<https://towardsdatascience.com/k-nearest-neighbors-algorithm-with-examples-in-r-simply-explained-knn-1f2c88da405c>

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. 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. 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. 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. 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. 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.

## load data

library(class)

data("iris") # load the iris data

> str(iris)

'data.frame': 150 obs. of 5 variables:

$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...

$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...

$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...

$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...

$ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

> head(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

1 5.1 3.5 1.4 0.2 setosa

2 4.9 3.0 1.4 0.2 setosa

3 4.7 3.2 1.3 0.2 setosa

4 4.6 3.1 1.5 0.2 setosa

5 5.0 3.6 1.4 0.2 setosa

6 5.4 3.9 1.7 0.4 setosa

2. Preprocess the dataset

Since classification is a type of Supervised Learning, we would require two sets of data i.e. Training and Testing Data(generally in 80:20 ratio). We would load Iris Dataset which is available in RStudio by default and then divide the dataset into two subsets. Our knn classification model would then be trained using subset iris.train and tested using iris.test. Since the iris dataset is sorted by “Species” by default, we will first jumble the data rows and then take subset.

> set.seed(123) # required to reproduce the results

> rnum = sample(rep(1:150)) # randomly generate numbers from 1 to 150

> rnum

[1]14 50 118 43 150 148 90 91 143 92 137 99 72 26 7 78 81 147 103 117 76 32

[23] 106 109 136 9 41 74 23 27 60 53 126 119 121 96 38 89 34 93 69 138 130 63

[45] 13 82 97 142 25 114 21 79 124 47 144 120 16 6 127 86 132 39 31 134 149 112

[67] 4 128 110 102 52 22 129 87 35 40 30 12 88 123 64 146 67 122 37 8 51 10

[89] 115 42 44 85 107 139 73 20 46 17 54 108 75 80 71 15 24 68 133 145 29 104

[111] 45 140 101 135 95 116 5 111 94 49 100 62 11 84 56 105 19 36 57 125 61 2

[133] 66 83 3 65 70 55 77 1 98 113 28 33 18 48 59 131 58 141

> iris = iris[rnum,] #randomize "iris" dataset

> head(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

14 4.3 3.0 1.1 0.1 setosa

50 5.0 3.3 1.4 0.2 setosa

118 7.7 3.8 6.7 2.2 virginica

43 4.4 3.2 1.3 0.2 setosa

150 5.9 3.0 5.1 1.8 virginica

148 6.5 3.0 5.2 2.0 virginica

> normalize <- function(x){

+ return ((x-min(x))/(max(x)-min(x)))

+ }

> iris.new = as.data.frame(apply(iris[c(1,2,3,4)], MARGIN=2, normalize))

> head(iris.new)

Sepal.Length Sepal.Width Petal.Length Petal.Width

14 0.00000000 0.4166667 0.01694915 0.00000000

50 0.19444444 0.5416667 0.06779661 0.04166667

118 0.94444444 0.7500000 0.96610169 0.87500000

43 0.02777778 0.5000000 0.05084746 0.04166667

150 0.44444444 0.4166667 0.69491525 0.70833333

148 0.61111111 0.4166667 0.71186441 0.79166667

> # Divide the data in train and test data i.e. subset the data set

> iris.train = iris.new[1:100,]

> iris.test = iris.new[101:150,]

> iris.train.target = iris[1:100,5]

> iris.test.target = iris[101:150,5]

### 3. Apply k-NN classification algorithm

> model1 = knn(train = iris.train, test = iris.test, cl = iris.train.target, k=3)

### 4. Verify results confusion matrix

> table(iris.test.target, model1)

model1

iris.test.target setosa versicolor virginica

setosa 16 0 0

versicolor 0 19 2

virginica 0 1 12

> accuracy = function(x){sum(diag(x)/(sum(rowSums(x))))}

> accuracy(tab)

[1] 0.94

The values on the diagonal shows number of correctly classified instances out of total 153 instances. The values not on the diagonal implies that they have been incorrectly instances. Hence, there is a scope of further improvement in classifier model. Improvement may be done in terms of trying different values of “k” and choosing the one with maximum accuracy. However, other classification algorithms may also be tried to get a better result. There is no stopping in Optimization!

> model1 = knn(train = iris.train, test = iris.test, cl = iris.train.target, k=16)

> table(iris.test.target, model1)

model1

iris.test.target setosa versicolor virginica

setosa 16 0 0

versicolor 0 19 2

virginica 0 0 13

> accuracy = function(x){sum(diag(x)/(sum(rowSums(x))))}

> accuracy(tab)

[1] 0.96