

# Dynamic report k-NN

*Rubén Sánchez Fernández*

*6 de octubre de 2018*

## Contents

<b>Introduction to k-NN algorithm (with code)</b>	<b>1</b>
Introduction . . . . .	1
How it works? . . . . .	1
Strengths and weakness table . . . . .	1
Measuring distance . . . . .	2
Choosing the right k . . . . .	2
Preparing data for k-NN . . . . .	2
Implementing k-NN with R . . . . .	2
<b>References</b>	<b>7</b>

## Introduction to k-NN algorithm (with code)

### Introduction

This dynamic report is created as an exercise for the Machine Learning course from Bioinformatics and Biostatistics Msc.

This report aims to introduce one of the simplest machine learning algorithms, the k-Nearest Neighbors. k-NN is a supervised, non-parametric method used both for classification and regression. It's simplicity and effectiveness makes it one of the widest used ML algorithms.

### How it works?

K-NN method is based on distance measure. Each unlabeled point is classified according to which class has the highest frequency from the  $k$  nearest points. When performing regression, the output is the mean or median from the  $k$  nearest points.

### Strengths and weakness table

The following table is extracted from (Lantz 2015).

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

## Measuring distance

As previously mentioned, k-NN measures ‘similarity’ by calculating the distance between points. There are several distance measures that can be implemented with k-NN, being **Euclidean distance** the more popular.

Euclidean distance is calculated following the next equation:

$$dist(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2}$$

Where  $p$  and  $q$  are the two samples and  $n$  is the feature. Therefore,  $p_1$  is the point of the sample  $p$  on the first feature, and  $p_n$  is the point of sample  $p$  on the last feature.

## Choosing the right k

When choosing what  $k$  we use it is important to know that low  $k$  in most cases could lead to overfitting meanwhile high  $k$  could lead to underfitting. Finding the balance is key to achieve the highest accuracy possible.

In practice, usually the best approach is to choose an evaluation method and evaluate the performance of the model across different  $k$  values.

## Preparing data for k-NN

We explained that k-NN is based on distance measure. Obviously, this measure is heavily dependent on what scale are the features. That’s why it is important to transform the data to a standard scale across all features.

Usually, the method of rescaling for k-NN is **min-max normalization** following the next equation:

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(x)}$$

Another common method of rescaling is the **z-score standarization**:

$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - \text{mean}(X)}{\text{StdDev}(X)}$$

## Implementing k-NN with R

Let’s create a k-NN model to perform classification.

```
#importing data
ds<-read.csv("wisc_bc_data.csv", stringsAsFactors = FALSE)

print(paste0("The dataset has ", nrow(ds), " examples and ", ncol(ds), " features"))

## [1] "The dataset has 569 examples and 32 features"
```

