EXPERIMENT NO 7

NAME: Vishal Shashikant Salvi **UID: 2019230069**

Class: TE Comps **BATCH:C**

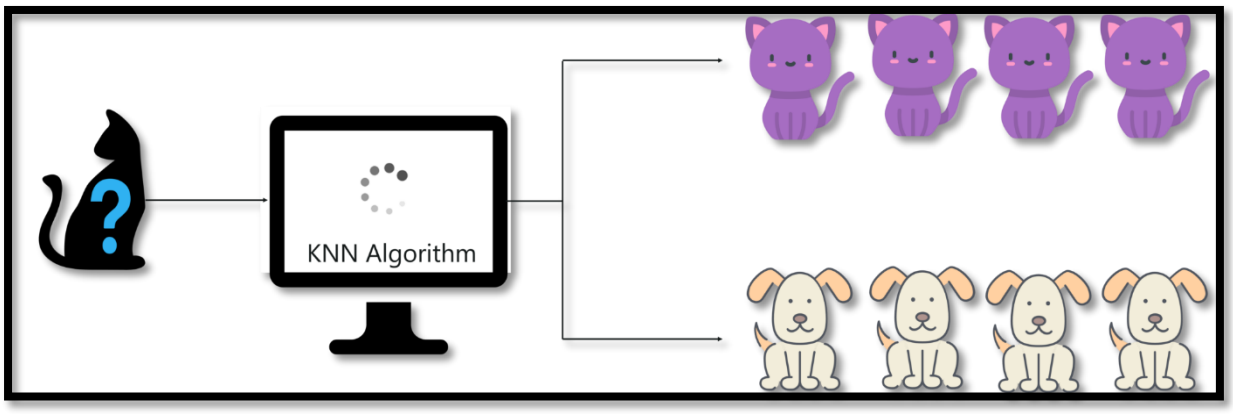
Aim: To implement KNN Algorithm of Classification using R-Programming.

Theory:

What Is KNN Algorithm?

KNN which stand for K Nearest Neighbor is a Supervised Machine Learning algorithm that classifies a new data point into the target class, depending on the features of its neighboring data points.

Let's try to understand the KNN algorithm with a simple example. Let's say we want a machine to distinguish between images of cats & dogs. To do this we must input a dataset of cat and dog images and we have to train our model to detect the animals based on certain features. For example, features such as pointy ears can be used to identify cats and similarly we can identify dogs based on their long ears.



What is KNN Algorithm? – KNN Algorithm In R

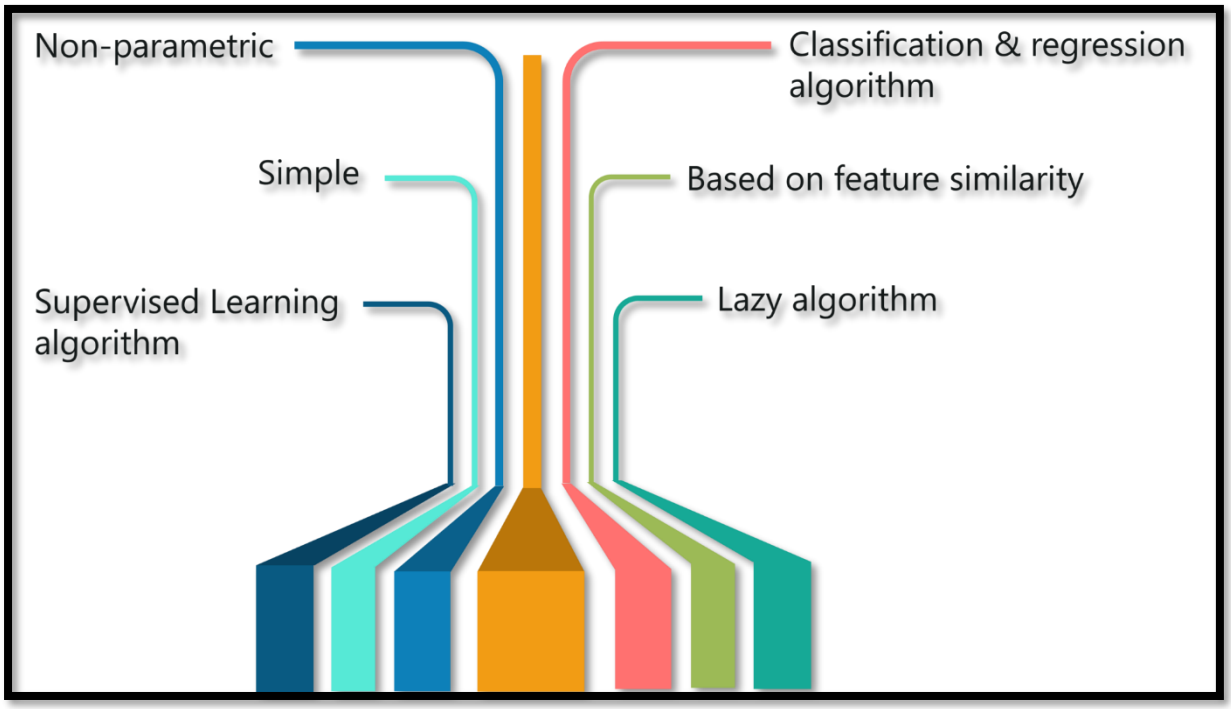
After studying the dataset during the training phase, when a new image is given to the model, the KNN algorithm will classify it into either cats or dogs depending on the similarity in their features. So if the new image has pointy ears, it will classify that image as a cat because it is similar to the cat images. In this manner, the KNN algorithm classifies data points based on how similar they are to their neighboring data points.

Now let's discuss the features of the KNN algorithm.

Features Of KNN Algorithm

The KNN algorithm has the following features:

- KNN is a Supervised Learning algorithm that uses labeled input data set to predict the output of the data points.
- It is one of the most simple Machine learning algorithms and it can be easily implemented for a varied set of problems.
- It is mainly based on feature similarity. KNN checks how similar a data point is to its neighbor and classifies the data point into the class it is most similar to.

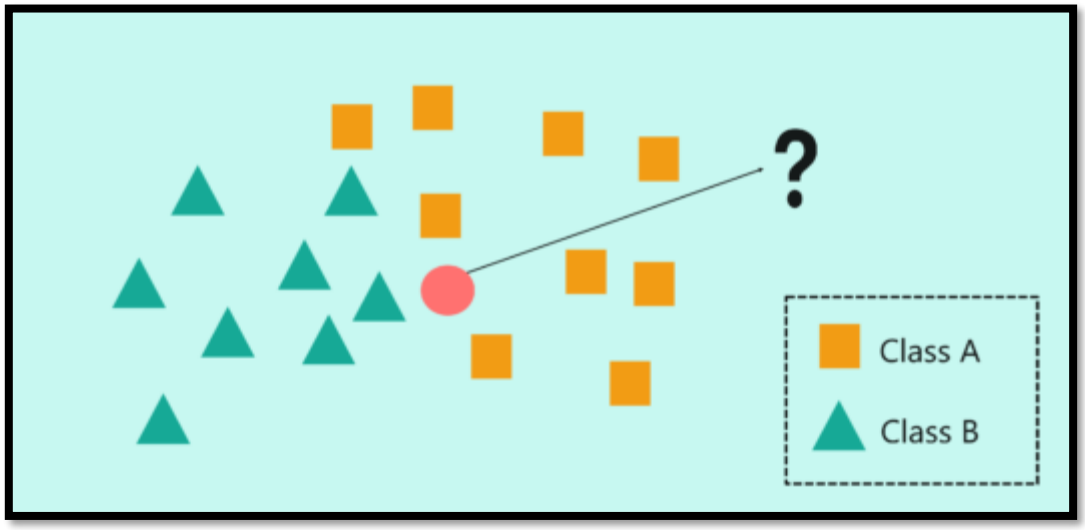


Features of KNN – KNN Algorithm In R

- Unlike most algorithms, KNN is a non-parametric model which means that it does not make any assumptions about the data set. This makes the algorithm more effective since it can handle realistic data.
- KNN is a lazy algorithm, this means that it memorizes the training data set instead of learning a discriminative function from the training data.
- KNN can be used for solving both classification and regression problems.

