Comparison between K-Means Clustering & KNN Algorithm

Project submitted in partial fulfillment of requirements

For the degree of

B.Sc. in Computer Science under CBCS

of

Computer Science Department of Asutosh College

of

University of Calcutta

Submitted by

Subham De	183012-21-0104	012-1111-1315-18
Santanu Mondal	183012-21-0136	012-1115-0977-18
Sagnik Dutta	183012-21-0134	012-1114-1086-18

under the supervision of

Tonmoy Mete

Assistant Professor

Department of Computer Science

Asutosh College

University of Calcutta

Contents

SUPERVISOR'S CERTIFICATE

This is to certify that Sagnik Dutta, Shantanu Mondol and Subham De, student of B.Sc. Honors in Computer Science of Asutosh College under the University of Calcutta was under my supervision and guidance for their project work and prepared a project report with the title Comparison between K-means clustering & KNN algorithm using Python which she is submitting is her genuine and original work to the best of knowledge.

Place: Kolkata

Date:

Signature:

Name: Tanmoy Mete

Designation: S.A.C.T - I

Introduction

Machine Learning is a self explanatory keyword. The term is made up of two words, “Machine” and “Learning”. So now the question arises that how we make a machine learn. Well there are a numerous answer to that varying from person to person. But the two techniques that have been used for Machine Learning are:

1. Learn for Past Interactions
2. Learn from pre-defined algorithms

Both these techniques work very well.

Now the question is why we need learned machines when we are the human being, the greatest creature known? Well as we are the greatest creature known to ourselves in the whole universe, we thrive for knowledge and innovation. We first built machines to solve our mathematical calculations, then to store data for future use. But we didn’t stop at that. We kept on going to make a better version of those machines, we kept on going to make them one of us, to make them the future of mankind. For that, we needed to make intelligence out of our bare minds. We made them. We made artificial intelligence by the help of machine learning. We built machines that can learn, grow, and solve real world problems. We built the human inside the machine. The term Machine Learning was coined by Arthur Samuel in 1959, an American pioneer in the field of computer gaming and artificial intelligence, and stated that “it gives computers the ability to learn without being explicitly programmed”.

And in 1997, Tom Mitchell gave a “well-posed” mathematical and relational definition that “A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Background

In the field of Machine Learning we are novice. So coming to a new world of the same region is quite exciting. So like every other newcomer we tried to fit into the field by simply going through Youtube videos and Online Web contents about how to implement these two algorithms without knowing how actually they work. We saw that in the real world, these two algorithms are implanted using various libraies of python like “sklearn”, “pandas”, “numpy” etc. But starting with a new scope of computer science, starting with two completely new algorithms we weren’t totally ready for using those libraries for our work. Therefore we decided that we have to implement these algorithms from scratch. So we used our prior knowledge about python programming and implemented the core and naked structure of these two algorithms.

Now let’s get familiar with these algorithms and know how they work.

K-Means Clustering

Introduction:

Data has an important role in human activities. Data mining is a knowledge discovery process by analyzing large volumes of data from various perspectives and organizing them into useful information. Huge amount of data is generated in many organizations in a day. Data mining techniques are used to search for valuable information and identify hidden structures in large volumes of data. Organizations are now starting to realize the importance of data mining.

What is Clustering?

Clustering is the process of dividing data set into groups, consisting of similar data-points.

- Points in the same groups are as similar as possible.
- Points in different group are as different as possible.

Use of clustering:

In the field of Machine Learning, Data mining, or Data Analysis, we often find ourselves working with a big dataset. Working with a big dataset is hectic and often non-optimized also. Therefore we need to cluster those datasets into smaller groups as stated above. That's why we need clustering algorithm. Application of clustering is everywhere nowadays, like in:

1. Amazon recommendation System
2. Netflix recommendation of movies
3. Classification of people being affected by Covid-19
4. Clustering the products of an online store

What is K-means clustering?

K-means is a clustering algorithm whose main goals is to group similar element or data point into a cluster.
'K' in k-means represents the number of clusters.

K-means is an algorithm which is a part of partitioned clustering. The k-means algorithm can group data into k number of categories. K-means is a method of cluster analysis. Its aim is to partition n observations into k clusters and each observation will be a part of any one cluster with the nearest mean.

Working of K-means clustering Algorithm:

This algorithm takes the unlabelled/unorganized dataset as input, divides the dataset into k-number of clusters or k number of classes, and repeats the process until it does not find the best clusters. The value of k must be predefined or must be given at runtime.

Method:

1. First, we define the value of k randomly.
2. We define k no. of means which will initially be considered as the centroids of the clusters.
3. Depending upon the closest mean of a data point, the data point is clustered in one of the k clusters.
4. After clustering all the data points, we repeat this technique by changing the value of the k means, until the variation of the means from the last position is nil or negligible.

