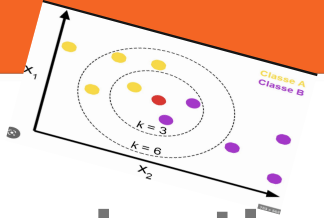
k-Nearest-Neighbors (k-NN) Algorithm

Python + kNN + Colab

Norina Akhtar 19643

Table of Contents



1. Introduction

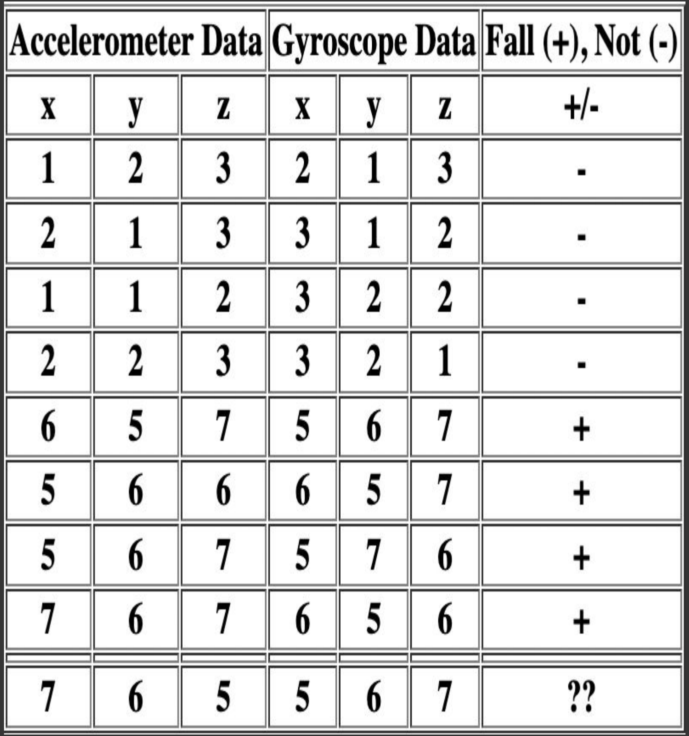
k-Nearest-Neighbors (k-NN) is a supervised machine learning model. Supervised learning is when a model learns from data that is already labeled. A supervised learning model takes in a set of input objects and output values. The model then trains on that data to learn how to map the inputs to the desired output so it can learn to make predictions on unseen data.

Design

Most mobile devices are equipped with different kind of sensors.

We can use the data sent from Gyroscope sensor and Accelerometer sensor to categorize any motion:

- 3 values(x1,y1,z1) from Accelerometer sensor.
- 3 values from(x2,y2,z2) Gyroscope sensor.



Accelerometer Data			Gyroscope Data			Fall (+), Not (-)
x	y	z	x	y	z	+/-
1	2	3	2	1	3	-
2	1	3	3	1	2	-
1	1	2	3	2	2	-
2	2	3	3	2	1	-
6	5	7	5	6	7	+
5	6	6	6	5	7	+
5	6	7	5	7	6	+
7	6	7	6	5	6	+
7	6	5	5	6	7	??

Find the Value of K?

A general rule of thumb: K = the closest odd number of the square root of the number of samples

Number of nearest neighbors

=sqrt(number of neighbors)

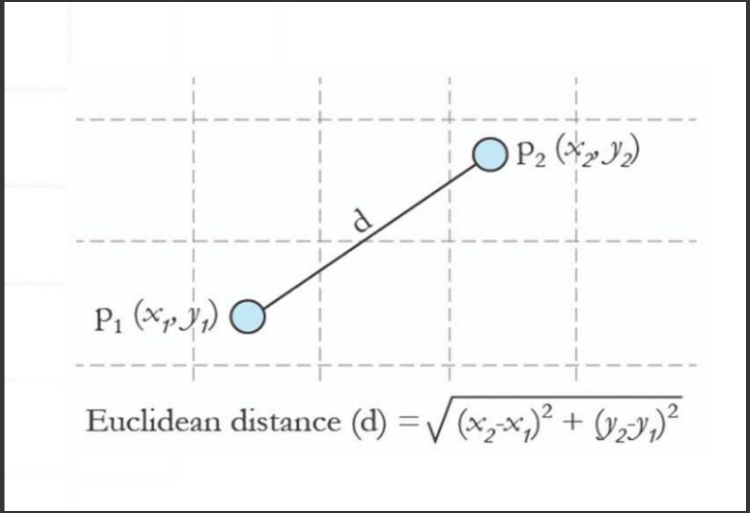
=sqrt(number of data samples)