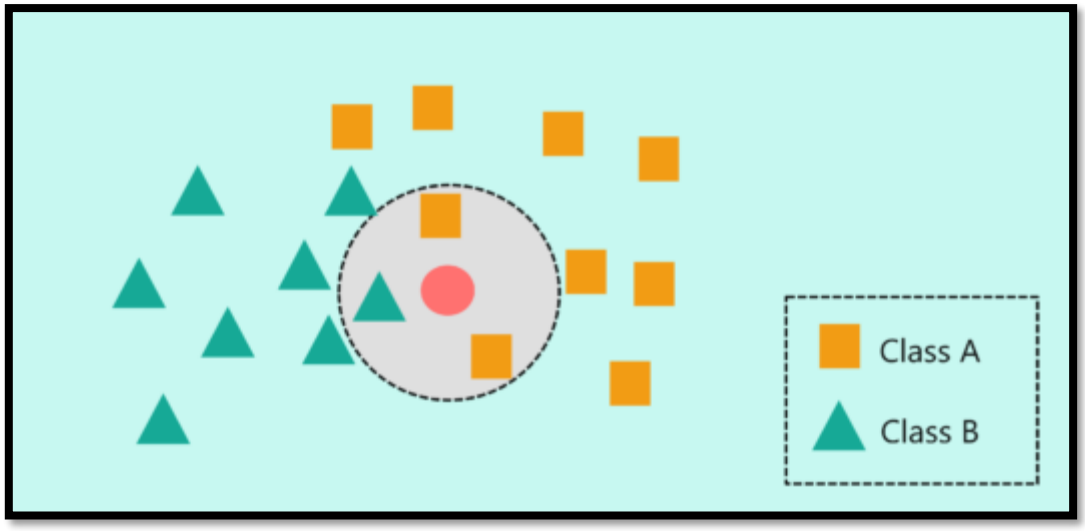
KNN Algorithm Example

To make you understand how KNN algorithm works, let's consider the following scenario:



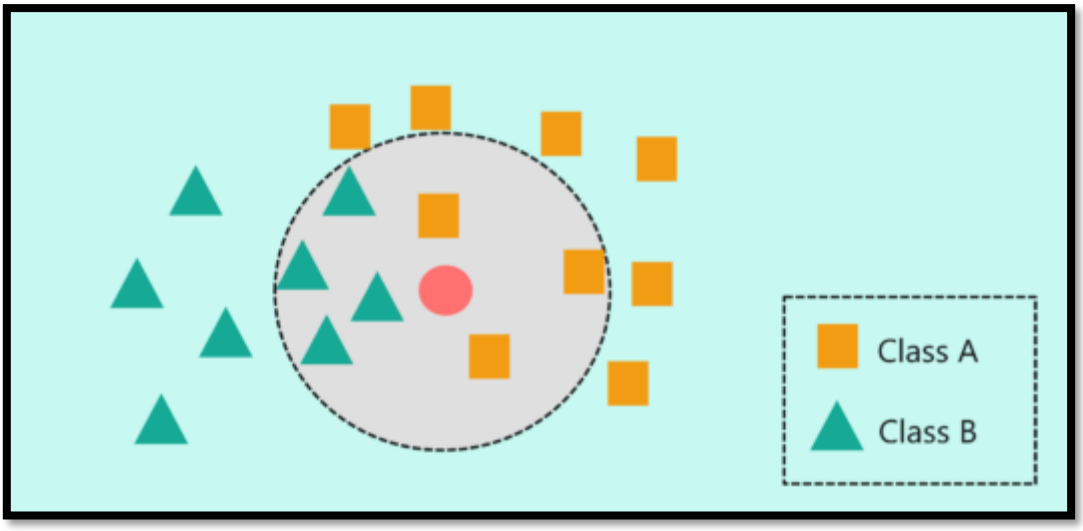
How does KNN Algorithm work? – KNN Algorithm In R

- In the above image, we have two classes of data, namely class A (squares) and Class B (triangles)
- The problem statement is to assign the new input data point to one of the two classes by using the KNN algorithm
- The first step in the KNN algorithm is to define the value of 'K'. But what does the 'K' in the KNN algorithm stand for?
- 'K' stands for the number of Nearest Neighbors and hence the name K Nearest Neighbors (KNN).



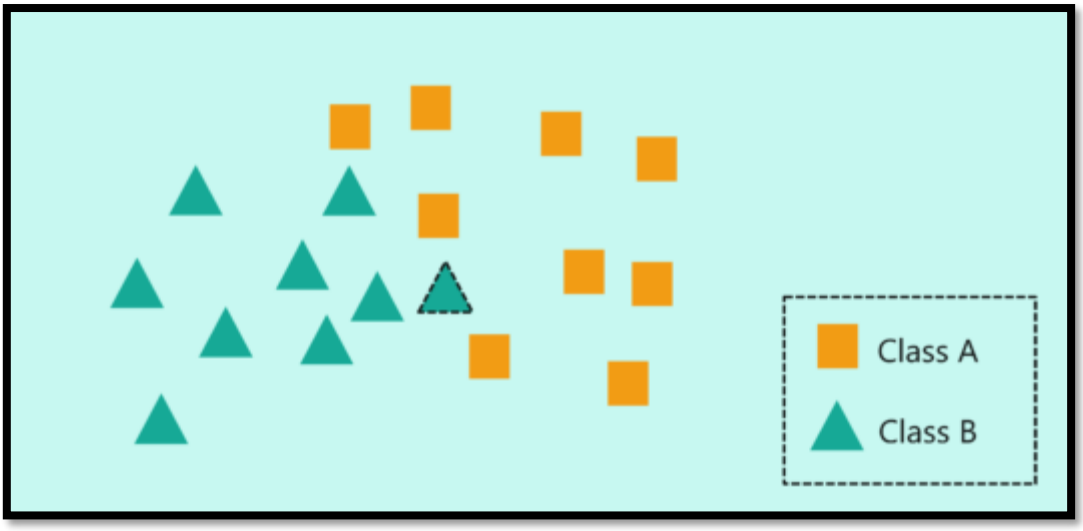
How does KNN Algorithm work? – KNN Algorithm In R

- In the above image, I've defined the value of 'K' as 3. This means that the algorithm will consider the three neighbors that are the closest to the new data point in order to decide the class of this new data point.
- The closeness between the data points is calculated by using measures such as Euclidean and Manhattan distance, which I'll be explaining below.
- At 'K' = 3, the neighbors include two squares and 1 triangle. So, if I were to classify the new data point based on 'K' = 3, then it would be assigned to Class A (squares).



