When Do We Use the KNN Algorithm?

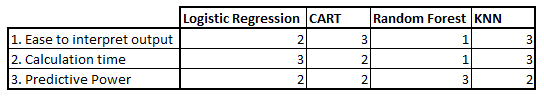
KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique, we generally look at 3 important aspects:

1. Ease of interpreting output

2. Calculation time

3. Predictive Power

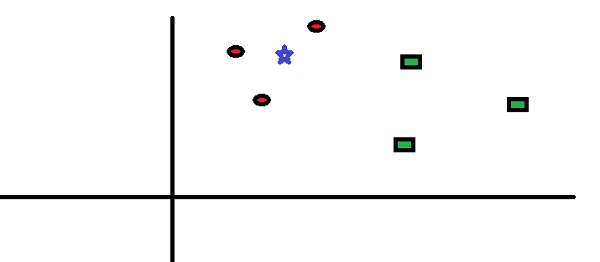
Let us take a few examples to  place KNN in the scale :

[](https://www.analyticsvidhya.com/wp-content/uploads/2014/10/Model-comparison.png)

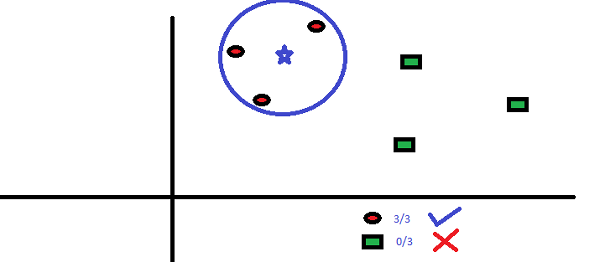
KNN classifier fairs across all parameters of consideration. It is commonly used for its ease of interpretation and low calculation time.

How Does the KNN Algorithm Work?

Let’s take a simple case to understand this algorithm. Following is a spread of red circles (RC) and green squares (GS):

[](https://www.analyticsvidhya.com/wp-content/uploads/2014/10/scenario1.png)

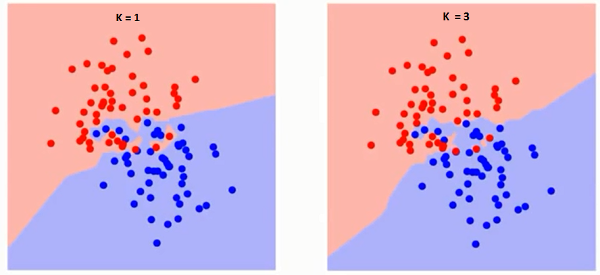
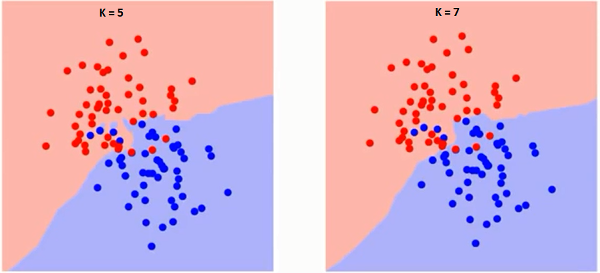
You intend to find out the class of the blue star (BS). BS can either be RC or GS and nothing else. The “K” in KNN algorithm is the nearest neighbor we wish to take the vote from. Let’s say K = 3. Hence, we will now make a circle with BS as the center just as big as to enclose only three data points on the plane. Refer to the following diagram for more details:

[](https://www.analyticsvidhya.com/wp-content/uploads/2014/10/scenario2.png)

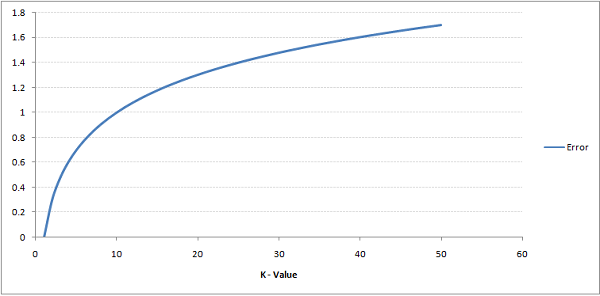
The three closest points to BS are all RC. Hence, with a good confidence level, we can say that the BS should belong to the class RC. Here, the choice became obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm. Next, we will understand the factors to be considered to conclude the best K.

How Do We Choose the Factor K?