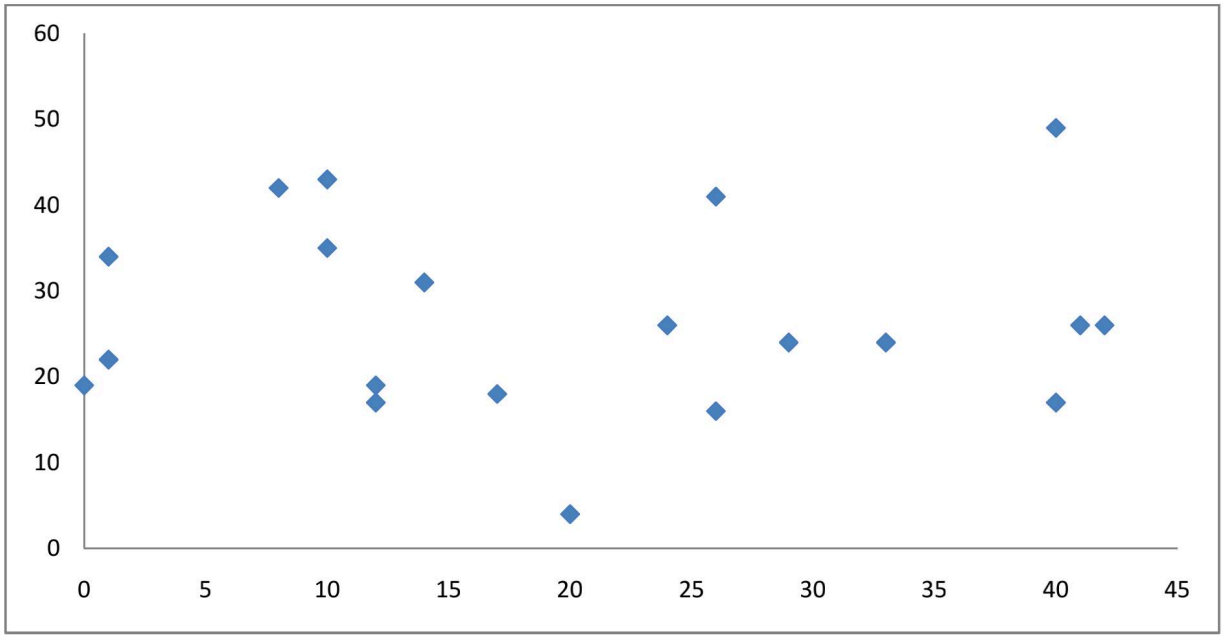
Algorithm:

Step-1: Define the value of K to decide the number of clusters.
Step-2: Select K no. of means randomly. It can differ from the input dataset.
Step-3: Now for each data point find their closest mean.
Step-4: Once the closest mean is found, assign the data point to the cluster of that mean.
Step-5: This process will follow for every single data points in the dataset and will cluster each data depending upon the given initial mean values.
Step-6: Now it will recompute the mean values with the help of the data points of each cluster.

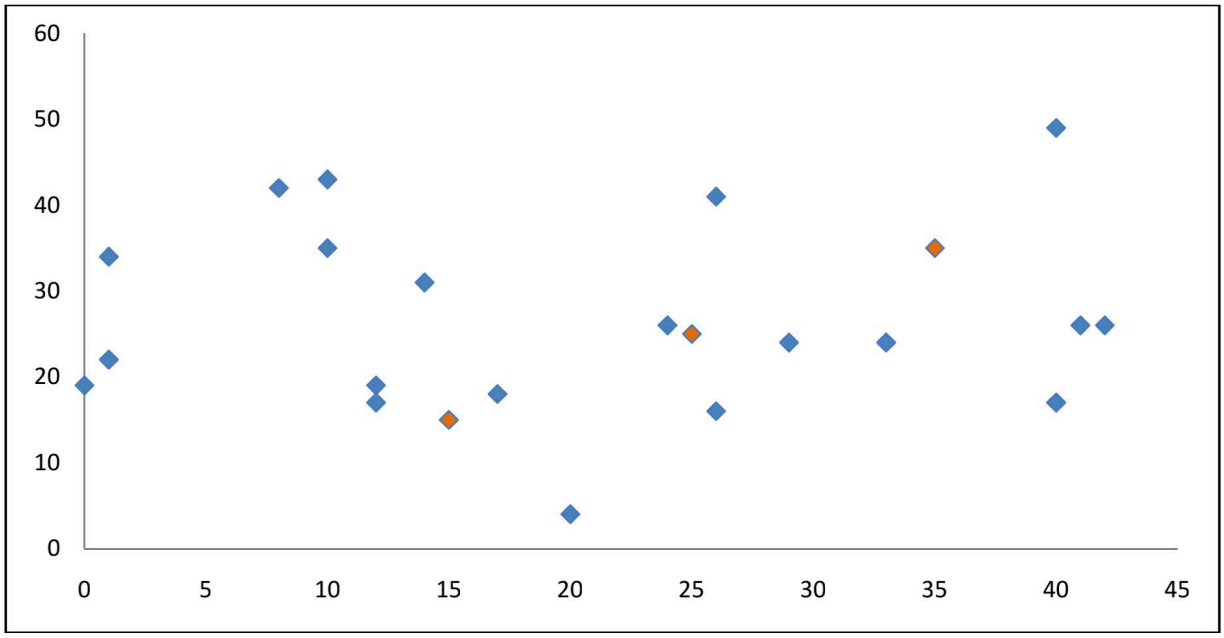
$$Mean_i = \frac{\sum d_i}{n}\ ;\ n\text{=no. of elements in cluster } i$$

Step-7: If the new mean values differ from the older mean values, it will repeat from step 3 to step 6, else it will return the clusters.

