ComputingTheEuclideanDistance

August 8, 2023

1 Computing the Euclidean Distance

[6]:
```
import pandas as pd
import numpy as np
import os
import math
import random
import matplotlib.pyplot as plt
```

1.0.1 Load a Data Set and Save it as a Pandas DataFrame

We will work with a new data set called "cell2cell." This data set is used to analyze cellular telephone customers and can be used to predict whether a customer will remain with their current telecom service or leave to another.

[7]:
```
filename = os.path.join(os.getcwd(), "data","cell2cell.csv")
df = pd.read_csv(filename, header=0)
```

1.0.2 Inspect the Data

[8]:
```
df.head()
```

[8]:
```
   CustomerID  Churn ServiceArea  ChildrenInHH  HandsetRefurbished  \
0     3000002   True   SEAPOR503         False               False
1     3000010   True   PITHOM412          True               False
2     3000014  False   MILMIL414          True               False
3     3000022  False   PITHOM412         False               False
4     3000026   True   OKCTUL918         False               False

   HandsetWebCapable  TruckOwner  RVOwner  HomeownershipKnown  \
0               True       False    False                True
1              False       False    False                True
2              False       False    False               False
3               True       False    False                True
4              False       False    False                True

   BuysViaMailOrder  ...  HandsetModels  CurrentEquipmentDays    AgeHH1  \
```

```
0              True  ...        0.487071              -0.077013  1.387766
1              True  ...       -0.616775               3.019920  0.392039
2             False  ...       -0.616775               3.019920 -0.241605
3              True  ...        2.694763               0.305179 -0.060564
4              True  ...        1.590917               1.857585  0.663601

     AgeHH2  RetentionCalls  RetentionOffersAccepted  \
0 -0.883541        4.662897                  -0.1283
1  0.871495       -0.180167                  -0.1283
2  0.202910       -0.180167                  -0.1283
3 -0.883541       -0.180167                  -0.1283
4  1.372934       -0.180167                  -0.1283

   ReferralsMadeBySubscriber  IncomeGroup  AdjustmentsToCreditRating  \
0                  -0.169283    -0.103411                  -0.140707
1                  -0.169283     0.215243                  -0.140707
2                  -0.169283     0.533896                  -0.140707
3                  -0.169283     0.533896                  -0.140707
4                  -0.169283     1.489856                   2.469282

  HandsetPrice
0    -0.864858
1    -0.864858
2    -0.368174
3    -1.195980
4    -1.195980

[5 rows x 58 columns]
```

[9]:
```
df.dtypes
```

[9]:
```
CustomerID                   int64
Churn                         bool
ServiceArea                 object
ChildrenInHH                  bool
HandsetRefurbished            bool
HandsetWebCapable             bool
TruckOwner                    bool
RVOwner                       bool
HomeownershipKnown            bool
BuysViaMailOrder              bool
RespondsToMailOffers          bool
OptOutMailings                bool
NonUSTravel                   bool
OwnsComputer                  bool
HasCreditCard                 bool
NewCellphoneUser              bool
NotNewCellphoneUser           bool
```

```
OwnsMotorcycle                 bool
MadeCallToRetentionTeam        bool
CreditRating                 object
PrizmCode                    object
Occupation                   object
Married                      object
MonthlyRevenue              float64
MonthlyMinutes              float64
TotalRecurringCharge        float64
DirectorAssistedCalls       float64
OverageMinutes              float64
RoamingCalls                float64
PercChangeMinutes           float64
PercChangeRevenues          float64
DroppedCalls                float64
BlockedCalls                float64
UnansweredCalls             float64
CustomerCareCalls           float64
ThreewayCalls               float64
ReceivedCalls               float64
OutboundCalls               float64
InboundCalls                float64
PeakCallsInOut              float64
OffPeakCallsInOut           float64
DroppedBlockedCalls         float64
CallForwardingCalls         float64
CallWaitingCalls            float64
MonthsInService             float64
UniqueSubs                  float64
ActiveSubs                  float64
Handsets                    float64
HandsetModels               float64
CurrentEquipmentDays        float64
AgeHH1                      float64
AgeHH2                      float64
RetentionCalls              float64
RetentionOffersAccepted     float64
ReferralsMadeBySubscriber   float64
IncomeGroup                 float64
AdjustmentsToCreditRating   float64
HandsetPrice                float64
dtype: object
```

[10]:
```
df.shape
```

[10]:
```
(51047, 58)
```

1.1 Euclidean Distance

KNN k-Nearest Neighbors (KNN) is an instance-based learning algorithm. To make a classification for a given unlabeled example A, we search the training data for the k nearest neighbors, as defined by some distance metric $d(A, B)$ in which B represents another example. We choose the most common label among the nearest neighbor examples to be our prediction (label) for the unlabeled example.

