Predicting the Occurrence of Cancer

Oracy Martos

November 21, 2018

Predicting the Occurrence of Cancer

This project is an integral part of the Big Data Analytics course with R and Microsoft Azure Data Education Training. The goal is to analyze actual data on breast cancer exams performed with women in the US and then predict the occurrence of new cases.

Breast cancer data include 569 observations of cancer biopsies, each with 32 (variable) characteristics. One characteristic is an identification number (ID), another is the diagnosis of cancer, and 30 are numerical laboratory measures. The diagnosis is coded as “M” to indicate malignant or “B” to indicate benign.

Step 1 - Data Gathering

Here is the data collection, in this case a csv file.

```
# http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29
# http://datascienceacademy.com.br/blog/aluno/RFundamentos/Datasets/ML/wisc_bc_data.csv

df <- read.csv(file = "/home/oracy/Documents/DSA_Projetos/DSA_Projetos/Big Data Analytics com R e Micros
str(df)
```

```
## 'data.frame':    569 obs. of  32 variables:
##  $ id               : int  87139402 8910251 905520 868871 9012568 906539 925291 87880 862989 89827 .
##  $ diagnosis        : Factor w/ 2 levels "B","M": 1 1 1 1 1 1 1 2 1 1 ...
##  $ radius_mean      : num  12.3 10.6 11 11.3 15.2 ...
##  $ texture_mean     : num  12.4 18.9 16.8 13.4 13.2 ...
##  $ perimeter_mean   : num  78.8 69.3 70.9 73 97.7 ...
##  $ area_mean        : num  464 346 373 385 712 ...
##  $ smoothness_mean  : num  0.1028 0.0969 0.1077 0.1164 0.0796 ...
##  $ compactness_mean : num  0.0698 0.1147 0.078 0.1136 0.0693 ...
##  $ concavity_mean   : num  0.0399 0.0639 0.0305 0.0464 0.0339 ...
##  $ points_mean      : num  0.037 0.0264 0.0248 0.048 0.0266 ...
##  $ symmetry_mean    : num  0.196 0.192 0.171 0.177 0.172 ...
##  $ dimension_mean   : num  0.0595 0.0649 0.0634 0.0607 0.0554 ...
##  $ radius_se        : num  0.236 0.451 0.197 0.338 0.178 ...
##  $ texture_se       : num  0.666 1.197 1.387 1.343 0.412 ...
##  $ perimeter_se     : num  1.67 3.43 1.34 1.85 1.34 ...
##  $ area_se          : num  17.4 27.1 13.5 26.3 17.7 ...
##  $ smoothness_se    : num  0.00805 0.00747 0.00516 0.01127 0.00501 ...
##  $ compactness_se   : num  0.0118 0.03581 0.00936 0.03498 0.01485 ...
##  $ concavity_se     : num  0.0168 0.0335 0.0106 0.0219 0.0155 ...
##  $ points_se        : num  0.01241 0.01365 0.00748 0.01965 0.00915 ...
##  $ symmetry_se      : num  0.0192 0.035 0.0172 0.0158 0.0165 ...
##  $ dimension_se     : num  0.00225 0.00332 0.0022 0.00344 0.00177 ...
##  $ radius_worst     : num  13.5 11.9 12.4 11.9 16.2 ...
##  $ texture_worst    : num  15.6 22.9 26.4 15.8 15.7 ...
##  $ perimeter_worst  : num  87 78.3 79.9 76.5 104.5 ...
##  $ area_worst       : num  549 425 471 434 819 ...
##  $ smoothness_worst : num  0.139 0.121 0.137 0.137 0.113 ...
##  $ compactness_worst: num  0.127 0.252 0.148 0.182 0.174 ...
##  $ concavity_worst  : num  0.1242 0.1916 0.1067 0.0867 0.1362 ...
##  $ points_worst     : num  0.0939 0.0793 0.0743 0.0861 0.0818 ...
##  $ symmetry_worst   : num  0.283 0.294 0.3 0.21 0.249 ...
##  $ dimension_worst  : num  0.0677 0.0759 0.0788 0.0678 0.0677 ...
```

```
head(df)
