Best way to learn kNN Algorithm using R Programming

Introduction

This algorithm is a supervised learning algorithm, where the destination is known, but the path to the destination is not. Understanding nearest neighbors forms the quintessence of machine learning. Just like Regression, this algorithm is also easy to learn and apply.

What is kNN Algorithm?

Let's assume we have several groups of labeled samples. The items present in the groups are homogeneous in nature. Now, suppose we have an unlabeled example which needs to be classified into one of the several labeled groups. How do you do that? Unhesitatingly, using kNN Algorithm.

k nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. This algorithms segregates unlabeled data points into well defined groups.

How to select appropriate k value?

Choosing the number of nearest neighbors i.e. determining the value of k plays a significant role in determining the efficacy of the model. Thus, selection of k will determine how well the data can be utilized to generalize the results of the kNN algorithm. A large k value has benefits which include reducing the variance due to the noisy data; the side effect being developing a bias due to which the learner tends to ignore the smaller patterns which may have useful insights.

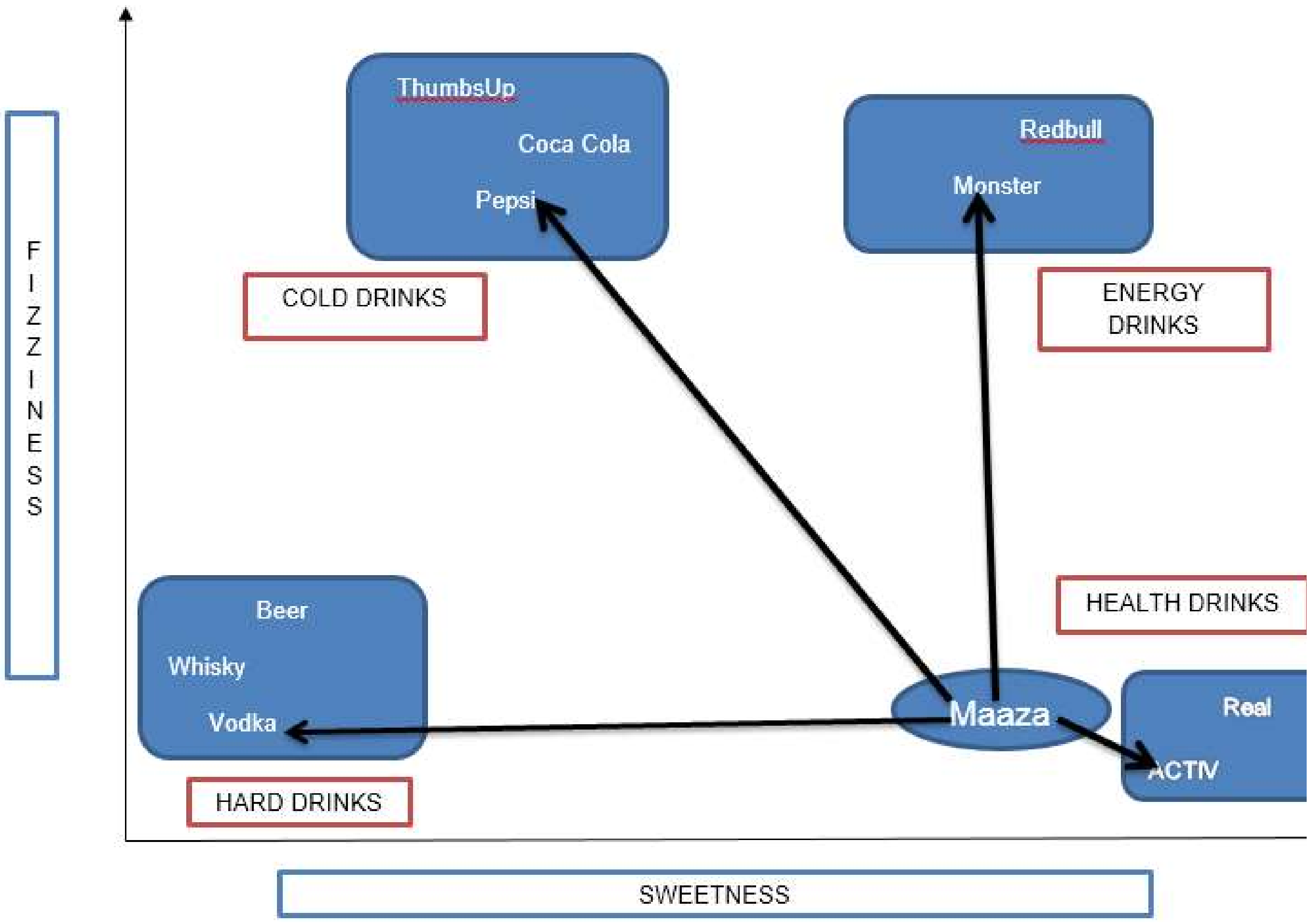
The following example will give you practical insight on selecting the appropriate k value.

