Falling Prediction using KNN

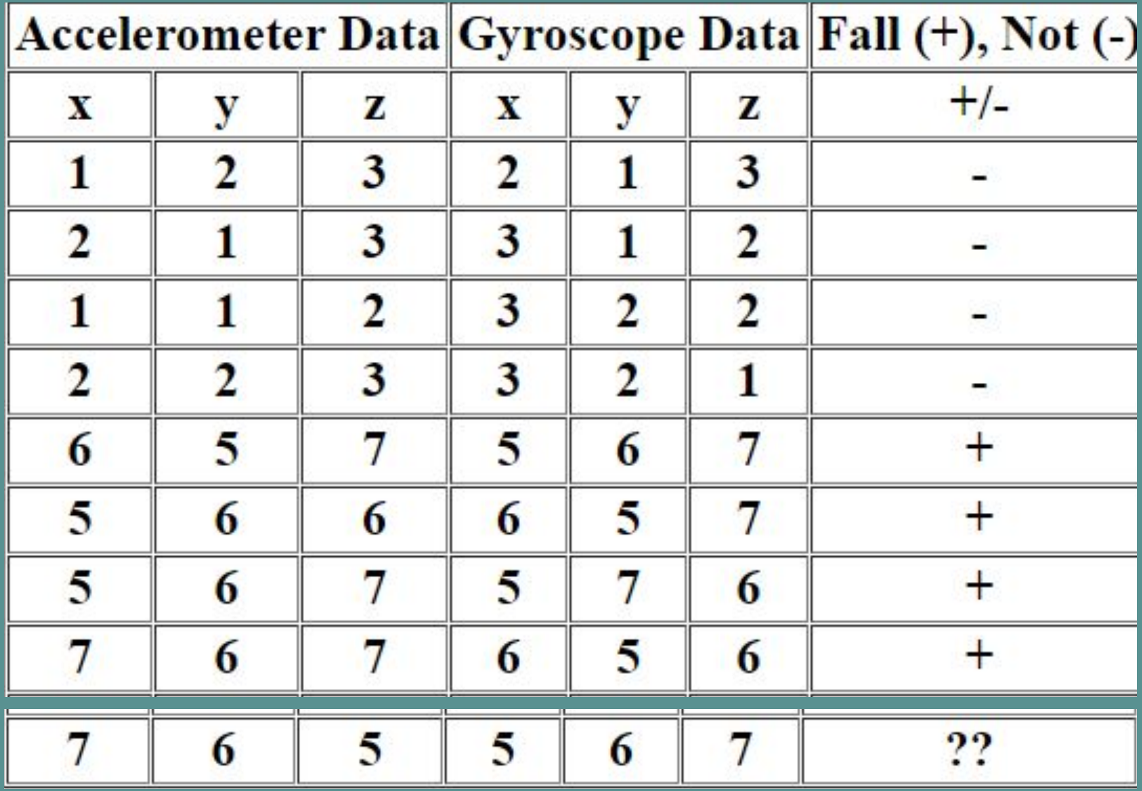
By Farnaz Golnam
MCS 19576

Introduction:

The k-nearest neighbors (KNN) is a supervised machine learning algorithm which relies on labled input data to learn how to label an unknown data and can be used for both regression and classification problems.

In the falling prediction problem we are going to use data points with (X-Y-Z) coordinate from sensors and classify whether a new unlabeled point is in the class of fall(-) or not.

Our data points from two different sensors:

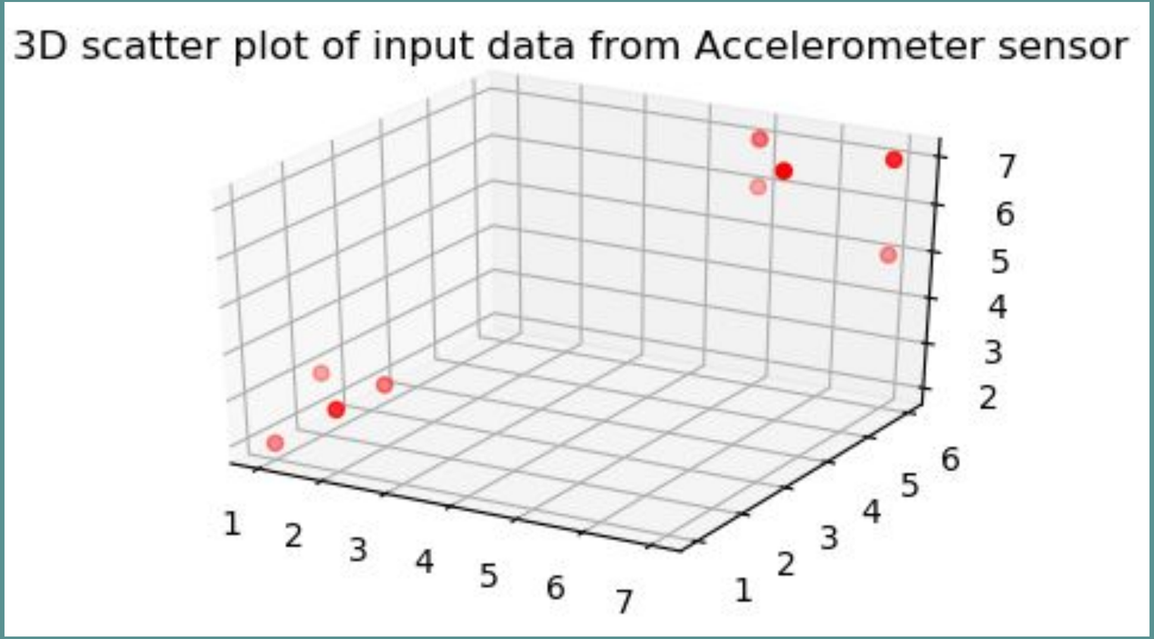


Accelerometer Data			Gyroscope Data			Fall (+), Not (-)
x	y	z	x	y	z	+/-
1	2	3	2	1	3	-
2	1	3	3	1	2	-
1	1	2	3	2	2	-
2	2	3	3	2	1	-
6	5	7	5	6	7	+
5	6	6	6	5	7	+
5	6	7	5	7	6	+
7	6	7	6	5	6	+
7	6	5	5	6	7	??

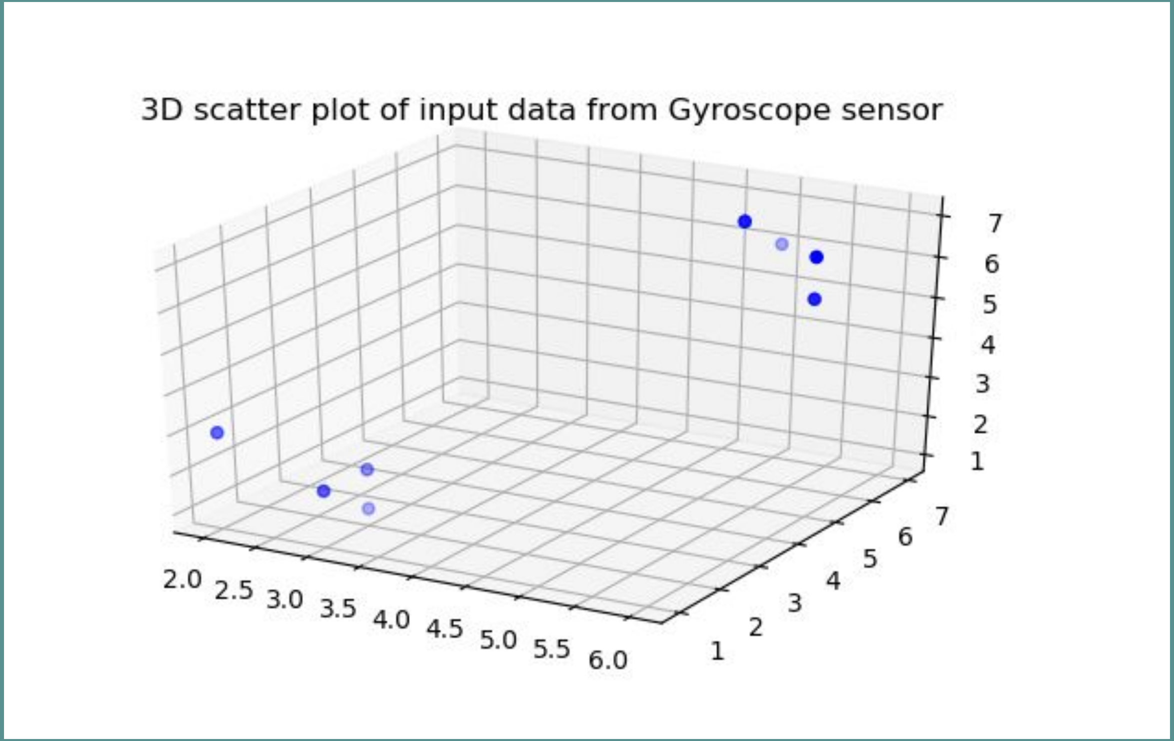
Data Visualization:

Creating scatter plots of the input data points using python to better understand the data

In accelerometer sensor,the unknown point to classify is (7,6,5)



In Gyroscope sensor,the unknown point to classify is (5,6,7)



Step1: choose a value for K

K = sqrt(n) where n is the total number of input data points
K should be an odd number to prevent confusion between two classes of data

