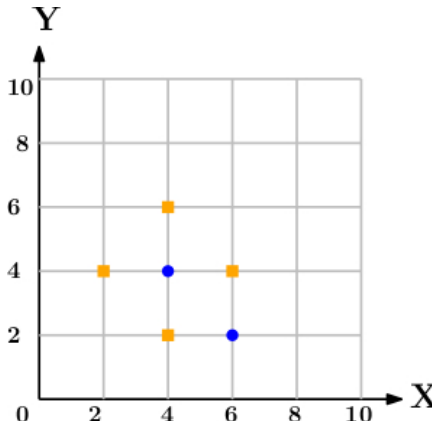
Assignment no: 3

Title:

k-NN Classification

Problem definition:

In the following diagram let blue circles indicate positive examples and orange squares indicate negative examples. We want to use k-NN algorithm for classifying the points. If k=3, find the class of the point (6,6). Extend the same example for Distance-Weighted k-NN and Locally weighted Averaging .



Prerequisite:

Basic Python programming, Concept of Decision Tree Classifier

Software Requirements:

Jupyter notebook python 2.7/3.5, ubuntu OS

Hardware Requirement:

2GB RAM, 500 GB HDD, Laptop or pc

Objectives:

Learn How to Apply KNN Classification for Classify Positive and Negative Points in given example.

Outcome:

After completion of this assignment we are able Implement code for KNN Classification for Classify Positive and Negative Points in given example also and find the class of the point (6,6)

Theory:

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

KNN algorithm fairs across all parameters of considerations. 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) :

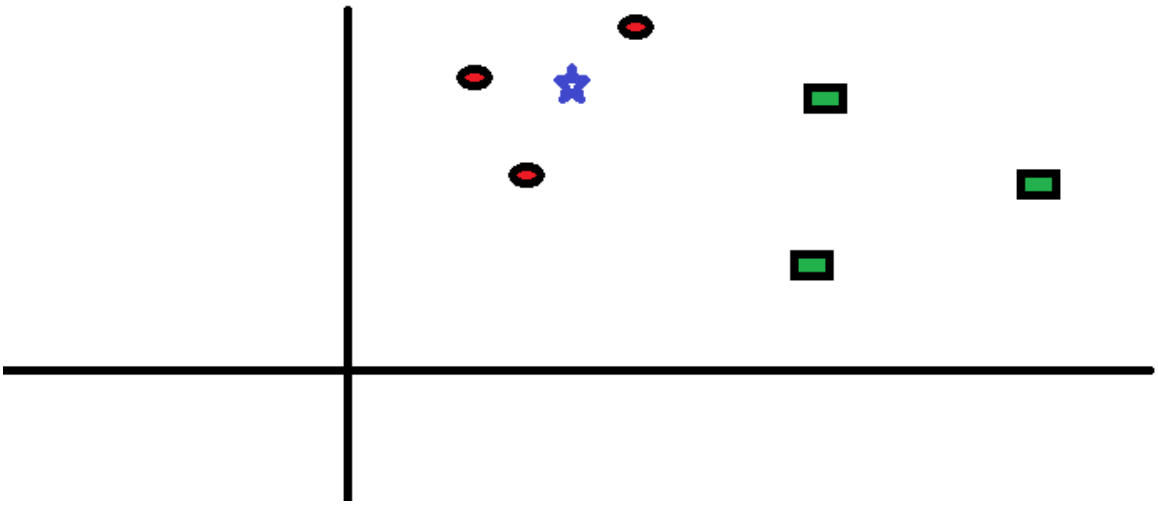


Fig.1

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 data points on the plane. Refer to following diagram for more details:

