A Complete Guide to K-Nearest Neighbors (Updated 2024)

ALGORITHM BIG DATA BUSINESS ANALYTICS CLASSIFICATION INTERMEDIATE MACHINE LEARNING PYTHON R STRUCTURED DATA SUPERVISED

Introduction

In the four years of my data science career, I have built more than 80% of classification models and just 15-20% of regression models. These ratios can be more or less generalized throughout the industry. The reason behind this bias towards classification models is that most analytical problems involve making decisions. In this article, we will talk about one such widely used machine learning **classification technique** called the k-nearest neighbors (KNN) algorithm. Our focus will primarily be on how the algorithm works on new data and how the input parameter affects the output/prediction.

Note: People who prefer to learn through videos can learn the same through our free course – K-Nearest Neighbors (KNN) Algorithm in Python and R. And if you are a complete beginner to Data Science and Machine Learning, check out our Certified BlackBelt program –

- Certified AI & ML Blackbelt+ Program

Learning Objectives

- Understand the working of KNN and how it operates in python and R.
- Get to know how to choose the right value of k for KNN
- Understand the difference between training error rate and validation error rate

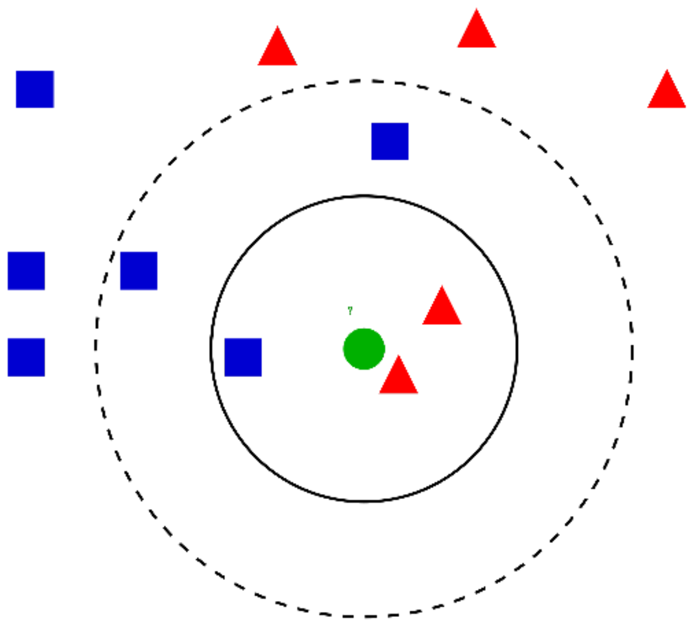


Table of contents

What is KNN (K-Nearest Neighbor) Algorithm?

The K-Nearest Neighbors (KNN) algorithm is a popular machine learning technique used for classification and regression tasks. It relies on the idea that similar data points tend to have similar labels or values.

During the training phase, the KNN algorithm stores the entire training dataset as a reference. When making predictions, it calculates the distance between the input data point and all the training examples, using a chosen distance metric such as Euclidean distance.

Next, the algorithm identifies the K nearest neighbors to the input data point based on their distances. In the case of classification, the algorithm assigns the most common class label among the K neighbors as the predicted label for the input data point. For regression, it calculates the average or weighted average of the target values of the K neighbors to predict the value for the input data point.

The KNN algorithm is straightforward and easy to understand, making it a popular choice in various domains. However, its performance can be affected by the choice of K and the distance metric, so careful parameter tuning is necessary for optimal results.

When Do We Use the KNN Algorithm?

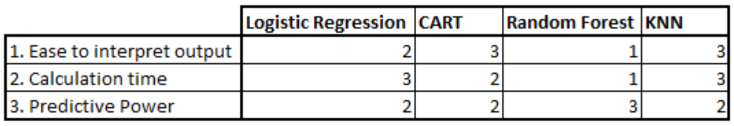
KNN Algorithm can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique, we generally look at 3 important aspects:

1. Ease of interpreting output

2. Calculation time

3. Predictive Power

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

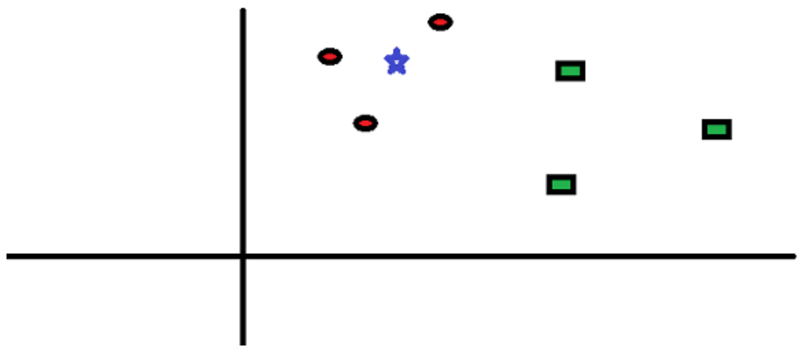


	Logistic Regression	CART	Random Forest	KNN
1. Ease to interpret output	2	3	1	3
2. Calculation time	3	2	1	3
3. Predictive Power	2	2	3	2

