BIG DATA (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/BIG-DATA/)

BUSINESS ANALYTICS (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/BUSINESS-ANALYTICS/)

MACHINE LEARNING (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/MACHINE-LEARNING/)

PYTHON (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PYTHON-2/)

Introduction to k-Nearest Neighbors: Simplified (with implementation in Python)

TAVISH SRIVASTAVA (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/TAVISH1/), MARCH 26, 2018

Note: This article was originally published on Oct 10, 2014 and updated on Mar 27th, 2018

Introduction

In four years of my career into analytics I have built more than 80% of classification models and just 15-20% regression models. These ratios can be more or less generalized throughout the industry. The reason of a bias towards classification models is that most analytical problem involves making a decision. For instance will a customer attrite or not, should we target customer X for digital campaigns, whether customer has a high potential or not etc. These analysis are more insightful and directly links to an implementation roadmap. In

this article, we will talk about another widely used classification technique called K-nearest neighbors (KNN) . Our focus will be primarily on how does the algorithm work and how does the input parameter effect the output/prediction.

Table of Contents

When do we use 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 to interpret output

2. Calculation time

3. Predictive Power

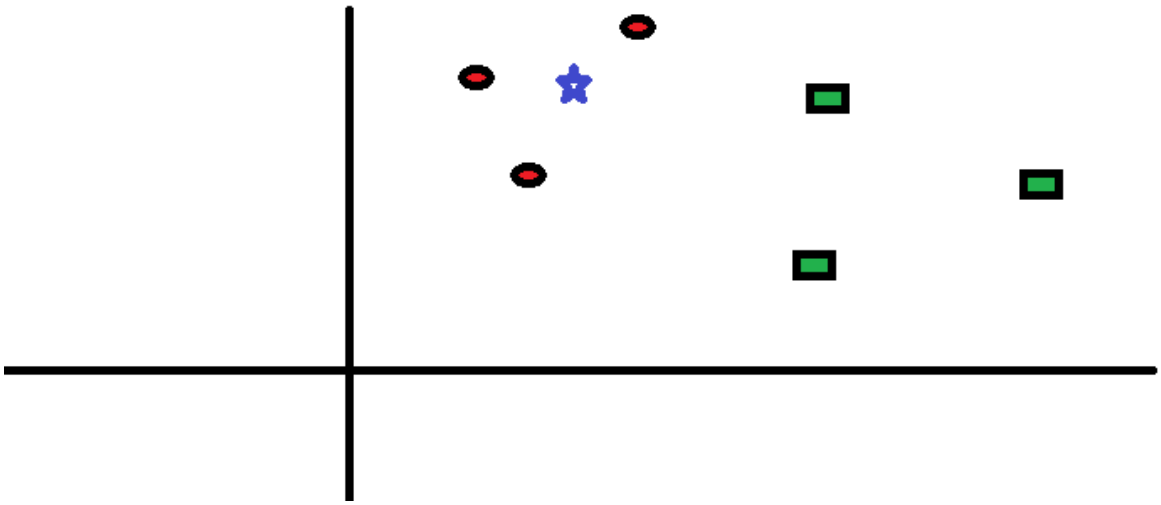
Let us take a few examples to place KNN in the scale :

	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

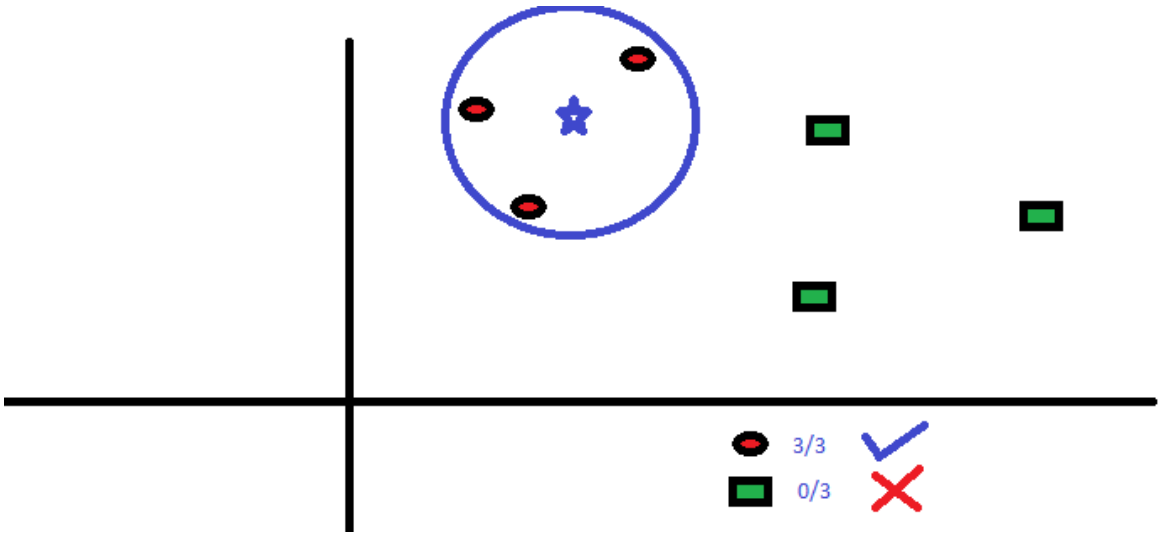
(https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/Model-comparison.png)KNN algorithm fairs across all parameters of considerations. It is commonly used for its easy of interpretation and low calculation time.

