Using K-NN to predict breast cancer

Contents

Introduction

We are going to use the "Breast Cancer Wisconsing Diagnostic" dataset from the *UCI Machine Learning Repository*. The data set donated by researchers includes measurements from digitized images of fine-needle aspirate of a breast mass. The features describe characteristics of the cell nuclei present in the images.

The dataset includes 569 samples with 32 features each. One of the features is an identification number and another is the cancer diagnosis, the rest are numeric values. The diagnosis is encoded as 'M' and 'B' representing "Malignant" and "Benign" respectively.

The measurements include the mean, standard error and worst value for the following measurements: Radius, Texture, Perimeter, Area, Smoothness, Compactness, Concavity, Concave points, Symmetry and Fractal dimension.

Exploring and preparing the data

We will start by importing the necessary libraries

```
library(caTools)
library(class)
```

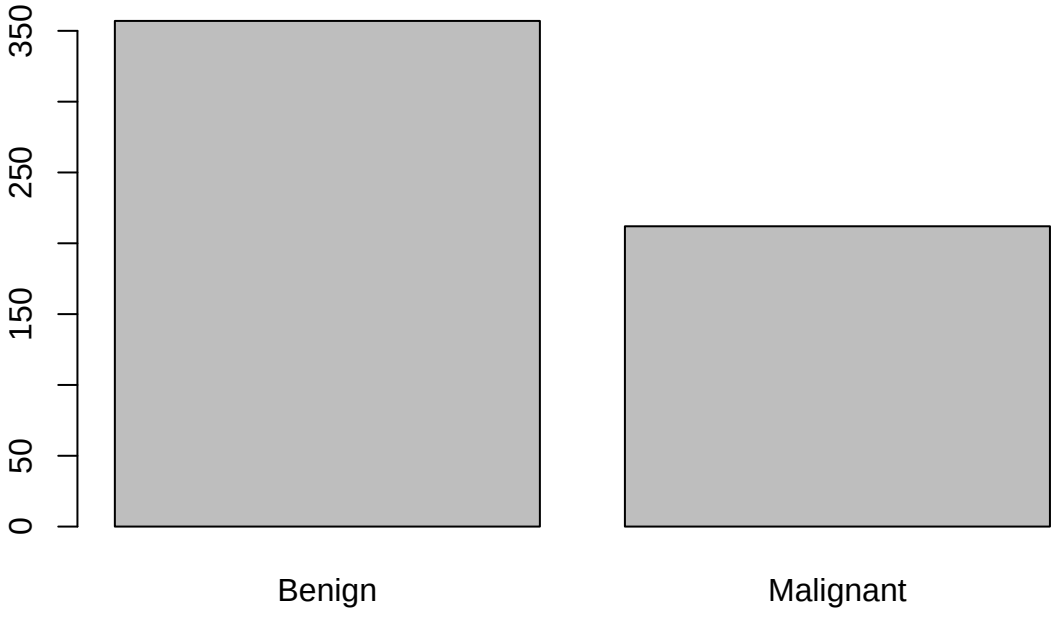
We import and inspect the data

```
## 'data.frame':    569 obs. of  32 variables:
##  $ id                      : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844
##  $ diagnosis               : chr  "M" "M" "M" "M" ...
##  $ radius_mean             : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean            : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean          : num  122.8 132.9 130 77.6 135.1 ...
##  $ area_mean               : num  1001 1326 1203 386 1297 ...
##  $ smoothness_mean         : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness_mean        : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity_mean          : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave.points_mean     : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
```

```
##  $ symmetry_mean          : num  0.242 0.181 0.207 0.26 0.181 ...
##  $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius_se              : num  1.095 0.543 0.746 0.496 0.757 ...
##  $ texture_se             : num  0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter_se           : num  8.59 3.4 4.58 3.44 5.44 ...
##  $ area_se                : num  153.4 74.1 94 27.2 94.4 ...
##  $ smoothness_se          : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness_se         : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity_se           : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave.points_se      : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry_se            : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal_dimension_se   : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ radius_worst           : num  25.4 25 23.6 14.9 22.5 ...
##  $ texture_worst          : num  17.3 23.4 25.5 26.5 16.7 ...
##  $ perimeter_worst        : num  184.6 158.8 152.5 98.9 152.2 ...
##  $ area_worst             : num  2019 1956 1709 568 1575 ...
##  $ smoothness_worst       : num  0.162 0.124 0.144 0.21 0.137 ...
##  $ compactness_worst      : num  0.666 0.187 0.424 0.866 0.205 ...
##  $ concavity_worst        : num  0.712 0.242 0.45 0.687 0.4 ...
##  $ concave.points_worst   : num  0.265 0.186 0.243 0.258 0.163 ...
##  $ symmetry_worst         : num  0.46 0.275 0.361 0.664 0.236 ...
##  $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
```

