Falling Prediction using KNN

By Fatema Nagori _19635_CS550

Table of Content:

INTRODUCTION

k-Nearest Neighbors (k-NN) is a supervised machine learning algorithm used for classification and regression. Given a set of labeled examples, the algorithm tries to predict the class or the value of an unseen sample by finding the "k" closest examples in the feature space and aggregating the results (e.g., majority voting for classification and average for regression). The "k" is a hyperparameter that the user must set and determines the number of neighbors considered for the prediction.

Design

Most mobile devices are equipped with different kind of sensors.

We can use the data sent from Gyroscope sensor and Accelerometer sensor 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	??

Prediction

Calculate Euclidean Distance

The Euclidean distance between two points in a multi-dimensional space is calculated as the square root of the sum of the squared differences between their coordinates.

Mathematically, the Euclidean distance "d" between two points "p" and "q" in an n-dimensional space can be expressed as:

d(p,q) = sqrt(sum((p_i - q_i)^2)) for i = 1 to n

where "p_i" and "q_i" are the i-th coordinates of the points "p" and "q".

Get Nearest Neighbors

- To locate the neighbors for a new piece of data within a dataset we must first calculate the distance between each record in the dataset to the new piece of data.
- Once distances are calculated, we must sort all of the records in the training dataset by their distance to the new data. We can then select the top k to return as the most similar neighbors.

IMPLEMENTATION

Get Nearest Neighbors - Formula

- A general rule of thumb: K = the closest odd number of the square root of the number of samples.
  - ➢ K = sqrt(number of data samples)
- If no winner, then pick the next odd number greater than K
- Since K = 5, we select the closest neighbors.

Example = sqrt(number of data samples)

=sqrt(25) = 5

PREDICTIONIONS

- The KNN prediction of the query instance is based on simple majority of the category of nearest neighbors.
  - In our example, the data is only binary, thus the majority can be taken as simple as counting the number of ‘+' and ‘-‘ signs.
  - If the number of plus is greater than minus, we predict the query instance as plus and vice versa.
  - If the number of plus is equal to minus, we can choose arbitrary or determine as one of the plus or minus.

Find K ???

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)	Distance to each neighbor = (Target$_{X1}$-Data$_{X1}$)^2 +(Target$_{X2}$-Data$_{X2}$)^2 +(Target$_{X2}$-Data$_{X2}$)^2 = (7-X1)^2+(6-Y1)^2+ (5-Z1)^2+(5-X2)^2+ (6-Y2)^2+(7-Z2)^2	K =Number of nearest neighbors =sqrt(number of neighbors) =sqrt(number of data samples) =sqrt(8) =4, but we take odd number to choose better so will take 5
X1	Y1	Z1	X2	Y2	Z2	+/-		
1	2	3	2	1	3	-	(7-1)^2+(6-2)^2+(5-3)^2+(5-2)^2+(6-1)^2+(7-3)^2=106	
2	1	3	3	1	2	-	(7-2)^2+(6-1)^2+(5-3)^2+(5-3)^2+(6-1)^2+(7-2)^2=108	
1	1	2	3	2	2	-	(7-1)^2+(6-1)^2+(5-2)^2+(5-3)^2+(6-2)^2+(7-2)^2=115	
2	2	3	3	2	1	-	(7-2)^2+(6-2)^2+(5-3)^2+(5-3)^2+(6-2)^2+(7-1)^2=101	-
6	5	7	5	6	7	+	(7-6)^2+(6-5)^2+(5-7)^2+(5-5)^2+(6-6)^2+(7-7)^2=6	+
5	6	6	6	5	7	+	(7-5)^2+(6-6)^2+(5-6)^2+(5-6)^2+(6-5)^2+(7-7)^2=7	+
5	6	7	5	7	6	+	(7-5)^2+(6-6)^2+(5-7)^2+(5-5)^2+(6-7)^2+(7-6)^2=10	+
7	6	7	6	5	6	+	(7-7)^2+(6-6)^2+(5-7)^2+(5-6)^2+(6-5)^2+(7-6)^2=7	+
7	6	5	5	6	7	??		+

TEST: Program to find the Euclidean Distance

```
# 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 Neighbors

```
# 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))
        distances.sort(key=lambda tup: tup[1])
        neighbors = list()
        for i in range(num_neighbors):
                neighbors.append(distances[i][0])
        return neighbors
```

Classification prediction with Neighbors

Make a classification prediction with neighbors.

- test_row is row 0

- num_neighbors is

```
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
```

Predict value on Test Data

```
# Test distance function
dataset =  [[7,6,5,5,6,7,1],
            [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]]
```

```
# Calculate euclidean_distance
print("Euclidean distance between two vectors")
for i in range(1,len(dataset)):
        print(euclidean_distance(dataset[0],dataset[i]))
# row 0 (i.e., dataset[0]) is the one to be predicted
prediction = predict_classification(dataset, dataset[0], 3)
# - dataset[0][-1] is the last element of row 0 of dataset
# - Display
#    Expected 1, Got 1.
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

Result : Run on Colab

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

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

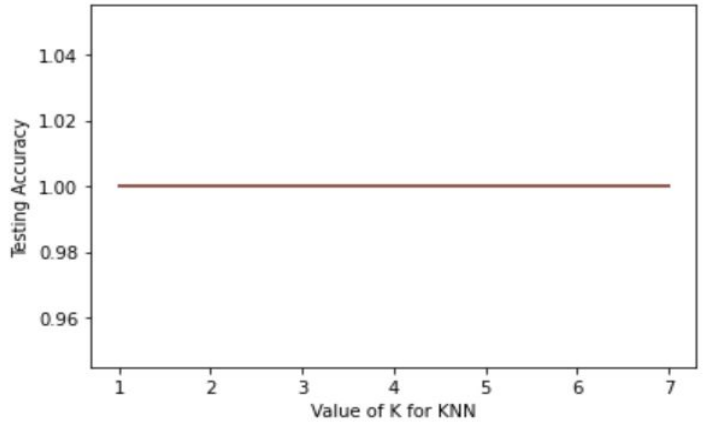
```
Euclidean distance between two vectors
10.295630140987
10.392304845413264
10.723805294763608
10.04987562112089
2.449489742783178
2.6457513110645907
3.1622776601683795
2.6457513110645907
Expected 1, Got 1.
```

RESULT: Run on Colab

```
# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(x_train, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

Out[22]: Text(0, 0.5, 'Testing Accuracy')



ENHANCEMENT

Ways to enhance the k-Nearest Neighbors (k-NN) algorithm include: feature scaling, hyperparameter tuning (choice of k), using alternative distance metrics, dimensionality reduction, ensemble methods, lazy learning techniques, and weighted voting.

CONCLUSION

The k-Nearest Neighbors (k-NN) algorithm is a simple yet effective method for both classification and regression problems. It finds the k closest examples in the feature space and aggregates their labels/values to make a prediction for a new sample. Advantages include easy understanding and implementation, and ability to handle non-linear relationships. Limitations include dependence on the choice of k and distance metric, slow prediction time for large datasets, and high memory requirement. Despite these limitations, k-NN remains a popular choice in pattern recognition and computer vision.

REFERENCES

COLAB

PROF HENRY'S MATERIAL

KNN ALGORITHM


Thank
You