head(habermans\_survival\_test\_pred)

## [1] Alive Alive Alive Alive Alive Alive  
## Levels: Alive Dead

For the k-NN algorithm, the training phase actually involves no model building; the process of training a lazy learner like k-NN simply involves storing the input data in a structured format. To classify our test instances, we will use a k-NN implementation from the class package, which provides a set of basic R functions for classification. The knn() function in the class package provides a standard, classic implementation of the k-NN algorithm. For each instance in the test data, the function will identify the k-Nearest Neighbors, using Euclidean distance, where k is a user-specified number. The test instance is classified by taking a "vote" among the k-Nearest Neighbors specifically, this involves assigning the class of the majority of the k neighbors. A tie vote is broken at random. The knn() function returns the predicted class using the training and the test datasets and the specified k value .As our training data includes 206 instances, we might try k = 13, an odd number roughly equal to the square root of 206. With a two-category outcome, using an odd number eliminates the chance of ending with a tie vote. Finally we used the knn() function to classify the test data and this function returned a factor vector of predicted labels as the levels of alive and dead for each of the examples in the test dataset, which we have assigned to Habermans\_test\_pred.

## Step 4: Evaluating model performance ----  
  
# load the "gmodels" library  
library(gmodels)  
# Create the cross tabulation of predicted vs. actual  
CrossTable(x = habermans\_survival\_test\_labels, y = habermans\_survival\_test\_pred,  
 prop.chisq = FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 100   
##   
##   
## | habermans\_survival\_test\_pred   
## habermans\_survival\_test\_labels | Alive | Dead | Row Total |   
## -------------------------------|-----------|-----------|-----------|  
## Alive | 69 | 6 | 75 |   
## | 0.920 | 0.080 | 0.750 |   
## | 0.793 | 0.462 | |   
## | 0.690 | 0.060 | |   
## -------------------------------|-----------|-----------|-----------|  
## Dead | 18 | 7 | 25 |   
## | 0.720 | 0.280 | 0.250 |   
## | 0.207 | 0.538 | |   
## | 0.180 | 0.070 | |   
## -------------------------------|-----------|-----------|-----------|  
## Column Total | 87 | 13 | 100 |   
## | 0.870 | 0.130 | |   
## -------------------------------|-----------|-----------|-----------|  
##   
##

The next step of the process is to evaluate how well the predicted classes in the habermans\_survival\_test\_pred vector match up with the known values in the habermans\_survival\_test\_labels vector. To do this, we can use the CrossTable() function in the gmodels. After using library(gmodels) command,we can create a cross tabulation indicating the agreement between the two vectors. Specifying prop.chisq = FALSE will remove the unnecessary chi-square values from the output.

We can split the results of CrossTable() into 4 values:

1.The top-left cell indicates the true negative results----69 of 100values were patients who were alive for 5 years or longer after the surgery and k-NN correctly identified it.

2.The bottom-right cell indicates the true positive results---7 of 100 values were patients who died within 5 years of the surgery and k-NN correctly identified it.

The cells falling on the other diagonal contain counts of examples where the k-NN approach disagreed with the true label.

3.The lower-left cell are false negative results , in this case ,k-NN predicted 18 of the 100 observations as Alive but they were Dead.

4.The top-right cell would contain the false positive results and K-NN predicted 6 out of the 100 observations as Dead even though they are alive .

Using confusion matrices to measure performance With the 2 x 2 confusion matrix, we can formalize our definition of prediction accuracy as: Accuracy =TP+TN/TP+TN+FP+FN; Error rate =FP+FN/TP+TN+FP+FN=1-Accuracy;

By using these two formulas ,we can calculate our model developed performance : Accuracy = 69+7/69+7+18+6 = 0.76 Error rate = 1-0.76 = 0.24;

A total of 24 out of 100, or 24 percent of Survival.status were incorrectly classified by the k-NN approach. While 76 percent accuracy seems impressive .But we might try another iteration of the model to see whether we can improve the performance and reduce the number of values that have been incorrectly classified, particularly because the errors were dangerous false negatives.

## Step 5: Improving model performance ----

We will attempt two simple variations on our previous classifier. First, we will employ an alternative method for rescaling our numeric features. Second, we will try several different values for k.

Transformation - z-score standardization

Although normalization is traditionally used for k-NN classification, it may not always be the most appropriate way to rescale features. Since the z-score standardized values have no predefined minimum and maximum, extreme values are not compressed towards the center.