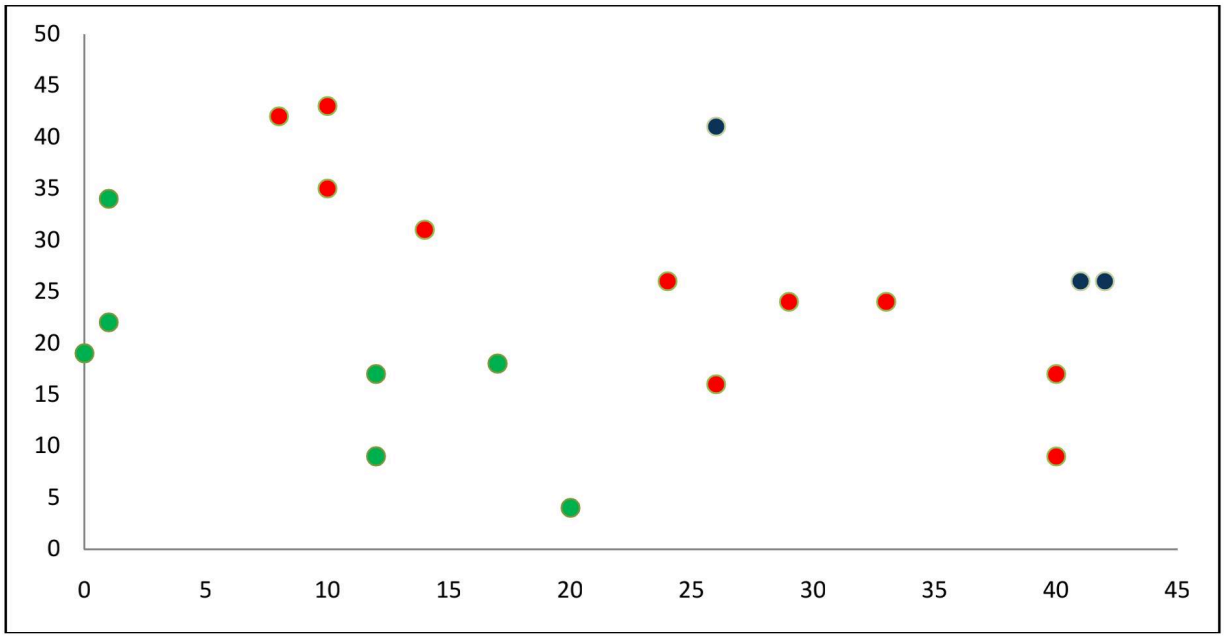
Example:



Suppose we have a dataset of spatial coordinates like the above figure. We need to cluster these points into 3 different classes. Now for that we need to define three different means.



Now we can find the nearest mean points of all the data points, depending upon which we can cluster those points to a certain cluster. By this technique we find the following clusters.



Now we will recompute the mean values with respect to their cluster elements using the previously mentioned formula. But here in this example we have chose such 3 mean value that at the second execution it will fail to cluster these values into 3 different clusters. Thus this is the optimal solution we can get from these mean values. That's why the efficiency of K-Means depends upon the defined mean values.

Source Code:

```
from math import sqrt
import random

class KMEANS:
	#Function for the K-Means algorithm
	def kmeans(self):
		print("The dataset is:")
		print(*self.data, sep = "\n")
		self.k = int(input("No. of clusters (K) = "))
		c = [
			[] for _ in range(self.k)
		]
		print("Enter ",self.k," centroids as follows:")
		for i in range(self.k):
			x,y = input("centroid {0} = " .format(i+1)).split(",")
			c[i] = [int(x),int(y)]
		print("The initial centroids are : ", *c, sep = "\n")
		cluster, length = self.clustering(c)
		ncluster = pcluster = cluster
		for n in range(10):
			print("\n\ncluster  {}:\n" .format(n))
			print(*cluster,sep="\n")
			if(0  in length):
				print("\noptimal position reached:")
				print("\nThe optimal clusters are:" .format(n))
				print(*pcluster, sep="\n")
				break
			else:
				cnew = self.means(ncluster, length)
				if c==cnew:
					print("\noptimal position reached:")
					print("\nThe optimal clusters are:" .format(n))
					print(*ncluster, sep="\n")
					break;
				else:
					c=cnew
					pcluster = ncluster
					ncluster,length =  self.clustering(c)

		return cluster

	#Function for clustering the dataset
	def clustering(self,c):
		cl=[
			[]	for _  in range(self.k)
		]
		length=[]
		for i in range(self.k):
			l=0
			for j in self.data:
```

```
                                if(self.check(j,c)==i):
                                        cl[i].append(j)
                                        l+=1
                        length.append(l)
                return cl, length

        #Function for finding new centroids
        def means(self,cluster,length):
                c=[[ [] for _ in range(2) ] for _ in range (self.k)]
                sx=sy=0
                for i in range(len(c)):
                        for j in range(length[i]):
                                sx,sy = sx+cluster[i][j][0],sy+cluster[i][j][1]
                        c[i] = [sx/length[i],sy/length[i]]
                return c
        '''Function for finding the cluster number
        for a certain point'''
        def check(self,point,c):
                d=[]
                [x,y]=point[:]
                for i in range(len(c)):
                        [x1,y1] = c[i]
                        d.append(sqrt(((x-x1)**2)+((y-y1)**2)))
                return d.index(min(d))

k1 = KMEANS()
k1.data = [
        [random.randrange(50),random.randrange(50)] for _ in range(20)
];
k1.cluster = k1.kmeans()
# print("The clusters after {0} repeatitions are: " .format(n))
#print(*k1.cluster, sep="\n")
```