```

```
##         id diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1 87139402         B       12.32        12.39          78.85     464.1
## 2  8910251         B       10.60        18.95          69.28     346.4
## 3   905520         B       11.04        16.83          70.92     373.2
## 4   868871         B       11.28        13.39          73.00     384.8
## 5  9012568         B       15.19        13.21          97.65     711.8
## 6   906539         B       11.57        19.04          74.20     409.7
##   smoothness_mean compactness_mean concavity_mean points_mean
## 1         0.10280          0.06981        0.03987     0.03700
## 2         0.09688          0.11470        0.06387     0.02642
## 3         0.10770          0.07804        0.03046     0.02480
## 4         0.11640          0.11360        0.04635     0.04796
## 5         0.07963          0.06934        0.03393     0.02657
## 6         0.08546          0.07722        0.05485     0.01428
##   symmetry_mean dimension_mean radius_se texture_se perimeter_se area_se
## 1        0.1959        0.05955    0.2360     0.6656        1.670   17.43
## 2        0.1922        0.06491    0.4505     1.1970        3.430   27.10
## 3        0.1714        0.06340    0.1967     1.3870        1.342   13.54
## 4        0.1771        0.06072    0.3384     1.3430        1.851   26.33
## 5        0.1721        0.05544    0.1783     0.4125        1.338   17.72
## 6        0.2031        0.06267    0.2864     1.4400        2.206   20.30
##   smoothness_se compactness_se concavity_se points_se symmetry_se
## 1      0.008045       0.011800      0.01683  0.012410     0.01924
## 2      0.007470       0.035810      0.03354  0.013650     0.03504
## 3      0.005158       0.009355      0.01056  0.007483     0.01718
## 4      0.011270       0.034980      0.02187  0.019650     0.01580
## 5      0.005012       0.014850      0.01551  0.009155     0.01647
## 6      0.007278       0.020470      0.04447  0.008799     0.01868
##   dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1     0.002248        13.50         15.64           86.97      549.1
## 2     0.003318        11.88         22.94           78.28      424.8
## 3     0.002198        12.41         26.44           79.93      471.4
## 4     0.003442        11.92         15.77           76.53      434.0
## 5     0.001767        16.20         15.73          104.50      819.1
## 6     0.003339        13.07         26.98           86.43      520.5
##   smoothness_worst compactness_worst concavity_worst points_worst
## 1           0.1385            0.1266         0.12420      0.09391
## 2           0.1213            0.2515         0.19160      0.07926
## 3           0.1369            0.1482         0.10670      0.07431
## 4           0.1367            0.1822         0.08669      0.08611
## 5           0.1126            0.1737         0.13620      0.08178
## 6           0.1249            0.1937         0.25600      0.06664
##   symmetry_worst dimension_worst
## 1         0.2827         0.06771
## 2         0.2940         0.07587
## 3         0.2998         0.07881
## 4         0.2102         0.06784
## 5         0.2487         0.06766
## 6         0.3035         0.08284
```