= sqrt(8)

= 2.828 ⇒ 3

Euclidean Distance :

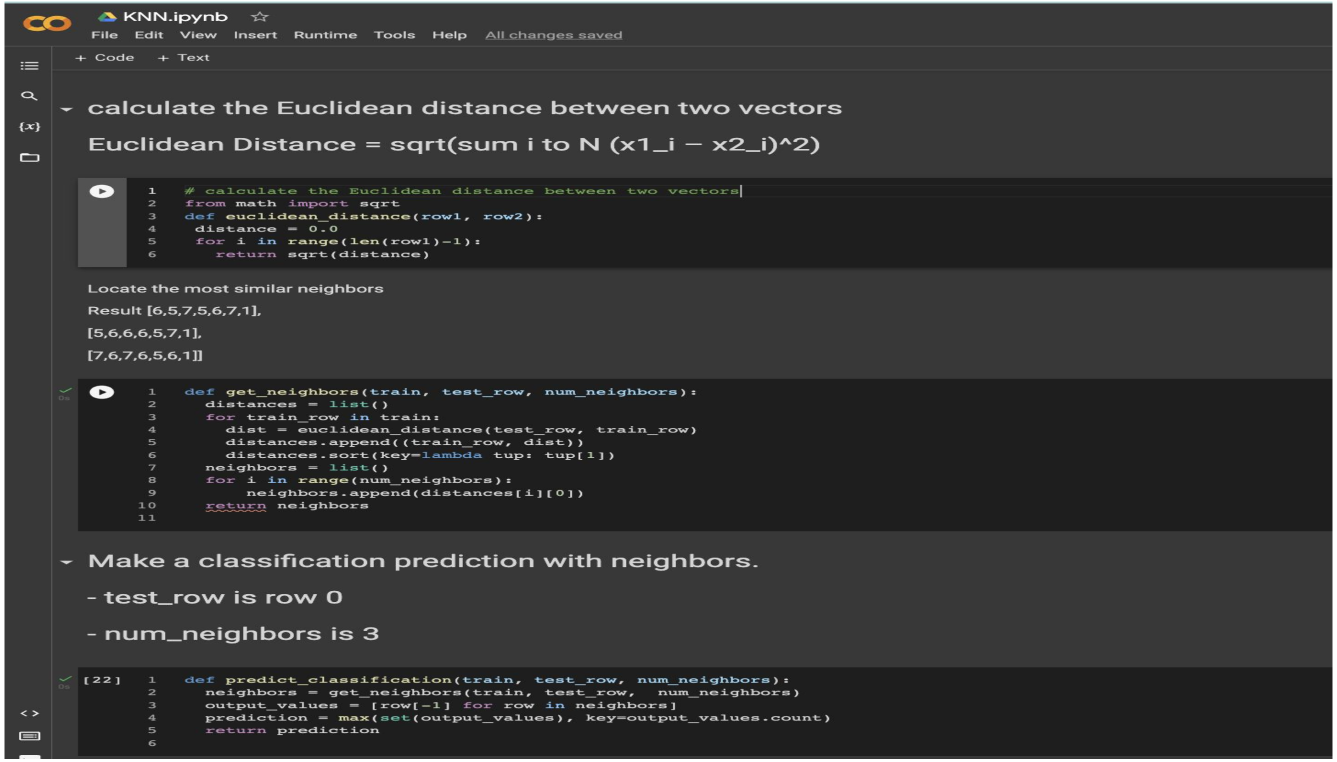
sqrt(sum i to N (x1_i – x2_i)^2)