The most commonly used distance metric for KNN is the Euclidean distance.

Euclidean Distance For two n-dimensional, real-valued vectors $A, B \in \mathbb{R}^n$, the Euclidean distance *eud* is defined as:

$$eud(A, B) = \sqrt{\sum_{i=1}^{n} (B_i - A_i)^2}$$

Euclidean distance finds the distance between two vectors of the same length. In this formula, A_i is the *ith* coordinate of vector A, and B_i is the *ith* coordinate of vector B.

Let's relate this to a dataset. Let's think of the vectors A and B as being two examples (rows) in a dataset.

Let $A =< x_1^a, ...x_n^a >$ be a n-dimensional vector (x_i^a is the *ith* feature in example A and n is the total number of features).

Then for two vectors (examples) A and B the Euclidean distance is defined as:

$$eud(A, B) = \sqrt{(x_1^b - x_1^a)^2 + (x_2^b - x_2^a)^2 + ... + (x_n^b - x_n^a)^2} = \sqrt{\sum_{i=1}^{n} (x_i^b - x_i^a)^2}$$

To visualize KNN, you can picture plotting the examples (also called data points) in our dataset and finding the distance between them. Let's create a visualization to see how we plot examples and find the distance between each example.

To easily visualize this, let's plot two examples from DataFrame `df`. Note that each example contains many features, but to make this visualization even simpler, we will work with two dimensions (that is, two features).

Euclidean distance is best used to calculate the distance between vectors containing numerical values. Therefore, we will choose two features that have numerical values.

Let us use row 0 and row 4 in DataFrame `df` and focus on features `HandsetModels` and `AgeHH1`. Run the code below to examine the two examples we will be plotting.

[11]:
```
display(df.loc[[0,4],['HandsetModels','AgeHH1']])
```

	HandsetModels	AgeHH1
0	0.487071	1.387766
4	1.590917	0.663601

Each example (row) can be viewed as a vector:

example 1: `(0.487071, 1.387766)`

example 2: `(1.590917, 0.663601)`

You will use the Euclidean distance formula to find the distance between these two vectors. First, let's plot these vectors. Run the code cell below to generate a plot. Examine the resulting plot.

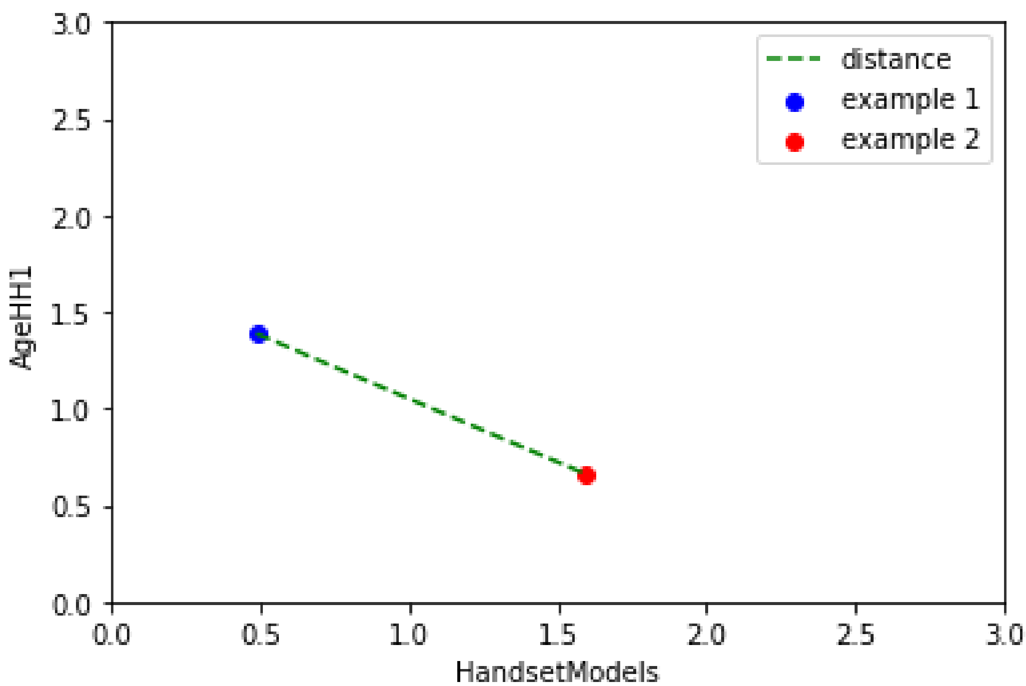
[12]:
```
# example 1 (row 0):
vector_A = [df.loc[0]['HandsetModels'], df.loc[0]['AgeHH1']]

# example 2 (row 4):
vector_B = [df.loc[4]['HandsetModels'], df.loc[4]['AgeHH1']]


plt.scatter(vector_A[0],vector_A[1] ,c='b',label='example 1')
plt.scatter(vector_B[0],vector_B[1], c='r', label='example 2')
plt.plot([vector_A[0],vector_B[0]], [vector_A[1],vector_B[1]], c='g',␣
↪linestyle='dashed', label ='distance')

plt.xlim([0, 3])
plt.ylim([0, 3])
plt.xlabel('HandsetModels')
plt.ylabel('AgeHH1')

plt.legend(loc='upper right');
plt.show()
```



You will use the Euclidean distance formula to find the distance between these two vectors.