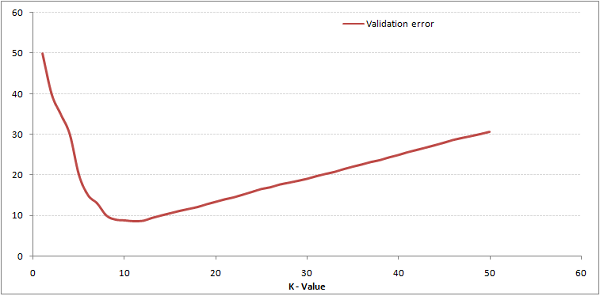
First, let us try to understand exactly the K influence in the algorithm. If we see the last example, given that all the 6 training observation remain constant, with a given K value we can make boundaries of each class. These decision boundaries will segregate RC from GS. In the same way, let’s try to see the effect of value “K” on the class boundaries. The following are the different boundaries separating the two classes with different values of K.

[](https://www.analyticsvidhya.com/wp-content/uploads/2014/10/K-judgement.png)[](https://www.analyticsvidhya.com/wp-content/uploads/2014/10/K-judgement2.png)

If you watch carefully, you can see that the boundary becomes smoother with increasing value of K. With K increasing to infinity it finally becomes all blue or all red depending on the total majority.  The training error rate and the validation error rate are two parameters we need to access different K-value. Following is the curve for the training error rate with a varying value of K :

[](https://www.analyticsvidhya.com/wp-content/uploads/2014/10/training-error.png)

As you can see, the error rate at K=1 is always zero for the training sample. This is because the closest point to any training data point is itself.Hence the prediction is always accurate with K=1. If validation error curve would have been similar, our choice of K would have been 1. Following is the validation error curve with varying value of K:

[](https://www.analyticsvidhya.com/wp-content/uploads/2014/10/training-error_11.png)

This makes the story more clear. At K=1, we were overfitting the boundaries. Hence, error rate initially decreases and reaches a minima. After the minima point, it then increase with increasing K. To get the optimal value of K, you can segregate the training and validation from the initial dataset. Now plot the validation error curve to get the optimal value of K. This value of K should be used for all predictions.

The above content can be understood more intuitively using our free course – [K-Nearest Neighbors (KNN) Algorithm in Python and R](https://courses.analyticsvidhya.com/courses/K-Nearest-Neighbors-KNN-Algorithm?utm_source=blog&utm_medium=knn_in_python&R)

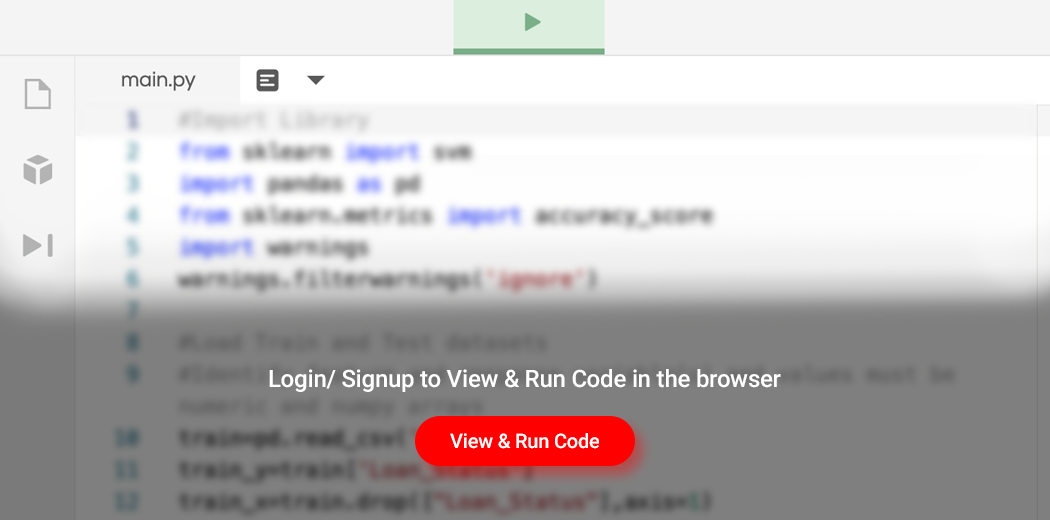
Breaking It Down – Pseudo Code of KNN

We can implement a KNN model by following the below steps:

1. Load the data
2. Initialise the value of k
3. For getting the predicted class, iterate from 1 to total number of training data points
   1. Calculate the distance between test data and each row of training dataset. Here we will use Euclidean distance as our distance metric since it’s the most popular method. The other distance function or metrics that can be used are Manhattan distance, Minkowski distance, Chebyshev, cosine, etc. If there are categorical variables, hamming distance can be used.
   2. Sort the calculated distances in ascending order based on distance values
   3. Get top k rows from the sorted array
   4. Get the most frequent class of these rows
   5. Return the predicted class

Implementation in Python From Scratch

We will be using the popular Iris dataset for building our KNN model. You can download it from [here](https://gist.githubusercontent.com/gurchetan1000/ec90a0a8004927e57c24b20a6f8c8d35/raw/fcd83b35021a4c1d7f1f1d5dc83c07c8ffc0d3e2/iris.csv).