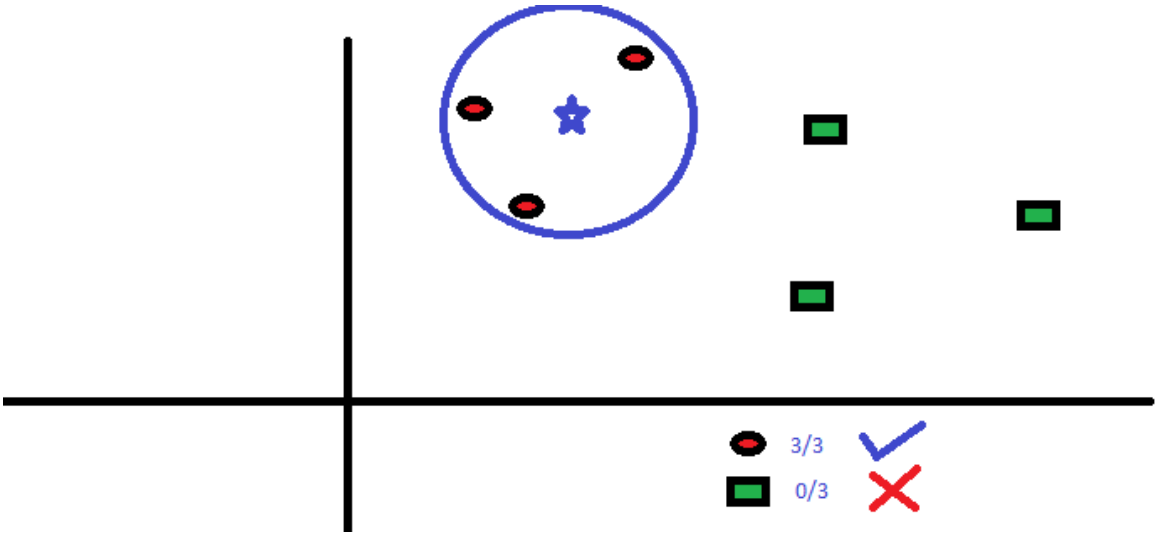


Fig.2

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.