Example of kNN Algorithm

Let's consider 10 'drinking items' which are rated on two parameters on a scale of 1 to 10. The two parameters are "sweetness" and "fizziness". This is more of a perception based rating and so may vary between individuals. I would be considering my ratings (which might differ) to take this illustration ahead. The ratings of few items look somewhat as:

Ingredient	Sweetness	Fizziness	Type of Drink
Monster	8	8	Energy booster
ACTIV	9	1	Health drink
Pepsi	4	8	Cold drink
Vodka	2	1	Hard drink

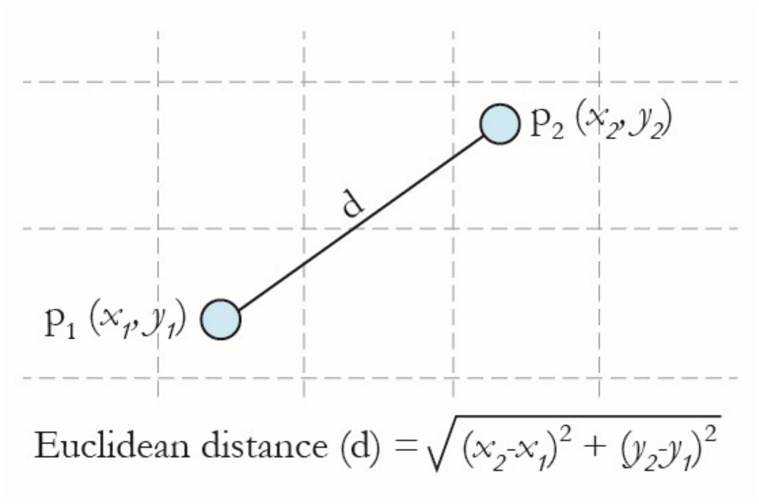
"Sweetness" determines the perception of the sugar content in the items. "Fizziness" ascertains the presence of bubbles in the drink due to the carbon dioxide content in the drink. Again, all these ratings used are based on personal perception and are strictly relative.



From the above figure, it is clear we have bucketed the 10 items into 4 groups namely, 'COLD DRINKS', 'ENERGY DRINKS', 'HEALTH DRINKS' and 'HARD DRINKS'. The question here is, to which group would 'Maaza' fall into? This will be determined by calculating distance.

Calculating Distance

Now, calculating distance between ‘Maaza’ and its nearest neighbors (‘ACTIV’, ‘Vodka’, ‘Pepsi’ and ‘Monster’) requires the usage of a distance formula, the most popular being Euclidean distance formula i.e. the shortest distance between the 2 points which may be obtained using a ruler.



Source: Gamesetmap

Using the co-ordinates of Maaza (8,2) and Vodka (2,1), the distance between ‘Maaza’ and ‘Vodka’ can be calculated as:

dist(Maaza,Vodka) = 6.08

Ingredient	Sweetness	Fizziness	Type of Drink	Distance to Maaza
Monster	7	8	Energy booster	6.08
ACTIV	9	1	Health drink	1.41
Pepsi	4	8	Cold drink	7.21
Vodka	2	1	Hard drink	6.08

Using Euclidean distance, we can calculate the distance of Maaza from each of its nearest neighbors. Distance between Maaza and ACTIV being the least, it may be inferred that Maaza is same as ACTIV in nature which in turn belongs to the group of drinks (Health Drinks).

If k=1, the algorithm considers the nearest neighbor to Maaza i.e, ACTIV; if k=3, the algorithm considers ‘3’ nearest neighbors to Maaza to compare the distances (ACTIV, Vodka, Monster) – ACTIV stands the nearest to Maaza.

kNN Algorithm – Pros and Cons

Pros: The algorithm is highly unbiased in nature and makes no prior assumption of the underlying data. Being simple and effective in nature, it is easy to implement and has gained good popularity.

Cons: Indeed it is simple but kNN algorithm has drawn a lot of flake for being extremely simple! If we take a deeper look, this doesn't create a model since there's no abstraction process involved. Yes, the training process is really fast as the data is stored verbatim (hence lazy learner) but the prediction time is pretty high with useful insights missing at times. Therefore, building this algorithm requires time to be invested in data preparation (especially treating the missing data and categorical features) to obtain a robust model.

Case Study: Detecting Prostate Cancer

Machine learning finds extensive usage in pharmaceutical industry especially in detection of oncogenic (cancer cells) growth. R finds application in machine learning to build models to predict the abnormal growth of cells thereby helping in detection of cancer and benefiting the health system.

Let's see the process of building this model using kNN algorithm in R Programming. Below you'll observe I've explained every line of code written to accomplish this task.

