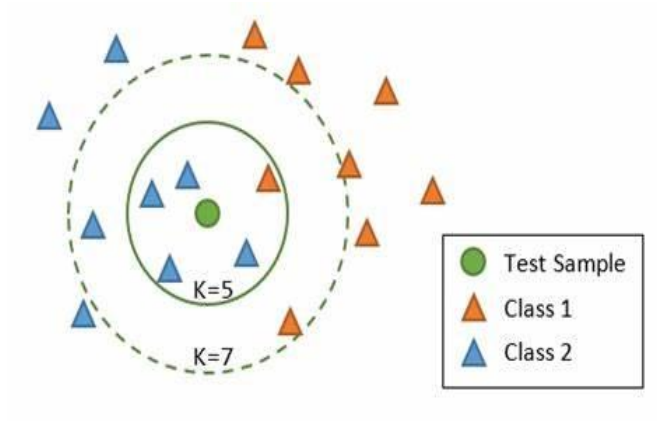
Falling Detection: Python + kNN + Colab

CS550 - Machine Learning and Business Intelligence



Submitted by: Divya Pandey(19665)
Instructor: Dr. Henry Chang

Table of Content

- Introduction
- Design
- Implementation
- Test
- Enhancement Ideas
- Conclusion
- References

Introduction

★ k-Nearest-Neighbors (k-NN) is a simple machine learning algorithm used for classification and regression tasks.

★ It works by finding the k number of closest training examples to a new data point and classifying the new data point based on the majority of the class labels of its k nearest neighbors.

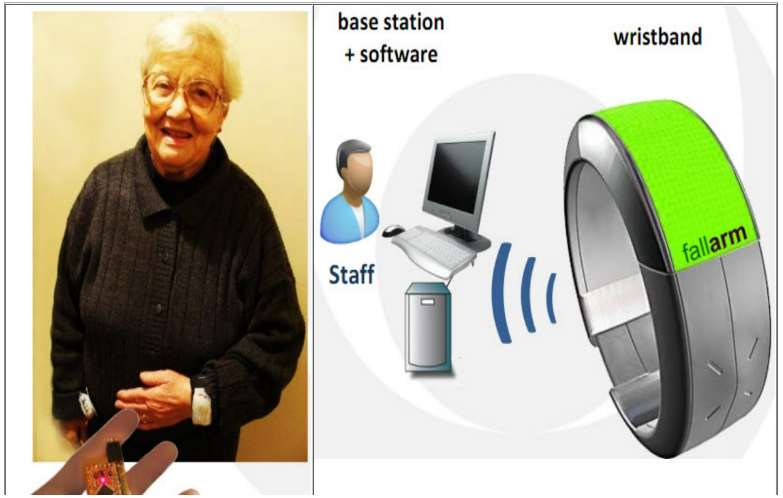
★ The distance metric used to calculate the closest neighbors is typically Euclidean distance.

Introduction

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.



Design (Calculate the distance to each Neighbor)

We calculate the distance between each neighbor in the dataset using Euclidean distance.

- It is calculated as the square root of the sum of the squared differences between the two vectors.
    - Euclidean Distance = sqrt(sum i to N (x1_i – x2_i)^2)

Where x1 is the first row of data, x2 is the second row of data and i is the index to a specific column as we sum across all columns.

Design (Get Nearest Neighbors)

★ To find the nearest neighbors for a new piece of data in a dataset, we must first calculate the distance between each record in the dataset and the new piece of data

★ Once the distances between each record in the training dataset and the new data have been calculated, the next step is to sort all of the records by their distance to the new data

Design(Get Nearest Neighbors)

- 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

Example = sqrt(number of data samples)

