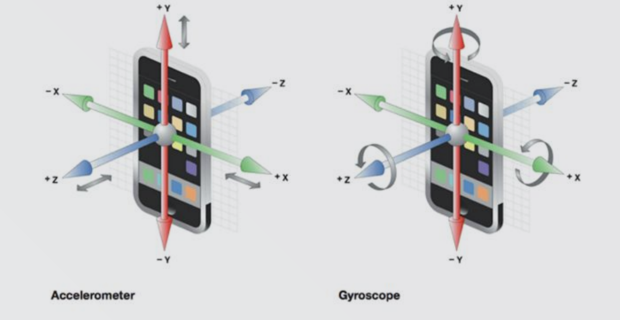
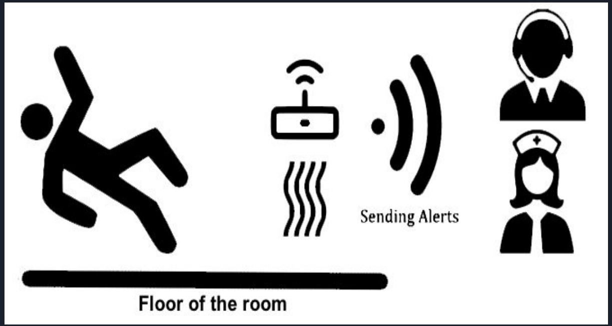
Machine Learning Project Falling Detection : kNN + Python + Colab

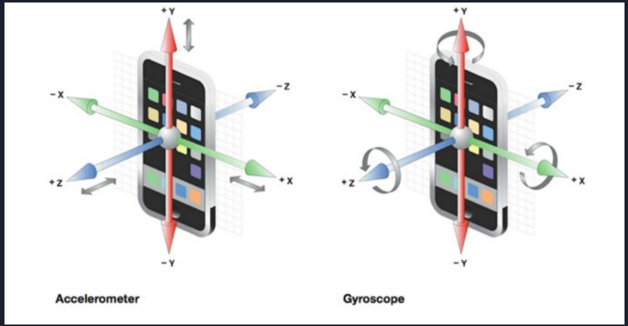


Shoumya Singh

Table of Content

Introduction





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

Classification of Fall Factors

- Fall prediction and prevention is a multifaceted problem that can be broadly categorized into two different domains: physiology and kinematics.
- In this project we are focusing on the kinematics which includes sensors.
- Accelerometer - An accelerometer is a device that measures acceleration, i.e, the rate of change of the velocity of an object.
- Gyroscope - A gyroscope gives the angular rate around one or more axes of the space. Angular measurement around lateral, longitudinal and vertical plane are referred to as pitch, roll and yaw, respectively.

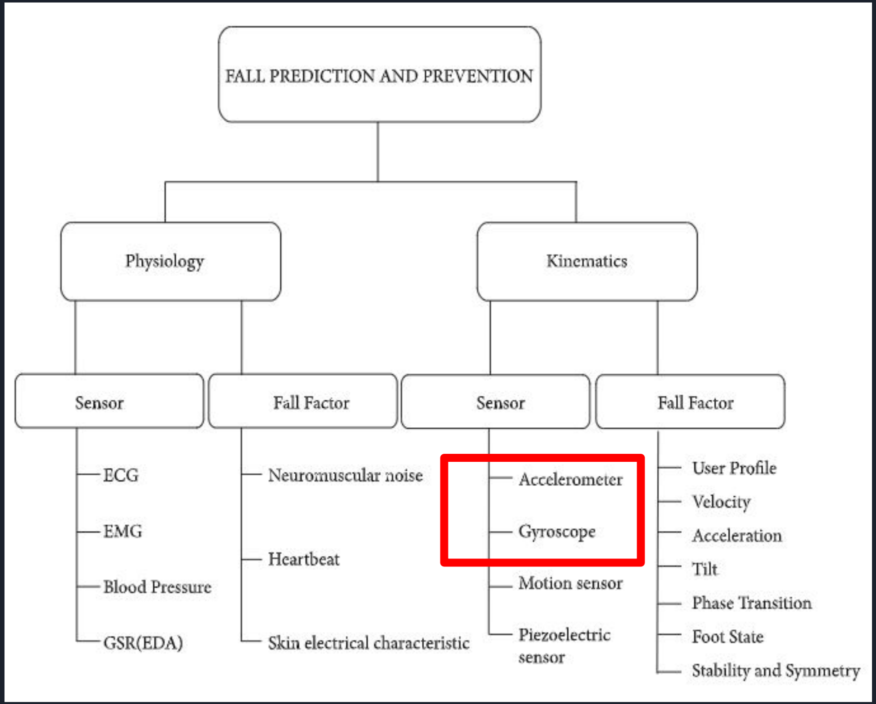
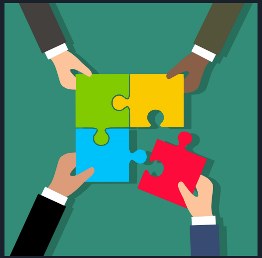


Table of Content

Project Description



- Most mobile devices are equipped with different kind of sensors.
- In this project we have the training data and test data from the accelerometer and gyroscope from a smartphone, we will apply the kNN (k-Nearest Neighbors) algorithm in two ways.
    - To manually calculate the distance using kNN and predict the results .
    - 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.