Step 1- Data collection

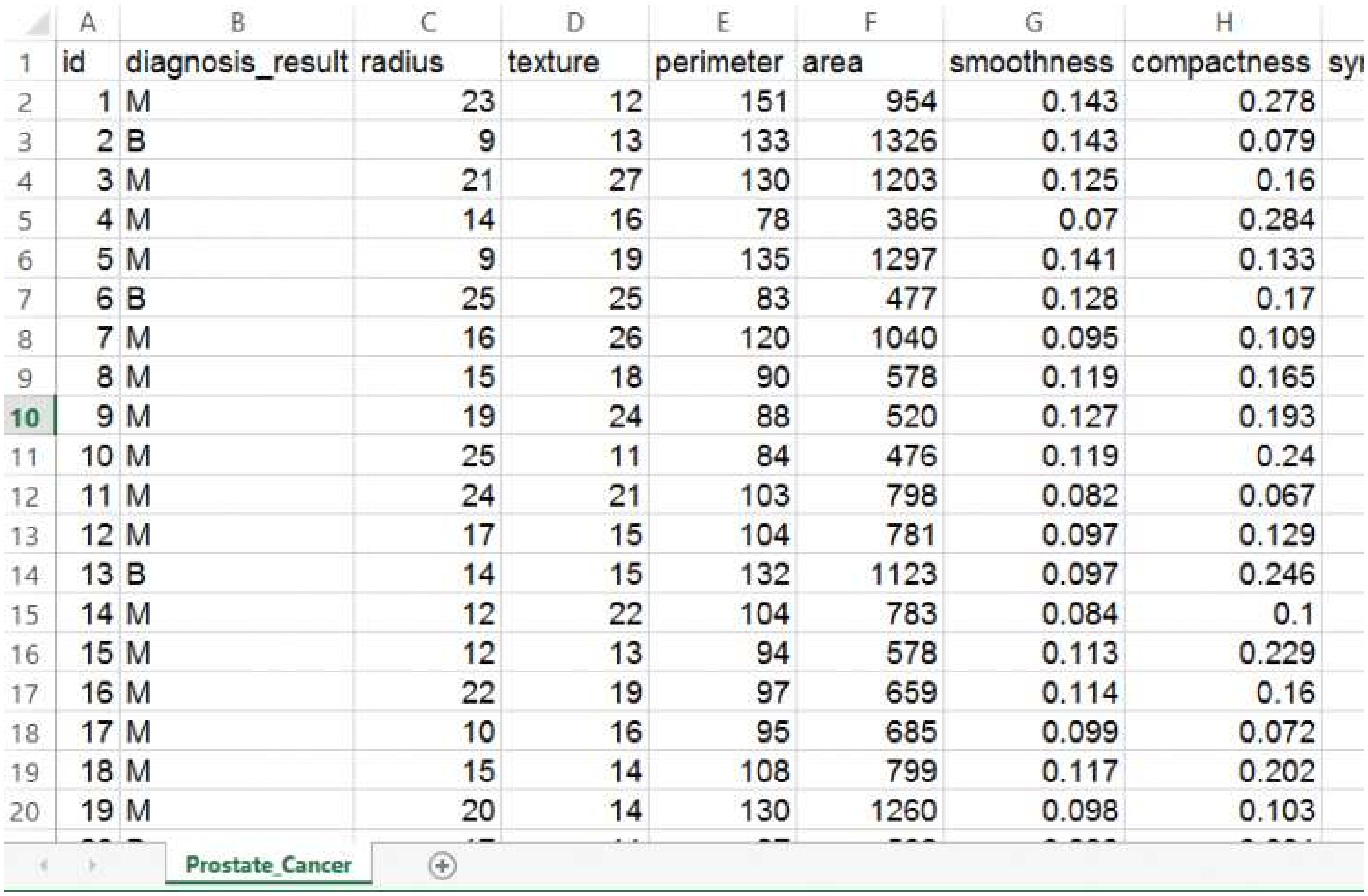
We will use a data set of 100 patients (created solely for the purpose of practice) to implement the knn algorithm and thereby interpreting results .The data set has been prepared keeping in mind the results which are generally obtained from DRE (Digital Rectal Exam). You can download the data set and practice these steps as I explain.

The data set consists of 100 observations and 10 variables (out of which 8 numeric variables and one categorical variable and is ID) which are as follows:

1. Radius
2. Texture
3. Perimeter
4. Area
5. Smoothness
6. Compactness
7. Symmetry
8. Fractal dimension

In real life, there are dozens of important parameters needed to measure the probability of cancerous growth but for simplicity purposes let's deal with 8 of them!