# use the scale() function to z-score standardize a data frame  
habermans\_survival\_z <- as.data.frame(scale(habermans\_survival[-4]))

Instead of using transformation for normalizing the dataset, z-score standardization can be used to allow the outliers to be weighted heavily in the distance calculation. Our goal is to standardize our dataset by using scale function . Our command rescales all the features, with the exception of survivalstatus and stores the result as thehabermans\_survival \_z data frame. The \_z suffix is a reminder that the values were z-score transformed.

# confirm that the transformation was applied correctly  
summary(habermans\_survival\_z$Operation.year)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -1.49300 -0.87800 0.04526 0.00000 0.89160 1.89200

To confirm that the transformation was applied correctly, we looked at the summary statistics of Operarion.year .The mean of a z-score standardized variable should always be zero, and the range should be fairly compact. A z-score greater than 3 or less than -3 indicates an extremely rare value.In our summarized data there is no value greater than 3 or less than 3 ,so we do not have to worry about an extermely value .

# create training and test datasets  
habermans\_survival\_train <-habermans\_survival\_z[1:206, ]  
habermans\_survival\_test <- habermans\_survival\_z[207:306, ]  
habermans\_survival\_train\_labels <- habermans\_survival[1:206, 4]  
habermans\_survival\_test\_labels <- habermans\_survival[207:306, 4]

As we had done earlier, we need to divide the data into training and test sets, and then classify the test instances using the knn() function.

# re-classify test cases  
habermans\_survival\_test\_pred <- knn(train = habermans\_survival\_train, test = habermans\_survival\_test,  
 cl = habermans\_survival\_train\_labels, k = 13)  
# Create the cross tabulation of predicted vs. actual  
CrossTable(x = habermans\_survival\_test\_labels, y = habermans\_survival\_test\_pred,  
 prop.chisq = FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 100   
##   
##   
## | habermans\_survival\_test\_pred   
## habermans\_survival\_test\_labels | Alive | Dead | Row Total |   
## -------------------------------|-----------|-----------|-----------|  
## Alive | 69 | 6 | 75 |   
## | 0.920 | 0.080 | 0.750 |   
## | 0.802 | 0.429 | |   
## | 0.690 | 0.060 | |   
## -------------------------------|-----------|-----------|-----------|  
## Dead | 17 | 8 | 25 |   
## | 0.680 | 0.320 | 0.250 |   
## | 0.198 | 0.571 | |   
## | 0.170 | 0.080 | |   
## -------------------------------|-----------|-----------|-----------|  
## Column Total | 86 | 14 | 100 |   
## | 0.860 | 0.140 | |   
## -------------------------------|-----------|-----------|-----------|  
##   
##

By using these two formulas ,we can calculate our model developed performance : Accuracy = 69+8/69+8+17+6 = 0.77 Error rate = 1-0.77 = 0.23; After applying the knn ()function to this newly standardized data, the accuracy increased to 77%, hence this clearly is a little bit better model than what I was created earlier.

Another method that can be followed to be able to better the model is by applying various k values, the following are the results of trying out multiple k values.

# try several different values of k  
habermans\_survival\_train <- habermans\_survival\_n[1:206, ]  
habermans\_survival\_test <- habermans\_survival\_n[207:306, ]  
habermans\_survival\_test\_pred <- knn(train = habermans\_survival\_train, test = habermans\_survival\_test, cl = habermans\_survival\_train\_labels, k=1)  
CrossTable(x = habermans\_survival\_test\_labels, y = habermans\_survival\_test\_pred, prop.chisq=FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 100   
##   
##   
## | habermans\_survival\_test\_pred   
## habermans\_survival\_test\_labels | Alive | Dead | Row Total |   
## -------------------------------|-----------|-----------|-----------|  
## Alive | 66 | 9 | 75 |   
## | 0.880 | 0.120 | 0.750 |   
## | 0.805 | 0.500 | |   
## | 0.660 | 0.090 | |   
## -------------------------------|-----------|-----------|-----------|  
## Dead | 16 | 9 | 25 |   
## | 0.640 | 0.360 | 0.250 |   
## | 0.195 | 0.500 | |   
## | 0.160 | 0.090 | |   
## -------------------------------|-----------|-----------|-----------|  
## Column Total | 82 | 18 | 100 |   
## | 0.820 | 0.180 | |   
## -------------------------------|-----------|-----------|-----------|  
##   
##