Step 2 - Data Exploration

Regardless of the machine learning method, ID variables should always be deleted. Otherwise, this can lead to erroneous results because the ID can be used to uniquely "predict" each example. Therefore, a model that includes an identifier may suffer from over-adjustment, and it will be very difficult to use it for generalize other data.

```r
# Remove ID Column
# Check if there is some NA value
# Font: https://stackoverflow.com/questions/6286313/remove-an-entire-column-from-a-data-frame-in-r
# Font: https://discuss.analyticsvidhya.com/t/how-can-i-check-whether-my-data-frame-contains-na-inf-val
df$id <- NULL
str(df)
```

```
## 'data.frame':    569 obs. of  31 variables:
##  $ diagnosis        : Factor w/ 2 levels "B","M": 1 1 1 1 1 1 1 2 1 1 ...
##  $ radius_mean      : num  12.3 10.6 11 11.3 15.2 ...
##  $ texture_mean     : num  12.4 18.9 16.8 13.4 13.2 ...
##  $ perimeter_mean   : num  78.8 69.3 70.9 73 97.7 ...
##  $ area_mean        : num  464 346 373 385 712 ...
##  $ smoothness_mean  : num  0.1028 0.0969 0.1077 0.1164 0.0796 ...
##  $ compactness_mean : num  0.0698 0.1147 0.078 0.1136 0.0693 ...
##  $ concavity_mean   : num  0.0399 0.0639 0.0305 0.0464 0.0339 ...
##  $ points_mean      : num  0.037 0.0264 0.0248 0.048 0.0266 ...
##  $ symmetry_mean    : num  0.196 0.192 0.171 0.177 0.172 ...
##  $ dimension_mean   : num  0.0595 0.0649 0.0634 0.0607 0.0554 ...
##  $ radius_se        : num  0.236 0.451 0.197 0.338 0.178 ...
##  $ texture_se       : num  0.666 1.197 1.387 1.343 0.412 ...
##  $ perimeter_se     : num  1.67 3.43 1.34 1.85 1.34 ...
##  $ area_se          : num  17.4 27.1 13.5 26.3 17.7 ...
##  $ smoothness_se    : num  0.00805 0.00747 0.00516 0.01127 0.00501 ...
##  $ compactness_se   : num  0.0118 0.03581 0.00936 0.03498 0.01485 ...
##  $ concavity_se     : num  0.0168 0.0335 0.0106 0.0219 0.0155 ...
##  $ points_se        : num  0.01241 0.01365 0.00748 0.01965 0.00915 ...
##  $ symmetry_se      : num  0.0192 0.035 0.0172 0.0158 0.0165 ...
##  $ dimension_se     : num  0.00225 0.00332 0.0022 0.00344 0.00177 ...
##  $ radius_worst     : num  13.5 11.9 12.4 11.9 16.2 ...
##  $ texture_worst    : num  15.6 22.9 26.4 15.8 15.7 ...
##  $ perimeter_worst  : num  87 78.3 79.9 76.5 104.5 ...
##  $ area_worst       : num  549 425 471 434 819 ...
##  $ smoothness_worst : num  0.139 0.121 0.137 0.137 0.113 ...
##  $ compactness_worst: num  0.127 0.252 0.148 0.182 0.174 ...
##  $ concavity_worst  : num  0.1242 0.1916 0.1067 0.0867 0.1362 ...
##  $ points_worst     : num  0.0939 0.0793 0.0743 0.0861 0.0818 ...
##  $ symmetry_worst   : num  0.283 0.294 0.3 0.21 0.249 ...
##  $ dimension_worst  : num  0.0677 0.0759 0.0788 0.0678 0.0677 ...
```

```r
head(df)
```

```
##   diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1         B       12.32        12.39          78.85     464.1
## 2         B       10.60        18.95          69.28     346.4
## 3         B       11.04        16.83          70.92     373.2
## 4         B       11.28        13.39          73.00     384.8
## 5         B       15.19        13.21          97.65     711.8
## 6         B       11.57        19.04          74.20     409.7
##   smoothness_mean compactness_mean concavity_mean points_mean
## 1         0.10280          0.06981        0.03987     0.03700
## 2         0.09688          0.11470        0.06387     0.02642
## 3         0.10770          0.07804        0.03046     0.02480
## 4         0.11640          0.11360        0.04635     0.04796
## 5         0.07963          0.06934        0.03393     0.02657
## 