How does KNN Algorithm work? – KNN Algorithm In R

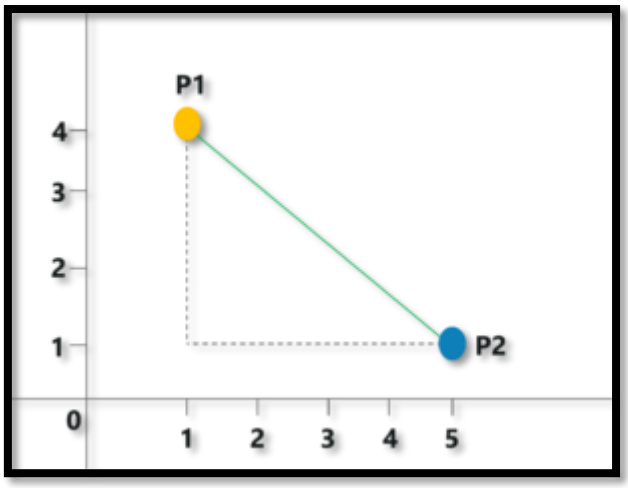
- But what if the 'K' value is set to 7? Here, I'm basically telling my algorithm to look for the seven nearest neighbors and classify the new data point into the class it is most similar to.
- At 'K' = 7, the neighbors include three squares and four triangles. So, if I were to classify the new data point based on 'K' = 7, then it would be assigned to Class B (triangles) since the majority of its neighbors were of class B.



How does KNN Algorithm work? – KNN Algorithm In R

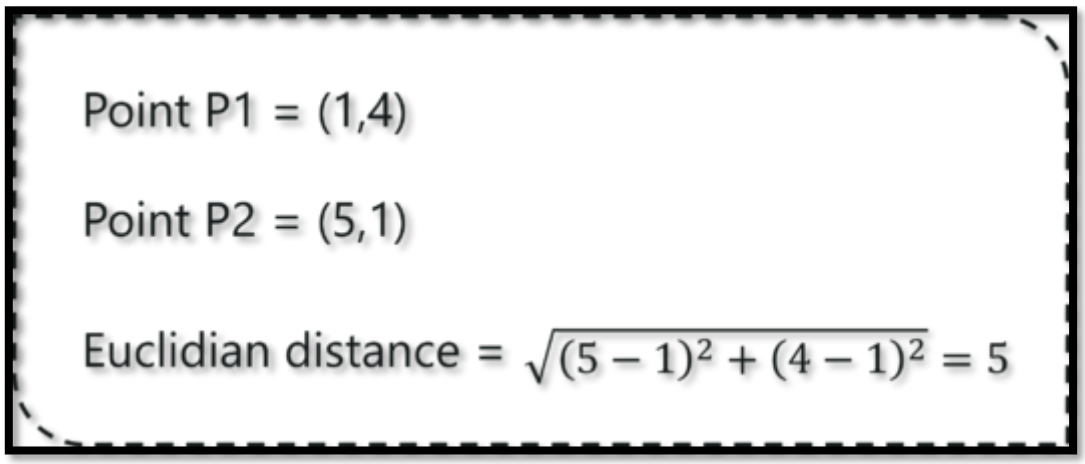
In practice, there's a lot more to consider while implementing the KNN algorithm. This will be discussed in the demo section of the blog.

Earlier I mentioned that KNN uses Euclidean distance as a measure to check the distance between a new data point and its neighbors, let's see how.



Euclidian Distance – KNN Algorithm In R

- Consider the above image, here we're going to measure the distance between P1 and P2 by using the Euclidian Distance measure.
- The coordinates for P1 and P2 are (1,4) and (5,1) respectively.
- The Euclidian Distance can be calculated like so:



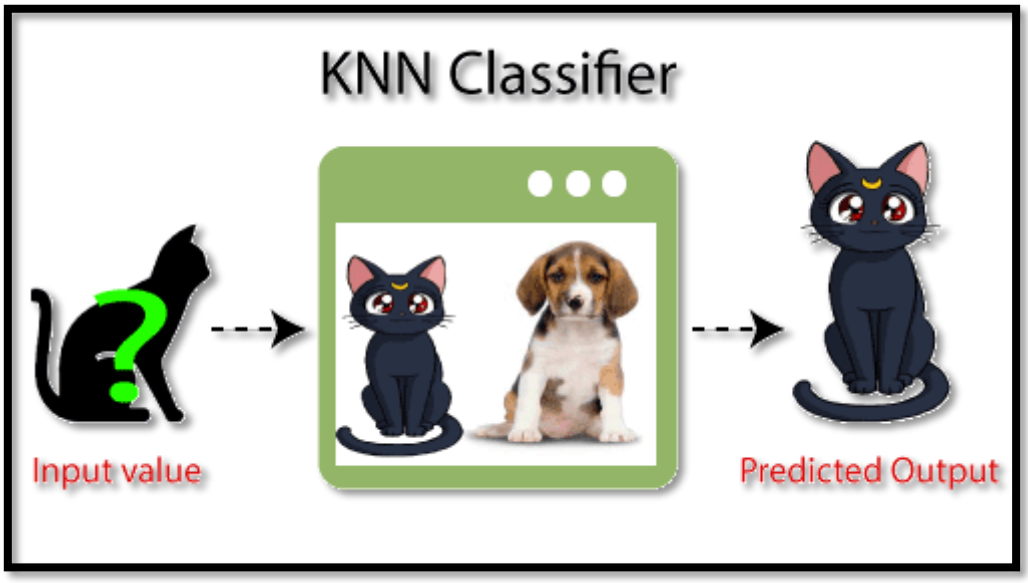
Euclidian Distance Calculations – KNN Algorithm In R

It is as simple as that! KNN makes use of simple measure in order to solve complex problems, this is one of the reasons why KNN is such a commonly used algorithm.

K-Nearest Neighbor(KNN) Algorithm for

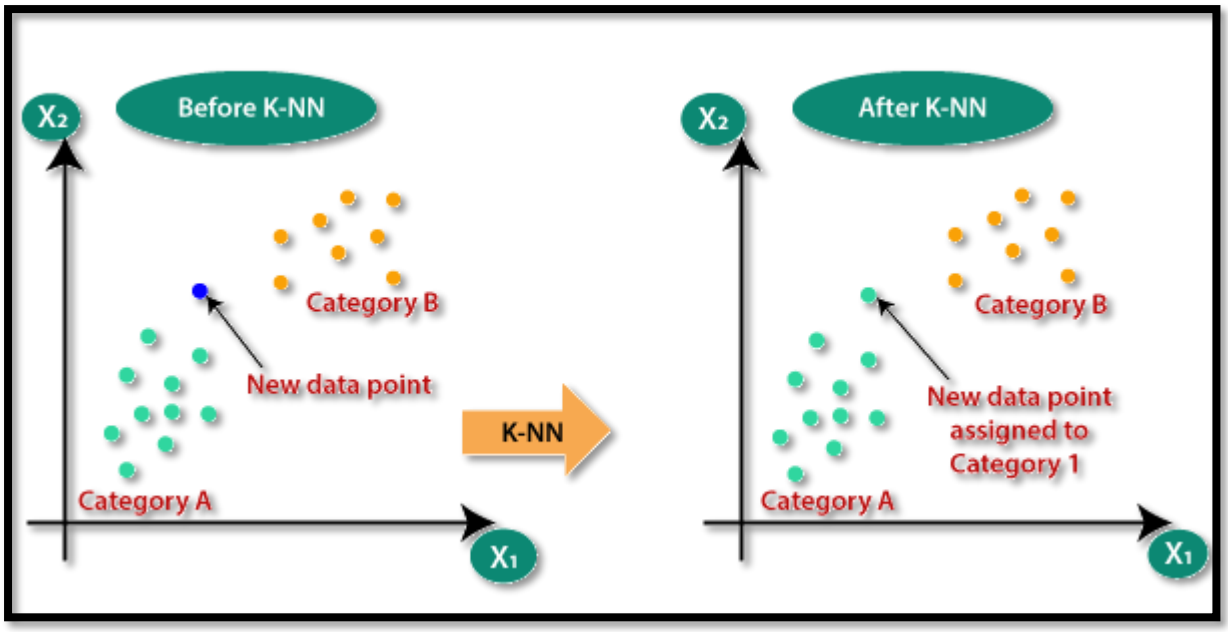
- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