[](https://id.analyticsvidhya.com/auth/login/?next=https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/?&utm_source=coding-window-blog&source=coding-window-blog)

Comparing Our Model With Scikit-learn

from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n\_neighbors=3)

neigh.fit(data.iloc[:,0:4], data['Name'])

# Predicted class

print(neigh.predict(test))

-> ['Iris-virginica']

# 3 nearest neighbors

print(neigh.kneighbors(test)[1])

-> [[141 139 120]]

We can see that both the models predicted the same class (‘Iris-virginica’) and the same nearest neighbors ( [141 139 120] ). Hence we can conclude that our model runs as expected.

Implementation of KNN in R

**Step 1: Importing the data**  
**Step 2: Checking the data and calculating the data summary**

|  |  |
| --- | --- |
|  | data<-read.table(file.choose(), header = T, sep = ",", dec = ".")#Importing the data |
|  | head(data) #Top observations present in the data |
|  | dim(data) #Check the dimensions of the data |
|  | summary(data) #Summarise the data |

[view raw](https://gist.github.com/Harshit1694/bfc073fa1c57767c99a0b8e49fdf7ef0/raw/698f09e0156405053d62270dd0e989a6e16a6478/import_knn.R)[import\_knn.R](https://gist.github.com/Harshit1694/bfc073fa1c57767c99a0b8e49fdf7ef0#file-import_knn-r)hosted with  by [GitHub](https://github.com/)

**Output**

#Top observations present in the data

SepalLength SepalWidth PetalLength PetalWidth Name

1 5.1 3.5 1.4 0.2 Iris-setosa

2 4.9 3.0 1.4 0.2 Iris-setosa

3 4.7 3.2 1.3 0.2 Iris-setosa

4 4.6 3.1 1.5 0.2 Iris-setosa

5 5.0 3.6 1.4 0.2 Iris-setosa

6 5.4 3.9 1.7 0.4 Iris-setosa

#Check the dimensions of the data

[1] 150 5

#Summarise the data

SepalLength SepalWidth PetalLength PetalWidth Name

Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100 Iris-setosa :50

1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300 Iris-versicolor:50

Median :5.800 Median :3.000 Median :4.350 Median :1.300 Iris-virginica :50

Mean :5.843 Mean :3.054 Mean :3.759 Mean :1.199

3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800

Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500

**Step 3: Splitting the Data**

|  |  |
| --- | --- |
|  | #Splitting the data set into train and test |
|  | set.seed(2) |
|  |  |
|  | part <- sample(2, nrow(data), replace = TRUE, prob = c(0.7, 0.3)) |
|  |  |
|  | train<- data[part == 1,] |
|  |  |
|  | test<- data[part == 2,] |

[view raw](https://gist.github.com/Harshit1694/449771ddaa71af871bb85f14d9c5c743/raw/b21c3822d146600bd20ff111bcd149cfc2d40e0d/split_kNN.R)[split\_kNN.R](https://gist.github.com/Harshit1694/449771ddaa71af871bb85f14d9c5c743#file-split_knn-r)hosted with  by [GitHub](https://github.com/)

**Step 4: Calculating the Euclidean Distance**

|  |  |
| --- | --- |
|  | #Calculating the euclidean distance |
|  |  |
|  | ED<-function(data1,data2){ |
|  | distance=0 |
|  | for (i in (1:(length(data1)-1))){ |
|  | distance=distance+(data1[i]-data2[i])^2 |
|  | } |
|  | return(sqrt(distance)) |
|  | } |

[view raw](https://gist.github.com/Harshit1694/e7f45054499e015ff75034958ff7d40c/raw/d517bfa117cc6838303f0ee0df3de57fe9cdc53c/euc_kNN.R)[euc\_kNN.R](https://gist.github.com/Harshit1694/e7f45054499e015ff75034958ff7d40c#file-euc_knn-r)hosted with  by [GitHub](https://github.com/)