As we can see the first feature is a simple identifier which we can drop since it does not provide useful information to us. The next variable, `diagnosis`, is what we hope to predict. We first encode is into numerical values 1 and 2 and then we get an idea of the diagnostic distribution.

```
##
##    Benign Malignant
## 0.6274165 0.3725835
```




```
# Creating train-
```

ing and test datasets and training a model on them

Before splitting the data we will scale it since K-NN is heavily affected by the measurement scale of the input features (we are omitting the diagnosis since that cannot be scaled)

```
dataset[-1] = scale(dataset[-1])
```

We can now split the data

```
set.seed(123)
split = sample.split(dataset$diagnosis, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

We are going to start with k = 5 (the default value) and we will later come back and check whether different values give better results

```
y_pred = knn(train = training_set[, -1],
             test = test_set[, -1],
             cl = training_set[, 1],
             k = 5)
```

Evaluating model performance

It is time to evaluate our model's performance. We'll start with making a simple confusion matrix for the test set

```
cm = table(test_set[, 1], y_pred)
cm
```

```
##            y_pred
##             Benign Malignant
##   Benign        88         1
##   Malignant      5        48
```

Our model appears very accurate. Let's calculate the Odds Ratio

```
OR = (cm[1] * cm[4])/(cm[2] * cm[3])
OR
```

```
## [1] 844.8
```

Even though the Odds Ratio is really high we notice that we have more false negatives (FN) than false positives (FP) which although in terms for accuracy does not make a difference, it can be dangerous in a real life scenario to diagnose someone as Benign when in fact they are Malignant.

Imrpoving model performance

We will first try to improve our result by changing our scaling method. Earlier we used z-score standardization, we will now try normalization.

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

dataset[-1] = as.data.frame(lapply(dataset[-1], normalize))
```

We will now repeat the previous steps and see if that improves our accuracy

```
##            y_pred2
##             Benign Malignant
##   Benign        87         2
##   Malignant      4        49
```

```
## [1] 532.875
```

We see that our Odds Ratio decreased however, so did our False Negatives and in this particular case, that is more important than a simple accuracy measurement.

We can also, as mentioned earlier, test different values of K and see how that affects our model performance

```
tmp.res.OR = integer()
tmp.res.FN = integer()
tmp.res.FP = integer()

set.seed(123)
for (i in c(1:25)) {
  tmp.pred = knn(train = training_set2[, -1],
                 test = test_set2[, -1],
                 cl = training_set2[, 1],
                 k = i)

  tmp.cm = table(test_set2[, 1], tmp.pred)

  tmp.OR = (tmp.cm[1] * tmp.cm[4])/(tmp.cm[2] * tmp.cm[3])

  tmp.res.OR[i] = tmp.OR
  tmp.res.FN[i] = tmp.cm[2]
  tmp.res.FP[i] = tmp.cm[3]
}

results = data.frame(tmp.res.OR, tmp.res.FN, tmp.res.FP)
colnames(results) = c('OR', 'FN', 'FP')
results = results[order(results$OR, decreasing = T),]
results
```

```
##           OR FN FP
## 22 1466.6667  3  1
## 8  1078.0000  4  1
## 12 1078.0000  4  1
## 13 1078.0000  4  1
## 15 1078.0000  4  1
## 16 1078.0000  4  1
## 20 1078.0000  4  1
## 21 1078.0000  4  1
## 23 1078.0000  4  1
## 24 1078.0000  4  1
## 25 1078.0000  4  1
## 1   844.8000  5  1
## 14  844.8000  5  1
## 17  844.8000  5  1
## 18  844.8000  5  1
## 19  844.8000  5  1
## 3   532.8750  4  2
## 4   532.8750  4  2
## 5   532.8750  4  2
## 7   532.8750  4  2
## 9   532.8750  4  2
## 10  532.8750  4  2
## 11  532.8750  4  2
```

```
## 6   351.1667  4  3
## 2   275.2000  5  3
```

Examining the results we observe that the the iteration with k = 22 yielded not only the best Odds Ratio but also the least amount of False Negatives. While this does seem like the best iteration on paper, because of our low sample size this could be the result of overfitting. The next best result in terms of both OR and FN is held by various values of K from which I'll be picking K = 8 as it is the smallest and therefore less likely to overfit.

Conclusion

We see that K-NN alrogorithm can be very accurate in modeling this data set. Although the classifier was never perfect the 8NN approach was able to avoid some of the False Negatives that the default 5NN had and as a result, increase the Odds Ratio. It is important to remember however that the size of this dataset is fairly limiting and a much larger sample size would be needed to obtain concrete and reliable results concrete