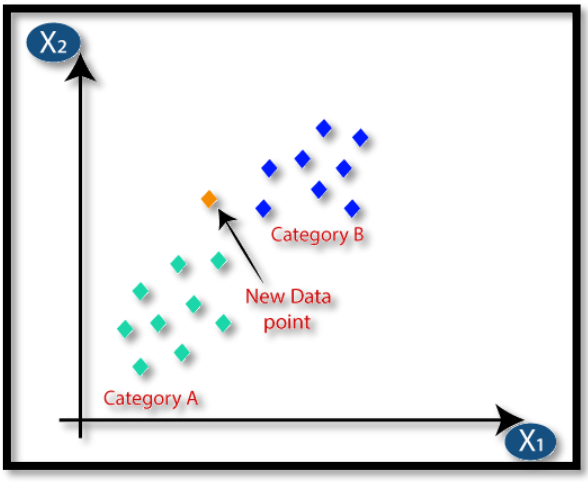


How does K-NN work?

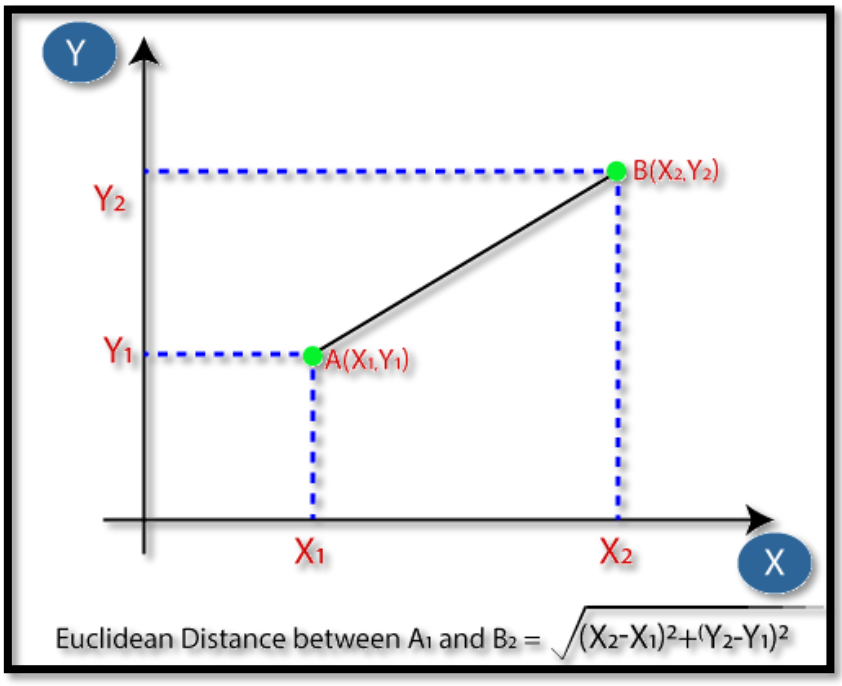
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

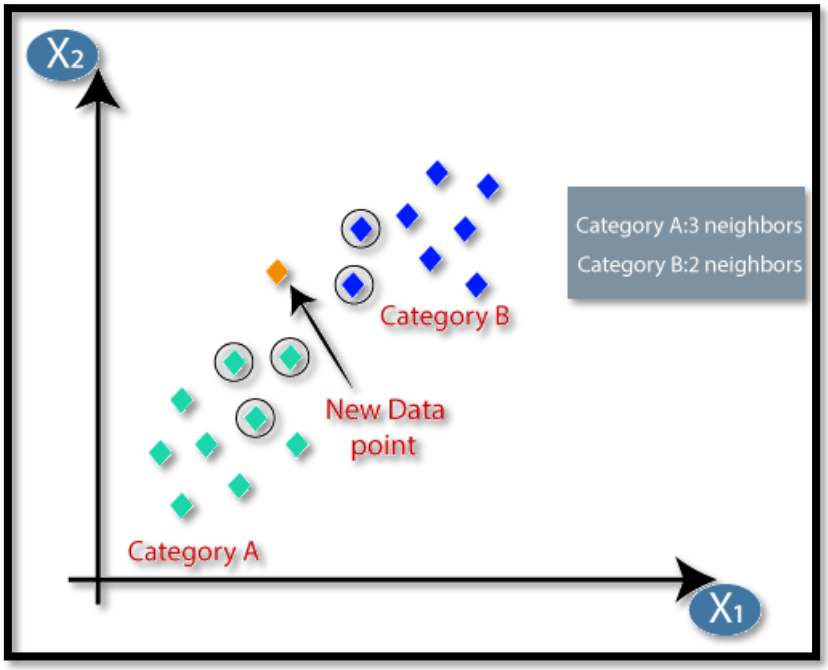
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Code:

```
loan <- read.csv("C:/Users/Vishal/Desktop/Vishal/Third Year 5th
Sem/DWM/german_credit.csv")

head (loan)

str(loan)


loan.subset <-
loan[c('Creditability','Age..years.','Sex...Marital.Status','Occupation','Account.Balance','Credit
.Amount','Length.of.current.employment','Purpose')]

str(loan.subset)

head(loan.subset)


#Normalization

normalize <- function(x) {

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

```
loan.subset.n <- as.data.frame(lapply(loan.subset[,2:8], normalize))
head(loan.subset.n)

set.seed(123)
dat.d <- sample(1:nrow(loan.subset.n),size=nrow(loan.subset.n)*0.7,replace = FALSE)
#random selection of 70% data.

train.loan <- loan.subset[dat.d,] # 70% training data
test.loan <- loan.subset[-dat.d,] # remaining 30% test data

#Creating seperate dataframe for 'Creditability' feature which is our target.
train.loan_labels <- loan.subset[dat.d,1]
test.loan_labels <-loan.subset[-dat.d,1]

#Install class package
install.packages('class')
# Load class package
library(class)

NROW(train.loan_labels)

knn.26 <- knn(train=train.loan, test=test.loan, cl=train.loan_labels, k=26)
knn.27 <- knn(train=train.loan, test=test.loan, cl=train.loan_labels, k=27)

ACC.26 <- 100 * sum(test.loan_labels == knn.26)/NROW(test.loan_labels)
ACC.27 <- 100 * sum(test.loan_labels == knn.27)/NROW(test.loan_labels)

ACC.26
ACC.27
```

```
# Check prediction against actual value in tabular form for k=26
table(knn.26 ,test.loan_labels)
knn.26

