K-NN

Laura Sudupe Medinilla

14 de marzo, 2021

Contents

Introduction

In this markdown we are going to learn about classification using k-NN. Unlike many classification algorithms, k-NN does not do any learning. It simply stores the training data verbatim. Unlabeled test examples are then matched to the most similar records in the training set usin a distance function, and the unlabeled example is assignaed the label of its neighbors.

The strengths and weaknesses of this algorithm are as follows:

strengths	weaknesses
• Simple and effective	• Does not produce a model, limiting the ability to understand how the features are related to the class
• Makes no assumptions about the underlying data distribution	• Requires selection of an appropriate k
• Fast training phase	• Slow classification phase • Nominal features and missing data require additional processing

Step 1

Collect the data

```
# Input / Output variables
# Tuning parameters
# ...
file1 <- "usedcars.csv"
```

Step 2

Exploring and preparing the data

Import the CSV data file to the `cancer_mama` dataframe.

```
cancer_mama <- read.csv("wisc_bc_data.csv", stringsAsFactors = FALSE)
```

Using the `str(cancer_mama)`command, we can see the data structure.

```
str(cancer_mama)
```

```
## 'data.frame':    569 obs. of  32 variables:
##  $ id               : int  87139402 8910251 905520 868871 9012568 906539 925291 87880 862989 89827 .
##  $ diagnosis        : chr  "B" "B" "B" "B" ...
##  $ 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 ...
```

We can see the stdy has 32 examples and 569 features. The first feature `id` is a unique identifier for each patient in the data, we will exclude it from the model.

```
cancer_mama <- cancer_mama[-1]
```

The variable `diagnosis` is going to be our label, the oucome we hope to predict. This feature indicates whether the exmaple is from a benign or malignant mass. With the `table()` output we can see that 357 are benign while 212 are malignant.

```
table(cancer_mama$diagnosis)
```

```
##
##   B   M
## 357 212
```

Many R machine learning classifiers require the target feature is coded as a factor, so we will recode `diagnosis` feature.

```
cancer_mama$diagnosis <- factor(cancer_mama$diagnosis, levels = c("B", "M"),
                                labels = c("Benign", "Malignant"))
```

Let´s check the `Benign` and `Malignant` percentages with `prop.table()`

```
round(prop.table(table(cancer_mama$diagnosi)) * 100, digits=1)
```

```
##
##    Benign Malignant
##      62.7      37.3
```

All the remaining features are numeric, they consist of three different measurements of ten characteristics. We will take a closser look of three of these features

```
summary(cancer_mama[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
```

The distance calculation for k-NN is heavily dependent upon the measurement sclae of the input features. Since `smoothness_mean` ranges from 0.05 to 0.16 and `area_mean` ranges from 143.5 to 2501, the impact of area is going to be much larger than smoothness in the distance calculation. This cpuld potentially cause problems for our classifier, so let´s apply normalization to rescale the features to a standard range of values.

Transformation - normalizing numeric data

To normalize these features, we need to create a `normalize()` function.

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

The `lapply()` function take a list and applies a specified function to each list element. As a data frame is a list of equal-length vectors, we can use `lapply()` to apply ´normalize()´ to each feature in the data frame. The final step is to convert the list returned by `lapply()` to a data frame, using the`as.data.frame()` function

```
cancer_mama_n <-as.data.frame(lapply(cancer_mama[2:31], normalize))
```

Check the transformation was applied correctly

```
summary(cancer_mama_n[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
```

Data preparation - creating training and test datasets

We can simulate unkown label data by dividing our data 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. We will use the first 469 records for the training dataset and the remaining 100 to simulate new patients.We can do these because the data is already randomly ordered.

```
cancer_train <- cancer_mama_n[1:469,]
cancer_test <- cancer_mama_n[470:569,]
```

The next step is to exclude the target variable `diagnosis`. For training the k–NN model, we will need to store these class labels in factor vectors, split between the training and test datasets

```
cancer_train_labels <- cancer_mama[1:469, 1]
cancer_test_lables <- cancer_mama[470:569, 1]
```

With these code we create the vectors `cancer_train_labels` and `cancer_test_labels`. We will use these in the next steps of training and evaluating our classifier.

Step 3

Training a model on the data

For the k-NN algorithm, the training phase actually involves no model building; the process of training 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.

```
#install.packages("class")
library("class")
```

The `knn()` function in the ´r 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 Eucledian distance, where k is a user-specified number. The test instance is classified by taking a "vote" among the k-Neares Neighbors- specifically, this involves assigning the class of the majority of the k neighbors.

```
cancer_test_pred <- knn(train = cancer_train, test = cancer_test,
                        cl = cancer_train_labels, k = 21)
```

The `knn()` function returns a factor vector of predicted labels for each of the examples in the test dataset, which we hace assigned to `cancer_test_pred`.