Using the k value as 1 avoided some of the false negatives at the cost of adding false positives.So I used k value as 1 to improve my dataset .after using these values ,I obtained followed accuracy and error rate : Accuracy = 66+9/66+9+16+9 = 0.75 Error rate = 1-0.75 = 0.25; accuracy rate 75% and error rate 25 % .

habermans\_survival\_train <- habermans\_survival\_n[1:206, ]  
habermans\_survival\_test <- habermans\_survival\_n[207:306, ]  
habermans\_survival\_test\_pred <- knn(train = habermans\_survival\_train, test = habermans\_survival\_test, cl = habermans\_survival\_train\_labels, k=5)  
CrossTable(x = habermans\_survival\_test\_labels, y = habermans\_survival\_test\_pred, prop.chisq=FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 100   
##   
##   
## | habermans\_survival\_test\_pred   
## habermans\_survival\_test\_labels | Alive | Dead | Row Total |   
## -------------------------------|-----------|-----------|-----------|  
## Alive | 68 | 7 | 75 |   
## | 0.907 | 0.093 | 0.750 |   
## | 0.800 | 0.467 | |   
## | 0.680 | 0.070 | |   
## -------------------------------|-----------|-----------|-----------|  
## Dead | 17 | 8 | 25 |   
## | 0.680 | 0.320 | 0.250 |   
## | 0.200 | 0.533 | |   
## | 0.170 | 0.080 | |   
## -------------------------------|-----------|-----------|-----------|  
## Column Total | 85 | 15 | 100 |   
## | 0.850 | 0.150 | |   
## -------------------------------|-----------|-----------|-----------|  
##   
##

Using k=5 ,the accuracy and error rate are : Accuracy = 68+8/68+8+17+7 = 0.76 Error rate = 1-0.76 = 0.24; accuracy rate 76% and error rate 24 % .

habermans\_survival\_train <- habermans\_survival\_n[1:206, ]  
habermans\_survival\_test <- habermans\_survival\_n[207:306, ]  
habermans\_survival\_test\_pred <- knn(train = habermans\_survival\_train, test = habermans\_survival\_test, cl = habermans\_survival\_train\_labels, k=11)  
CrossTable(x = habermans\_survival\_test\_labels, y = habermans\_survival\_test\_pred, prop.chisq=FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 100   
##   
##   
## | habermans\_survival\_test\_pred   
## habermans\_survival\_test\_labels | Alive | Dead | Row Total |   
## -------------------------------|-----------|-----------|-----------|  
## Alive | 69 | 6 | 75 |   
## | 0.920 | 0.080 | 0.750 |   
## | 0.784 | 0.500 | |   
## | 0.690 | 0.060 | |   
## -------------------------------|-----------|-----------|-----------|  
## Dead | 19 | 6 | 25 |   
## | 0.760 | 0.240 | 0.250 |   
## | 0.216 | 0.500 | |   
## | 0.190 | 0.060 | |   
## -------------------------------|-----------|-----------|-----------|  
## Column Total | 88 | 12 | 100 |   
## | 0.880 | 0.120 | |   
## -------------------------------|-----------|-----------|-----------|  
##   
##

Using k=11 ,the accuracy and error rate are : Accuracy = 69+6/69+6+19+6 = 0.75 Error rate = 1-0.75 = 0.25; accuracy rate 75% and error rate 25% .

habermans\_survival\_train <- habermans\_survival\_n[1:206, ]  
habermans\_survival\_test <- habermans\_survival\_n[207:306, ]  
habermans\_survival\_test\_pred <- knn(train = habermans\_survival\_train, test = habermans\_survival\_test, cl = habermans\_survival\_train\_labels, k=15)  
CrossTable(x = habermans\_survival\_test\_labels, y = habermans\_survival\_test\_pred, prop.chisq=FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 100   
##   
##   
## | habermans\_survival\_test\_pred   
## habermans\_survival\_test\_labels | Alive | Dead | Row Total |   
## -------------------------------|-----------|-----------|-----------|  
## Alive | 70 | 5 | 75 |   
## | 0.933 | 0.067 | 0.750 |   
## | 0.787 | 0.455 | |   
## | 0.700 | 0.050 | |   
## -------------------------------|-----------|-----------|-----------|  
## Dead | 19 | 6 | 25 |   
## | 0.760 | 0.240 | 0.250 |   
## | 0.213 | 0.545 | |   
## | 0.190 | 0.060 | |   
## -------------------------------|-----------|-----------|-----------|  
## Column Total | 89 | 11 | 100 |   
## | 0.890 | 0.110 | |   
## -------------------------------|-----------|-----------|-----------|  
##   
##

