Manually calculate with KNN:

Using KNN to manually calculate the distance and predict the result.

- This is the tranining data and the test data:

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	??

Ans:

K = sqrt(samples) = sqrt(8) = 3 because "The choice of K equal to the odd number cloest to the square root of the number of instances is an empirical rule-of-thumb"

We calculate the distance with "(TargetX1-DataX1)^2+(TargetX2-DataX2)^2":

(7-1)^2+(6-2)^2+(5-3)^2+(5-2)^2+(6-1)^2+(7-3)^2 = 106

(7-2)^2+(6-1)^2+(5-3)^2+(5-3)^2+(6-1)^2+(7-2)^2 = 108

(7-1)^2+(6-1)^2+(5-2)^2+(5-3)^2+(6-2)^2+(7-2)^2 = 115

(7-2)^2+(6-2)^2+(5-3)^2+(5-3)^2+(6-2)^2+(7-1)^2 = 101

(7-6)^2+(6-5)^2+(5-7)^2+(5-5)^2+(6-6)^2+(7-7)^2 = 6

(7-5)^2+(6-6)^2+(5-6)^2+(5-6)^2+(6-5)^2+(7-7)^2 = 7

(7-5)^2+(6-6)^2+(5-7)^2+(5-5)^2+(6-7)^2+(7-6)^2 = 10

(7-7)^2+(6-6)^2+(5-7)^2+(5-6)^2+(6-5)^2+(7-6)^2 = 7

Then we pick smallest 3 numbers, so

(7-6)^2+(6-5)^2+(5-7)^2+(5-5)^2+(6-6)^2+(7-7)^2 = 6 +

(7-5)^2+(6-6)^2+(5-6)^2+(5-6)^2+(6-5)^2+(7-7)^2 = 7 +

(7-7)^2+(6-6)^2+(5-7)^2+(5-6)^2+(6-5)^2+(7-6)^2 = 7 +

Thus answer should be :

7	6	5	5	6	7	+

Python with KNN:

First, google search the Colab , then create new

Type with below:

```
from math import sqrt

# calculate the Euclidean distance between two vectors
#     Euclidean Distance = sqrt(sum i to N (x1_i - x2_i)^2)
# use sqrt to avoid exceeding the maximum value calculated by the
computer
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

# Make a classification prediction with neighbors
# - test_row is row 0
# - num_neighbors is 3
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 distance function
# 0 means not fall (-), 1 means fall (+)
dataset = [[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],
]

# the dataset that i wanna predecting and predect it will be fall
testdata = [[7, 6, 5, 5, 6, 7, 0]]

# row 0 (i.e., testdata[0]) is the one to be predicted
# 3 is the K
prediction = predict_classification(dataset, testdata[0], 3)

# - dataset[0][-1] is the last element of row 0 of dataset
# - Display
print('Expected %d, Got %d.' % (testdata[0][-1], prediction))
```

We can see the prediction of [7,6,5,5,6,7] is fall, and my guess is not fall

```
# 0 means not fall (-), 1 means fall (+)
dataset = [[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],
]

# the dataset that i wanna predecting and predect it will be fall
testdata = [[7, 6, 5, 5, 6, 7, 0]]

# row 0 (i.e., testdata[0]) is the one to be predicted
# 3 is the K
prediction = predict_classification(dataset, testdata[0], 3)

# - dataset[0][-1] is the last element of row 0 of dataset
# - Display
print('Expected %d, Got %d.' % (testdata[0][-1], prediction))
```

```
Expected 0, Got 1.
```

Comparison:

If we do it with hand, we do not need to sqrt the distance, but for using computer we need do it to avoid exceeding the maximum value calculated by the computer. And for the huge, big, complicated data, using python will be faster.