Output:

```
H:\SD's Workspace\Project\Major Project\Resources>python KMeans.py
The dataset is:
[44, 36]
[25, 35]
[6, 6]
[48, 48]
[27, 14]
[49, 36]
[36, 31]
[19, 12]
[11, 25]
[28, 26]
[38, 19]
[34, 37]
[39, 34]
[13, 24]
```

```
[2, 29]
[19, 49]
[22, 34]
[2, 2]
[41, 32]
[42, 47]
No. of clusters (K) = 3
Enter  3  centroids as follows:
centroid 1 = 10,10
centroid 2 = 20,20
centroid 3 = 30,30
The initial centroids are :
[10, 10]
[20, 20]
[30, 30]


cluster  0:

[[6, 6], [2, 2]]
[[27, 14], [19, 12], [11, 25], [13, 24], [2, 29]]
[[44, 36], [25, 35], [48, 48], [49, 36], [36, 31], [28, 26], [38, 19], [34, 37], [39, 34], [19, 49], [22, 34], [41,
32], [42, 47]]


cluster  1:

[[6, 6], [2, 2]]
[[27, 14], [19, 12], [11, 25], [13, 24], [2, 29]]
[[44, 36], [25, 35], [48, 48], [49, 36], [36, 31], [28, 26], [38, 19], [34, 37], [39, 34], [19, 49], [22, 34], [41,
32], [42, 47]]


cluster  2:

[[6, 6], [2, 2]]
[[27, 14], [19, 12], [11, 25], [13, 24], [2, 29]]
[[44, 36], [25, 35], [48, 48], [49, 36], [36, 31], [28, 26], [38, 19], [34, 37], [39, 34], [19, 49], [22, 34], [41,
32], [42, 47]]


cluster  3:

[[6, 6], [2, 2]]
[[27, 14], [19, 12], [11, 25], [13, 24], [2, 29]]
[[44, 36], [25, 35], [48, 48], [49, 36], [36, 31], [28, 26], [38, 19], [34, 37], [39, 34], [19, 49], [22, 34], [41,
32], [42, 47]]

optimal position reached:

The optimal clusters are:
[[6, 6], [2, 2]]
[[44, 36], [25, 35], [27, 14], [49, 36], [36, 31], [19, 12], [11, 25], [28, 26], [38, 19], [34, 37], [39, 34], [13,
24], [2, 29], [19, 49], [22, 34], [41, 32]]
```

```
[[48, 48], [42, 47]]
```

Advantage:

- It is very easy to understand and implement.
- If we have large number of variables, K-Means may be computationally faster than hierarchical clustering (if K is small).
- An instance can change cluster (move to another cluster) when the centroids are recomputed.
- Tighter clusters are formed with K-means as compared to Hierarchical clustering.

Disadvantage:

- Difficult to predict the number of clusters (K-Value).
- Initial seeds have a strong impact on the final results.
- The order of the data has an impact on the final results.

K NEAREST NEIGHBOUR CLASSIFICATION

Introduction:

K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.

It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

The Supervised machine learning (The KNN algorithm) relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data.

Working of KNN Algorithm:

K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps –

Step 1 – For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 – Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