Manual Implementation

Accelerometer Data			**Gyroscope Data**			**Fall / No**	**Distance to each neighbor** (Target$_{X1}$-Data$_{X1}$)^2 + (Target$_{y1}$-Data$_{y1}$)^2 + (Target$_{z1}$-Data$_{z1}$)^2 + (Target$_{X2}$-Data$_{X2}$)^2 + (Target$_{y2}$-Data$_{y2}$)^2 + (Target$_{z2}$-Data$_{z2}$)^2 **(7-x1)^2+(6-y1)^2+(5-z1)^2+(5-x2)^2+(6-y2)^2+(7-z2)^2**	**K =Number of nearest neighbors =sqrt(8) =3**
X1	**Y1**	**Z1**	**X2**	**Y2**	**Z2**			
1	2	3	2	1	3	-	(7-1)^2+(6-2)^2+(5-3)^2+(5-2)^2+(6-1)^2+(7-3)^2= 106	
2	1	3	3	1	2	-	(7-2)^2+(6-1)^2+(5-3)^2+(5-3)^2+(6-1)^2+(7-2)^2= 108	-
1	1	2	3	2	2	-	(7-1)^2+(6-1)^2+(5-2)^2+(5-3)^2+(6-2)^2+(7-2)^2= 115	-
2	2	3	3	2	1	-	(7-2)^2+(6-2)^2+(5-3)^2+(5-3)^2+(6-2)^2+(7-1)^2= 101	-
6	5	7	5	6	7	+	(7-6)^2+(6-5)^2+(5-7)^2+(5-5)^2+(6-6)^2+(7-7)^2= 6	+
5	6	6	6	5	7	+	(7–5)^2+(6–6)^2+(5–6)^2+(5–6)^2+(6–5)^2+(7–7)^2= 7	+
5	6	7	5	7	6	+	(7-5)^2+(6-6)^2+(5-7)^2+(5-5)^2+(6-7)^2+(7-6)^2 = 10	+
7	6	7	6	5	6	+	(7-7)^2+(6-6)^2+(5-7)^2+(5-6)^2+(6-5)^2+(7-6)^2 = 7	+
7	6	5	5	6	7	?		**+**

Implementation(Python Program)



calculate the Euclidean distance between two vectors

Euclidean Distance = sqrt(sum i to N (x1_i – x2_i)^2)

```
# calculate the Euclidean distance between two vectors
from math import sqrt
def euclidean_distance(row1, row2):
 distance = 0.0
 for i in range(len(row1)-1):
   return sqrt(distance)
```

Locate the most similar neighbors

Result [6,5,7,5,6,7,1],

[5,6,6,6,5,7,1],

[7,6,7,6,5,6,1]]

```
def get_neighbors(train, test_row, num_neighbors):
  distances = list()
  for train_row in train:
    dist = euclidean_distance(test_row, train_row)
    distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
  neighbors = list()
  for i in range(num_neighbors):
      neighbors.append(distances[i][0])
  return neighbors

```

Make a classification prediction with neighbors.