6         0.08546          0.07722        0.05485     0.01428
##   symmetry_mean dimension_mean radius_se texture_se perimeter_se area_se
## 1        0.1959        0.05955    0.2360     0.6656        1.670   17.43
## 2        0.1922        0.06491    0.4505     1.1970        3.430   27.10
## 3        0.1714        0.06340    0.1967     1.3870        1.342   13.54
## 4        0.1771        0.06072    0.3384     1.3430        1.851   26.33
## 5        0.1721        0.05544    0.1783     0.4125        1.338   17.72
## 6        0.2031        0.06267    0.2864     1.4400        2.206   20.30
##   smoothness_se compactness_se concavity_se points_se symmetry_se
## 1      0.008045       0.011800      0.01683  0.012410     0.01924
## 2      0.007470       0.035810      0.03354  0.013650     0.03504
## 3      0.005158       0.009355      0.01056  0.007483     0.01718
## 4      0.011270       0.034980      0.02187  0.019650     0.01580
## 5      0.005012       0.014850      0.01551  0.009155     0.01647
## 6      0.007278       0.020470      0.04447  0.008799     0.01868
##   dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1     0.002248        13.50         15.64           86.97      549.1
## 2     0.003318        11.88         22.94           78.28      424.8
## 3     0.002198        12.41         26.44           79.93      471.4
## 4     0.003442        11.92         15.77           76.53      434.0
## 5     0.001767        16.20         15.73          104.50      819.1
## 6     0.003339        13.07         26.98           86.43      520.5
##   smoothness_worst compactness_worst concavity_worst points_worst
## 1           0.1385            0.1266         0.12420      0.09391
## 2           0.1213            0.2515         0.19160      0.07926
## 3           0.1369            0.1482         0.10670      0.07431
## 4           0.1367            0.1822         0.08669      0.08611
## 5           0.1126            0.1737         0.13620      0.08178
## 6           0.1249            0.1937         0.25600      0.06664
##   symmetry_worst dimension_worst
## 1         0.2827         0.06771
## 2         0.2940         0.07587
## 3         0.2998         0.07881
## 4         0.2102         0.06784
## 5         0.2487         0.06766
## 6         0.3035         0.08284
```

```r
any(is.na(df))
```

```
## [1] FALSE
```

```r
# Many classifiers require that the variables be Factor type
# Font: https://stats.idre.ucla.edu/r/modules/factor-variables/
# Font: https://www.statmethods.net/input/valuelabels.html
df$diagnosis <- factor(df$diagnosis, levels = c("B", "M"), labels = c("Benigno", "Maligno"))

str(df)
```

```
## 'data.frame':    569 obs. of  31 variables:
##  $ diagnosis        : Factor w/ 2 levels "Benigno","Maligno": 1 1 1 1 1 1 1 2 1 1 ...
##  $ radius_mean      : num  12.3 10.6 11 11.3 15.2 ...
##  $ texture_mean     : num  12.4 18.9 16.8 13.4 13.2 ...
##  $ perimeter_mean   : num  78.8 69.3 70.9 73 97.7 ...
##  $ area_mean        : num  464 346 373 385 712 ...
##  $ smoothness_mean  : num  0.1028 0.0969 0.1077 0.1164 0.0796 ...
##  $ compactness_mean : num  0.0698 0.1147 0.078 0.1136 0.0693 ...
##  $ concavity_mean   : num  0.0399 0.0639 0.0305 0.0464 0.0339 ...
##  $ points_mean      : num  0.037 0.0264 0.0248 0.048 0.0266 ...
##  $ symmetry_mean    : num  0.196 0.192 0.171 0.177 0.172 ...
##  $ dimension_mean   : num  0.0595 0.0649 0.0634 0.0607 0.0554 ...
##  $ radius_se        : num  0.236 0.451 0.197 0.338 0.178 ...
##  $ texture_se       : num  0.666 1.197 1.387 1.343 0.412 ...
##  $ perimeter_se     : num  1.67 3.43 1.34 1.85 1.34 ...