Let’s present an overview of the data.

```
#summary
summary(ds)
```

##	id	diagnosis	radius_mean	texture_mean
##	Min. :	8670	Length:569	Min. : 6.981
##	1st Qu.: 869218	Class :character	1st Qu.:11.700	Min. : 9.71
##	Median : 906024	Mode :character	Median :13.370	1st Qu.:16.17
##	Mean : 30371831		Mean :14.127	Median :18.84
##	3rd Qu.: 8813129		3rd Qu.:15.780	Mean :19.29
##	Max. :911320502		Max. :28.110	3rd Qu.:21.80
				Max. :39.28

```

## perimeter_mean      area_mean      smoothness_mean      compactness_mean
## Min.      : 43.79    Min.      : 143.5    Min.      :0.05263    Min.      :0.01938
## 1st Qu.: 75.17    1st Qu.: 420.3    1st Qu.:0.08637    1st Qu.:0.06492
## Median : 86.24    Median : 551.1    Median :0.09587    Median :0.09263
## Mean      : 91.97    Mean      : 654.9    Mean      :0.09636    Mean      :0.10434
## 3rd Qu.:104.10    3rd Qu.: 782.7    3rd Qu.:0.10530    3rd Qu.:0.13040
## Max.      :188.50    Max.      :2501.0    Max.      :0.16340    Max.      :0.34540
## concavity_mean      points_mean      symmetry_mean      dimension_mean
## Min.      :0.00000    Min.      :0.00000    Min.      :0.1060    Min.      :0.04996
## 1st Qu.:0.02956    1st Qu.:0.02031    1st Qu.:0.1619    1st Qu.:0.05770
## Median :0.06154    Median :0.03350    Median :0.1792    Median :0.06154
## Mean      :0.08880    Mean      :0.04892    Mean      :0.1812    Mean      :0.06280
## 3rd Qu.:0.13070    3rd Qu.:0.07400    3rd Qu.:0.1957    3rd Qu.:0.06612
## Max.      :0.42680    Max.      :0.20120    Max.      :0.3040    Max.      :0.09744
## radius_se      texture_se      perimeter_se      area_se
## Min.      :0.1115    Min.      :0.3602    Min.      : 0.757    Min.      : 6.802
## 1st Qu.:0.2324    1st Qu.:0.8339    1st Qu.: 1.606    1st Qu.: 17.850
## Median :0.3242    Median :1.1080    Median : 2.287    Median : 24.530
## Mean      :0.4052    Mean      :1.2169    Mean      : 2.866    Mean      : 40.337
## 3rd Qu.:0.4789    3rd Qu.:1.4740    3rd Qu.: 3.357    3rd Qu.: 45.190
## Max.      :2.8730    Max.      :4.8850    Max.      :21.980    Max.      :542.200
## smoothness_se      compactness_se      concavity_se
## Min.      :0.001713    Min.      :0.002252    Min.      :0.00000
## 1st Qu.:0.005169    1st Qu.:0.013080    1st Qu.:0.01509
## Median :0.006380    Median :0.020450    Median :0.02589
## Mean      :0.007041    Mean      :0.025478    Mean      :0.03189
## 3rd Qu.:0.008146    3rd Qu.:0.032450    3rd Qu.:0.04205
## Max.      :0.031130    Max.      :0.135400    Max.      :0.39600
## points_se      symmetry_se      dimension_se      radius_worst
## Min.      :0.000000    Min.      :0.007882    Min.      :0.0008948    Min.      : 7.93
## 1st Qu.:0.007638    1st Qu.:0.015160    1st Qu.:0.0022480    1st Qu.:13.01
## Median :0.010930    Median :0.018730    Median :0.0031870    Median :14.97
## Mean      :0.011796    Mean      :0.020542    Mean      :0.0037949    Mean      :16.27
## 3rd Qu.:0.014710    3rd Qu.:0.023480    3rd Qu.:0.0045580    3rd Qu.:18.79
## Max.      :0.052790    Max.      :0.078950    Max.      :0.0298400    Max.      :36.04
## texture_worst      perimeter_worst      area_worst      smoothness_worst
## Min.      :12.02    Min.      : 50.41    Min.      : 185.2    Min.      :0.07117
## 1st Qu.:21.08    1st Qu.: 84.11    1st Qu.: 515.3    1st Qu.:0.11660
## Median :25.41    Median : 97.66    Median : 686.5    Median :0.13130
## Mean      :25.68    Mean      :107.26    Mean      : 880.6    Mean      :0.13237
## 3rd Qu.:29.72    3rd Qu.:125.40    3rd Qu.:1084.0    3rd Qu.:0.14600
## Max.      :49.54    Max.      :251.20    Max.      :4254.0    Max.      :0.22260
## compactness_worst      concavity_worst      points_worst      symmetry_worst
## Min.      :0.02729    Min.      :0.0000    Min.      :0.00000    Min.      :0.1565
## 1st Qu.:0.14720    1st Qu.:0.1145    1st Qu.:0.06493    1st Qu.:0.2504
## Median :0.21190    Median :0.2267    Median :0.09993    Median :0.2822
## Mean      :0.25427    Mean      :0.2722    Mean      :0.11461    Mean      :0.2901
## 3rd Qu.:0.33910    3rd Qu.:0.3829    3rd Qu.:0.16140    3rd Qu.:0.3179
## Max.      :1.05800    Max.      :1.2520    Max.      :0.29100    Max.      :0.6638
## dimension_worst
## Min.      :0.05504
## 1st Qu.:0.07146
## Median :0.08004
## Mean      :0.08395