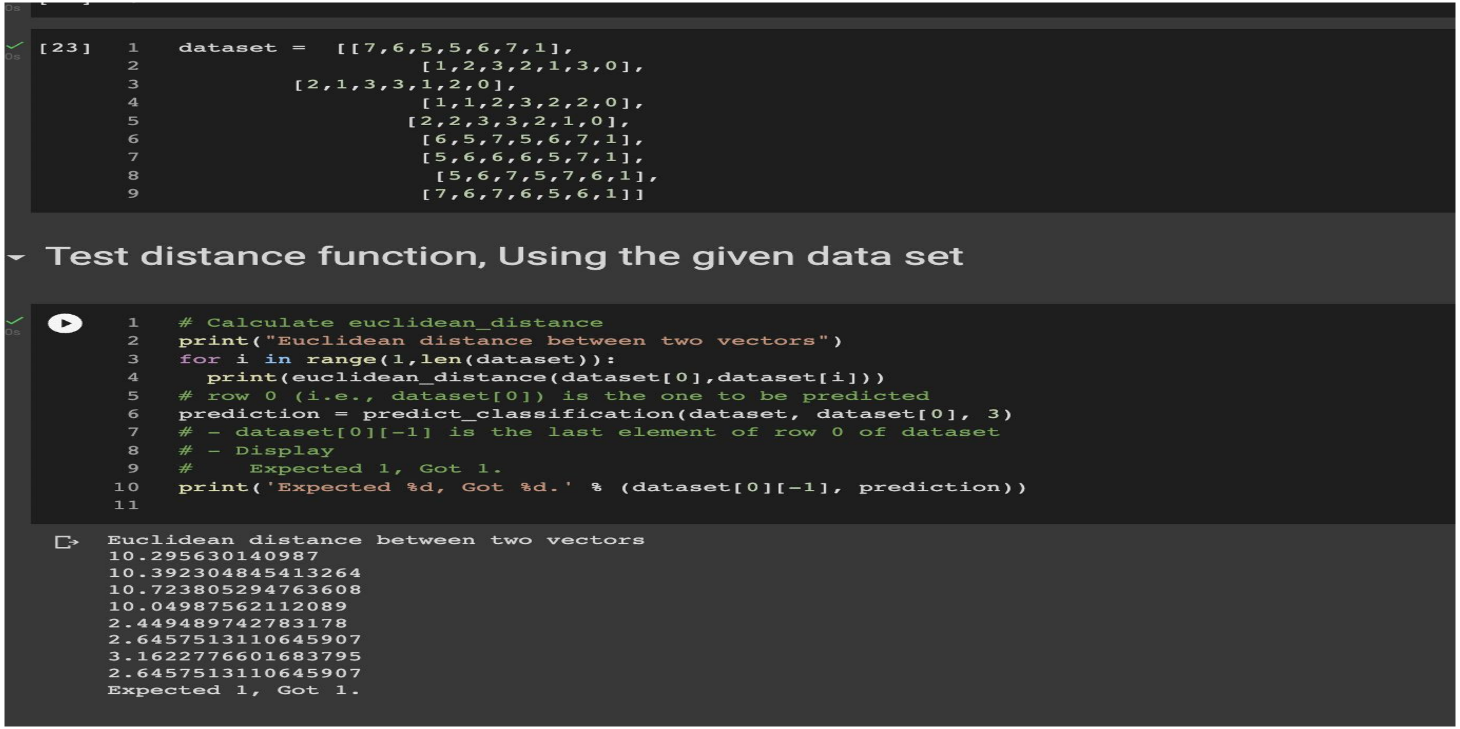
- test_row is row 0

- num_neighbors is 3

```
[22]
def predict_classification(train, test_row, num_neighbors):
  neighbors = get_neighbors(train, test_row,  num_neighbors)
  output_values = [row[-1] for row in neighbors]
  prediction = max(set(output_values), key=output_values.count)
  return prediction

```

Test



```
[23]
dataset =  [[7,6,5,5,6,7,1],
                     [1,2,3,2,1,3,0],
            [2,1,3,3,1,2,0],
                     [1,1,2,3,2,2,0],
                    [2,2,3,3,2,1,0],
                     [6,5,7,5,6,7,1],
                     [5,6,6,6,5,7,1],
                      [5,6,7,5,7,6,1],
                     [7,6,7,6,5,6,1]]
```