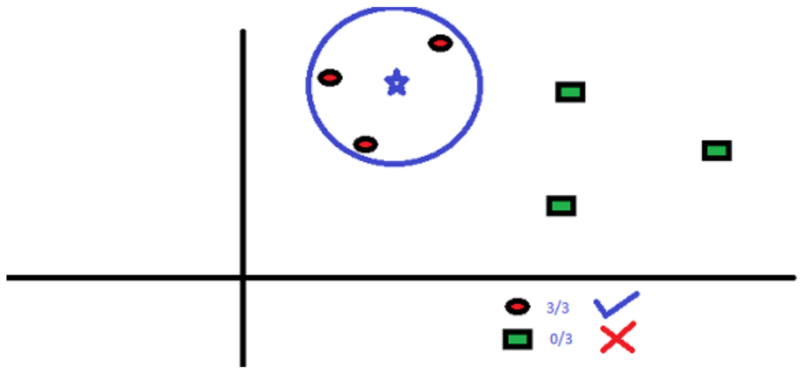
KNN classifier fairs across all parameters of consideration. It is commonly used for its ease of interpretation and low calculation time.

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):



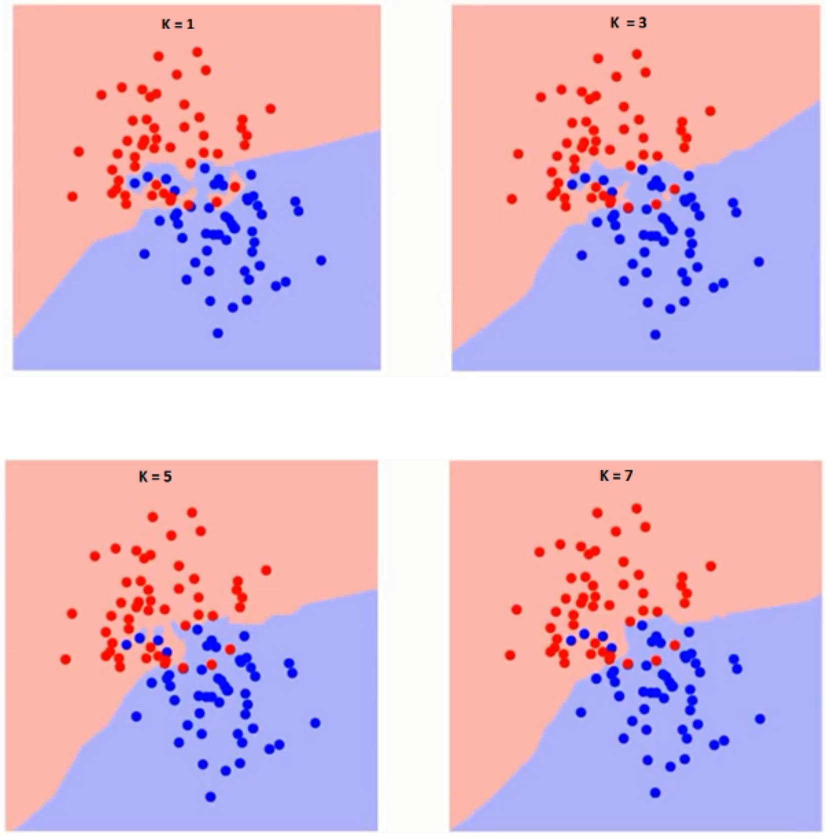
You intend to find out the class of the blue star (BS). BS can either be RC or GS and nothing else. The "K" in KNN algorithm is the nearest neighbor we wish to take the vote from. Let's say K = 3. Hence, we will now make a circle with BS as the center just as big as to enclose only three data points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



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.

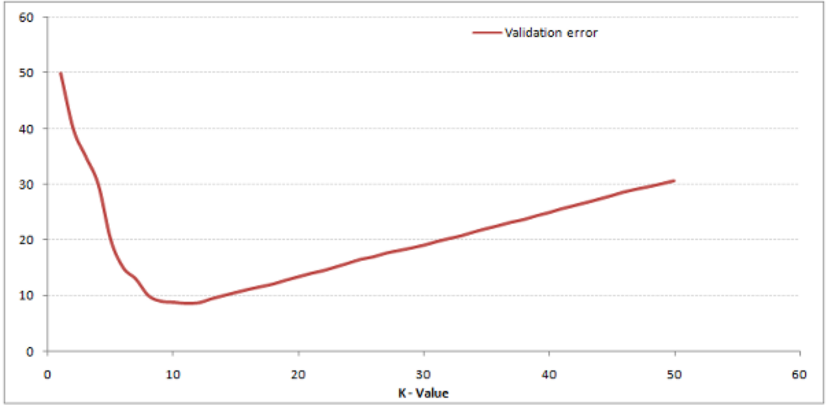
How Do We Choose the Factor K?

First, let us try to understand the influence of the K-nearest neighbors (KNN) in the algorithm. If we consider the last example, keeping all 6 training observations constant, a given K value allows us to establish boundaries for each class. These decision boundaries effectively segregate, for instance, RC from GS. Similarly, let's examine the impact of the value "K" on these class boundaries. The following illustrates the distinct boundaries that separate the two classes, each corresponding to different values of K.