```

```
## 3rd Qu.:0.09208
## Max.    :0.20750
```

Let's also check the structure.

```
#internal structure
str(ds)
```

```
## '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 ...
```

Now, we will remove the first column. Usually, the first column represents an *id* or a sample number that doesn't provide useful information for the model. If the dataset doesn't have an *id* feature, skip this part.

```
ds<-ds[-1]
```

In R, is a requirement for a lot of Machine Learning algorithms to input the target feature as a factor.

```
#converting target feature to factor and changing labels
ds$diagnosis<-factor(ds$diagnosis, levels = c("B", "M"), labels = c("Benign", "Malignant"))
```

As mentioned before, it is important to have normalized scaled data to implement k-NN. To do so, we will create a function and apply it to our dataset.

```
#creating a function to normalize data
normalize<-function(x){
  return(( x-min(x)) / (max(x - min(x))))
```

```
}
```

```
#applying the function to our data
```

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

Finally, let's check that we have the data normalized.

```
summary(ds_n)
```

```
##      radius_mean      texture_mean      perimeter_mean      area_mean
## Min.      :0.0000    Min.      :0.0000    Min.      :0.0000    Min.      :0.0000
## 1st Qu.:0.2233    1st Qu.:0.2185    1st Qu.:0.2168    1st Qu.:0.1174
## Median :0.3024    Median :0.3088    Median :0.2933    Median :0.1729
## Mean      :0.3382    Mean      :0.3240    Mean      :0.3329    Mean      :0.2169
## 3rd Qu.:0.4164    3rd Qu.:0.4089    3rd Qu.:0.4168    3rd Qu.:0.2711
## Max.      :1.0000    Max.      :1.0000    Max.      :1.0000    Max.      :1.0000
## smoothness_mean compactness_mean concavity_mean      points_mean
## Min.      :0.0000    Min.      :0.0000    Min.      :0.00000    Min.      :0.0000
## 1st Qu.:0.3046    1st Qu.:0.1397    1st Qu.:0.06926    1st Qu.:0.1009
## Median :0.3904    Median :0.2247    Median :0.14419    Median :0.1665
## Mean      :0.3948    Mean      :0.2606    Mean      :0.20806    Mean      :0.2431
## 3rd Qu.:0.4755    3rd Qu.:0.3405    3rd Qu.:0.30623    3rd Qu.:0.3678
## Max.      :1.0000    Max.      :1.0000    Max.      :1.00000    Max.      :1.0000
## symmetry_mean  dimension_mean      radius_se      texture_se
## Min.      :0.0000    Min.      :0.0000    Min.      :0.00000    Min.      :0.0000
## 1st Qu.:0.2823    1st Qu.:0.1630    1st Qu.:0.04378    1st Qu.:0.1047
## Median :0.3697    Median :0.2439    Median :0.07702    Median :0.1653
## Mean      :0.3796    Mean      :0.2704    Mean      :0.10635    Mean      :0.1893
## 3rd Qu.:0.4530    3rd Qu.:0.3404    3rd Qu.:0.13304    3rd Qu.:0.2462
## Max.      :1.0000    Max.      :1.0000    Max.      :1.00000    Max.      :1.0000
## perimeter_se      area_se      smoothness_se      compactness_se
## Min.      :0.00000    Min.      :0.00000    Min.      :0.0000    Min.      :0.00000
## 1st Qu.:0.04000    1st Qu.:0.02064    1st Qu.:0.1175    1st Qu.:0.08132
## Median :0.07209    Median :0.03311    Median :0.1586    Median :0.13667
## Mean      :0.09938    Mean      :0.06264    Mean      :0.1811    Mean      :0.17444
## 3rd Qu.:0.12251    3rd Qu.:0.07170    3rd Qu.:0.2187    3rd Qu.:0.22680
## Max.      :1.00000    Max.      :1.00000    Max.      :1.0000    Max.      :1.00000
## concavity_se      points_se      symmetry_se      dimension_se
## Min.      :0.00000    Min.      :0.0000    Min.      :0.0000    Min.      :0.00000
## 1st Qu.:0.03811    1st Qu.:0.1447    1st Qu.:0.1024    1st Qu.:0.04675
## Median :0.06538    Median :0.2070    Median :0.1526    Median :0.07919
## Mean      :0.08054    Mean      :0.2235    Mean      :0.1781    Mean      :0.10019
## 3rd Qu.:0.10619    3rd Qu.:0.2787    3rd Qu.:0.2195    3rd Qu.:0.12656
## Max.      :1.00000    Max.      :1.0000    Max.      :1.0000    Max.      :1.00000
## radius_worst      texture_worst      perimeter_worst      area_worst
## Min.      :0.0000    Min.      :0.0000    Min.      :0.0000    Min.      :0.00000
## 1st Qu.:0.1807    1st Qu.:0.2415    1st Qu.:0.1678    1st Qu.:0.08113
## Median :0.2504    Median :0.3569    Median :0.2353    Median :0.12321
## Mean      :0.2967    Mean      :0.3640    Mean      :0.2831    Mean      :0.17091
## 3rd Qu.:0.3863    3rd Qu.:0.4717    3rd Qu.:0.3735    3rd Qu.:0.22090
## Max.      :1.0000    Max.      :1.0000    Max.      :1.0000    Max.      :1.00000
## smoothness_worst compactness_worst concavity_worst      points_worst
## Min.      :0.0000    Min.      :0.0000    Min.      :0.00000    Min.      :0.0000
## 1st Qu.:0.3000    1st Qu.:0.1163    1st Qu.:0.09145    1st Qu.:0.2231
## Median :0.3971    Median :0.1791    Median :0.18107    Median :0.3434
```

```
## Mean :0.4041 Mean :0.2202 Mean :0.21740 Mean :0.3938
## 3rd Qu.:0.4942 3rd Qu.:0.3025 3rd Qu.:0.30583 3rd Qu.:0.5546
## Max. :1.0000 Max. :1.0000 Max. :1.00000 Max. :1.0000
## symmetry_worst dimension_worst
## Min. :0.0000 Min. :0.0000
## 1st Qu.:0.1851 1st Qu.:0.1077
## Median :0.2478 Median :0.1640
## Mean :0.2633 Mean :0.1896
## 3rd Qu.:0.3182 3rd Qu.:0.2429
## Max. :1.0000 Max. :1.0000
```