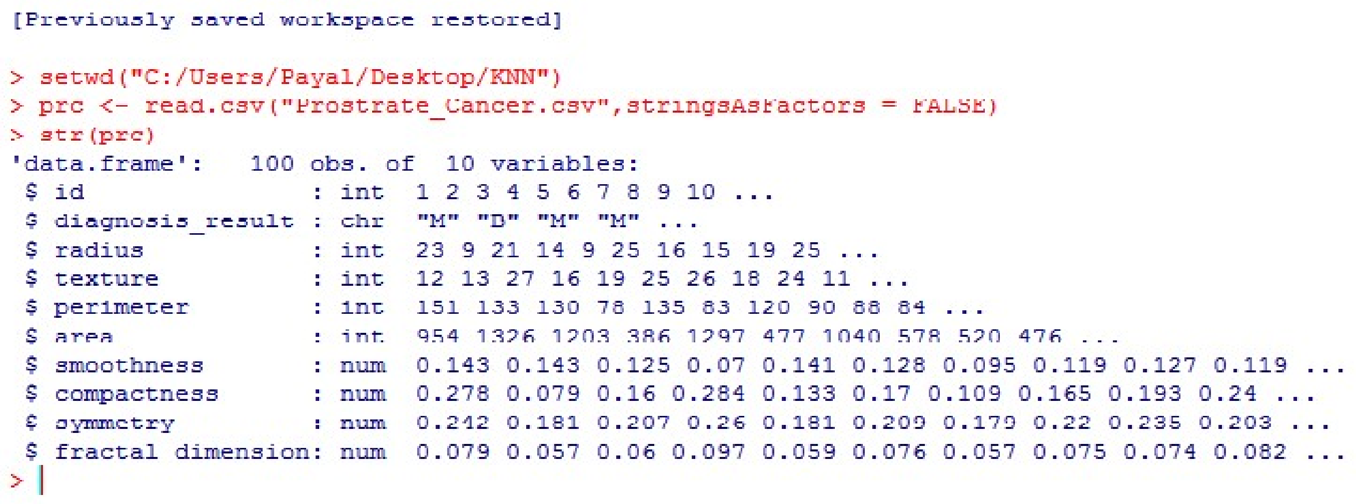
Here's how the data set looks like:



	A	B	C	D	E	F	G	H	
1	id	diagnosis_result	radius	texture	perimeter	area	smoothness	compactness	sy
2	1	M	23	12	151	954	0.143	0.278	
3	2	B	9	13	133	1326	0.143	0.079	
4	3	M	21	27	130	1203	0.125	0.16	
5	4	M	14	16	78	386	0.07	0.284	
6	5	M	9	19	135	1297	0.141	0.133	
7	6	B	25	25	83	477	0.128	0.17	
8	7	M	16	26	120	1040	0.095	0.109	
9	8	M	15	18	90	578	0.119	0.165	
10	9	M	19	24	88	520	0.127	0.193	
11	10	M	25	11	84	476	0.119	0.24	
12	11	M	24	21	103	798	0.082	0.067	
13	12	M	17	15	104	781	0.097	0.129	
14	13	B	14	15	132	1123	0.097	0.246	
15	14	M	12	22	104	783	0.084	0.1	
16	15	M	12	13	94	578	0.113	0.229	
17	16	M	22	19	97	659	0.114	0.16	
18	17	M	10	16	95	685	0.099	0.072	
19	18	M	15	14	108	799	0.117	0.202	
20	19	M	20	14	130	1260	0.098	0.103	