# Check prediction against actual value in tabular form for k=27
table(knn.27 ,test.loan_labels)
knn.27

install.packages('caret')
install.packages('e1071', dependencies=TRUE)

library(caret)
confusionMatrix(table(knn.26 ,test.loan_labels))

i=1
k.optm=1

for (i in 1:28){
+     knn.mod <- knn(train=train.loan, test=test.loan,cl=train.loan_labels, k=i)
+     k.optm[i] <- 100 * sum(test.loan_labels==knn.mod)/NROW(test.loan_labels)
+     k=i
+     cat(k,'=',k.optm[i],'\n')
+ }
#Accuracy plot
plot(k.optm, type="b", xlab="K- Value",ylab="Accuracy level")
```

Console:

```
> loan <- read.csv("C:/Users/Vishal/Desktop/Vishal/Third Year 5th Sem/DWM/german_credit.csv")
> head (loan)
  Creditability Account.Balance Duration.of.Credit..month. Payment.Status.of.Previous.Credit Purpose Credit.Amount
1             1               1                         18                                4       2          1049
2             1               1                          9                                4       0          2799
3             1               2                         12                                2       9           841
4             1               1                         12                                4       0          2122
5             1               1                         12                                4       0          2171
6             1               1                         10                                4       0          2241
  Value.Savings.Stocks Length.of.current.employment Instalment.per.cent Sex...Marital.Status Guarantors
1                    1                            2                    4                    2          1
2                    1                            3                    2                    3          1
3                    2                            4                    2                    2          1
4                    1                            3                    3                    3          1
5                    1                            3                    4                    3          1
6                    1                            2                    1                    3          1
  Duration.in.Current.address Most.valuable.available.asset Age..years. Concurrent.Credits Type.of.apartment
1                           4                              2          21                  3                  1
2                           2                              1          36                  3                  1
3                           4                              1          23                  3                  1
4                           2                              1          39                  3                  1
5                           4                              2          38                  1                  2
6                           3                              1          48                  3                  1
  No.of.Credits.at.this.Bank Occupation No.of.dependents Telephone Foreign.Worker
1                          1          3                1         1              1
2                          2          3                2         1              1
3                          1          2                1         1              1
4                          2          2                2         1              2
5                          2          2                1         1              2
6                          2          2                2         1              2
```

```
Console  Terminal  Jobs
~/

R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> loan <- read.csv("C:/Users/Vishal/Desktop/Vishal/Third Year 5th Sem/DWM/german_credit.csv")
> head (loan)
  Creditability Account.Balance Duration.of.Credit..month. Payment.Status.of.Previous.Credit Purpose Credit.Amount Value.Savings.Stocks Length.of.current.employment
1             1               1                         18                                4       2          1049                    1                            2
2             1               1                          9                                4       0          2799                    1                            3
3             1               2                         12                                2       9           841                    2                            4
4             1               1                         12                                4       0          2122                    1                            3
5             1               1                         12                                4       0          2171                    1                            3
6             1               1                         10                                4       0          2241                    1                            2
  Instalment.per.cent Sex...Marital.Status Guarantors Duration.in.Current.address Most.valuable.available.asset Age..years. Concurrent.Credits Type.of.apartment
1                   4                    2          1                           4                              2          21                  3                  1
2                   2                    3          1                           2                              1          36                  3                  1
3                   2                    2          1                           4                              1          23                  3                  1
4                   3                    3          1                           2                              1          39                  3                  1
5                   4                    3          1                           4                              2          38                  1                  2
6                   1                    3          1                           3                              1          48                  3                  1
  No.of.Credits.at.this.Bank Occupation No.of.dependents Telephone Foreign.Worker
1                          1          3                1         1              1
2                          2          3                2         1              1
3                          1          2                1         1              1
4                          2          2                2         1              2
5                          2          2                1         1              2
6                          2          2                2         1              2
```

```
> str(loan)
'data.frame':   1000 obs. of  21 variables:
 $ Creditability                  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Account.Balance                : int  1 1 2 1 1 1 1 1 4 2 ...
 $ Duration.of.Credit..month.     : int  18 9 12 12 12 10 8 6 18 24 ...
 $ Payment.Status.of.Previous.Credit: int  4 4 2 4 4 4 4 4 4 2 ...
 $ Purpose                        : int  2 0 9 0 0 0 0 0 3 3 ...
 $ Credit.Amount                  : int  1049 2799 841 2122 2171 2241 3398 1361 1098 3758 ...
 $ Value.Savings.Stocks           : int  1 1 2 1 1 1 1 1 1 3 ...
 $ Length.of.current.employment   : int  2 3 4 3 3 2 4 2 1 1 ...
 $ Instalment.per.cent            : int  4 2 2 3 4 1 1 2 4 1 ...
 $ Sex...Marital.Status           : int  2 3 2 3 3 3 3 3 2 2 ...
 $ Guarantors                     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Duration.in.Current.address    : int  4 2 4 2 4 3 4 4 4 4 ...
 $ Most.valuable.available.asset  : int  2 1 1 1 2 1 1 1 3 4 ...
 $ Age..years.                    : int  21 36 23 39 38 48 39 40 65 23 ...
 $ Concurrent.Credits             : int  3 3 3 3 1 3 3 3 3 3 ...
 $ Type.of.apartment              : int  1 1 1 1 2 1 2 2 2 1 ...
 $ No.of.Credits.at.this.Bank     : int  1 2 1 2 2 2 2 1 2 1 ...
 $ Occupation                     : int  3 3 2 2 2 2 2 2 1 1 ...
 $ No.of.dependents               : int  1 2 1 2 1 2 1 2 1 1 ...
 $ Telephone                      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Foreign.Worker                 : int  1 1 1 2 2 2 2 2 1 1 ...
> loan.subset <- loan[c('Creditability','Age..years.','Sex...Marital.Status','Occupation','Account.Balance','Credit.Amount','Length.of.current.employment','Purpose')]
> str(loan.subset)
'data.frame':   1000 obs. of  8 variables:
 $ Creditability              : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Age..years.                : int  21 36 23 39 38 48 39 40 65 23 ...
 $ Sex...Marital.Status       : int  2 3 2 3 3 3 3 3 2 2 ...
 $ Occupation                 : int  3 3 2 2 2 2 2 2 1 1 ...
 $ Account.Balance            : int  1 1 2 1 1 1 1 1 4 2 ...
 $ Credit.Amount              : int  1049 2799 841 2122 2171 2241 3398 1361 1098 3758 ...
 $ Length.of.current.employment: int  2 3 4 3 3 2 4 2 1 1 ...
 $ Purpose                    : int  2 0 9 0 0 0 0 0 3 3 ...
> head(loan.subset)
  Creditability Age..years. Sex...Marital.Status Occupation Account.Balance Credit.Amount Length.of.current.employment Purpose
1             1          21                    2          3               1         1049                           2       2
2             1          36                    3          3               1         2799                           3       0
3             1          23                    2          2               2          841                           4       9
4             1          39                    3          2               1         2122                           3       0
5             1          38                    3          2               1         2171                           3       0
6             1          48                    3          2               1         2241                           2       0
> normalize <- function(x) {
+   return ((x - min(x)) / (max(x) - min(x))) }
> loan.subset.n <- as.data.frame(lapply(loan.subset[,2:8], normalize))
> head(loan.subset.n)
  Age..years. Sex...Marital.Status Occupation Account.Balance Credit.Amount Length.of.current.employment Purpose
1  0.03571429            0.3333333  0.6666667       0.0000000    0.04396390                        0.25     0.2
2  0.30357143            0.6666667  0.6666667       0.0000000    0.14025531                        0.50     0.0
3  0.07142857            0.3333333  0.3333333       0.3333333    0.03251898                        0.75     0.9
4  0.35714286            0.6666667  0.3333333       0.0000000    0.10300429                        0.50     0.0
5  0.33928571            0.6666667  0.3333333       0.0000000    0.10570045                        0.50     0.0
6  0.51785714            0.6666667  0.3333333       0.0000000    0.10955211                        0.25     0.0
> set.seed(123)
> dat.d <- sample(1:nrow(loan.subset.n),size=nrow(loan.subset.n)*0.7,replace = FALSE)
> train.loan <- loan.subset[dat.d,]
> test.loan <- loan.subset[-dat.d,]
```