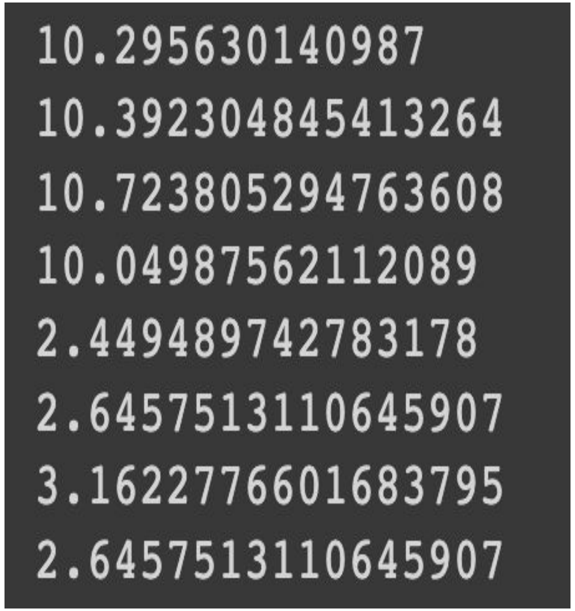
Test distance function, Using the given data set

```
# Calculate euclidean_distance
print("Euclidean distance between two vectors")
for i in range(1,len(dataset)):
  print(euclidean_distance(dataset[0],dataset[i]))
# row 0 (i.e., dataset[0]) is the one to be predicted
prediction = predict_classification(dataset, dataset[0], 3)
# - dataset[0][-1] is the last element of row 0 of dataset
# - Display
#    Expected 1, Got 1.
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

```
Euclidean distance between two vectors
10.295630140987
10.392304845413264
10.723805294763608
10.04987562112089
2.449489742783178
2.6457513110645907
3.1622776601683795
2.6457513110645907
Expected 1, Got 1.
```

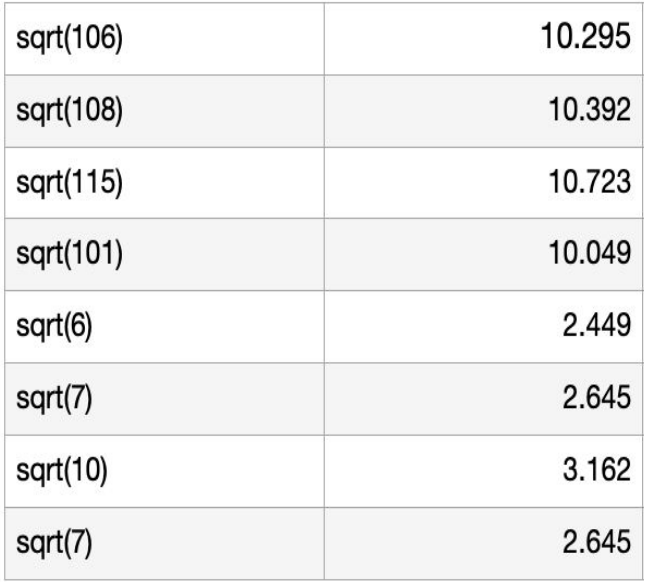
Comparing the results

Python program result