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 :

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:



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

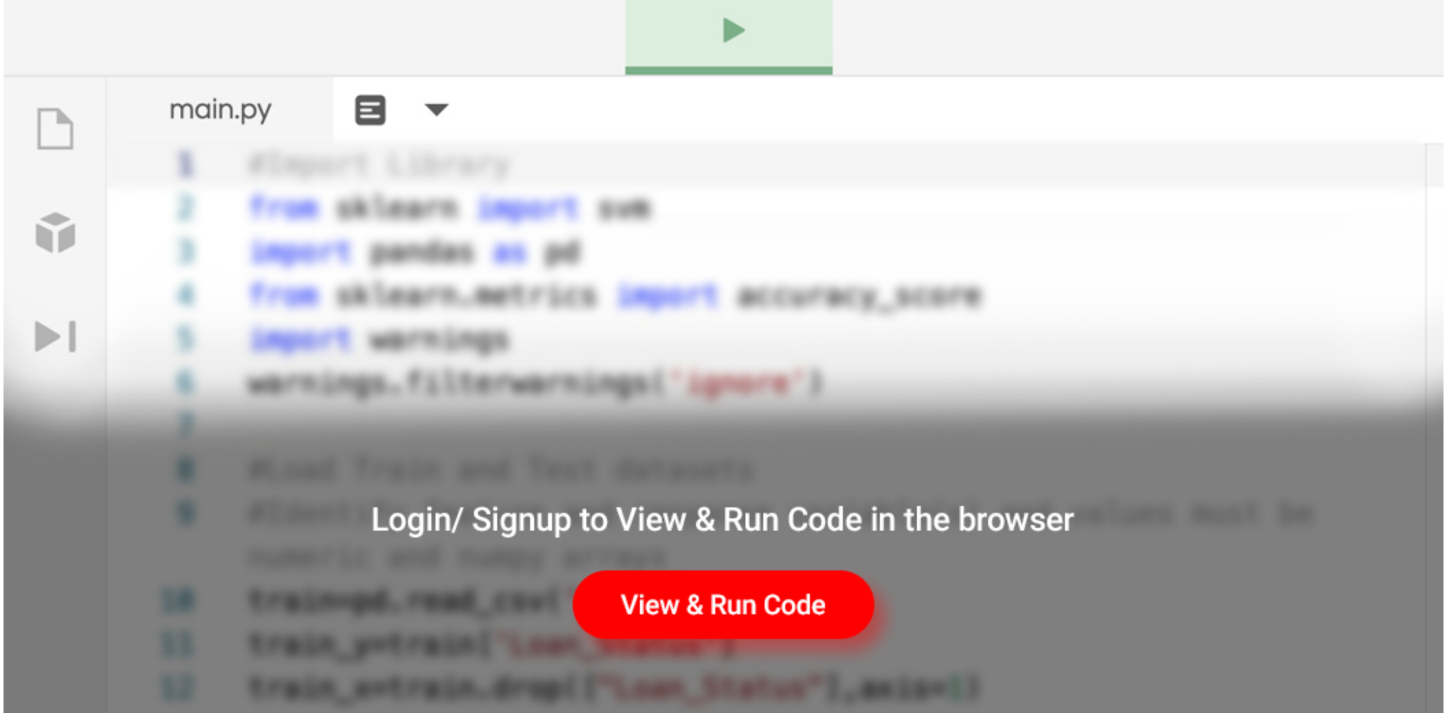
Breaking It Down – Pseudo Code of KNN

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
    - 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.
    - Sort the calculated distances in ascending order based on distance values
    - Get top k rows from the sorted array
    - Get the most frequent class of these rows
    - 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.



```
#Import Library
from sklearn import svm
import pandas as pd
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')

#Load Train and Test datasets
#Identi[illegible] values must be
numeric and numpy arrays
train=pd.read_csv(
train_y=train['Loan_Status']
train_x=train.drop(["Loan_Status"],axis=1)
```

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

import_knn.R hosted with ❤ by **GitHub**

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
```

```r
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

split_kNN.R hosted with ❤ by **GitHub**

Step 4: Calculating the Euclidean Distance

```r
#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

euc_kNN.R hosted with ❤ by **GitHub**

Step 5: Writing the function to predict kNN

Step 6: Calculating the label(Name) for K=1

```r
#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

kNN_func.R hosted with ❤ by **GitHub**

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

```r
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)
```

view raw

classlib_kNN.R hosted with ❤ by **GitHub**

Output

```
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 k-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.

Article Url - https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

Tavish Srivastava

Tavish Srivastava, co-founder and Chief Strategy Officer of Analytics Vidhya, is an IIT Madras graduate and a passionate data-science professional with 8+ years of diverse experience in markets including the US, India and Singapore, domains including Digital Acquisitions, Customer Servicing and Customer Management, and industry including Retail Banking, Credit Cards and Insurance. He is fascinated by the idea of artificial intelligence inspired by human intelligence and enjoys every discussion, theory or even movie related to this idea.