**Step 5: Writing the function to predict kNN**  
**Step 6: Calculating the label(Name) for K=1**

|  |  |
| --- | --- |
|  | #Writing the function to predict kNN |
|  | knn\_predict <- function(test, train, k\_value){ |
|  | pred <- c() |
|  | #LOOP-1 |
|  | for(i in c(1:nrow(test))){ |
|  | dist = c() |
|  | char = c() |
|  | setosa =0 |
|  | versicolor = 0 |
|  | virginica = 0 |
|  | } |
|  |  |
|  | #LOOP-2-looping over train data |
|  | for(j in c(1:nrow(train))){} |
|  |  |
|  | dist <- c(dist, ED(test[i,], train[j,])) |
|  | char <- c(char, as.character(train[j,][[5]])) |
|  |  |
|  |  |
|  | df <- data.frame(char, dist$SepalLength) |
|  | df <- df[order(df$dist.SepalLength),] #sorting dataframe |
|  | df <- df[1:k\_value,] |
|  |  |
|  |  |
|  | #Loop 3: loops over df and counts classes of neibhors. |
|  | for(k in c(1:nrow(df))){ |
|  | if(as.character(df[k,"char"]) == "setosa"){ |
|  | setosa = setosa + 1 |
|  | }else if(as.character(df[k,"char"]) == "versicolor"){ |
|  | versicolor = versicolor + 1 |
|  | }else |
|  | virginica = virginica + 1 |
|  | } |
|  |  |
|  |  |
|  | n<-table(df$char) |
|  | pred=names(n)[which(n==max(n))] |
|  |  |
|  | return(pred) #return prediction vector |
|  | } |
|  |  |
|  | #Predicting the value for K=1 |
|  | K=1 |
|  | predictions <- knn\_predict(test, train, K) |

[view raw](https://gist.github.com/Harshit1694/e784618fd626ea3161523fbfdae47d19/raw/5f9395c6b227d1bf1950d7159e8afe1a94ec25e7/kNN_func.R)[kNN\_func.R](https://gist.github.com/Harshit1694/e784618fd626ea3161523fbfdae47d19#file-knn_func-r)hosted with  by [GitHub](https://github.com/)

**Output**

For K=1

[1] "Iris-virginica"

**In the same way, you can compute for other values of K.**

Comparing Our KNN Predictor Function With “Class” Library

|  |  |
| --- | --- |
|  | install.packages("class") |
|  | library(class) |
|  |  |
|  | #Normalization |
|  | normalize <- function(x) { |
|  | return ((x - min(x)) / (max(x) - min(x))) } |
|  | norm <- as.data.frame(lapply(data[,1:4], normalize)) |
|  |  |
|  | set.seed(123) |
|  | data\_spl <- sample(1:nrow(norm),size=nrow(norm)\*0.7,replace = FALSE) |
|  |  |
|  | train2 <- data[data\_spl,] # 70% training data |
|  | test2 <- data[-data\_spl,] # remaining 30% test data |
|  |  |
|  | train\_labels <- data[data\_spl,5] |
|  | test\_labels <-data[-data\_spl,5] |
|  | knn\_pred <- knn(train=train2, test=test2, cl=train\_labels, k=1) |

Conclusion

For K=1

[1] "Iris-virginica"

We can see that both models predicted the same class (‘Iris-virginica’).

Conclusion

The KNN algorithm is one of the simplest classification algorithms. Even with such simplicity, it can give highly competitive results. KNN algorithm can also be used for regression problems. The only difference from the discussed methodology will be using averages of nearest neighbors rather than voting from nearest neighbors. KNN can be coded in a single line on R. I am yet to explore how we can use the KNN algorithm on SAS.

**Key Takeaways**

* KNN classifier operates by finding the k nearest neighbors to a given data point, and it takes the majority vote to classify the data point.
* The value of k is crucial, and one needs to choose it wisely to prevent overfitting or underfitting the model.
* One can use cross-validation to select the optimal value of k for the k-NN algorithm, which helps improve its performance and prevent overfitting or underfitting. Cross-validation is also used to identify the outliers before applying the KNN algorithm.
* The above article provides implementations of KNN in Python and R, and it compares the result with scikit-learn and the “Class” library in R.

Frequently Asked Questions

**Q1. What is K nearest neighbors algorithm?**

A. KNN classifier is a machine learning algorithm used for classification and regression problems. It works by finding the K nearest points in the training dataset and uses their class to predict the class or value of a new data point. It can handle complex data and is also easy to implement, which is why KNN has become a popular tool in the field of artificial intelligence.

**Q2. What is KNN algorithm used for?**

A. KNN algorithm is most commonly used for:  
1. Disease prediction – predicts the likelihood of a disease based on symptoms and the data available.  
2. Handwriting recognition – recognizes the handwritten characters.  
3. Image classification – recognizes images in computer vision.

**Q3. What is the difference between KNN and Artificial Neural Networks?**

A. K-nearest neighbors (KNN) are mainly used for classification and regression problems, while Artificial Neural Networks (ANN) are used for complex function approximation and pattern recognition problems. Moreover, ANN has a higher computational cost than KNN.