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) :



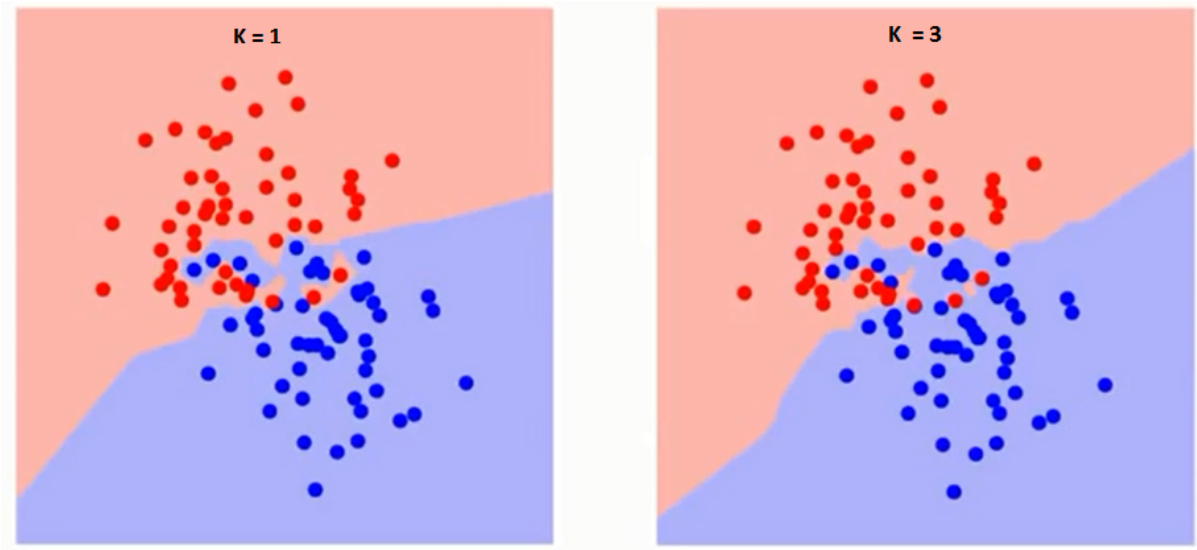
(https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/scenario1.png)You intend to find out the class of the blue star (BS) . BS can either be RC or GS and nothing else. The "K" is KNN algorithm is the nearest neighbors we wish to take vote from. Let's say K = 3. Hence, we will now make a circle with BS as center just as big as to enclose only three datapoints on the plane. Refer to following diagram for more details:



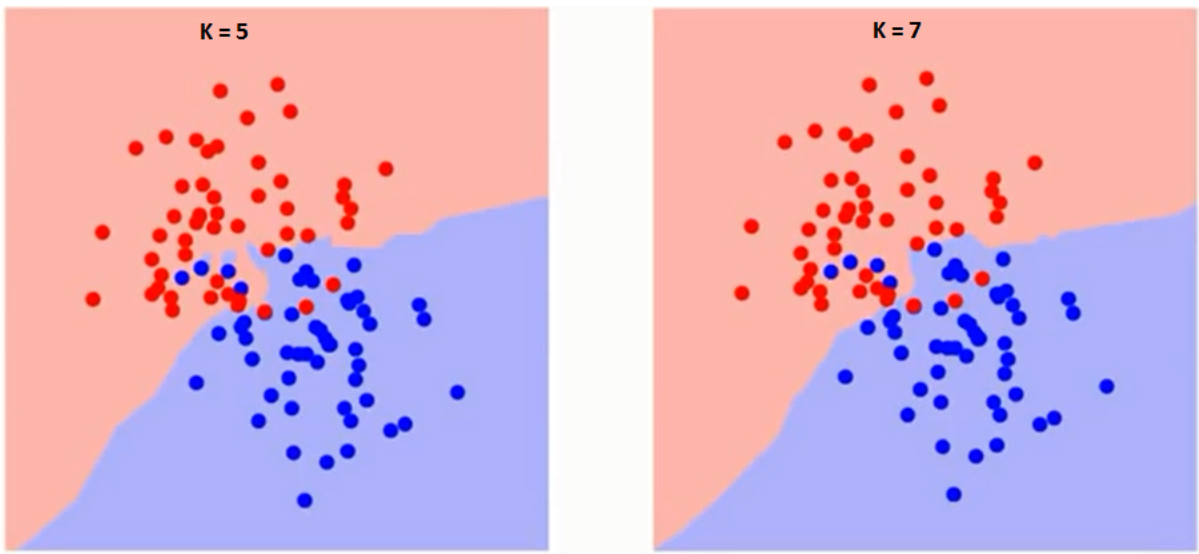
(https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/scenario2.png) The three closest points to BS is all RC. Hence, with good confidence level we can say that the BS should belong to the class RC. Here, the choice became very obvious as all three votes from the closest neighbor went to RC. The choice of the parameter K is very crucial in this algorithm. Next we will understand what are the factors to be considered to conclude the best K.