##  $ area_se          : num  17.4 27.1 13.5 26.3 17.7 ...
##  $ smoothness_se    : num  0.00805 0.00747 0.00516 0.01127 0.00501 ...
##  $ compactness_se   : num  0.0118 0.03581 0.00936 0.03498 0.01485 ...
##  $ concavity_se     : num  0.0168 0.0335 0.0106 0.0219 0.0155 ...
##  $ points_se        : num  0.01241 0.01365 0.00748 0.01965 0.00915 ...
##  $ symmetry_se      : num  0.0192 0.035 0.0172 0.0158 0.0165 ...
##  $ dimension_se     : num  0.00225 0.00332 0.0022 0.00344 0.00177 ...
##  $ radius_worst     : num  13.5 11.9 12.4 11.9 16.2 ...
##  $ texture_worst    : num  15.6 22.9 26.4 15.8 15.7 ...
##  $ perimeter_worst  : num  87 78.3 79.9 76.5 104.5 ...
##  $ area_worst       : num  549 425 471 434 819 ...
##  $ smoothness_worst : num  0.139 0.121 0.137 0.137 0.113 ...
##  $ compactness_worst: num  0.127 0.252 0.148 0.182 0.174 ...
##  $ concavity_worst  : num  0.1242 0.1916 0.1067 0.0867 0.1362 ...
##  $ points_worst     : num  0.0939 0.0793 0.0743 0.0861 0.0818 ...
##  $ symmetry_worst   : num  0.283 0.294 0.3 0.21 0.249 ...
##  $ dimension_worst  : num  0.0677 0.0759 0.0788 0.0678 0.0677 ...
```

```r
# Checking the proportion
# dfProp <- prop.table(table(df$diagnosis))
# dfProp <- dfProp * 100
# dfProp <- round(dfProp, digits = 2)

dfProp <- round(prop.table(table(df$diagnosis)) * 100, digits = 2)
dfProp
```

```
##
## Benigno Maligno
##   62.74   37.26
```

```
# Central Tendency Measures
# We detected here a problem of scale between the data, which then need to be normalized
# The distance calculation made by kNN is dependent on the scale measurements in the inupt data.
summary(df[c("radius_mean", "area_mean", "smoothness_mean")])
```

```
##   radius_mean       area_mean      smoothness_mean
##  Min.   : 6.981   Min.   : 143.5   Min.   :0.05263
##  1st Qu.:11.700   1st Qu.: 420.3   1st Qu.:0.08637
##  Median :13.370   Median : 551.1   Median :0.09587
##  Mean   :14.127   Mean   : 654.9   Mean   :0.09636
##  3rd Qu.:15.780   3rd Qu.: 782.7   3rd Qu.:0.10530
##  Max.   :28.110   Max.   :2501.0   Max.   :0.16340
```

```
# Normalize function
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

# Testing normalize function - the results should be equals
normalize(c(1, 2, 3, 4, 5))
```

```
## [1] 0.00 0.25 0.50 0.75 1.00
```

```
normalize(c(10, 20, 30, 40, 50))
```

```
## [1] 0.00 0.25 0.50 0.75 1.00
```

```
# Normalizing data
df_norm <- as.data.frame(lapply(df[2:31], normalize))
#df_norm

# Checking that the normalize worked
summary(df[c("radius_mean", "area_mean", "smoothness_mean")])
```

```
##   radius_mean       area_mean      smoothness_mean
##  Min.   : 6.981   Min.   : 143.5   Min.   :0.05263
##  1st Qu.:11.700   1st Qu.: 420.3   1st Qu.:0.08637
##  Median :13.370   Median : 551.1   Median :0.09587
##  Mean   :14.127   Mean   : 654.9   Mean   :0.09636
##  3rd Qu.:15.780   3rd Qu.: 782.7   3rd Qu.:0.10530
##  Max.   :28.110   Max.   :2501.0   Max.   :0.16340
```

```
summary(df_norm[c("radius_mean", "area_mean", "smoothness_mean")])
```

```
##   radius_mean       area_mean      smoothness_mean
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.2233   1st Qu.:0.1174   1st Qu.:0.3046
##  Median :0.3024   Median :0.1729   Median :0.3904
##  Mean   :0.3382   Mean   :0.2169   Mean   :0.3948
##  3rd Qu.:0.4164   3rd Qu.:0.2711   3rd Qu.:0.4755
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

Step 3: Training the model

With the data properly normalized, we can now begin the process of training the model. To do this, let's split our data set into training data and test data.