=sqrt(9) = 3

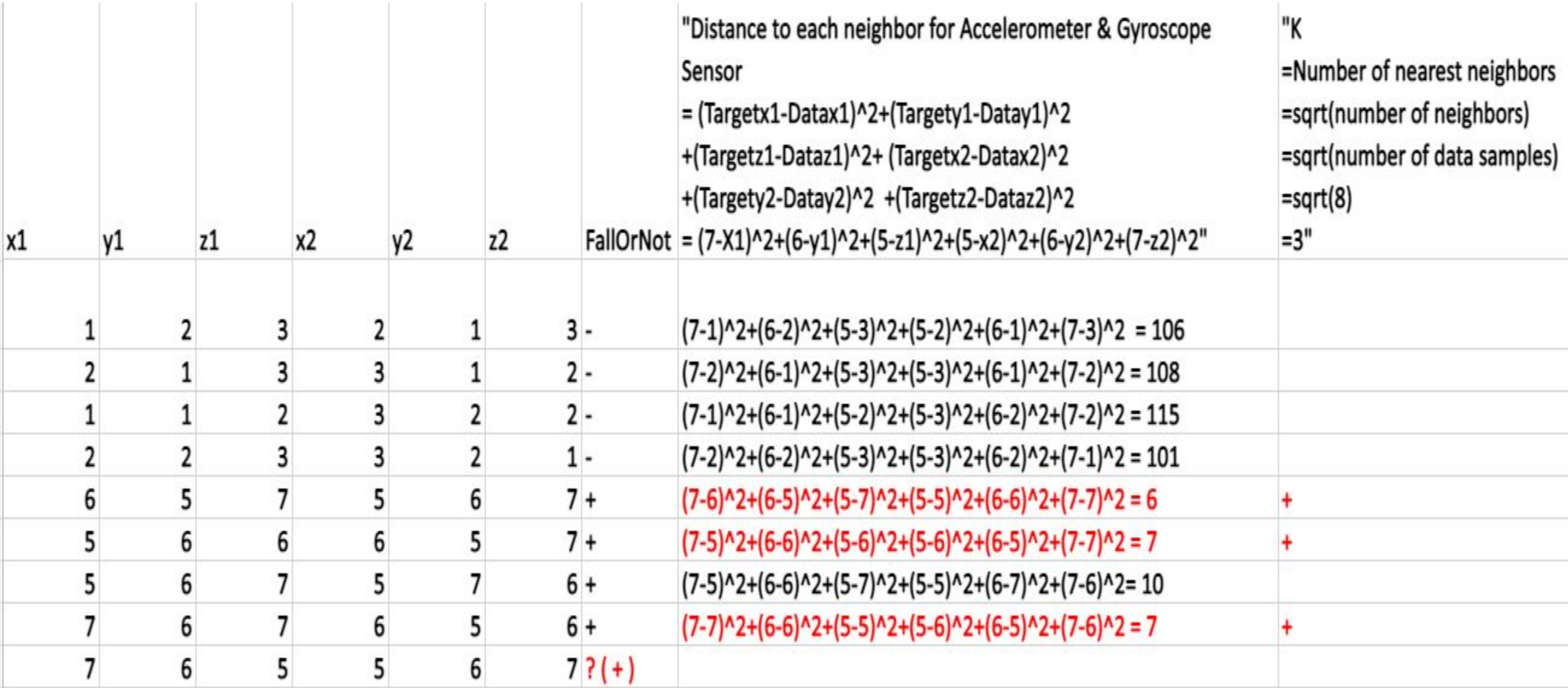
Design(Make Predictions)

- 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.

Design(Find the value of K)



x1	y1	z1	x2	y2	z2	FallOrNot	"Distance to each neighbor for Accelerometer & Gyroscope Sensor = (Targetx1-Datax1)^2+(Targety1-Datay1)^2 +(Targetz1-Dataz1)^2+ (Targetx2-Datax2)^2 +(Targety2-Datay2)^2 +(Targetz2-Dataz2)^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) =3"
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+(5-5)^2+(5-6)^2+(6-5)^2+(7-6)^2 = 7	+
7	6	5	5	6	7	? (+)		