```
> str(loan)
'data.frame':   1000 obs. of  21 variables:
 $ Creditability                  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Account.Balance                : int  1 1 2 1 1 1 1 1 4 2 ...
 $ Duration.of.Credit..month.     : int  18 9 12 12 12 10 8 6 18 24 ...
 $ Payment.Status.of.Previous.Credit: int  4 4 2 4 4 4 4 4 4 2 ...
 $ Purpose                        : int  2 0 9 0 0 0 0 0 3 3 ...
 $ Credit.Amount                  : int  1049 2799 841 2122 2171 2241 3398 1361 1098 3758 ...
 $ Value.Savings.Stocks           : int  1 1 2 1 1 1 1 1 1 3 ...
 $ Length.of.current.employment   : int  2 3 4 3 3 2 4 2 1 1 ...
 $ Instalment.per.cent            : int  4 2 2 3 4 1 1 2 4 1 ...
 $ Sex...Marital.Status           : int  2 3 2 3 3 3 3 3 2 2 ...
 $ Guarantors                     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Duration.in.Current.address    : int  4 2 4 2 4 3 4 4 4 4 ...
 $ Most.valuable.available.asset  : int  2 1 1 1 2 1 1 1 3 4 ...
 $ Age..years.                    : int  21 36 23 39 38 48 39 40 65 23 ...
 $ Concurrent.Credits             : int  3 3 3 3 1 3 3 3 3 3 ...
 $ Type.of.apartment              : int  1 1 1 1 2 1 2 2 2 1 ...
 $ No.of.Credits.at.this.Bank     : int  1 2 1 2 2 2 2 1 2 1 ...
 $ Occupation                     : int  3 3 2 2 2 2 2 2 1 1 ...
 $ No.of.dependents               : int  1 2 1 2 1 2 1 2 1 1 ...
 $ Telephone                      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Foreign.Worker                 : int  1 1 1 2 2 2 2 2 1 1 ...
```

```
> loan.subset <- loan[c('Creditability','Age..years.','Sex...Marital.Status','Occupation','Account.Balance','Credit.Amount','Length.of.current.employment','Purpose')]
> str(loan.subset)
'data.frame':   1000 obs. of  8 variables:
 $ Creditability              : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Age..years.                : int  21 36 23 39 38 48 39 40 65 23 ...
 $ Sex...Marital.Status       : int  2 3 2 3 3 3 3 3 2 2 ...
 $ Occupation                 : int  3 3 2 2 2 2 2 2 1 1 ...
 $ Account.Balance            : int  1 1 2 1 1 1 1 1 4 2 ...
 $ Credit.Amount              : int  1049 2799 841 2122 2171 2241 3398 1361 1098 3758 ...
 $ Length.of.current.employment: int  2 3 4 3 3 2 4 2 1 1 ...
 $ Purpose                    : int  2 0 9 0 0 0 0 0 3 3 ...
```

```
> str(loan.subset)
'data.frame':	1000 obs. of  8 variables:
 $ Creditability                    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Age..years.                      : int  21 36 23 39 38 48 39 40 65 23 ...
 $ Sex...Marital.Status             : int  2 3 2 3 3 3 3 3 2 2 ...
 $ Occupation                       : int  3 3 2 2 2 2 2 2 1 1 ...
 $ Account.Balance                  : int  1 1 2 1 1 1 1 1 4 2 ...
 $ Credit.Amount                    : int  1049 2799 841 2122 2171 2241 3398 1361 1098 3758 ...
 $ Length.of.current.employment: int  2 3 4 3 3 2 4 2 1 1 ...
 $ Purpose                          : int  2 0 9 0 0 0 0 0 3 3 ...
```

```
> head(loan.subset)
  Creditability Age..years. Sex...Marital.Status Occupation Account.Balance Credit.Amount Length.of.current.employment
1             1          21                    2          3               1          1049                            2
2             1          36                    3          3               1          2799                            3
3             1          23                    2          2               2           841                            4
4             1          39                    3          2               1          2122                            3
5             1          38                    3          2               1          2171                            3
6             1          48                    3          2               1          2241                            2
  Purpose
1       2
2       0
3       9
4       0
5       0
6       0
```

```
> normalize <- function(x) {
+   return ((x - min(x)) / (max(x) - min(x))) }
> loan.subset.n <- as.data.frame(lapply(loan.subset[,2:8], normalize))
> head(loan.subset.n)
  Age..years. Sex...Marital.Status Occupation Account.Balance Credit.Amount Length.of.current.employment Purpose
1  0.03571429            0.3333333  0.6666667       0.0000000    0.04396390                         0.25     0.2
2  0.30357143            0.6666667  0.6666667       0.0000000    0.14025531                         0.50     0.0
3  0.07142857            0.3333333  0.3333333       0.3333333    0.03251898                         0.75     0.9
4  0.35714286            0.6666667  0.3333333       0.0000000    0.10300429                         0.50     0.0
5  0.33928571            0.6666667  0.3333333       0.0000000    0.10570045                         0.50     0.0
6  0.51785714            0.6666667  0.3333333       0.0000000    0.10955211                         0.25     0.0
```

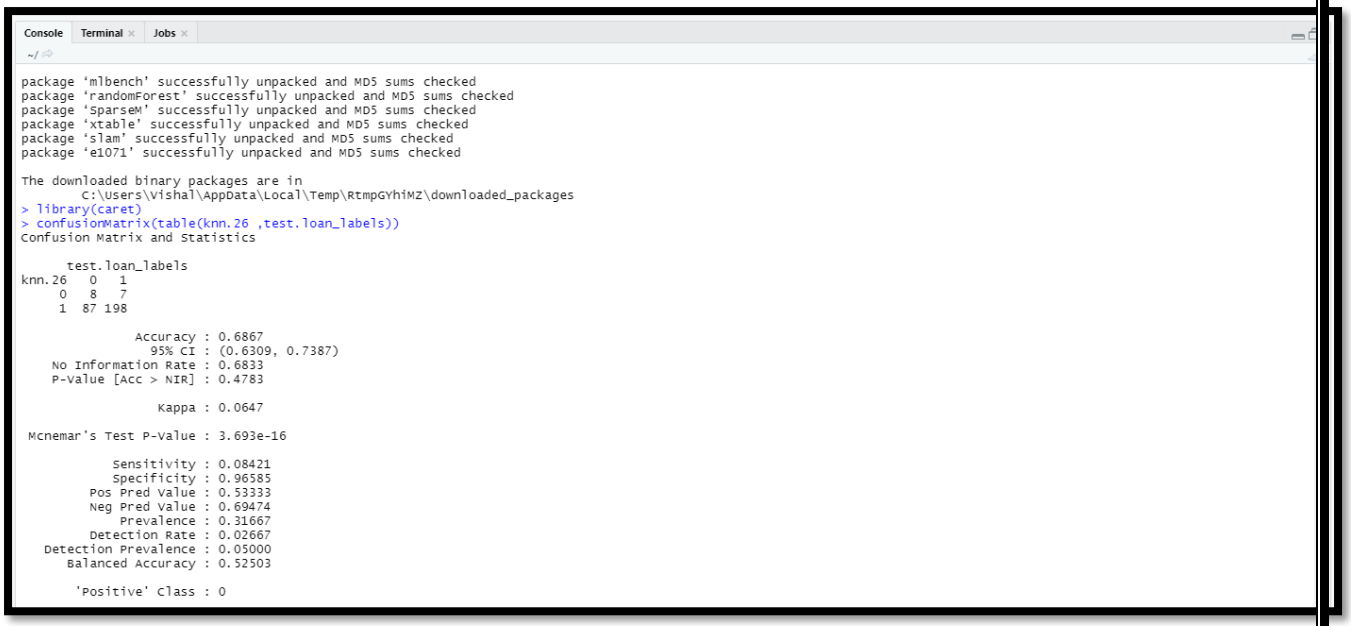
```
Source
Console   Terminal ×   Jobs ×
~/
> test.loan_labels <-loan.subset[-dat.d,1]
> install.packages('class')
WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropriate version of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into 'C:/Users/Vishal/Documents/R/win-library/4.0'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/class_7.3-17.zip'
Content type 'application/zip' length 108460 bytes (105 KB)
downloaded 105 KB