How do we choose the factor K?

First let us try to understand what exactly does 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 boundaries will segregate RC from GS. The same way, let's try to see the effect of value "K" on the class boundaries. Following are the different boundaries separating the two classes with different values of K.

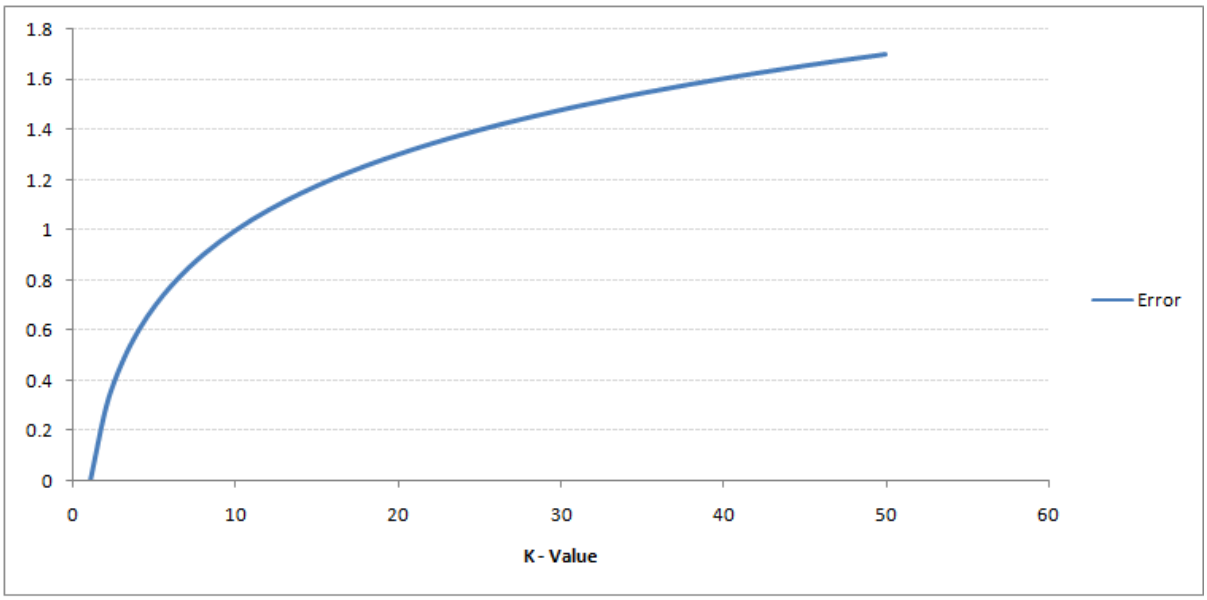


(https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/K-judgement.png)