```
10.295630140987
10.392304845413264
10.723805294763608
10.04987562112089
2.449489742783178
2.6457513110645907
3.1622776601683795
2.6457513110645907
```

Manual solution



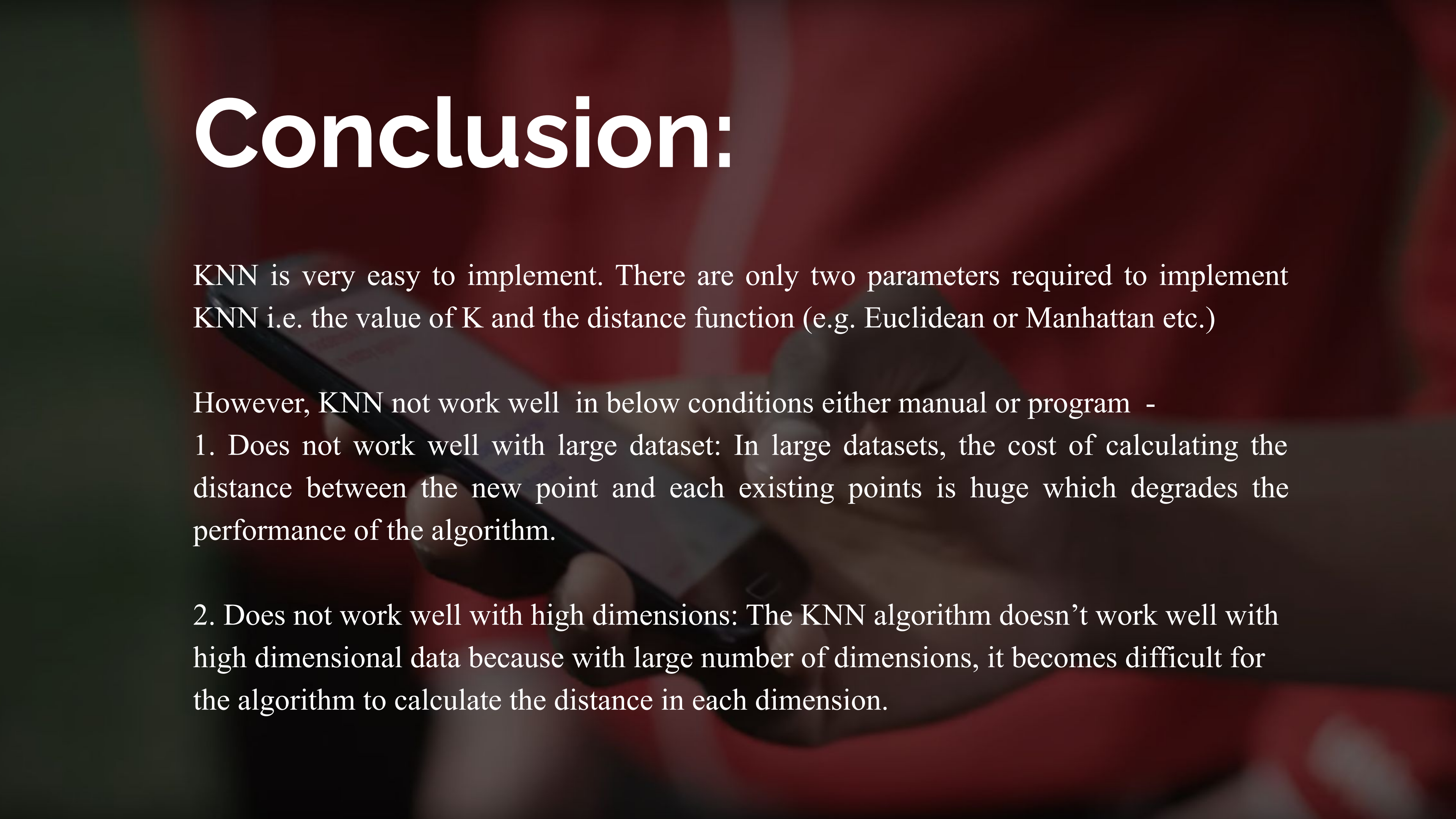
sqrt(106)	10.295
sqrt(108)	10.392
sqrt(115)	10.723
sqrt(101)	10.049
sqrt(6)	2.449
sqrt(7)	2.645
sqrt(10)	3.162
sqrt(7)	2.645

Conclusion:

KNN is very easy to implement. There are only two parameters required to implement KNN i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)

However, KNN not work well in below conditions either manual or program -
1. Does not work well with large dataset: In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.

2. Does not work well with high dimensions: The KNN algorithm doesn’t work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.



Enhancement Ideas

The key to improve the algorithm is to add a preprocessing stage to make the final algorithm run with more efficient data and then improve the effect of classification. The experimental results show that the improved KNN algorithm improves the accuracy and efficiency of classification.

References

https://hc.labnet.sfbu.edu/~henry/sfbu/course/data_science/algorithm/slide/knn_from_scratch.html

https://hc.labnet.sfbu.edu/~henry/sfbu/course/data_science/algorithm/slide/k_nn_example.html

https://hc.labnet.sfbu.edu/~henry/npu/classes/data_science/algorithm/slide/index_slide.html

https://hc.labnet.sfbu.edu/~henry/sfbu/course/android/sensor/slide/gyroscope_rotation.html

https://www.youtube.com/watch?v=ralBdVcNbml&ab_channel=GetScience%26Technology

https://www.researchgate.net/post/How_can_we_find_the_optimum_K_in_K-Nearest_Neighbor