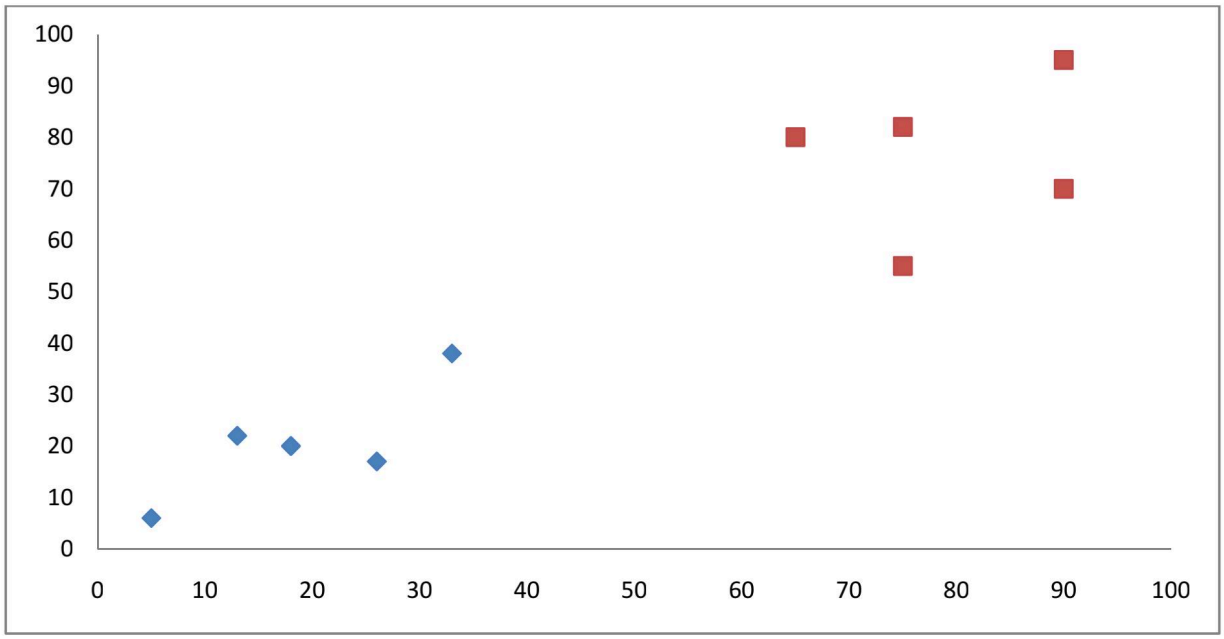
Step 3 – For each point in the test data do the following –

- **3.1** – Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
- **3.2** – Now, based on the distance value, sort them in ascending order.
- **3.3** – Next, it will choose the top K rows from the sorted array.
- **3.4** – Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4 – End

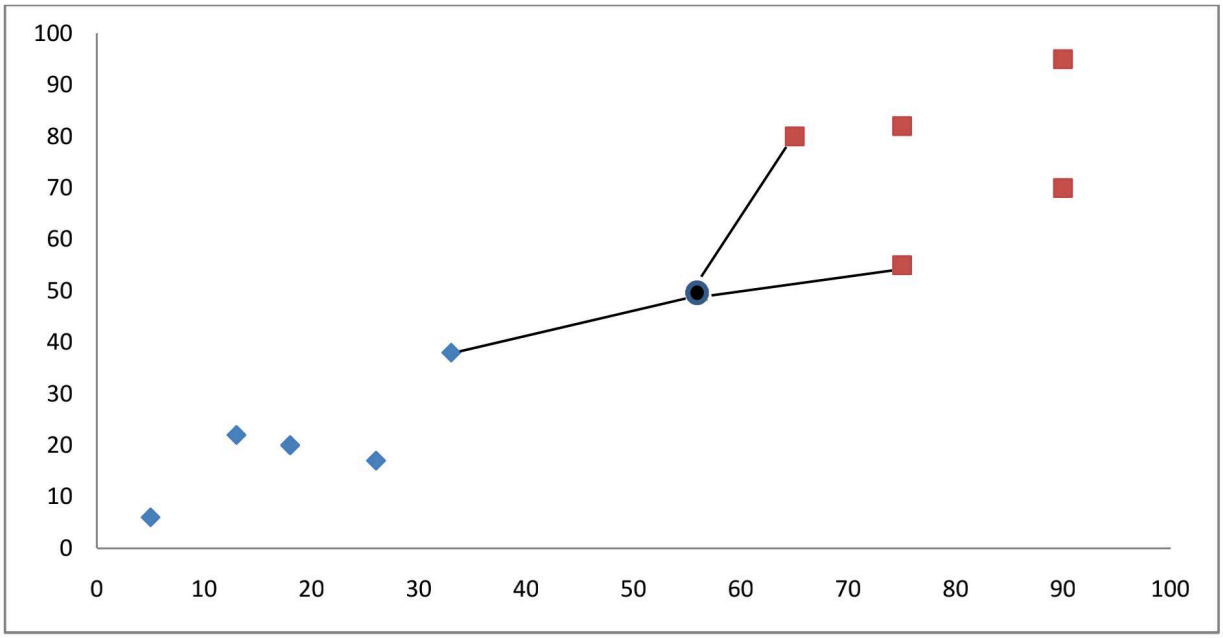
EXAMPLE:

The following is an example to understand the concept of K and working of KNN algorithm –



Suppose we have a dataset which can be plotted as follows –

Now, we need to classify new data point with black dot (at point 56,50) into blue or brown class. We are assuming K = 3 i.e. it would find three nearest data points. It is shown in the next diagram –



We can see in the above diagram the three nearest neighbors of the data point with black dot. Among those three, two of them lies in brown class hence the black dot will also be assigned in brown class.