Prostate_Cancer

Step 2- Preparing and exploring the data

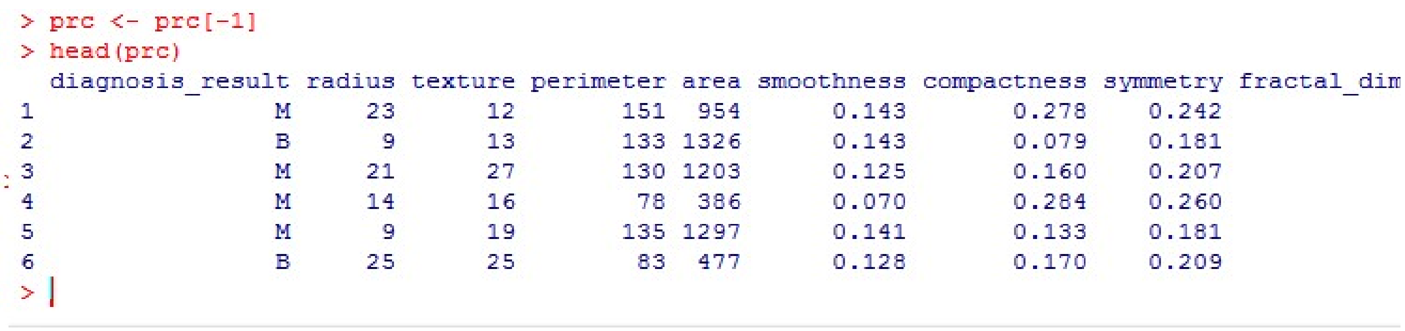


```
[Previously saved workspace restored]

> setwd("C:/Users/Payal/Desktop/KNN")
> prc <- read.csv("Prostrate_Cancer.csv",stringsAsFactors = FALSE)
> str(prc)
'data.frame':   100 obs. of  10 variables:
 $ id               : int  1 2 3 4 5 6 7 8 9 10 ...
 $ diagnosis_result : chr  "M" "B" "M" "M" ...
 $ radius           : int  23 9 21 14 9 25 16 15 19 25 ...
 $ texture          : int  12 13 27 16 19 25 26 18 24 11 ...
 $ perimeter        : int  151 133 130 78 135 83 120 90 88 84 ...
 $ area             : int  954 1326 1203 386 1297 477 1040 578 520 476 ...
 $ smoothness       : num  0.143 0.143 0.125 0.07 0.141 0.128 0.095 0.119 0.127 0.119 ...
 $ compactness      : num  0.278 0.079 0.16 0.284 0.133 0.17 0.109 0.165 0.193 0.24 ...
 $ symmetry         : num  0.242 0.181 0.207 0.26 0.181 0.209 0.179 0.22 0.235 0.203 ...
 $ fractal_dimension: num  0.079 0.057 0.06 0.097 0.059 0.076 0.057 0.075 0.074 0.082 ...
> |
```

Let's make sure that we understand every line of code before proceeding to the next stage:

```
setwd("C:/Users/Payal/Desktop/KNN")    #Using this command, we've imported the 'Prostate_Cancer
.csv' data file. This command is used to point to the folder containing the required file. Do k
eep in mind, that it's a common mistake to use "\" instead of "/" after the setwd command.

prc <- read.csv("Prostate_Cancer.csv",stringsAsFactors = FALSE)    #This command imports the re
quired data set and saves it to the prc data frame.

stringsAsFactors = FALSE   #This command helps to convert every character vector to a factor wh
erever it makes sense.

str(prc)    #We use this command to see whether the data is structured or not.
```

We find that the data is structured with 10 variables and 100 observations. If we observe the data set, the first variable 'id' is unique in nature and can be removed as it does not provide useful information.



```
> prc <- prc[-1]
> head(prc)
  diagnosis_result radius texture perimeter area smoothness compactness symmetry fractal_dim
1                M     23      12       151  954      0.143       0.278    0.242
2                B      9      13       133 1326      0.143       0.079    0.181
3                M     21      27       130 1203      0.125       0.160    0.207
4                M     14      16        78  386      0.070       0.284    0.260
5                M      9      19       135 1297      0.141       0.133    0.181
6                B     25      25        83  477      0.128       0.170    0.209
> 
```

```
prc <- prc[-1]  #removes the first variable(id) from the data set.
```

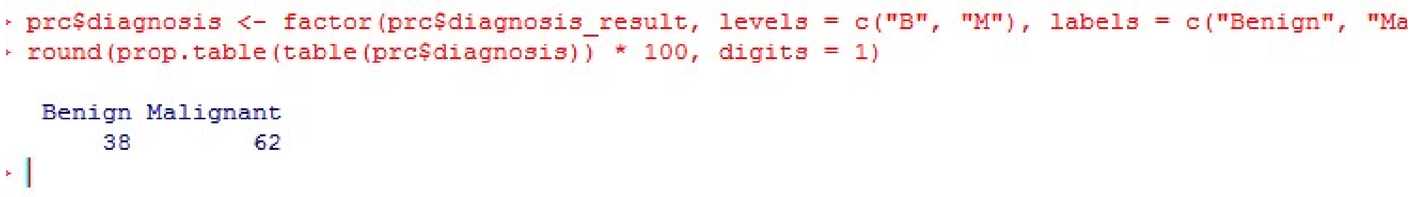
The data set contains patients who have been diagnosed with either Malignant (M) or Benign (B) cancer

```
table(prc$diagnosis_result)  # it helps us to get the numbers of patients
```

(The variable diagnosis_result is our target variable i.e. this variable will determine the results of the diagnosis based on the 8 numeric variables)

In case we wish to rename B as”Benign” and M as “Malignant” and see the results in the percentage form, we may write as:

```
prc$diagnosis <- factor(prc$diagnosis_result, levels = c("B", "M"), labels = c("Benign", "Malig

nant"))

round(prop.table(table(prc$diagnosis)) * 100, digits = 1)  # it gives the result in the percent

age form rounded of to 1 decimal place( and so it’s digits = 1)
```



```
> prc$diagnosis <- factor(prc$diagnosis_result, levels = c("B", "M"), labels = c("Benign", "Ma
> round(prop.table(table(prc$diagnosis)) * 100, digits = 1)

 Benign Malignant 
     38        62 
> |
```

