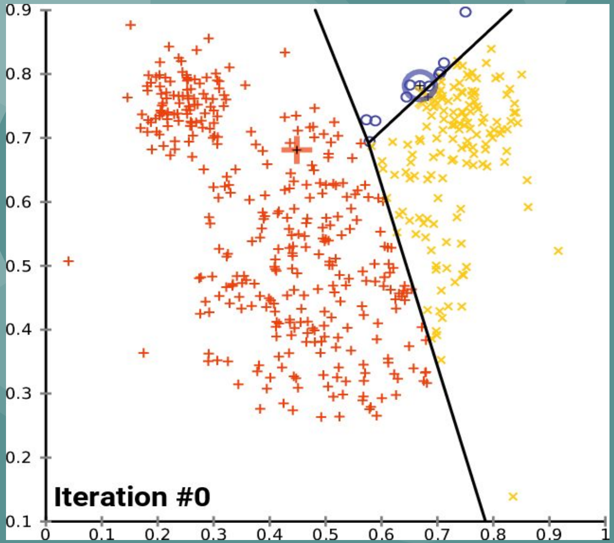
CS550 - Machine Learning and Business Intelligence



Falling Prediction using KNN

By: Maranata Muluneh

Instructor: Dr. Chang, Henry

2023

Table of Content

Introduction

Fall detection technology is critical for the elderly people. In order to avoid the need of full time care giving service, the actual trend is to encourage elderly to stay living autonomously in their homes as long as possible. Reliable fall detection methods can enhance life safety of the elderly and boost their confidence by immediately alerting fall cases to caregivers. This slide presents an algorithm of fall detection, which detects fall events by using data-mining approach. It uses k-nearest neighbour algorithm, that is, well-known lazy learning algorithm to detect fall occurrences. It detects falls by identifying the fall patterns in the data stream.

Theory

KNN is supervised learning (training data provides labels) and non-parametric Algorithm (there is no assumptions or defined function, it only learns from the data patterns).

The value of K should ideally be odd. If sqrt of the number of data points is even, then add or subtract one.

KNN requires feature scaling and is highly sensitive to outliers since it calculates the euclidean distance between the rows and would end up giving more weightage to bigger values.

Theory

- KNN requires feature scaling and is highly sensitive to outliers since it calculates the euclidean distance between the rows and would end up giving more weightage to bigger values.
- KNN can be used for continuous output variables, however, it would be KNN Regression and the output value would be calculated by taking the average of the nearest neighbors
- KNN is called Lazy Learner algorithm because it has instance-based learning, which means it does not immediately learn the model, but stores the training data and only use it while making predictions

Theory

- Choosing K value: This can be done in two ways (a) Take sqrt of the number of data points (b) Use cross-validation for hyperparameter tuning the value of K is a range e.g. (1, 21) and select the K value which gave a minimal error.
- Bias-Variance tradeoff can be noted in KNN too. Ideally the bigger the value of K, the better the prediction would be, however, this could also lead to overfitting, hence resulting in high variance.

Implementation

We have this table for the Accelerometer and Gyroscope 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	??

Implementation (Manually)

We have the predicted value manually taking

Number of training data: 8

K= sqr(8) = 2.82

K= 3

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)	Distance for Acce	Distance for Gyros
x	y	z	x	y	z	+/-		
1	2	3	2	1	3	-	(7-1)^2 + (6-2)^2 + (5-3)^2= 56	(5-2)^2 + (6-1)^2 + (7-3)^2= 50
2	1	3	3	1	2	-	(7-2)^2 + (6-1)^2 + (5-3)^2= 54	(5-3)^2 + (6-1)^2 + (7-2)^2= 54
1	1	2	3	2	2	-	(7-1)^2 + (6-1)^2 + (5-2)^2= 70	(5-3)^2 + (6-2)^2 + (7-2)^2= 45
2	2	3	3	2	1	-	(7-2)^2 + (6-2)^2 + (5-3)^2= 45	(5-3)^2 + (6-2)^2 + (7-1)^2= 56

Implementation

Hence, we have three values closest to K, highlighted in yellow. Therefore, we have '+' predicted for the last row.

6	5	7	5	6	7	+	(7-6)^2 + (6-5)^2 + (5-7)^2= 6	(5-5)^2 + (6-6)^2 + (7-7)^2= 0
5	6	6	6	5	7	+	(7-5)^2 + (6-6)^2 + (5-6)^2= 5	(5-6)^2 + (6-5)^2 + (7-7)^2= 2
5	6	7	5	7	6	+	(7-5)^2 + (6-6)^2 + (5-7)^2= 8	(5-5)^2 + (6-7)^2 + (7-6)^2= 2
7	6	7	6	5	6	+	(7-7)^2 + (6-6)^2 + (5-7)^2= 4	(5-6)^2 + (6-5)^2 + (7-6)^2= 3
7	6	5	5	6	7	+		

Implementation

Hence, we have three values closest to K, highlighted in yellow. Therefore, we have '+' predicted for the last row.

6	5	7	5	6	7	+	(7-6)^2 + (6-5)^2 + (5-7)^2= 6	(5-5)^2 + (6-6)^2 + (7-7)^2= 0
5	6	6	6	5	7	+	(7-5)^2 + (6-6)^2 + (5-6)^2= 5	(5-6)^2 + (6-5)^2 + (7-7)^2= 2
5	6	7	5	7	6	+	(7-5)^2 + (6-6)^2 + (5-7)^2= 8	(5-5)^2 + (6-7)^2 + (7-6)^2= 2
7	6	7	6	5	6	+	(7-7)^2 + (6-6)^2 + (5-7)^2= 4	(5-6)^2 + (6-5)^2 + (7-6)^2= 3
7	6	5	5	6	7	+		

Implementation (Using Colab)

Environment : Colab, Tensorflow 2

Programming Language: Python

Libraries

```
from math import sqrt
```

Implementation (Using Colab)

Using Python to implement the application of using kNN to predict falls.