Source Code:

```
from math import sqrt
class KNN:
        def knn(self):
                print("\nThe distance between the test value and the trained value:")
                dis = []
                for cluster in self.dataset:
                        for point in cluster:
                                d = self.euclidean_distance(point)
                                dis.append(d)
                                print(point," : ",round(d,4))
                self.dist = dis
                neighbour_value=int(input("\nHow many neighbours do you want?: "))

                print("\nThe {0} neighbours of {1} are:" .format(neighbour_value,self.test_data))
                self.neighbours = self.get_neighbours(neighbour_value)
                for neighbour in self.neighbours:
```

```
                print(neighbour[0]," of class ",neighbour[1])

        self.cls = self.predict_classification()
    '''Function for  finding the distance bewtween
    the test_data and the points in the dataset'''
    def euclidean_distance(self, point2):
        distance = 0.0
        for i in range(len(self.test_data)-1):
            distance = distance + (self.test_data[i] - point2[i])**2
        return sqrt(distance)

    '''Function for finding the nearest neighbours
    of the test_data point'''
    def get_neighbours(self, num_neighbors):
        distances = list()
        cluster_value = i = 0
        for cluster in self.dataset:
            cluster_value+=1
            for point in cluster:
                distances.append((point, self.dist[i], cluster_value))
                i+=1
        distances.sort(key=lambda tup: tup[1])
        neighbours = list()
        for i in range(0,num_neighbors):
            neighbours.append((distances[i][0],distances[i][2]))
        return neighbours

    #Function for finding the class value of the test_data
    def predict_classification(self):
        output_values = [row[-1] for row in self.neighbours]
        prediction = max(set(output_values), key=output_values.count)
        return prediction

k2 = KNN()
k2.dataset = [[[17, 18], [12, 9], [12, 17], [1, 34], [0, 19], [1, 22], [20, 4]],
```

```
[[33, 24], [40, 9], [29, 24], [40, 17], [8, 42], [14, 31], [26, 16], [10, 43], [10, 35], [24, 26]],
[[26, 41], [41, 26], [42, 26]]]
x,y = input("Enter test data point:").split(",")
k2.test_data = [int(x),int(y)]
k2.knn()
print("\nThe Class Value of {0} is {1}: " .format(k2.test_data,k2.cls))
```

Output:

```
H:\SD's Workspace\Project\Major Project\Resources>python findmeans.py

The reference dataset is:

[[17, 18], [12, 9], [12, 17], [1, 34], [0, 19], [1, 22], [20, 4]]
[[33, 24], [40, 9], [29, 24], [40, 17], [8, 42], [14, 31], [26, 16], [10, 43], [10, 35], [24, 26]]
[[26, 41], [41, 26], [42, 26]]
Enter test data point:46,30
The distance between the test value and the trained value:
[17, 18] : 29.0
[12, 9] : 34.0
[12, 17] : 34.0
[1, 34] : 45.0
[0, 19] : 46.0
[1, 22] : 45.0
[20, 4] : 26.0
[33, 24] : 13.0
[40, 9] : 6.0
[29, 24] : 17.0
[40, 17] : 6.0
[8, 42] : 38.0
[14, 31] : 32.0
[26, 16] : 20.0
[10, 43] : 36.0
[10, 35] : 36.0
[24, 26] : 22.0
```

```
[26, 41] : 20.0
[41, 26] : 5.0
[42, 26] : 4.0

How many neighbours do you want?: 7

The 7 neighbours of [46, 30] are:
[42, 26] of class 3
[41, 26] of class 3
[40, 9] of class 2
[40, 17] of class 2
[33, 24] of class 2
[29, 24] of class 2
[26, 16] of class 2
The Class Value of [46, 30] is 2:
```

Advantage:

1. It is very simple algorithm to understand and interpret.
2. It is very useful for nonlinear data because there is no assumption about data in this algorithm.
3. It is a versatile algorithm as we can use it for classification as well as regression.
4. It has relatively high accuracy but there are much better supervised learning models than KNN.

Disadvantage:

1. It is computationally a bit expensive algorithm because it stores all the training data.
2. High memory storage required as compared to other supervised learning algorithms.
3. Prediction is slow in case of big N.
4. It is very sensitive to the scale of data as well as irrelevant features.

Applications of KNN:

The following are some of the areas in which KNN can be applied successfully –

1. Banking System

KNN can be used in banking system to predict whether an individual is fit for loan approval? Does that individual have the characteristics similar to the defaulters one?

2. Calculating Credit Ratings

KNN algorithms can be used to find an individual's credit rating by comparing with the persons having similar traits.