Design (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
```

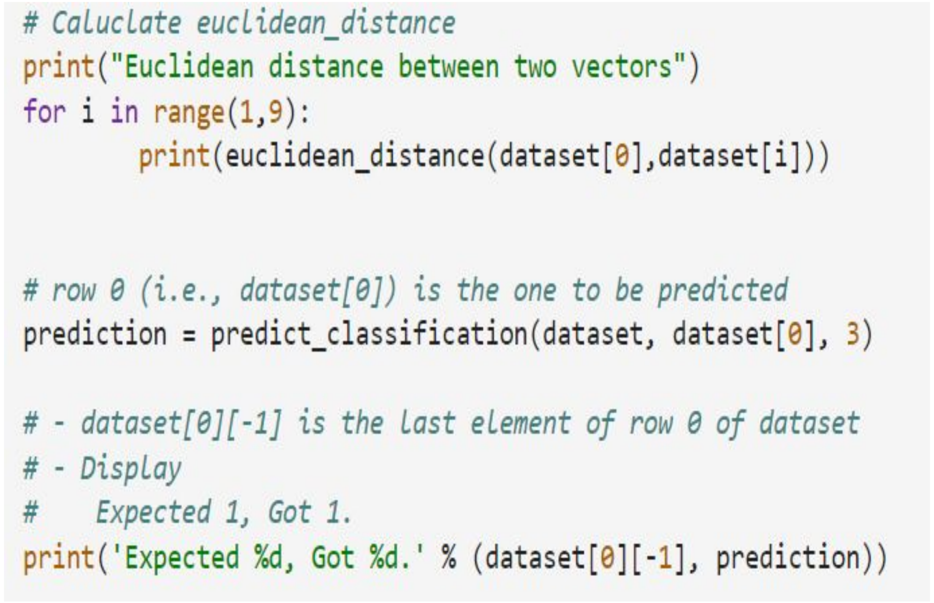
Design (Locate the most Similar Neighbors)

# Result

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

Design (Predict value on the Test Data)

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]]

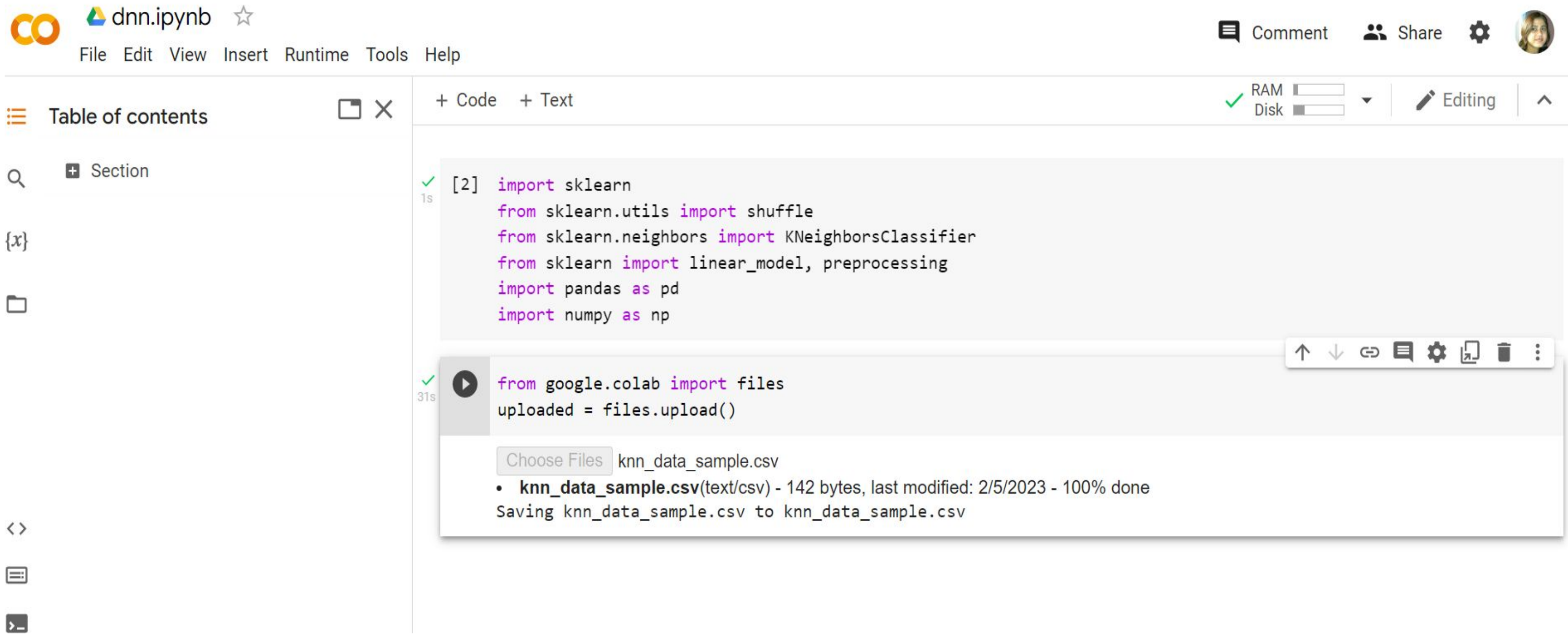


```
# Caluclate euclidean_distance
print("Euclidean distance between two vectors")
for i in range(1,9):
        print(euclidean_distance(dataset[0],dataset[i]))