N for inputs of each sensor=8
k= sqrt(8) = 2.8 ~ 3

We will start our calculation with k=3

Step 2:

calculating the distance between the unlabeled point to each input data (training data) using the Euclidean distance formula

$$d_{tq}^2 = (x^t - x^q)^2 + (y^t - y^q)^2 + (z^t - z^q)^2$$

For Accelerometer data

Accelerometer data					
X	Y	Z	Distance from (x=7,y=6,z=5)	d	Fall (+), Not (-)
1	2	3	$D_{tq}^2 = (1 - 7)^2 + (2 - 6)^2 + (3 - 5)^2$	7.4	-
2	1	3	$D_{tq}^2 = (2 - 7)^2 + (1 - 6)^2 + (3 - 5)^2$	7.3	-
1	1	2	$D_{tq}^2 = (1 - 7)^2 + (1 - 6)^2 + (2 - 5)^2$	8.3	-
2	2	3	$D_{tq}^2 = (2 - 7)^2 + (2 - 6)^2 + (3 - 5)^2$	6.7	-
6	5	7	$D_{tq}^2 = (6 - 7)^2 + (5 - 6)^2 + (7 - 5)^2$	2.4	+
5	6	6	$D_{tq}^2 = (5 - 7)^2 + (6 - 6)^2 + (6 - 5)^2$	2.2	+
5	6	7	$D_{tq}^2 = (5 - 7)^2 + (6 - 6)^2 + (7 - 5)^2$	2.8	+
7	6	7	$D_{tq}^2 = (7 - 7)^2 + (6 - 6)^2 + (7 - 5)^2$	2	+

For Gyroscope data

Gyroscope data					
X	Y	Z	Distance from (x=5,y=6,z=7)	d	Fall (+), Not (-)
2	1	3	$D_{tq}^2 = (2 - 5)^2 + (1 - 6)^2 + (3 - 7)^2$	7.01	-
3	1	2	$D_{tq}^2 = (3 - 5)^2 + (1 - 6)^2 + (2 - 7)^2$	7.3	-
3	2	2	$D_{tq}^2 = (3 - 5)^2 + (2 - 6)^2 + (2 - 7)^2$	6.7	-
3	2	1	$D_{tq}^2 = (3 - 5)^2 + (2 - 6)^2 + (1 - 7)^2$	7.4	-
5	6	7	$D_{tq}^2 = (5 - 5)^2 + (6 - 6)^2 + (7 - 7)^2$	0	+
6	5	7	$D_{tq}^2 = (6 - 5)^2 + (5 - 6)^2 + (7 - 7)^2$	1.4	+
5	7	6	$D_{tq}^2 = (5 - 5)^2 + (7 - 6)^2 + (6 - 7)^2$	1.4	+
6	5	6	$D_{tq}^2 = (6 - 5)^2 + (5 - 6)^2 + (6 - 7)^2$	1.7	+

Initial Results:

The class of our unlabeled data is the same as the class of its 3 nearest points (k=3) and they all show positive result which means Fall

Now we are going to implement the KNN problem in Python and compare the results.

Step 4: KNN classification problem using Python

Defining 3 main functions to do the following steps:

- Organizing training(labeled points) and test data(unlabeled point)
- Calculating the euclidean distance between training points and test point
- Get closest neighbors to our test data based on number of k
- Predict the class of the test data based on class of its neighbors

```
# Step1: organizing training and test data:

# for Accelerometer data:
data_points =
[(1,2,3,0),(2,1,3,0),(1,1,2,0),(2,2,3,0),(6,5,7,1),(5,6,6,1),(5,6,7,1),(7,6,7,1)]
#training data last index is 0 and 1 for - and + fall
query_point = (7,6,5) #test data

# for Gyroscope data:
data_points =
[(2,1,3,0),(3,1,2,0),(3,2,2,0),(3,2,1,0),(5,6,7,1),(6,5,7,1),(5,7,6,1),(6,5,6,1)]
#training data
query_point = (5,6,7)  #test data
```

```
# Step2: calculating the Euclidean distance between two vectors:
"Euclidean Distance = sqrt(sum i to N (x1_i - x2_i)^2)"

def euclidean_distance(tup1, tup2):
  distance = 0.0
  for i in range(len(tup1)-1): # last index is about fall(+) or
not showing by 0 and 1
    distance += (tup1[i] - tup2[i])** 2
  return round(sqrt(distance),3)
```

```
#step3:Finding closest neighbors to the test data based on the distance between
training and test data set

def get_neighbors(train_lst, test_point, num_neighbors):
#list of training data points as tuples, test_data as tuple, num_k
  distances = list()
  for point in train_lst:
    dist = euclidean_distance(point, test_point)
    distances.append((point, dist))
    distances.sort(key= lambda tup: tup[1])

  neighbors = list()
  for i in range(num_neighbors):
    neighbors.append(distances[i][ 0])
  # print(distances)
  print("the closest data points to the test data:" , neighbors)
  return distances
  # return neighbors
```

```python
# step4: predict the classification based on the neighbors class

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
```