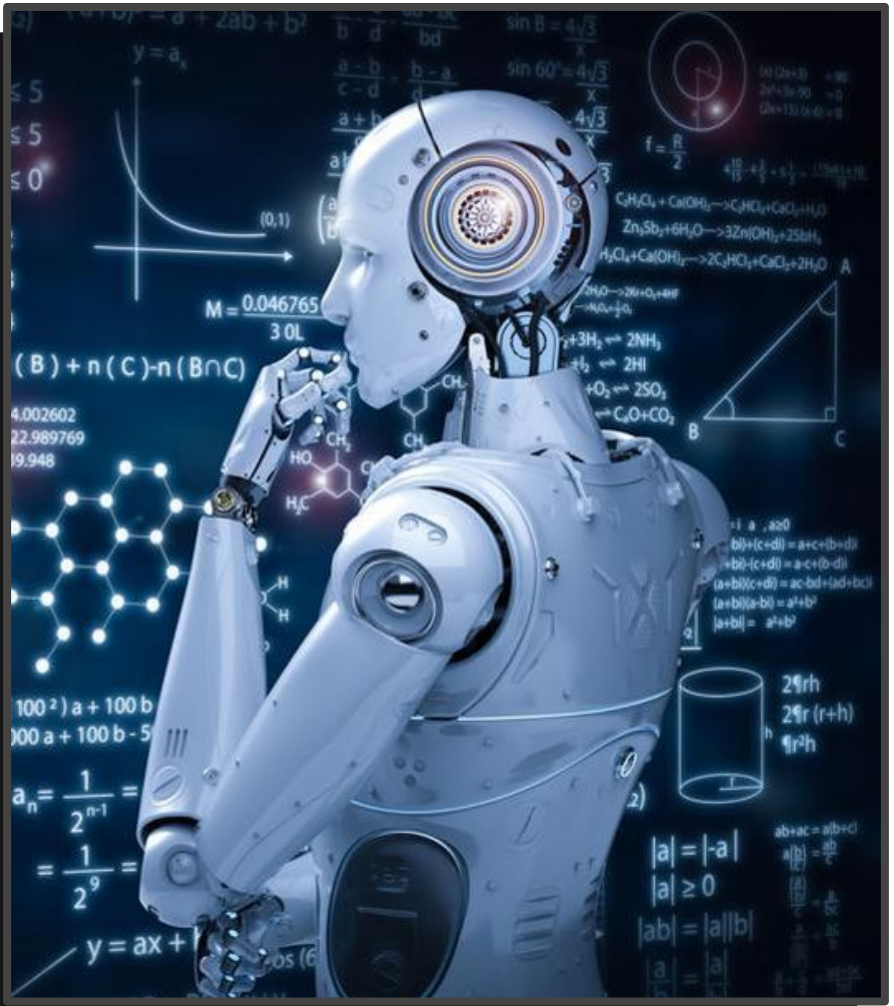

Machine Learning

Falling Prediction using KNN

Project Explanation:

Assume that we want to predict if a person was fall down.

1. Assume the person carried a smart-phone.
2. Get data from Gyroscope and Accelerometer sensor from that smart-phone.
3. Collect data and get x, y, z axis position (3 numbers) both from Gyroscope sensor and Accelerometer sensor (total 6 numbers) and record the real date if the person was fall down (fall down = 1, not fall down = 2).
4. Repeat No.3 and get enough data for testing. (Demo will only use 8 sets of data)

K-NN (K Nearest Neighbors)

A algorithm used for classification and regression in statistics.

1. Get data.
2. Calculate distance to each neighbor. =>sqrt((Target1 - data1)^2 + (Target2 - data2)^2 …)
3. Find Number “K” of nearest neighbors. => equal to the odd number closest to sqrt(number of neighbors)
4. Compare nearest neighbors and make the predict.

Accelerometer Data			Gyroscope Data			Fall(1), Not Fall(0)
X	Y	Z	X	Y	Z	1 / 0
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

```
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 ]]
```

1

Get The Data

Assume the smartphone provide the data [7, 6, 5, 5, 6, 7] , use K-nn algorithm to predict if the person who carry the phone was fall-down.

2

Accelerometer Data			Gyroscope Data			Distance	Fall(1), Not Fall(0) 1 / 0
X	Y	Z	X	Y	Z		
1	2	3	2	1	3	106	0
2	1	3	3	1	2	108	0
1	1	2	3	2	2	115	0
2	2	3	3	2	1	101	0
6	5	7	5	6	7	6	1
5	6	6	6	5	7	7	1
5	6	7	5	7	6	10	1
7	6	7	6	5	6	6	1

```
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 ]]
Target=
 [ 7, 6, 5, 5, 6, 7, ??]
```

```
sqrt((1-7)^2+(2-6)^2+(3-5)^2+(2-5)^2+(1-6)^2+(3-7)^2) = 9.486832980505138
sqrt((2-7)^2+(1-6)^2+(3-5)^2+(3-5)^2+(1-6)^2+(2-7)^2) = 9.1104335791443
sqrt((1-7)^2+(1-6)^2+(2-5)^2+(3-5)^2+(2-6)^2+(2-7)^2) = 9.486832980505138
sqrt((2-7)^2+(2-6)^2+(3-5)^2+(3-5)^2+(2-6)^2+(1-7)^2) = 8.06225774829855
sqrt((6-7)^2+(5-6)^2+(7-5)^2+(5-5)^2+(6-6)^2+(7-7)^2) = 2.449489742783178
sqrt((5-7)^2+(6-6)^2+(6-5)^2+(6-5)^2+(5-6)^2+(7-7)^2) = 2.6457513110645907
sqrt((5-7)^2+(6-6)^2+(7-5)^2+(5-5)^2+(7-6)^2+(6-7)^2) = 3.1622776601683795
sqrt((7-7)^2+(6-6)^2+(7-5)^2+(6-5)^2+(6-6)^2+(6-7)^2) = 2.449489742783178
```

Calculate Distance To Each Neighbor

In this case, there are 7 numbers in each data are: X1, y1, z1, x2, y2, z2 and fall or not (1 or 0)

The distance =
sqrt(
(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)

3

Find K

K equal to the odd number closest to sqrt(number of neighbors)

K = sqrt(8) close to 3

4

Compare nearest neighbors and make predict

3 of the nearest (smallest distance) of data are:

1. Distance 2.449 -> [6, 5, 7, 5, 6, 7, 1] -> Falled
2. Distance 2.449 -> [7, 6, 7, 6, 5, 6, 1] -> Falled
3. Distance 2.645 -> [5, 6, 6, 6, 5, 7, 1] -> Falled

Since most of the nearest neighbors are fall, then predict the test set [7, 6, 5, 5, 6, 7] is fall.

Python code – find distance

```
# -*- coding: utf-8 -*-
# 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)

#result:
#9.486832980505138
#9.1104335791443
#9.486832980505138
#8.06225774829855
#2.449489742783178
#2.6457513110645907
#3.1622776601683795
#2.449489742783178

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

  return sqrt(distance)
```

Python code - get nearest neighbors

```python
# 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
```

Python code - make 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
```

Python code - put real data and target data

```
# Test distance function
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 0, Got 1.
print('Expected %d, Got %d.' % ([7,6,5,5,6,7,1][-1], prediction))
```

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

Thank you!