# 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))
```

Implementation

- ★ Go to **Colab**
- ★ Download the **Sample Data**
- ★ Run Python Code of Scikit-learn library
- ★ Run Python Code to Predict kNN Value

Implementation



```
[2] import sklearn
    from sklearn.utils import shuffle
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn import linear_model, preprocessing
    import pandas as pd
    import numpy as np
```

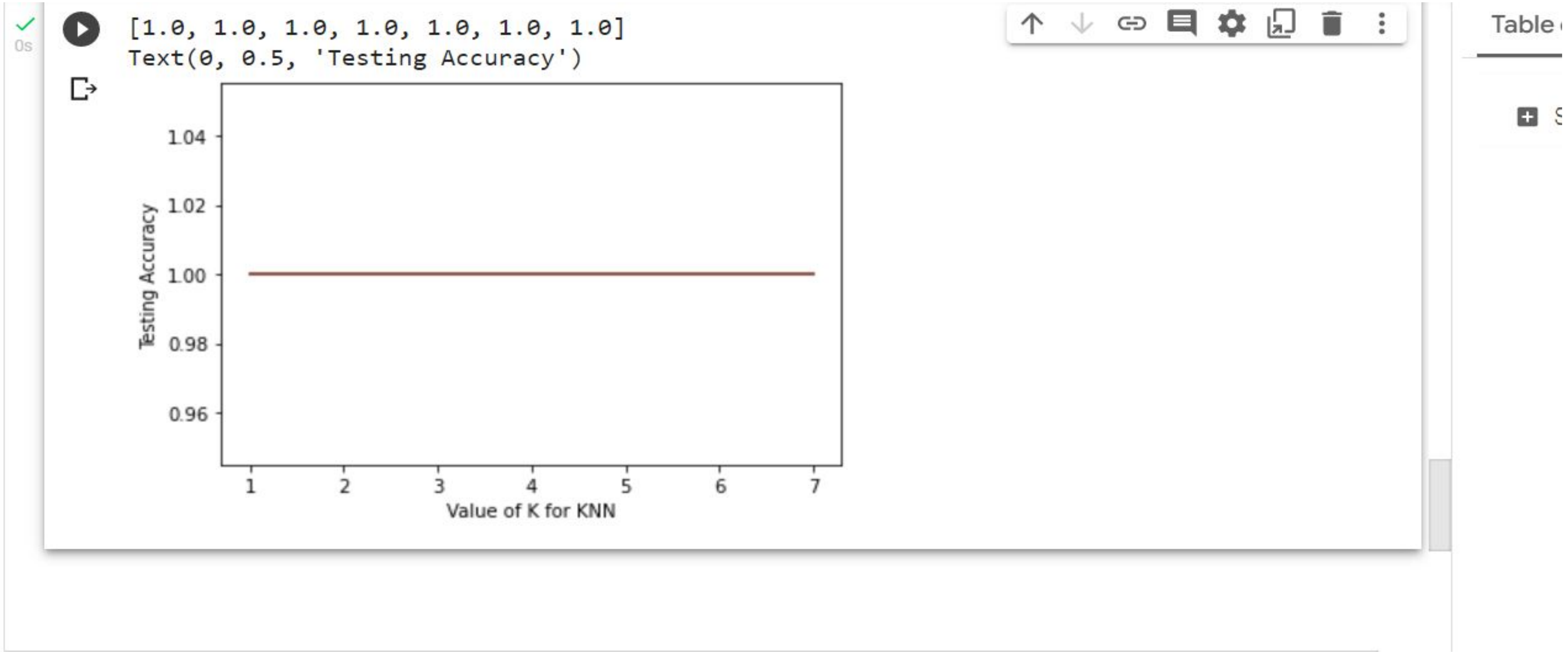
```
from google.colab import files
uploaded = files.upload()
```

Choose Files knn_data_sample.csv

- **knn_data_sample.csv**(text/csv) - 142 bytes, last modified: 2/5/2023 - 100% done

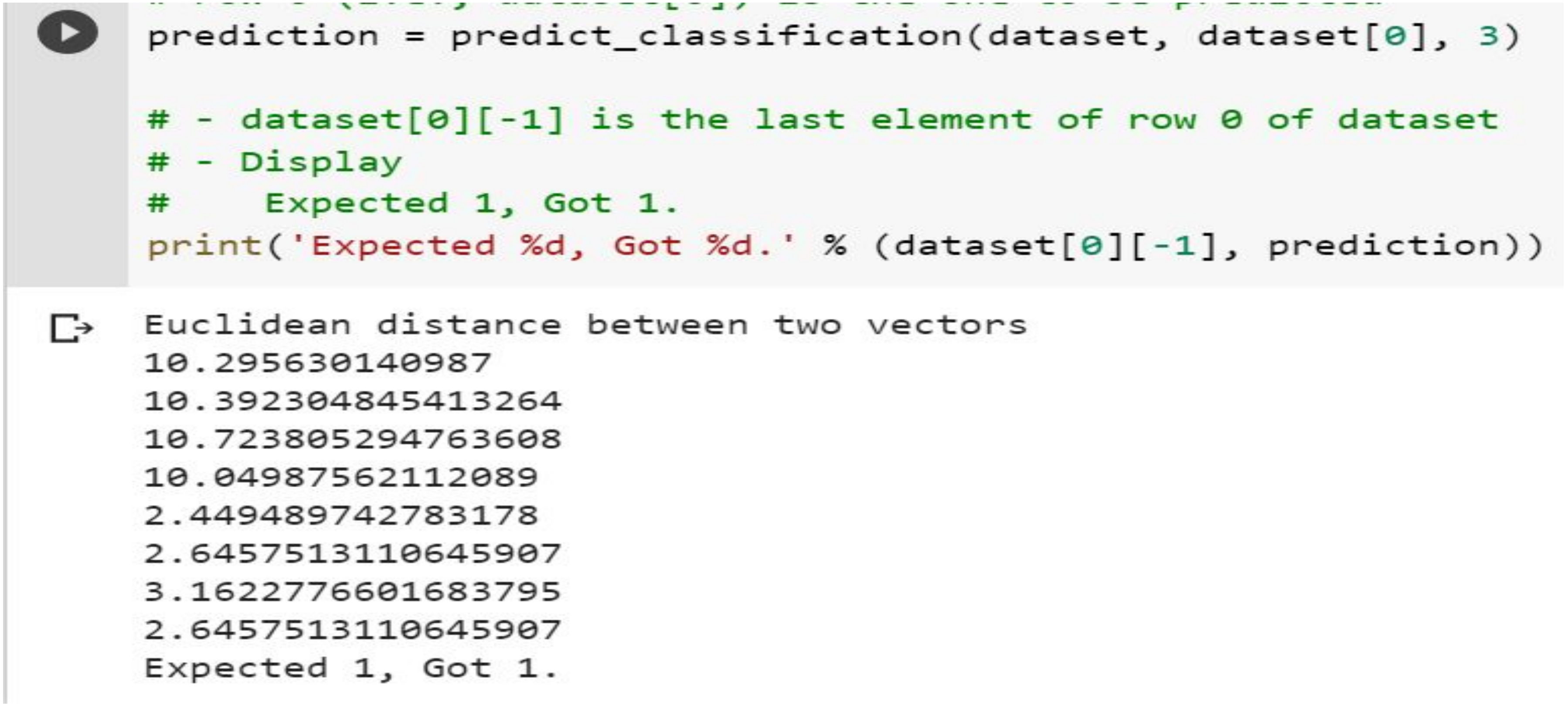
```
Saving knn_data_sample.csv to knn_data_sample.csv
```

Test (Scikit-learn library)



```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Text(0, 0.5, 'Testing Accuracy')
```

Test -Predict kNN Value



```
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.
```

Enhancement Ideas

★ We can use weighted k-NN.In traditional k-NN algorithm, each of the k nearest neighbors has equal weight in determining the class label or target variable. However, weighted k-NN gives more weight to the nearest neighbors and less weight to the farther neighbors. This can lead to improved performance.

Conclusion

★ The value of k, the number of nearest neighbors to consider, is a hyperparameter that can be tuned to achieve the best performance on a specific dataset.

★ In general, a small value of k may result in overfitting, while a large value of k may result in underfitting.

References

★ *Santhinagalla. (n.d.). Machine-learning/supervised learning/falling prediction using KNN at main · santhinagalla/machine-learning. Retrieved February 5, 2023, from https://github.com/santhinagalla/Machine-Learning/tree/main/Supervised%20Learning/Falling%20Prediction%20using%20KNN*

★ *Brownlee, J. (2020, February 23). Develop K-nearest neighbors in python from scratch. Retrieved February 5, 2023, from https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/*