Normalizing numeric data

This feature is of paramount importance since the scale used for the values for each variable might be different. The best practice is to normalize the data and transform all the values to a common scale.

```
normalize <- function(x) {

return ((x - min(x)) / (max(x) - min(x))) }
```

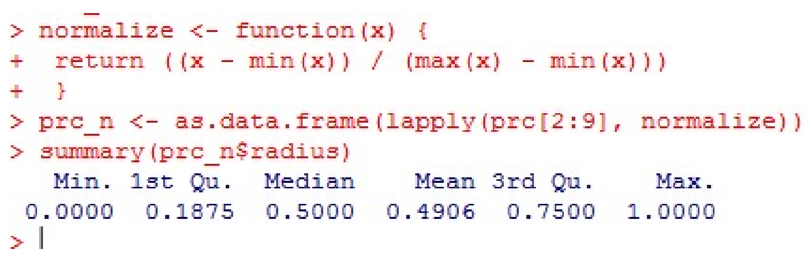
Once we run this code, we are required to normalize the numeric features in the data set. Instead of normalizing each of the 8 individual variables we use:

```
prc_n <- as.data.frame(lapply(prc[2:9], normalize))
```

The first variable in our data set (after removal of id) is ‘diagnosis_result’ which is not numeric in nature. So, we start from 2nd variable. The function lapply() applies normalize() to each feature in the data frame. The final result is stored to prc_n data frame using as.data.frame() function

Let’s check using the variable ‘radius’ whether the data has been normalized.

```
summary(prc_n$radius)
```



```
> normalize <- function(x) {
+   return ((x - min(x)) / (max(x) - min(x)))
+   }
> prc_n <- as.data.frame(lapply(prc[2:9], normalize))
> summary(prc_n$radius)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.0000  0.1875  0.5000  0.4906  0.7500  1.0000 
> 
```

The results show that the data has been normalized. Do try with the other variables such as perimeter, area etc.

Creating training and test data set

The kNN algorithm is applied to the training data set and the results are verified on the test data set.

For this, we would divide the data set into 2 portions in the ratio of 65: 35 (assumed) for the training and test data set respectively. You may use a different ratio altogether depending on the business requirement!

We shall divide the prc_n data frame into prc_train and prc_test data frames

```
prc_train <- prc_n[1:65,]


prc_test <- prc_n[66:100,]
```

A blank value in each of the above statements indicate that all rows and columns should be included.

Our target variable is ‘diagnosis_result’ which we have not included in our training and test data sets.

```
prc_train_labels <- prc[1:65, 1]
```

```
prc_test_labels <- prc[66:100, 1]   #This code takes the diagnosis factor in column 1 of the pr
c data frame and on turn creates prc_train_labels and prc_test_labels data frame.
```

Step 3 – Training a model on data

The knn () function needs to be used to train a model for which we need to install a package 'class'. The knn() function identifies the k-nearest neighbors using Euclidean distance where k is a user-specified number.

You need to type in the following commands to use knn()