```
# Installing and loading class package
# install.packages("class")
library(class)
?knn

# Creating training data and test data
# Font: https://cran.r-project.org/web/packages/dataPreparation/vignettes/train_test_prep.html
# Font: https://www.analyticsvidhya.com/blog/2015/08/learning-concept-knn-algorithms-programming/
#nrow(df_norm)
train_index <- df_norm[1:398,]
test_index <- df_norm[399:569,]
#length(train_index)
#length(test_index)

# Creating labels for training and test data.
df_train_labels <- df[1:398, 1]
df_test_labels <- df[399:569, 1]
#length(df_train_labels)
#length(df_test_labels)

# Creating the model
?knn
model <- knn(train = train_index, test = test_index, cl = df_train_labels, k = 10)

# The function knn() returns a factor type object with the predictions for each example in the test dat
class(model)
```

```
## [1] "factor"
```

Step 4: Evaluating and Interpreting the Model

Let us now evaluate the performance of the model.

```
# Installing and loading gmodels package
# Font: https://www.analyticsvidhya.com/blog/2015/08/learning-concept-knn-algorithms-programming/
#install.packages("gmodels")
library(gmodels)

# Creating cross table of predicted data vs. current data
# We will use sample with 171 observations: length (data_teste_labels)
length(df_test_labels)
```

```
## [1] 171
```

```r
# Check the definition of confusion matrix
CrossTable(x = df_test_labels, y = model, prop.chisq = FALSE)
```

```
## 
## 
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
## 
## 
## Total Observations in Table:  171 
## 
## 
##                 | model 
## df_test_labels |   Benigno |   Maligno | Row Total | 
## ---------------|-----------|-----------|-----------|
##        Benigno |       107 |         0 |       107 | 
##                |     1.000 |     0.000 |     0.626 | 
##                |     0.973 |     0.000 |           | 
##                |     0.626 |     0.000 |           | 
## ---------------|-----------|-----------|-----------|
##        Maligno |         3 |        61 |        64 | 
##                |     0.047 |     0.953 |     0.374 | 
##                |     0.027 |     1.000 |           | 
##                |     0.018 |     0.357 |           | 
## ---------------|-----------|-----------|-----------|
##   Column Total |       110 |        61 |       171 | 
##                |     0.643 |     0.357 |           | 
## ---------------|-----------|-----------|-----------|
## 
## 
```

```r
# Interpreting results
# The cross tabel shows 4 possible values, which represent the false/true positive and negative
# The first columns list the originals labels on the observed data.
# The two columns of the model (Benign and Malignant) of the model, show the results of the forecast
# We have:
# Schenario 1: Benign cell (label) x Benign (Model) - 107 cases - true negative
# Schenario 2: Benign cell (label) x Malignant (Model) - 00 cases - false positive
# Schenario 3: Malignant Cell (label) x Benign (Model) - 03 cases - false negative (model missed)
# Schenario 4: Malignant Cell (label) x Malignant (Model) - 61 cases - true positive

# Reading the Confusing Matrix (Perspect of having the disease or not)

# True Negative  = Our model predicted that the person did NOT have the disease and the data showed tha
# False Positive = Our model predicted that the person had the disease and the data showed that NO, the
# False Negative = Our model predicted that the person did NOT have the disease and the data showed tha
# True Positive = Our model predicted that the person had the disease and the data showed that YES, the

# False Positive - Type I Error
# False Negative - Type II Error

# Model Accuracy: 98.24% (168 out of 171)
```

Step 5: Optimizing model performance

```
# Using the scale() function to standardize the z-score
?scale()
df_z <- as.data.frame(scale(df[-1]))
#df_z

# Confirming transformation performed successfully
summary(df_z[c("radius_mean", "area_mean", "smoothness_mean")])
```

```
##   radius_mean        area_mean       smoothness_mean
##  Min.   :-2.0279   Min.   :-1.4532   Min.   :-3.10935
##  1st Qu.:-0.6888   1st Qu.:-0.6666   1st Qu.:-0.71034
##  Median :-0.2149   Median :-0.2949   Median :-0.03486
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.4690   3rd Qu.: 0.3632   3rd Qu.: 0.63564
##  Max.   : 3.9678   Max.   : 5.2459   Max.   : 4.76672
```

