Project Falling Detection: Python + kNN + Colab

CS550 Homework
Shoumya Singh
ID-19566

- Process
  1. Understand the project
     - Most mobile devices are equipped with different kind of sensors.
     - We can use the data sent from Gyroscope senso and Accelerometer sensor to categorize any motion:
       - 3 numbers from Accelerometer sensor
       - 3 numbers from Gyroscope sensor
     - References
       - A Review on Fall Prediction and Prevention System for Personal Devices: Evaluation and Experimental Results
       - Andorid FallArm Project
         - Sensors
           - Difference Between an Accelerometer and a Gyroscope

  2. Use this heuristic to decide the value of K

     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 popularized by the "Pattern Classification" book by Duda et al.

     - References
       - How can we find the optimum K in K-Nearest Neighbor?

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

- This is the training 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**	**??**

4. Use Python to implement the application of using kNN to predict falling.
   a. Create the code by modifying KNN from scratch
      - Explain the code
   b. Run the code on Colab
      - References
        - Get Start with Colab
5. Comparing the result from the Python program and the result of manual calculation.
6. Adding the project to your portofolio
   . Please use Google Slides to document the project
   a. Please link your presentation on GitHub using this structure

```
b.
c. Machine Learning
d.   - Supervised Learning
e.     + Falling Prediction using KNN
```

7. Submit
   . The URLs of the Google Slides and GitHub web pages related to this project.
   a. A PDF file of your Google Slides

<u>Understanding the project</u>

- This project is an evaluation on the fall prediction and prevention system from personal device.

- Injuries due to unintentional falls cause high social cost in which several systems have been developed to reduce them. Recently, two trends can be recognized.

- Firstly, the market is dominated by fall detection systems, which activate an alarm after a fall occurrence, but the focus is moving towards predicting and preventing a fall, as it is the most promising approach to avoid a fall injury.

- Secondly, personal devices, such as smartphones, are being exploited for implementing fall systems, because they are commonly carried by the user most of the day.

- This project reviews various fall prediction and prevention systems, with a particular interest to the ones that can rely on the sensors embedded in a smartphone, i.e., **accelerometer and gyroscope.**

- Kinematic features obtained from the data collected from accelerometer and gyroscope have been evaluated in combination with different machine learning algorithms.

- In this project we have the training data and test data from the accelerometer and gyroscope, we will apply the kNN (k-Nearest Neighbors) algorithm in two ways i.e., to manually calculate the distance and predict the results and use Python to implement the application of using kNN to predict falling then compare both the results.

- We will be using Colab by Google as a tool to write and execute python code through the browser, which is especially well suited for machine learning.

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

Step 1: Table of Input data –

- Most mobile devices are equipped with different kind of sensors.
- We can use the data sent from Gyroscope and Accelerometer sensors to categorize any motion:
  - 3 numbers from Accelerometer sensor
  - 3 numbers from 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**	??

Step 2:
Suppose we determine K = 8 (we will use 8 nearest neighbors) as parameter of this algorithm.

- Then we calculate the distance between the query-instance and all the training samples.
  - Because we use only quantitative Xi, we can use Euclidean distance.
- Suppose the query instance have coordinates (X1 q, X2 q) and the coordinate of training sample is (X1 t , X2 t) then square Euclidean distance is
- dtq ^2 = (X1 t - X1 q) ^2 + (X2 t - X2 q) ^2
  N = 8
  K = sqrt (N) =sqrt (8) = 3

Step 3: Manually calculate the distance and predict the result using step 2 methods.

Accelerometer Data			**Gyroscope Data**			**Fall (+), Not (-)**	**Distance to each neighbor = $(Target_{X1}-Data_{X1})^2 + (Target_{Y1}-Data_{Y1})^2 + (Target_{Z1}-Data_{Z1})^2 + (Target_{X2}-Data_{X2})^2 + (Target_{Y2}-Data_{Y2})^2 + (Target_{Z2}-Data_{Z2})^2$**	**K =Number of nearest neighbors =sqrt (number of neighbors) =sqrt (number of data samples) =sqrt (8) =3**
X1	**Y1**	**Z1**	**X2**	**Y2**	**Z2**	**+/-**	**= (7-X1) ^2+(6-Y1) ^2+(5-Z1) ^2+(5-X2) ^2+(6-Y2) ^2+(7-Z2) ^2**	
1	2	3	2	1	3	-	106	
2	1	3	3	1	2	-	108	
1	1	2	3	2	2	-	115	
2	2	3	3	2	1	-	101	
6	5	7	5	6	7	+	6	+
5	6	6	6	5	7	+	7	+
5	6	7	5	7	6	+	10	
7	6	7	6	5	6	+	7	+
7	6	5	5	6	7	??		+

Distance to each neighbor

$$= (Target_{X1}-Data_{X1})^2 + (Target_{Y1}-Data_{Y1})^2 + (Target_{Z1}-Data_{Z1})^2 + (Target_{X2}-Data_{X2})^2 + (Target_{Y2}-Data_{Y2})^2 + (Target_{Z2}-Data_{Z2})^2$$

= (7-1) ^2+(6-2) ^2+(5-3) ^2+(5-2) ^2+(6-1) ^2+(7-3) ^2
= 36 + 16 + 4 + 9 + 25 + 16
= 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

K = Number of nearest neighbors
= sqrt (number of neighbors)
= sqrt (number of data samples)
= sqrt (8)
= 3

- So, we check the distance to each neighbor, and select K=3 minimum values from the distance to each neighbor column.
- Then the corresponding sign in **Fall (+), Not (-)** column is checked, and the majority sign is the predicted result.
- In this case its 6,7 and 7 and all the three corresponds to '+' sign in the **Fall (+), Not (-)** column so the prediction result will also be **'+'**. This indicates a Fall with '+' sign.

Step 4: Using Python to implement the application of using kNN to predict falling.

```
# Example of making predictions
from math import sqrt
# calculate the Euclidean distance between two vectors
# Euclidean Distance = sqrt(sum i to N (x1_i - x2_i)^2)

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

  return sqrt(distance)

# Locate the most similar neighbors
# Result
# [6, 5, 7, 5, 6, 7, 1]
# [7, 6, 7, 6, 5, 6, 1]
# [5, 6, 6, 6, 5, 7, 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 [7,6,5,5,6,7]
# - 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],
  [5,6,7,6,5,6,1],
  [7,6,7,6,5,6,1]]

prediction = predict_classification(dataset, [7,6,5,5,6,7], 3)
# - Display
# Expected 1, Got 1.
print('Expected %d, Got %d.' % ([7,6,5,5,6,7,1][-1], prediction))
```

Output:

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

- In the manual method of kNN algorithm, we do not need to square root the distance.
- In case of Python coding, we need to use square root to avoid exceeding the maximum value calculated by the computer.
- Python method could be easier and faster for complicated and big data.