- We have the data sent from Gyroscope sensor and Accelerometer sensor to categorize any motion:
    - 3 numbers from Accelerometer sensor.
    - 3 numbers from Gyroscope sensor.
    - Total 6 numbers of sensor data.

Desired Result

Accelerometer Data			Gyroscope Data			Fall (+), Not Fall (-)
X1	Y1	Z1	X2	Y2	Z2	+/-
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	???

Prediction

In case of Python code , we will use Fall (1) /Not Fall (0)

```
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]]

#for predicting fall
testdata = [[7,6,5,5,6,7,1]]
```

kNN - k Nearest Neighbors Algorithm

- The k-nearest neighbors (**KNN**) **algorithm** is a simple, supervised machine learning **algorithm** that can be used to solve both classification and regression problems.
- In this project we have we determine K = 8 (given in the table 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

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

Table of Content

Implementation (1)-Manually calculate the distance and predict the result using kNN.

Accelerometer Data			Gyroscope Data			Fall (+), Not Fall (-)
X1	Y1	Z1	X2	Y2	Z2	+/-
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	???

Distance to each neighbor = (Target X1-Data X1) ^2 +(Target Y1-DataY1) ^2 + (Target Z1-Data Z1) ^2 +(Target X2-DataX2) ^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

Modified table with complete data in in the next slide.

Accelerometer Data			Gyroscope Data			Fall (+), Not Fall (-)	Distance to each neighbor = (Target X1-Data X1) ^2 +(Target Y1-DataY1) ^2 + (Target Z1-Data Z1) ^2 +(Target X2-DataX2) ^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	???		+

Implementation (1)-Manually calculate the distance and predict the result using kNN.

- 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 , 6,7 and 7 are selected and the corresponding to '+' sign in the **Fall (+), Not (-)** column so the prediction result will also be **'+'**.
- This indicates a Fall with '+' sign.
- Now let check Using Python to implement the application of using kNN to predict falling then compare them.

Table of Content

Implementation (2)- Using Python to implement the application of using kNN to predict falling.

1. **Using Python code to find the Euclidean distance**

```
# 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)
```

Implementation (2)- Using Python to implement the application of using kNN to predict falling.

2. Using Python code to find the nearest neighbors

```
# 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
```

Implementation (2)- Using Python to implement the application of using kNN to predict falling.

3. **Using Python code to make the prediction.**

```
# 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
```

Implementation (2)- Using Python to implement the application of using kNN to predict falling.

4. **Using Python code to find the result by adding the real and target data in the code**

```
# 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))
```

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

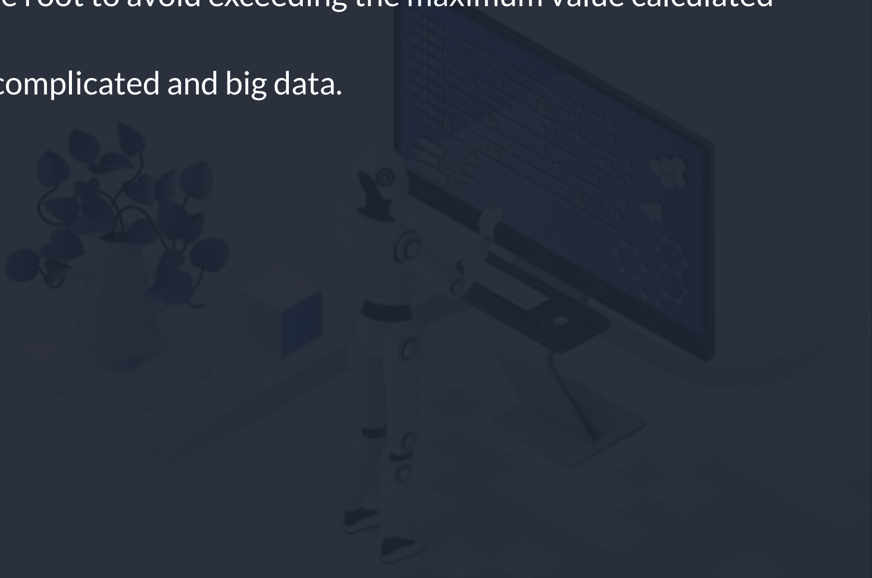
Table of Content

Test Result

- **Manually**
  - Distance to each neighbor
  - Number of nearest neighbors
  - Compare the majority sign (-/+)
  - **It is a Fall**
- **Python**
  - Find the Distance
  - Get the nearest neighbors
  - Make a Prediction
  - Add the real data and target data in the code.
  - **It is a Fall**

Conclusion

- ➔ 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.



Bibliography/References

- https://www.hindawi.com/journals/ahci/2019/9610567/
- https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
- https://npu85.npu.edu/~henry/npu/classes/data_science/algorithm/slide/exercise_algorithm.html