Other areas in which KNN algorithm can be used are

3. Speech Recognition.

4. Handwriting Detection.

5. Image Recognition and Video Recognition.

K MEANS CLUSTERING v/s K NEAREST NEIGHBOUR

K-Means Clustering	K Nearest Neighbour
It is an Unsupervised learning technique.	It is a Supervised learning technique
It is used for Clustering.	It is used mostly for Classification, and sometimes even for Regression.
'K' in K-Means is the number of clusters the algorithm is trying to identify/learn from the data. The clusters are often unknown since this is used with Unsupervised learning.	'K' in KNN is the number of nearest neighbors used to classify or (predict in case of continuous variable/regression) a test sample.
In training phase of K-Means, K observations are arbitrarily selected (known as centroids). Each point in the vector space is assigned to a cluster represented by nearest (Euclidean distance) centroid. Once the clusters are formed, for each cluster the centroid is updated to the mean of all cluster members. And the cluster formation restarts with new centroids. This repeats until the centroids themselves become mean of clusters,	K-NN doesn't have a training phase as such. But the prediction of a test observation is done based on the K-Nearest (often Euclidean distance) Neighbors (observations) based on weighted averages/votes.

It is typically used for scenarios like understanding the population demographics, market segmentation, social media trends, anomaly detection, etc. where the clusters are unknown to begin with.	It is used for classification and regression of known data where usually the target attribute/variable is known before hand.

Association of K Means Clustering and K Nearest Neighbour

Now it's time to merge this two techniques for a better automation.
This will be achieved by –

1. Applying K Means to cluster a certain dataset into K clusters or class.
2. Next we will send this clustered dataset to KNN along with a test data point to test.
3. Then KNN will find which class will the test data lie into.

Source Code:

```
from math import sqrt
import random

class KMEANS:
	#Function for the K-Means algorithm
	def kmeans(self):
		print("The dataset is:")
		print(*self.data, sep = "\n")
		self.k = int(input("No. of clusters (K) = "))
		c = [
			[] for _ in range(self.k)
		]
		print("Enter ",self.k," centroids as follows:")
		for i in range(self.k):
			x,y = input("centroid {0} = " .format(i+1)).split(",")
			c[i] = [int(x),int(y)]
		print("The initial centroids are : ", *c, sep = "\n")
		cluster, length = self.clustering(c)
		ncluster = pcluster = cluster
		for n in range(10):
			if(0  in length):
				print("\noptimal position reached:")
				print("\nThe optimal clusters are:" .format(n))
				print(*pcluster, sep="\n")
				break
```

```
                else:
                        cnew = self.means(ncluster, length)
                        if c==cnew:
                                print("\noptimal position reached:")
                                print("\nThe optimal clusters are:" .format(n))
                                print(*ncluster, sep="\n")
                                break;
                        else:
                                c=cnew
                                pcluster = ncluster
                                ncluster,length =  self.clustering(c)

        return cluster

    #Function for clustering the dataset
    def clustering(self,c):
        cl=[
                []      for _  in range(self.k)
        ]
        length=[]
        for i in range(self.k):
                l=0
                for j in self.data:
                        if(self.check(j,c)==i):
                                cl[i].append(j)
                                l+=1
                length.append(l)
        return cl, length

    #Function for finding new centroids
    def means(self,cluster,length):
        c=[[ [] for _ in range(2) ] for _ in range (self.k)]
        sx=sy=0
        for i in range(len(c)):
                for j in range(length[i]):
                        sx,sy = sx+cluster[i][j][0],sy+cluster[i][j][1]
                c[i] = [sx/length[i],sy/length[i]]
        return c
    '''Function for finding the cluster number
    for a certain point'''
    def check(self,point,c):
        d=[]
        [x,y]=point[:]
        for i in range(len(c)):
                [x1,y1] = c[i]
                d.append(sqrt(((x-x1)**2)+((y-y1)**2)))
        return d.index(min(d))

class KNN:
    def knn(self):
        print("\nThe distance between the test value and the trained value:")
        dis = []
        for cluster in self.dataset:
                for point in cluster:
```

```
                                        d = self.euclidean_distance(point)
                                        dis.append(d)
                                        print(point," : ",round(d,4))
                        self.dist = dis
                        neighbour_value=int(input("\nHow many neighbours do you want?: "))

                        print("\nThe {0} neighbours of {1} are:"
.format(neighbour_value,self.test_data))
                        self.neighbours = self.get_neighbours(neighbour_value)
                        for neighbour in self.neighbours:
                                print(neighbour[0]," of class ",neighbour[1])

                        self.cls = self.predict_classification()
                '''Function for  finding the distance bewtween
                the test_data and the points in the dataset'''
                def euclidean_distance(self, point2):
                        distance = 0.0
                        for i in range(len(self.test_data)-1):
                                distance = distance + (self.test_data[i] - point2[i])**2
                        return sqrt(distance)

                '''Function for finding the nearest neighbours
                of the test_data point'''
                def get_neighbours(self, num_neighbors):
                        distances = list()
                        cluster_value = i = 0
                        for cluster in self.dataset:
                                cluster_value+=1
                                for point in cluster:
                                        distances.append((point, self.dist[i], cluster_value))
                                        i+=1
                        distances.sort(key=lambda tup: tup[1])
                        neighbours = list()
                        for i in range(0,num_neighbors):
                                neighbours.append((distances[i][0],distances[i][2]))
                        return neighbours

                #Function for finding the class value of the test_data
                def predict_classification(self):
                        output_values = [row[-1] for row in self.neighbours]
                        prediction = max(set(output_values), key=output_values.count)
                        return prediction

k1 = KMEANS()
k1.data = [
        [random.randrange(50),random.randrange(50)] for _ in range(20)
];
k1.cluster = k1.kmeans()

k2 = KNN()
k2.dataset = k1.cluster

print("\nThe reference dataset is:\n")
print(*k2.dataset, sep="\n")
```

```
        x,y = input("Enter test data point:").split(",")
        k2.test_data = [int(x),int(y)]
        k2.knn()
print("\nThe Class Value of {0} is {1}: " .format(k2.test_data,k2.cls))
```

