Falling Prediction using KNN

Table of Contents

Introduction

- In this assignment we will be using KNN classification algorithm to make a fall prediction, we will be doing the prediction in two ways, the first one through regular calculations using excel, and the second one will be using python and colab to make the prediction.

Design

- We first has to take our already preprocessed data and then transform it into an array form.
- Our data for this project is list of features for the Abalone Animal, we are Interested in the Length and Height features, we will try to use the length data for our features and use the height data as our labels, and then train the linear model using this data set

Implementation in Excel

- We want to predict the fall of a certain individual from the gyroscope and acceleration data that we gathered from the gyroscope and the accelerometer

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	??

Implementation - Excel

- The first thing we should do is to calculate the K value
- In our case we had 8 samples so our k would be 3 in this case, since A general rule of thumb: K = the closest odd number of the square root of the number of samples

$$K = \text{sqrt}(8) = 3$$

Implementation - Excel

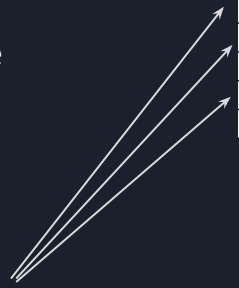
- K nearest neighbor algorithm is based on minimum distance from the query instance to the training samples to determine the K-nearest neighbors.
- We have to Calculate the euclidean distance following this convention.

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)	Distance Accelerometer	Distance Gyroscope
x	y	z	x	y	z	+/-		
1	2	3	2	1	3	-	56	50
2	1	3	3	1	2	-	54	54
1	1	2	3	2	2	-	70	45
2	2	3	3	2	1	-	45	56
6	5	7	5	6	7	+	6	0
5	6	6	6	5	7	+	5	2
5	6	7	5	7	6	+	8	2
7	6	7	6	5	6	+	4	3

```
=($B$14-B6)^2 + ($C$14-C6)^2 + ($D$14-D6)^2
```

Implementation - Excel

- The next step is to find the K-nearest neighbors.
- In other words, we sort the distance of all training samples to the query instance and determine the K-th minimum distance.
- Since K=3 then the neighbors would be

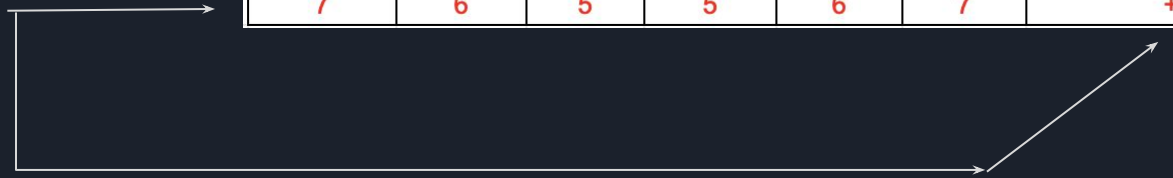


Accelerometer Data			Gyroscope Data			Fall (+), Not (-)	Distance Accelerometer	Distance Gyroscope
x	y	z	x	y	z	+/-		
1	2	3	2	1	3	-	56	50
2	1	3	3	1	2	-	54	54
1	1	2	3	2	2	-	70	45
2	2	3	3	2	1	-	45	56
6	5	7	5	6	7	+	6	0
5	6	6	6	5	7	+	5	2
5	6	7	5	7	6	+	8	2
7	6	7	6	5	6	+	4	3
7	6	5	5	6	7	+		

Test - Excel

- If the number of plus is greater than minus, we predict the query instance as plus and vice versa.
- If the number of plus is equal to minus, we can choose arbitrary or determine as one of the plus or minus.
- Then our final prediction would be a plus

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	+



Implementation - Python

- We first import the packages and modules we will need for the duration of our implementation

```
# using python knn to do a prediction
from math import sqrt
```

Implementation - Python

- We then define the functions we need in order to implement KNN Classification

```
def euclidean_distance(row1, row2):
  distance = 0.0
  for i in range(len(row1)-1):
    distance += (row1[i] - row2[i])**2
  return sqrt(distance)


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

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
```

Implementation - Python

- We then set up our dataset and convert it into a list format.
- We then test our euclidean distance function on our dataset to get the distances from the dataset

```
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]]

#we test our function and get the distances
print("Euclidean distance between two vectors")
for data in range(1, len(dataset)):
  print(euclidean_distance(dataset[0],dataset[data]))
```

```
Euclidean distance between two vectors
10.295630140987
10.392304845413264
10.723805294763608
10.04987562112089
2.449489742783178
2.6457513110645907
3.1622776601683795
2.6457513110645907
```

Implementation - Python

- The next step if to test get_neighbors function and generate a list of neighbors from our dataset that are kth distance from our test data

```
print("The nearest neighbors are: ")
print(get_neighbors(dataset[1:], dataset[0], 3))
```

```
The nearest neighbors are:
[[6, 5, 7, 5, 6, 7, 1], [5, 6, 6, 6, 5, 7, 1], [7, 6, 7, 6, 5, 6, 1]]
```

Test - Python

- The final step is to use our program to make a final prediction and then format our answer.

```
# In our dataset we had the falling prediction as either + or -, we represented that using the binary numbers 1 for + and o for -.
# in this dataset we will finally predict our output from the given dataset


print("Our Dataset is :")
for data in range(1, len(dataset)):
  print(dataset[data])
print()
print("We want to make a prediciton for: ")
print(dataset[0][:len(dataset[0]) - 1])
print()
predict = predict_classification(dataset, dataset[0], 3)
print(f"We expected {dataset[0][-1]} and we got {predict}")
```

```
Our Dataset is :
[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]

We want to make a prediciton for:
[7, 6, 5, 5, 6, 7]

We expected 1 and we got 1
```

Conclusion

- As we can see from our results, both KNN using Excel and Python correctly predicted from the given data of the gyroscope and accelerometer that the individual is experiencing a fall.