**K-nearest Neighbors Algorithm**

Let’ s say we are given certain number of objects and each object has its own unique attribute associated with it, for example, you have 5 chairs 10 beds and 15 tables and each for each one of them we know the length, breadth and height. Now, if someone give us a new object with new attributes and ask us to predict as to which category does that new object belongs to, that means we are given the dimension and asked to predict if it is a chair, bed or table, then we would have to use knn algorithm to determine that.

Therefore, attributes means the property of each object, and each object can be considered as a category. Our job is to check how closely are the properties of the new object is related to any one of the already known categories.

When we are given the attributes, we try and plot them on graph. The graphical representation of those attributes will help us to calculate Euclidian distance between the new value and the ones that we already know we have. By doing so, we can be sure as to which category does the new object is closest to.

# Euclidean distance

It just simply means the distance between two points in a plane. By simple using this formula you can calculate distance between two points no matter how many attributes or properties you are given like height, breadth, width, weight and so on upto n where n could be the last property of the object you have. The formula is √(x2−x1)²+(y2−y1)²+(z2−z1)² …… (n2-n1)²

Before we move on to implementing them in R, be aware of these following notes:

1- The nearest neighbor you want to check will be called defined by value “k”. If k is 5 then you will check 5 closest neighbors in order to determine the category. If majority of neighbor belongs to a certain category from within those five nearest neighbors, then that will be chosen as the category of upcoming object.

2- Different variables have different scaling units, like weight in kg and height in cm. Then how do we suppose to use them in the Euclidean formula? Well, we normalize each one of the variables using the formula (x-min(x))/(min(x) — max(x)) that we will also see in the examples below. Now if you have one variable of 200kg and another with 50kg, after normalization both will be represented by the value between 0 and 1. Just imagine you have created a box that can hold things that are in range of 0 and 1 only, but since you have used normalization formula to converted everything into values between 0 and 1, you are good to use that box. Note: That box is your plot.

3- The knn algorithm works well with the numeric variables, this is not to say that it cannot work with categorical variables, but it’s just if you have mix of both categorical and numeric variables as the predictors then it demands little bit of different approach. But if all predictors are numeric, then knn is best because we are dealing with the distance and for that we need hard numbers.

4- When we split our data into training and testing sets, the data should have already be normalized. That mean we first normalize the data and then split it.

5- The knn algorithm does not works with ordered-factors in R but rather with factors. We will see that in the code below.

6- The k-mean algorithm is different than K- nearest neighbor algorithm. K-mean is used for clustering and is a unsupervised learning algorithm whereas Knn is supervised leaning algorithm that works on classification problems.

df <- data(iris) ##load data  
head(iris) ## see the structure

##Generate a random number that is 90% of the total number of rows in dataset.  
ran <- sample(1:nrow(iris), 0.9 \* nrow(iris))   
##the normalization function is created  
nor <-function(x) { (x -min(x))/(max(x)-min(x)) }  
##Run nomalization on first 4 coulumns of dataset because they are the predictors  
iris\_norm <- as.data.frame(lapply(iris[,c(1,2,3,4)], nor))  
summary(iris\_norm)

##extract training set  
iris\_train <- iris\_norm[ran,]

##extract testing set  
iris\_test <- iris\_norm[-ran,]

##extract 5th column of train dataset because it will be used as 'cl' argument in knn function.  
iris\_target\_category <- iris[ran,5]

##extract 5th column if test dataset to measure the accuracy  
iris\_test\_category <- iris[-ran,5]

##load the package class  
library(class)

##run knn function  
pr <- knn(iris\_train,iris\_test,cl=iris\_target\_category,k=13)

#**train -** matrix or data frame of training set cases.

#test - matrix or data frame of test set cases.

#cl - factor of true classifications of training set

#**k -** number of neighbors considered.

##create confusion matrix  
tab <- table(pr,iris\_test\_category)

##this function divides the correct predictions by total number of predictions that tell us how accurate teh model is.

accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) \* 100}  
accuracy(tab)

In the iris dataset that is already available in R, we have run the k-nearest neighbor algorithm that gave us 80% accurate result. First, we normalized the data to convert petal.length, sepal.length, petal.width and sepal.length into a standardized 0-to-1 form so that we can fit them into one box (one graph) and also because our main objective is to predict whether a flower is virginica, Versicolor, or setosa and that is why we excluded the column 5 and stored it into another variable called iris\_target\_category. Then, we separated the normalized values into training and testing dataset. Imagine it this way, that the values from training dataset are firstly drawn on a graph and after we run knn function with all the necessary arguments, we introduce testing dataset’s values into the graph and calculate Euclidean distance with each and every already stored point in graph. Now, although we know which flower it is in testing dataset, we still predict the values and store them in variable called ‘pr’ so that we can compare predicted values with original testing dataset’s values. This way we understand the accuracy of our model and if we are to get new 50 values in future and we are asked to predict the category of those 50 values, we can do that with this model.