Step 4

Evaluating model performance

We have to evaluate how well the predicted classes in the `cancer_test_pred` vector match up with the known values in the `cancer_test_labels` vector. From these we can use the `CrossTable()` function.

```
#install.packages("gmodels")
library("gmodels")
```

We can create a cross tabulation indicating the agreement between the two vectors. Specifiying will remove the unnecessary chi-square values from the output

```
CrossTable(x=cancer_test_lables, y=cancer_test_pred, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  100
##
##
##                     | cancer_test_pred
## cancer_test_lables |    Benign | Malignant | Row Total |
## -------------------|-----------|-----------|-----------|
##              Benign |        61 |         0 |        61 |
##                     |     1.000 |     0.000 |     0.610 |
##                     |     0.968 |     0.000 |           |
##                     |     0.610 |     0.000 |           |
## -------------------|-----------|-----------|-----------|
##           Malignant |         2 |        37 |        39 |
##                     |     0.051 |     0.949 |     0.390 |
##                     |     0.032 |     1.000 |           |
##                     |     0.020 |     0.370 |           |
## -------------------|-----------|-----------|-----------|
##       Column Total |        63 |        37 |       100 |
##                     |     0.630 |     0.370 |           |
## -------------------|-----------|-----------|-----------|
##
##
```

The cell percentages in the table indicate the proportion of values that fall into four categories.

The tip-left cell indicates the **true negative** results. These 61 of 100 values are cases where the mass was benign and the k-NN algorithm correctly identified it such.

The bottom-right cell indicates the **true positive** results, where the classifier and the clinically determined label agree that mass is malignant. A total of 37 of 100 predictions were true positives.

The cells falling on the other diagonal contain counts of examples wher-left cell are **false negative** results; in this case, the predicted value was benign, but the tumor was actually malignant.

The top-right cell would contain the **false positive** results if there were any. Model classifies a mass as malignant, but in reality it was benign.

A 98 percent accuracy seems very good, 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

The z-score standardized values have no predefined minimun and maximum, extreme values are not compressed towards the center. One might suspect that with a malignant tumor, we might see some very extreme outliers as the tumors grow uncontrollably. It might, therefore, be reasonable to allow the outliers to be weighted more heavily in the distance calculation.

Let´s see whether z-score standardization can improve our predictive accuracy.

To standardize a vector, we can use the `scale()` function, which rescales values using the z-score standardization. The `scale()` function offers the additional benefit that it can be applied directly to a data frame, so we can avoid the use of the `lapply()` function.

```
cancer_z <- as.data.frame(scale(cancer_mama[-1]))
```

This command rescales all the features, with the exception of `diagnosis` and stores the result as the `cancer_z` dataframe.

Check the transformation

```
summary(cancer_z$area_mean)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.4532 -0.6666 -0.2949  0.0000  0.3632  5.2459
```

The mean of a z-score standardized variable should always be zero, and the range should ne fairly compact. A z-score greater than 3 or less than -3 indicates an extremely rare value. With this in mind, the trasformation seems to have worked.

As we had done earlier, we need to divide the data into training and test sets, and the classify the test instances using the `knn()` function. We´ll then compare the predicted labels to the actual labels using `CrossTable()`

```
cancer_z_train <- cancer_z[1:469, ]
cancer_z_test <- cancer_z[470:569, ]
```

```
cancer_z_test_pred <- knn(train = cancer_z_train, test = cancer_z_test,
                          cl= cancer_train_labels, k = 21)
```

```
CrossTable(x= cancer_test_lables, y=cancer_z_test_pred,
           prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
```

```
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  100
##
##
##                     | cancer_z_test_pred
## cancer_test_lables |    Benign | Malignant | Row Total |
## -------------------|-----------|-----------|-----------|
##              Benign |        61 |         0 |        61 |
##                     |     1.000 |     0.000 |     0.610 |
##                     |     0.924 |     0.000 |           |
##                     |     0.610 |     0.000 |           |
## -------------------|-----------|-----------|-----------|
##           Malignant |         5 |        34 |        39 |
##                     |     0.128 |     0.872 |     0.390 |
##                     |     0.076 |     1.000 |           |
##                     |     0.050 |     0.340 |           |
## -------------------|-----------|-----------|-----------|
##       Column Total |        66 |        34 |       100 |
##                     |     0.660 |     0.340 |           |
## -------------------|-----------|-----------|-----------|
##
##
```

the results of the new transformation show a slight decline in accuracy. The instances where we had correctly classified 98 percent of examples previously, we classified only 95 percent correctly this time. Also, we have more **false negative** so we didn´t better at classifying the dangerous false negatives.

Testing alternative values of k

We are going to examine the performance of various k values. Using the normalized training and test datasets, the same 100 records were classified using several different k values.

	k value	False negatives	False positives	Percent classified incorrectly
A	1	1	3	4 percent
B	5	2	0	2 percent
C	11	3	0	3 percent
D	15	3	0	3 perent
E	21	2	0	2 percent
F	27	4	0	4 percent

The classifier is never perfect, the 1-NN approach was able to avoid some of the false negatives at the expense of adding false positives. It is important to keep in mind, that it would be unwise to tailor our approach too closely to or test data; after all, a different set of 100 patient records is likely to be somewhat different from those used to measure our performance.