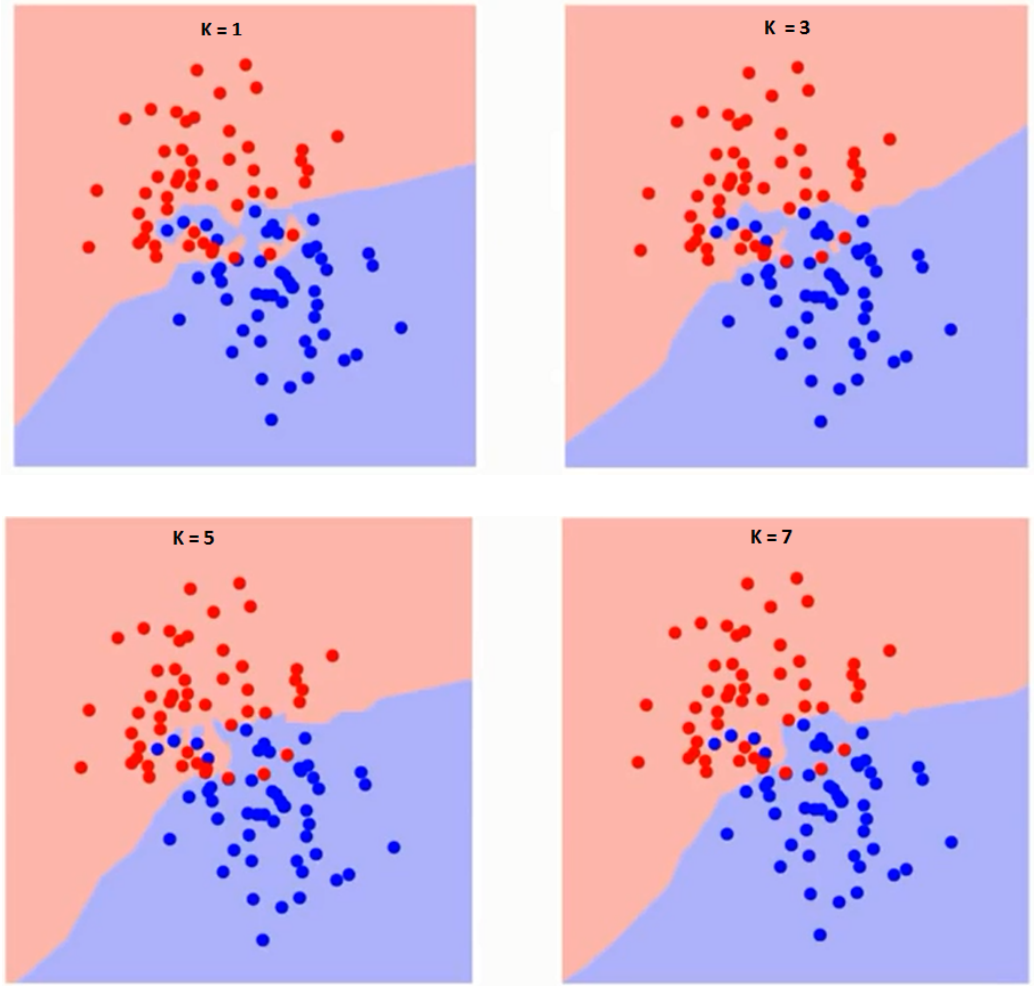


Fig.3

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 :

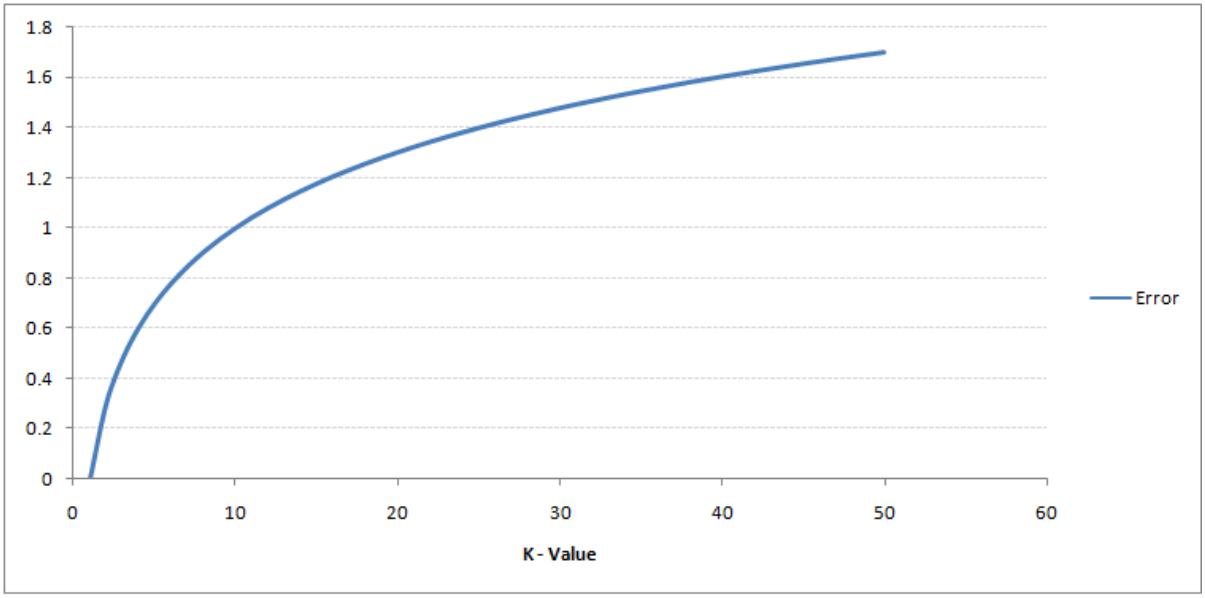


Fig.4

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:

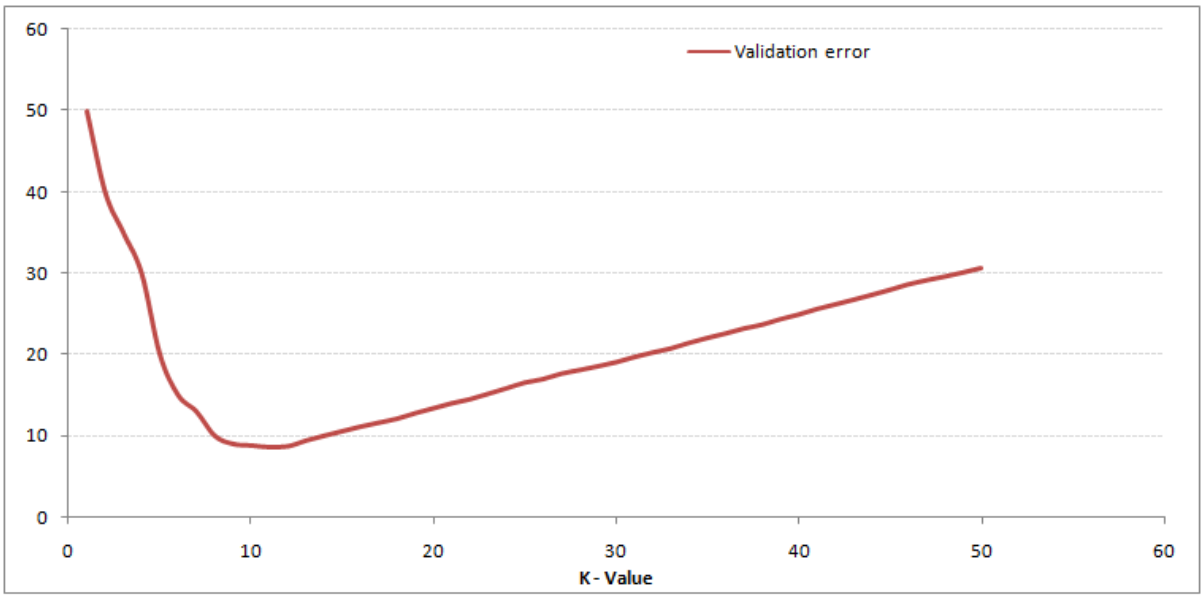


Fig.5

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.

scikit-learn implements two different nearest neighbors classifiers: **KneighborsClassifier** implements learning based on the k-nearest neighbors of each query point, where k is an integer value specified by the user.

RadiusNeighborsClassifier implements learning based on the number of neighbors within a fixed radius r of each training point, where r is a floating-point value specified by the user.

The K neighbors classification in **KneighborsClassifier** is the most commonly used technique. The optimal choice of the value k is highly data-dependent: in general a larger k suppresses the effects of noise, but makes the classification boundaries less distinct.

In cases where the data is not uniformly sampled, radius-based neighbors classification in **RadiusNeighborsClassifier** can be a better choice. The user specifies a fixed radius r, such that

points in sparser neighborhoods use fewer nearest neighbors for the classification. For high-dimensional parameter spaces, this method becomes less effective due to the so-called “curse of dimensionality”.

Breaking it Down – Pseudo Code of KNN

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

- Load the data
- Initialize the value of k
- 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 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.
    - 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

Conclusion:

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.

KNN.py

```
import pandas as pd
import numpy as np
#Read dataset
dataset=pd.read_csv('dataset3.csv')
X=dataset.iloc[:,:-1].values
y=dataset.iloc[:,2].values
#import KNeighborshood Classifier and create object of it
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=3)
classifier.fit(X,y)
#predict the class for the point(6,6)
X_test=np.array([6,6])
y_pred=classifier.predict([X_test])
print('General KNN',y_pred)
classifier=KNeighborsClassifier(n_neighbors=3,weights='distance')
classifier.fit(X,y)
#predict the class for the point(6,6)
X_test=np.array([6,2])
y_pred=classifier.predict([X_test])
print('Distance Weighted KNN',y_pred)
```

output:

python KNN.py

('General KNN', array(['negative'], dtype=object))

('Distance Weighted KNN', array(['positive'], dtype=object)