package 'class' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Vishal\AppData\Local\Temp\RtmpCYuRBT\downloaded_packages
> library(class)
> NROW(train.loan_labels)
[1] 700
> knn.26 <- knn(train=train.loan, test=test.loan, cl=train.loan_labels, k=26)
> knn.27 <- knn(train=train.loan, test=test.loan, cl=train.loan_labels, k=27)
> ACC.26 <- 100 * sum(test.loan_labels == knn.26)/NROW(test.loan_labels)
> ACC.27 <- 100 * sum(test.loan_labels == knn.27)/NROW(test.loan_labels)
> ACC.26
[1] 68.66667
> ACC.27
[1] 69
> table(knn.26 ,test.loan_labels)
      test.loan_labels
knn.26   0   1
     0   8   7
     1  87 198
> knn.26
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1
 [81] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[161] 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[241] 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0
Levels: 0 1
> table(knn.27 ,test.loan_labels)
      test.loan_labels
knn.27   0   1
     0   8   6
     1  87 199
> knn.27
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1
 [81] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[161] 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[241] 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0
Levels: 0 1
```

```
> set.seed(123)
> dat.d <- sample(1:nrow(loan.subset.n),size=nrow(loan.subset.n)*0.7,replace = FALSE)
> train.loan <- loan.subset[dat.d,]
> test.loan <- loan.subset[-dat.d,]
> train.loan_labels <- loan.subset[dat.d,1]
> test.loan_labels <-loan.subset[-dat.d,1]
> install.packages('class')
```

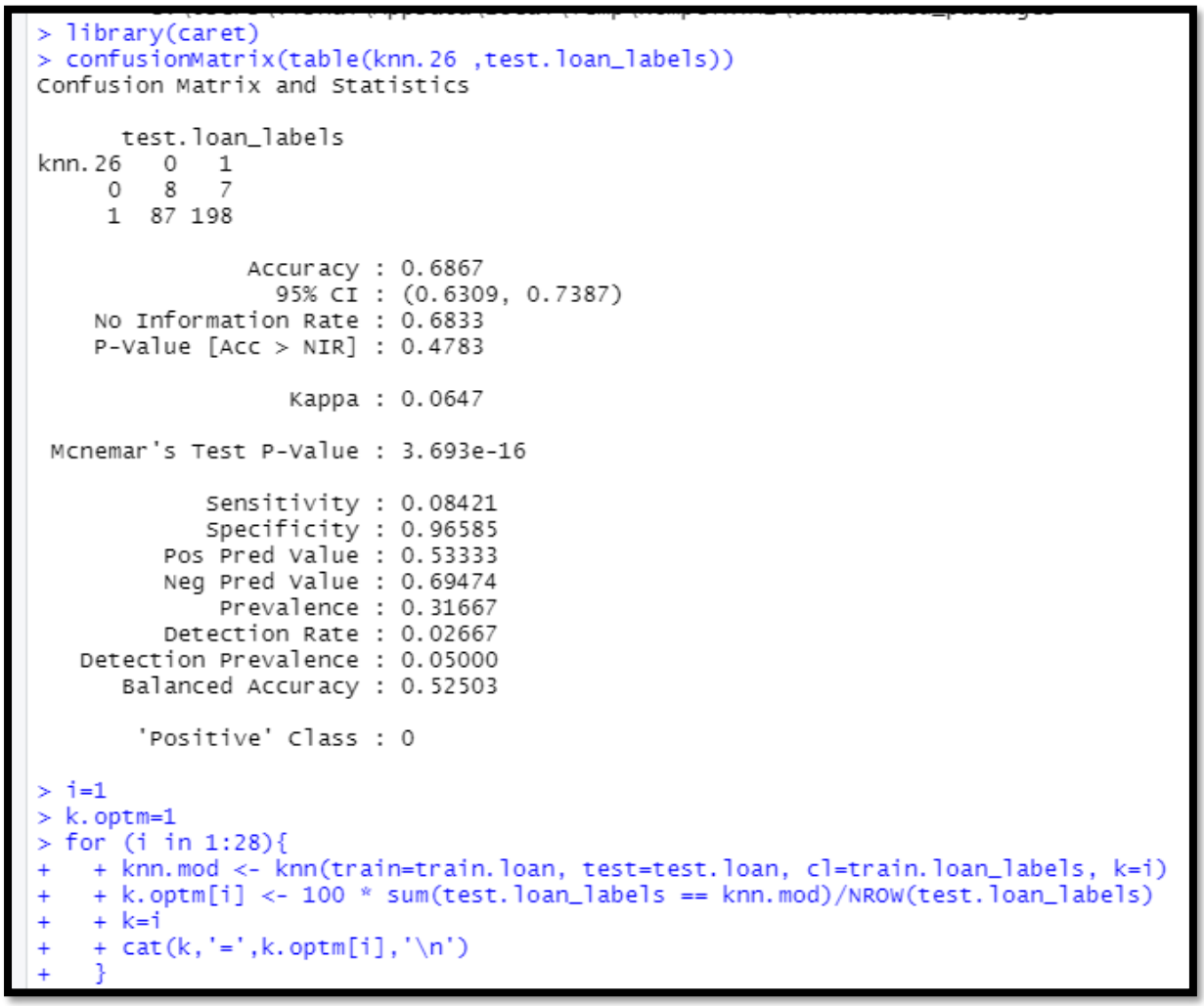
```
package 'class' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Vishal\AppData\Local\Temp\Rtmp6xC0j7\downloaded_packages
> library(class)
> NROW(train.loan_labels)
[1] 700
```