Now that we have our data ready, it is time to create the classification model. First, we will create the training and test sets. If the project already has a training and a test set, skip this part.

```
#training set
ds_train<-ds_n[1:469,]
#test set
ds_test<-ds_n[470:569,]

#training labels
ds_train_labels<-ds[1:469,1]
#test labels
ds_test_labels<-ds[470:569,1]
```

Next step is the training phase. K-NN is what we call a *lazy learner*, training phase only consists in storing the input data in a structured format, which is already done.

Now, we can classify the test data using the *knn()* function from the *class* package.

```
library(class)
ds_test_pred<-knn(train=ds_train, test=ds_test, cl=ds_train_labels, k=21) #k=21
```

We obtained a vector with the predicted labels. We need to assess how true are these predictions and therefore, how accurate is the model. To do so, we can use the *CrossTable()* function from the *gmodels* package.

```
library(gmodels)
CrossTable(x = ds_test_labels, y = ds_test_pred, prop.chisq=FALSE)
```

```
##
##
##      Cell Contents
## |-----|
## |                      N |
## |          N / Row Total |
## |          N / Col Total |
## |          N / Table Total |
## |-----|
##
##
## Total Observations in Table:  100
##
##
##              | ds_test_pred
## ds_test_labels |      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 |          |
## -----|-----|-----|-----|
##
##
```

Depending on the results we obtained, we can try to modify our model to achieve a better performance.

The easiest way to modify the model is by trying a different  $k$ . We must have in mind though, the possibility of overfitting and underfitting.

Another way to modify the model is by rescaling the data using *z-score standarization* instead of standard normalization. To standarize the data we use the `scale()` function.

```
ds_z<-as.data.frame(scale(ds[-1]))
```

## References

Lantz, Brett. 2015. *Machine Learning with R*. Packt Publishing Ltd.