```
# Creating new training and test datasets
# Font: https://cran.r-project.org/web/packages/dataPreparation/vignettes/train_test_prep.html
# Font: https://www.analyticsvidhya.com/blog/2015/08/learning-concept-knn-algorithms-programming/
train_index_z <- df_z[1:398,]
test_index_z <- df_z[399:569,]
#length(train_index_z)
#length(test_index_z)

# Creating labels for training and test data.
df_train_labels_z <- df[1:398, 1]
df_test_labels_z <- df[399:569, 1]
#length(df_train_labels_z)
#length(df_test_labels_z)

# Reclassifying
model_z <- knn(train = train_index, test = test_index, cl = df_train_labels, k = 10)

# Creating a cross table of predicted data vs. current data
CrossTable(x = df_test_labels_z, y = model_z, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  171
##
##
##                  | model_z
## df_test_labels_z |   Benigno |   Maligno | Row Total |
## -----------------|-----------|-----------|-----------|
##          Benigno |       107 |         0 |       107 |
##                  |     1.000 |     0.000 |     0.626 |
##                  |     0.973 |     0.000 |           |
##                  |     0.626 |     0.000 |           |
## -----------------|-----------|-----------|-----------|
##          Maligno |         3 |        61 |        64 |
##                  |     0.047 |     0.953 |     0.374 |
##                  |     0.027 |     1.000 |           |
##                  |     0.018 |     0.357 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |       110 |        61 |       171 |
##                  |     0.643 |     0.357 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

```
# Testing different values for K
# Creating training data and test data
train_index_2 <- df_z[1:398,]
test_index_2 <- df_z[399:569,]
#length(train_index_2)
#length(test_index_2)

# Creating labels for training and test data
df_train_labels_2 <- df[1:398, 1]
df_test_labels_2 <- df[399:569, 1]
#length(df_train_labels_2)
#length(df_test_labels_2)

# Different values for K
# model_v3 <- knn(train = train_index, test = test_index, cl = df_train_labels, k = 1)
# CrossTable(x = df_test_labels_z, y = model_v3, prop.chisq = FALSE)

# model_v4 <- knn(train = train_index, test = test_index, cl = df_train_labels, k = 5)
# CrossTable(x = df_test_labels_z, y = model_v4, prop.chisq = FALSE)

# model_v5 <- knn(train = train_index, test = test_index, cl = df_train_labels, k = 11)
# CrossTable(x = df_test_labels_z, y = model_v5, prop.chisq = FALSE)

# model_v6 <- knn(train = train_index, test = test_index, cl = df_train_labels, k = 15)
# CrossTable(x = df_test_labels_z, y = model_v6, prop.chisq = FALSE)
# model_v7 <- knn(train = train_index, test = test_index, cl = df_train_labels, k = 27)
# CrossTable(x = df_test_labels_z, y = model_v7, prop.chisq = FALSE)

# model_v8 <- knn(train = train_index, test = test_index, cl = df_train_labels, k = 21)
# CrossTable(x = df_test_labels_z, y = model_v8, prop.chisq = FALSE)
```