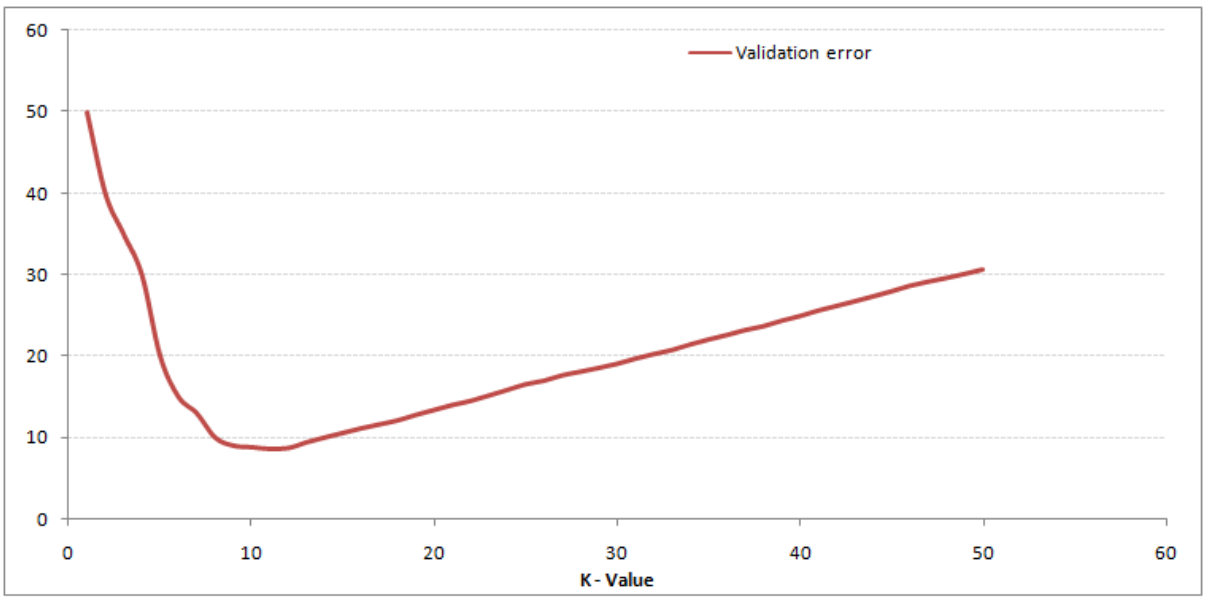


(https://www.analyticsvidhya.com/blog/wp-content/uploads/2014/10/K-judgement2.png)

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 on different K-value. Following is the curve for the training error rate with varying value of K :



(https://www.analyticsvidhya.com/blog/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/blog/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.

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 data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
    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/fcd83b35021a4c

```
# Importing libraries
import pandas as pd
import numpy as np
import math
import operator
```

```
#### Start of STEP 1
# Importing data
data = pd.read_csv("iris.csv")
#### End of STEP 1

data.head()
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
# Defining a function which calculates euclidean distance between two data points
def euclideanDistance(data1, data2, length):
    distance = 0
    for x in range(length):
        distance += np.square(data1[x] - data2[x])
    return np.sqrt(distance)

# Defining our KNN model
def knn(trainingSet, testInstance, k):

    distances = {}
    sort = {}

    length = testInstance.shape[1]

    #### Start of STEP 3
    # Calculating euclidean distance between each row of training data and test data
    for x in range(len(trainingSet)):

        #### Start of STEP 3.1
        dist = euclideanDistance(testInstance, trainingSet.iloc[x], length)

        distances[x] = dist[0]
        #### End of STEP 3.1

    #### Start of STEP 3.2
    # Sorting them on the basis of distance
    sorted_d = sorted(distances.items(), key=operator.itemgetter(1))
    #### End of STEP 3.2

    neighbors = []

    #### Start of STEP 3.3
    # Extracting top k neighbors
    for x in range(k):
        neighbors.append(sorted_d[x][0])
```

```
    #### End of STEP 3.3
    classVotes = {}

    #### Start of STEP 3.4
    # Calculating the most freq class in the neighbors
    for x in range(len(neighbors)):
        response = trainingSet.iloc[neighbors[x]][-1]

        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    #### End of STEP 3.4

    #### Start of STEP 3.5
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
    return(sortedVotes[0][0], neighbors)
    #### End of STEP 3.5
```

```
# Creating a dummy testset
testSet = [[7.2, 3.6, 5.1, 2.5]]
test = pd.DataFrame(testSet)
```

```
#### Start of STEP 2
# Setting number of neighbors = 1
k = 1
#### End of STEP 2
# Running KNN model
result,neigh = knn(data, test, k)

# Predicted class
print(result)
-> Iris-virginica
```

```
# Nearest neighbor
print(neigh)
-> [141]
```

Now we will try to alter the *k* values, and see how the prediction changes.

```
# Setting number of neighbors = 3
k = 3
# Running KNN model
result,neigh = knn(data, test, k)
# Predicted class
print(result) -> Iris-virginica
```

```
# 3 nearest neighbors
print(neigh)
-> [141, 139, 120]
```

```
# Setting number of neighbors = 5
k = 5
# Running KNN model
result,neigh = knn(data, test, k)
# Predicted class
print(result) -> Iris-virginica
```

```
# 5 nearest neighbors
print(neigh)
-> [141, 139, 120, 145, 144]
```

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.

End Notes

KNN algorithm is one of the simplest classification algorithm. 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 can we use KNN algorithm on SAS.

Did you find the article useful? Have you used any other machine learning tool recently? Do you plan to use KNN in any of your business problems? If yes, share with us how you plan to go about it.

If you like what you just read & want to continue your analytics learning, subscribe to our emails (http://feedburner.google.com/fb/a/mailverify?uri=analyticsvidhya), follow us on twitter (http://twitter.com/analyticsvidhya) or like our facebook page (http://facebook.com/analyticsvidhya).