Using k=15 ,the accuracy and error rate are : Accuracy = 70+6 /70+6+19+5 = 0.76 Error rate = 1-0.76 = 0.24; accuracy rate 76% and error rate 24 % .

habermans\_survival\_train <- habermans\_survival\_n[1:206, ]  
habermans\_survival\_test <- habermans\_survival\_n[207:306, ]  
habermans\_survival\_test\_pred <- knn(train = habermans\_survival\_train, test = habermans\_survival\_test, cl = habermans\_survival\_train\_labels, k=21)  
CrossTable(x = habermans\_survival\_test\_labels, y = habermans\_survival\_test\_pred, prop.chisq=FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 100   
##   
##   
## | habermans\_survival\_test\_pred   
## habermans\_survival\_test\_labels | Alive | Dead | Row Total |   
## -------------------------------|-----------|-----------|-----------|  
## Alive | 72 | 3 | 75 |   
## | 0.960 | 0.040 | 0.750 |   
## | 0.783 | 0.375 | |   
## | 0.720 | 0.030 | |   
## -------------------------------|-----------|-----------|-----------|  
## Dead | 20 | 5 | 25 |   
## | 0.800 | 0.200 | 0.250 |   
## | 0.217 | 0.625 | |   
## | 0.200 | 0.050 | |   
## -------------------------------|-----------|-----------|-----------|  
## Column Total | 92 | 8 | 100 |   
## | 0.920 | 0.080 | |   
## -------------------------------|-----------|-----------|-----------|  
##   
##

Using k=21 ,the accuracy and error rate are : Accuracy = 72+5/72+5+20+3 = 0.77 Error rate = 1-0.77 = 0.23; accuracy rate 77% and error rate 23 % .

habermans\_survival\_train <- habermans\_survival\_n[1:206, ]  
habermans\_survival\_test <- habermans\_survival\_n[207:306, ]  
habermans\_survival\_test\_pred <- knn(train = habermans\_survival\_train, test = habermans\_survival\_test, cl = habermans\_survival\_train\_labels, k=27)  
CrossTable(x = habermans\_survival\_test\_labels, y = habermans\_survival\_test\_pred, prop.chisq=FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 100   
##   
##   
## | habermans\_survival\_test\_pred   
## habermans\_survival\_test\_labels | Alive | Dead | Row Total |   
## -------------------------------|-----------|-----------|-----------|  
## Alive | 72 | 3 | 75 |   
## | 0.960 | 0.040 | 0.750 |   
## | 0.766 | 0.500 | |   
## | 0.720 | 0.030 | |   
## -------------------------------|-----------|-----------|-----------|  
## Dead | 22 | 3 | 25 |   
## | 0.880 | 0.120 | 0.250 |   
## | 0.234 | 0.500 | |   
## | 0.220 | 0.030 | |   
## -------------------------------|-----------|-----------|-----------|  
## Column Total | 94 | 6 | 100 |   
## | 0.940 | 0.060 | |   
## -------------------------------|-----------|-----------|-----------|  
##   
##

Using k=27 ,the accuracy and error rate are : Accuracy = 72+3/72+3+22+3 = 0.75 Error rate = 1-0.75 = 0.25; accuracy rate 75% and error rate 25 % .

Hence, we conclude the model with k as 21 which correctly identified whether the survival status was alive or dead as accuracy rate of 77% .

The number of false negatives and false positives are

shown for each iteration:

|  |  |  |  |
| --- | --- | --- | --- |
| **k value** | **False negatives** | **False positives** | **Percent classified incorrectly** |
| 1 | 16 | 9 | 25percent |
| 5 | 17 | 7 | 24 percent |
| 11 | 19 | 6 | 25percent |
| 15 | 19 | 5 | 24percent |
| 21 | 20 | 3 | 23percent |
| 27 | 22 | 3 | 25percent |

In spite of the fact that k-NN is a very simple algorithm, it is capable of tackling extremely complex tasks, such as identification of different status or cancerous masses as I learned from the book .The k-NN algorithm applied to the Haberman's Surgical Survival Data could classify the test data only with a 23% accuracy even after trying multiple k values and different transformation methods.