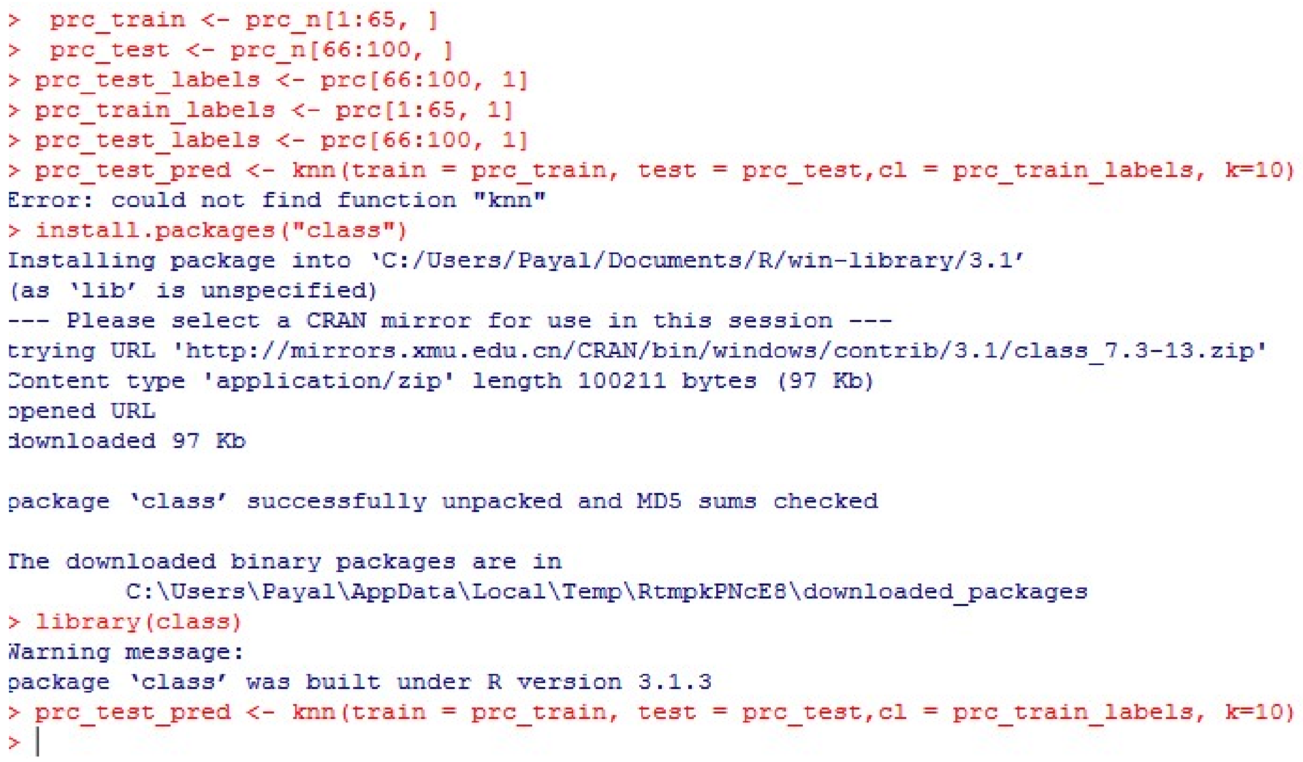
```
install.packages("class")

library(class)
```

Now we are ready to use the knn() function to classify test data

```
prc_test_pred <- knn(train = prc_train, test = prc_test,cl = prc_train_labels, k=10)
```

The value for k is generally chosen as the square root of the number of observations.

knn() returns a factor value of predicted labels for each of the examples in the test data set which is then assigned to the data frame prc_test_pred



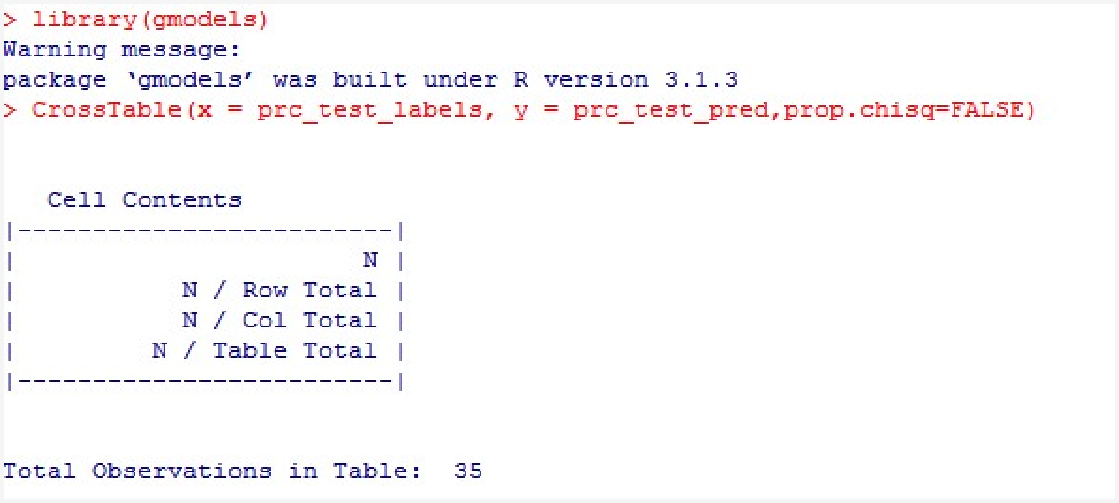
```
>  prc_train <- prc_n[1:65, ]
>  prc_test <- prc_n[66:100, ]
> prc_test_labels <- prc[66:100, 1]
> prc_train_labels <- prc[1:65, 1]
> prc_test_labels <- prc[66:100, 1]
> prc_test_pred <- knn(train = prc_train, test = prc_test,cl = prc_train_labels, k=10)
Error: could not find function "knn"
> install.packages("class")
Installing package into ‘C:/Users/Payal/Documents/R/win-library/3.1’
(as ‘lib’ is unspecified)
--- Please select a CRAN mirror for use in this session ---
trying URL 'http://mirrors.xmu.edu.cn/CRAN/bin/windows/contrib/3.1/class_7.3-13.zip'
Content type 'application/zip' length 100211 bytes (97 Kb)
opened URL
downloaded 97 Kb

package ‘class’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Payal\AppData\Local\Temp\RtmpkPNcE8\downloaded_packages
> library(class)
Warning message:
package ‘class’ was built under R version 3.1.3
> prc_test_pred <- knn(train = prc_train, test = prc_test,cl = prc_train_labels, k=10)
> |
```