```
> knn.26 <- knn(train=train.loan, test=test.loan, cl=train.loan_labels, k=26)
> knn.27 <- knn(train=train.loan, test=test.loan, cl=train.loan_labels, k=27)
> ACC.26 <- 100 * sum(test.loan_labels == knn.26)/NROW(test.loan_labels)
> ACC.27 <- 100 * sum(test.loan_labels == knn.27)/NROW(test.loan_labels)
> ACC.26
[1] 69
> ACC.27
[1] 69
```

```
> table(knn.26 ,test.loan_labels)
      test.loan_labels
knn.26   0   1
     0   8   6
     1  87 199
> knn.26
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [60] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
[119] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1
[178] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[237] 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1
[296] 1 1 1 1 0
Levels: 0 1
> table(knn.27 ,test.loan_labels)
      test.loan_labels
knn.27   0   1
     0   8   6
     1  87 199
> knn.27
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [60] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
[119] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1
[178] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[237] 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1
[296] 1 1 1 1 0
Levels: 0 1
```



```
Console  Terminal  Jobs
~/

package 'mlbench' successfully unpacked and MD5 sums checked
package 'randomForest' successfully unpacked and MD5 sums checked
package 'SparseM' successfully unpacked and MD5 sums checked
package 'xtable' successfully unpacked and MD5 sums checked
package 'slam' successfully unpacked and MD5 sums checked
package 'e1071' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Vishal\AppData\Local\Temp\RtmpGYhiMZ\downloaded_packages
> library(caret)
> confusionMatrix(table(knn.26 ,test.loan_labels))
Confusion Matrix and Statistics

      test.loan_labels
knn.26   0   1
     0   8   7
     1  87 198

               Accuracy : 0.6867
                 95% CI : (0.6309, 0.7387)
    No Information Rate : 0.6833
    P-Value [Acc > NIR] : 0.4783

                  Kappa : 0.0647

 Mcnemar's Test P-Value : 3.693e-16

            Sensitivity : 0.08421
            Specificity : 0.96585
         Pos Pred Value : 0.53333
         Neg Pred Value : 0.69474
             Prevalence : 0.31667
         Detection Rate : 0.02667
   Detection Prevalence : 0.05000
      Balanced Accuracy : 0.52503

       'Positive' Class : 0
```

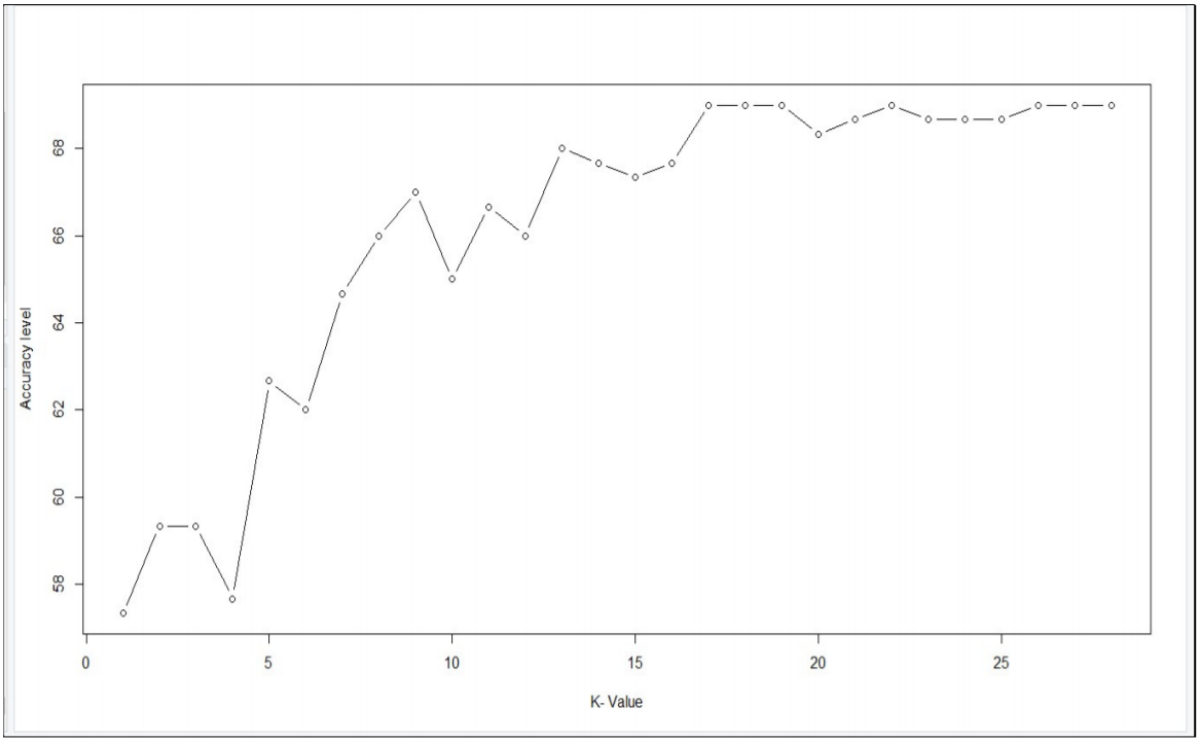


```
> library(caret)
> confusionMatrix(table(knn.26 ,test.loan_labels))
Confusion Matrix and Statistics

      test.loan_labels
knn.26   0   1
     0   8   7
     1  87 198

               Accuracy : 0.6867
                 95% CI : (0.6309, 0.7387)
    No Information Rate : 0.6833
    P-Value [Acc > NIR] : 0.4783

                  Kappa : 0.0647

 Mcnemar's Test P-Value : 3.693e-16

            Sensitivity : 0.08421
            Specificity : 0.96585
         Pos Pred Value : 0.53333
         Neg Pred Value : 0.69474
             Prevalence : 0.31667
         Detection Rate : 0.02667
   Detection Prevalence : 0.05000
      Balanced Accuracy : 0.52503

       'Positive' Class : 0

> i=1
> k.optm=1
> for (i in 1:28){
+   + knn.mod <- knn(train=train.loan, test=test.loan, cl=train.loan_labels, k=i)
+   + k.optm[i] <- 100 * sum(test.loan_labels == knn.mod)/NROW(test.loan_labels)
+   + k=i
+   + cat(k,'=',k.optm[i],'\n')
+   }
```

```
> for (i in 1:28){
+         knn.mod <- knn(train=train.loan,test=test.loan,cl=train.loan_labels,k=i)
+         k.optm[i] <-100* sum(test.loan_labels==knn.mod)/NROW(test.loan_labels)
+         k=i
+         cat(k,'=',k.optm[i],'\n')
+
+ }
1 = 57.33333
2 = 59.33333
3 = 59.33333
4 = 57.66667
5 = 62.66667
6 = 62
7 = 64.66667
8 = 66
9 = 67
10 = 65
11 = 66.66667
12 = 66
13 = 68
14 = 67.66667
15 = 67.33333
16 = 67.66667
17 = 69
18 = 69
19 = 69
20 = 68.33333
21 = 68.66667
22 = 69
23 = 68.66667
24 = 68.66667
25 = 68.66667
26 = 69
27 = 69
28 = 69
```

#Accuracy plot

plot(k.optm, type="b", xlab="K- Value",ylab="Accuracy level")



Conclusion:

Thus, I studied about KNN algorithm of classification on given dataset by using R programming.