Output:

```
H:\SD's Workspace\Project\Major Project>python 2k_Algo.py

The dataset is:

[13, 15]

[7, 29]

[23, 33]

[32, 38]

[45, 39]

[40, 11]

[40, 15]

[7, 8]

[39, 5]

[16, 4]

[21, 41]

[10, 7]

[24, 34]

[38, 24]

[25, 4]

[6, 20]

[31, 27]

[26, 3]

[28, 40]

[4, 26]

No. of clusters (K) = 3
```

```
Enter  3  centroids as follows:

centroid 1 = 10,10

centroid 2 = 20,20

centroid 3 = 30,30

The initial centroids are :

[10, 10]

[20, 20]

[30, 30]


optimal position reached:


The optimal clusters are:

[[13, 15], [7, 8], [16, 4], [10, 7], [25, 4], [6, 20], [26, 3], [4, 26]]

[[7, 29], [39, 5]]

[[23, 33], [32, 38], [45, 39], [40, 11], [40, 15], [21, 41], [24, 34], [38, 24], [31, 27], [28, 40]]


The reference dataset is:


[[13, 15], [7, 8], [16, 4], [10, 7], [25, 4], [6, 20], [26, 3], [4, 26]]

[[7, 29], [39, 5]]

[[23, 33], [32, 38], [45, 39], [40, 11], [40, 15], [21, 41], [24, 34], [38, 24], [31, 27], [28, 40]]

Enter test data point:25,30


The distance between the test value and the trained value:

[13, 15]  :  12.0

[7, 8]  :  18.0

[16, 4]  :  9.0

[10, 7]  :  15.0
```

```
[25, 4] : 0.0
[6, 20] : 19.0
[26, 3] : 1.0
[4, 26] : 21.0
[7, 29] : 18.0
[39, 5] : 14.0
[23, 33] : 2.0
[32, 38] : 7.0
[45, 39] : 20.0
[40, 11] : 15.0
[40, 15] : 15.0
[21, 41] : 4.0
[24, 34] : 1.0
[38, 24] : 13.0
[31, 27] : 6.0
[28, 40] : 3.0

How many neighbours do you want?: 5

The 5 neighbours of [25, 30] are:
[25, 4]  of class  1
[26, 3]  of class  1
[24, 34]  of class  3
[23, 33]  of class  3
[28, 40]  of class  3

The Class Value of [25, 30] is 3:
```

Discussion & Outcome

So, it wasn't so hard after all to understand and implement these two algorithms. But there are some problems that we had to face while developing these algorithm structures. These are:

1. While developing K means, we used a lot of dataset and a lot of mean values. But as we have known that the efficiency of K Means depend upon the dataset and the initial mean values, so there were some attempts when inappropriate dataset and mean values resulted inefficient clusters, making us think that our codes weren't accurate.

2. While working with KNN, we have faced the issue of finding the class value of the test data point because storing the class value of each data point inside them were efficient for implementation but not suitable for real life problems.

3. As we have done this project as a team therefore the implementation was divided among us. That's why at the time of merging the code we have faced some difficulties understanding the approach of the code blocks.

Conclusion

As we have been saying this that “Machine Learning” is te new part of our region that we never walked in to, working with these two algorithms gave us an overall idea of how machine learning algorithms look like and how to implement them. Moreover our interest in this context was very fascinating. Some of us would love to pursue “Machine Learning” as their future careers.

We also came to know how real life datasets are clustered and being useful for our daily lives, like the recommendation system of Amazon o Netflix. IF it was not for this project, we would never know that these algorithms are actually used in the well-known websites and applications.

Acknowledgement

We would like to give our special thanks and regards to our Supervisor Mr. TANMOY METE for guiding us and letting us have the ideas for choosing this topic and helping us to go through a thorough research.

Special thanks to our team members for their effort in this project.

Special thanks to our friends in college and university for sharing their knowledge with us.

Special thanks to GOOGLE for helping me acquire my data.

Special thanks to Microsoft Office Word for such a great typing experience.

References

http://abhijitanaldas.com/ml/kmeans-vs-knn-in-machine-learning.html

https://datascienceplus.com/k-means-clustering/

https://amete.github.io/DataSciencePortfolio/Udemy/Python-DS-and-ML-Bootcamp/K_Means_Clustering_Project.html

https://geeksforgeeks.com

https://tutorialspoint.com

https://w.saedsayad.com/clustering_kmeans.htm

Edureka Youtube Channel