The steps to implement the K- nearest neighbors can be broken down into three parts

Step 1: Calculate Euclidean Distance.

$$\textit{Euclidean Distance} = sqrt(sum\ i\ to\ N\ (x1\_i - x2\_i)^2)$$

Step 2: Get Nearest Neighbors.

Step 3: Make Predictions.

Implementation (Using Colab)

```
def euclidean_distance(row1, row2): # row 1 and row 2 are vectors

  distance = 0.0 # initializing distance variable as 0

  for i in range(len(row1)-1): # iterate along the index in the two vectors

    distance += (row1[i] - row2[i])**2  # sum of the total distance

  return sqrt(distance) # return sq. root of the distance
```

```
from math import sqrt
# calculate the Euclidean distance between two vectors
#     Euclidean Distance = sqrt(sum i to N (x1_i - x2_i)^2)
# Result:
#    10.295630140987
#    10.392304845413264
#    10.723805294763608
#    10.04987562112089
#    2.449489742783178
#    2.6457513110645907
#    3.1622776601683795
#    2.6457513110645907
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],
#   [5,6,6,6,5,7,1],
# [7,6,7,6,5,6,1]]
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))
```

Implementation (Using Colab)

KNN.ipynb

File Edit View Insert Runtime Tools Help All changes sa

Comment Share

+ Code + Text

RAM Disk Editing

```
  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
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],
             [7,6,5,5,6,7,1]]
```

Implementation (Using Colab)

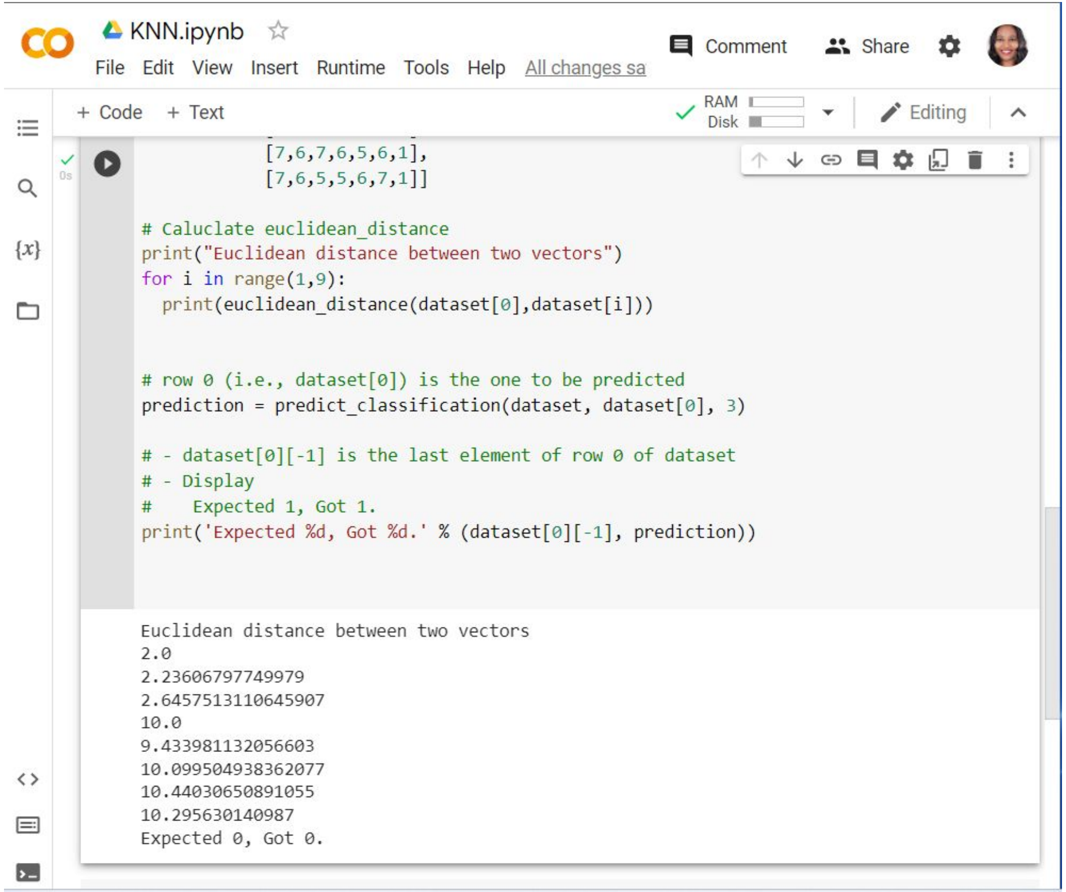
```
def
predict_classification(train,
test_row, num_neighbors):

  neighbors =
get_neighbors(train, test_row,
num_neighbors) # get the top K
neighbors

  output_values = [row[-1] for
row in neighbors] # prepare a
list of the distances of top K
neighbors

  prediction =
max(set(output_values),
key=output_values.count)

  return prediction # return
the label with a max count
```



References

- Paliwal, A. (2021, June 26). Writing KNN in python from scratch. Medium. Retrieved February 7, 2023, from https://medium.com/geekculture/writing-knn-in-python-from-scratch-67123907165
- A data mining approach for fall detection by using K-nearest neighbor ... (n.d.). Retrieved February 7, 2023, from https://www.researchgate.net/publication/260508814_A_data_mining_approach_for_fall_detection_by_using_k-nearest_neighbor_algorithm_on_wireless_sensor_network_data