Step 6 - Calculating the error rate

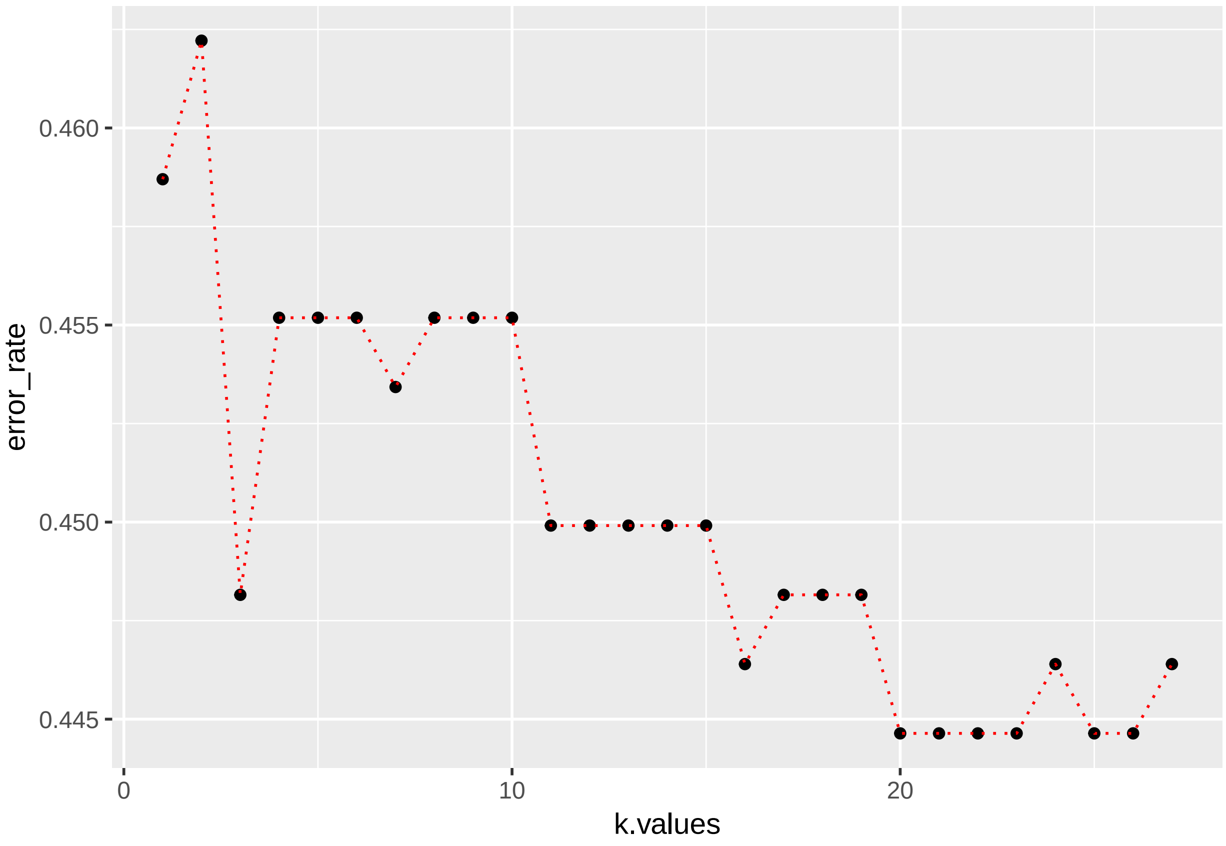
```
prev = NULL
error_rate = NULL

suppressWarnings(
  for(i in 1:27){
    set.seed(101)
    prev = knn(train = train_index_2, test = test_index_2, cl = df_train_labels_2, k = i)
    error_rate[i] = mean(df$diagnosis != prev)
  })

# Getting the k-values and error rates
#install.packages("ggplot2")
library(ggplot2)
k.values <- 1:27
df_error <- data.frame(error_rate, k.values)
df_error
```

```
##    error_rate k.values
## 1   0.4586995        1
## 2   0.4622144        2
## 3   0.4481547        3
## 4   0.4551845        4
## 5   0.4551845        5
## 6   0.4551845        6
## 7   0.4534271        7
## 8   0.4551845        8
## 9   0.4551845        9
## 10  0.4551845       10
## 11  0.4499121       11
## 12  0.4499121       12
## 13  0.4499121       13
## 14  0.4499121       14
## 15  0.4499121       15
## 16  0.4463972       16
## 17  0.4481547       17
## 18  0.4481547       18
## 19  0.4481547       19
## 20  0.4446397       20
## 21  0.4446397       21
## 22  0.4446397       22
## 23  0.4446397       23
## 24  0.4463972       24
## 25  0.4446397       25
## 26  0.4446397       26
## 27  0.4463972       27
```

```
# As we increase K, we decrease the model error rate
ggplot(df_error, aes(x = k.values, y = error_rate)) + geom_point()+ geom_line(lty = "dotted", color = '
```



Fim