Step 4 – Evaluate the model performance

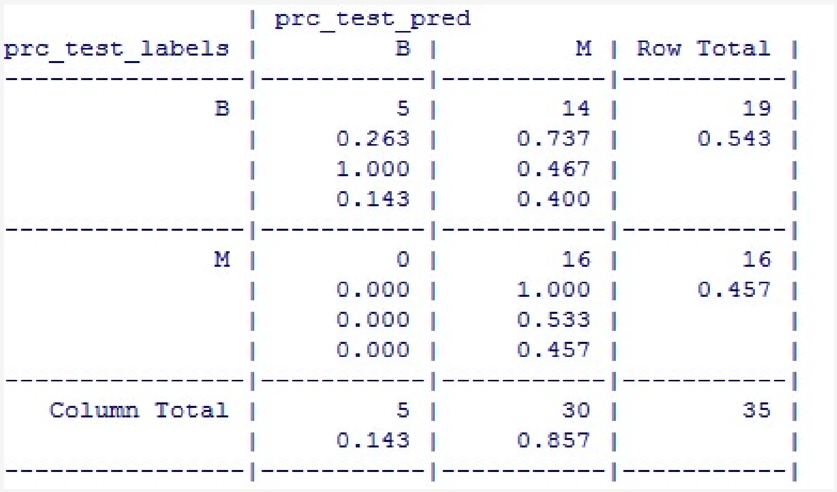
We have built the model but we also need to check the accuracy of the predicted values in prc_test_pred as to whether they match up with the known values in prc_test_labels. To ensure this, we need to use the CrossTable() function available in the package ‘gmodels’.

We can install it using:

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



```
> library(gmodels)
Warning message:
package ‘gmodels’ was built under R version 3.1.3
> CrossTable(x = prc_test_labels, y = prc_test_pred,prop.chisq=FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  35
```



```
                | prc_test_pred
prc_test_labels |         B |         M | Row Total |
----------------|-----------|-----------|-----------|
              B |         5 |        14 |        19 |
                |     0.263 |     0.737 |     0.543 |
                |     1.000 |     0.467 |           |
                |     0.143 |     0.400 |           |
----------------|-----------|-----------|-----------|
              M |         0 |        16 |        16 |
                |     0.000 |     1.000 |     0.457 |
                |     0.000 |     0.533 |           |
                |     0.000 |     0.457 |           |
----------------|-----------|-----------|-----------|
   Column Total |         5 |        30 |        35 |
                |     0.143 |     0.857 |           |
----------------|-----------|-----------|-----------|
```

The test data consisted of 35 observations. Out of which 5 cases have been accurately predicted (TN->True Negatives) as Benign (B) in nature which constitutes 14.3%. Also, 16 out of 35 observations were accurately predicted (TP-> True Positives) as Malignant (M) in nature which constitutes 45.7%. Thus a total of 16 out of 35 predictions where TP i.e, True Positive in nature.

There were no cases of False Negatives (FN) meaning no cases were recorded which actually are malignant in nature but got predicted as benign. The FN’s if any poses a potential threat for the same reason and the main focus to increase the accuracy of the model is to reduce FN’s.

There were 14 cases of False Positives (FP) meaning 14 cases were actually benign in nature but got predicted as malignant.

The total accuracy of the model is 60 %((TN+TP)/35) which shows that there may be chances to improve the model performance

Step 5 – Improve the performance of the model

This can be taken into account by repeating the steps 3 and 4 and by changing the k-value. Generally, it is the square root of the observations and in this case we took k=10 which is a perfect square root of 100.The k-value may be fluctuated in and around the value of 10 to check the increased accuracy of the model. Do try it out with values of your choice to increase the accuracy! Also remember, to keep the value of FN's as low as possible.

End Notes

For practice purpose, you can also solicit a dummy data set and execute the above mentioned codes to get a taste of the kNN algorithm. The results may not be that precise taking into account the nature of data but one thing for sure: the ability to understand the CrossTable() function and to interpret the results is what we should achieve.

In this article, I've explained kNN algorithm using an interesting example and a case study where I have demonstrated the process to apply kNN algorithm in building models.

Did you find this article helpful? Please share your opinions / thoughts in the comments section below.

This article was originally written by Payel Roy Choudhury, before Kunal did his experiment to set the tone. Payel has completed her MBA with specialization in Analytics from Narsee Monjee Institute of Management Studies (NMIMS) and is currently working with ZS Associates. She is looking forward to contribute regularly to Analytics Vidhya.