Use the Euclidean distance formula to calculate the distance between `vector_A` and `vector_B` by hand and save the result to variable `euc_distance`.

For simplicity, use the following rounded vector values in your calculation:

vector_A: `(0.5, 1.4)`

vector_B: (1.6, 0.7)

1.1.1 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[13]:
```
# YOUR CODE HERE
euc_distance=1.303
```

1.1.2 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell.

[14]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testEuc

try:
    p, err = testEuc(euc_distance)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

1.2 Step 1: Filter Numerical Features

We will now compute the Euclidean distance between two rows in DataFrame `df`, using all of their numerical feature values. Let us create a new DataFrame that contains only the numerically valued columns of the original `df` DataFrame.

[15]:
```
df_numerical = df.select_dtypes(include=['int64','float64'])

print(df_numerical.shape)
df_numerical.head()
```

```
(51047, 36)
```

[15]:

	CustomerID	MonthlyRevenue	MonthlyMinutes	TotalRecurringCharge	\
0	3000002	-0.782676	-0.578738	-1.041153	
1	3000010	-0.940180	-0.973177	-1.250809	
2	3000014	-0.468118	-0.976952	-0.370255	
3	3000022	0.526784	1.484048	1.181196	
4	3000026	-0.936810	-0.992050	-1.250809	

	DirectorAssistedCalls	OverageMinutes	RoamingCalls	PercChangeMinutes	\
0	-0.289532	-0.414422	-0.125914	-0.564836	

```
1             -0.401714       -0.414422     -0.125914          0.029311
2             -0.401714       -0.414422     -0.125914          0.037077
3              0.154708       -0.414422     -0.125914          0.654524
4             -0.401714       -0.414422     -0.125914          0.044844

   PercChangeRevenues  DroppedCalls  ...  HandsetModels  CurrentEquipmentDays  \
0           -0.449987     -0.587303  ...       0.487071             -0.077013
1            0.030120     -0.631532  ...      -0.616775              3.019920
2            0.030120     -0.664703  ...      -0.616775              3.019920
3            0.234797      4.012499  ...       2.694763              0.305179
4            0.025066     -0.664703  ...       1.590917              1.857585

     AgeHH1    AgeHH2  RetentionCalls  RetentionOffersAccepted  \
0  1.387766 -0.883541        4.662897                  -0.1283
1  0.392039  0.871495       -0.180167                  -0.1283
2 -0.241605  0.202910       -0.180167                  -0.1283
3 -0.060564 -0.883541       -0.180167                  -0.1283
4  0.663601  1.372934       -0.180167                  -0.1283

   ReferralsMadeBySubscriber  IncomeGroup  AdjustmentsToCreditRating  \
0                  -0.169283    -0.103411                  -0.140707
1                  -0.169283     0.215243                  -0.140707
2                  -0.169283     0.533896                  -0.140707
3                  -0.169283     0.533896                  -0.140707
4                  -0.169283     1.489856                   2.469282

   HandsetPrice
0     -0.864858
1     -0.864858
2     -0.368174
3     -1.195980
4     -1.195980

[5 rows x 36 columns]
```

We will exclude the CustomerID column, since it contains the customer ID and is not a feature that we want to consider.

[16]:
```
df_numerical = df_numerical.drop(columns=['CustomerID'])
df_numerical.head()
```

[16]:
```
   MonthlyRevenue  MonthlyMinutes  TotalRecurringCharge  \
0       -0.782676       -0.578738             -1.041153
1       -0.940180       -0.973177             -1.250809
2       -0.468118       -0.976952             -0.370255
3        0.526784        1.484048              1.181196
4       -0.936810       -0.992050             -1.250809

   DirectorAssistedCalls  OverageMinutes  RoamingCalls  PercChangeMinutes  \
```

```
0              -0.289532       -0.414422     -0.125914             -0.564836
1              -0.401714       -0.414422     -0.125914              0.029311
2              -0.401714       -0.414422     -0.125914              0.037077
3               0.154708       -0.414422     -0.125914              0.654524
4              -0.401714       -0.414422     -0.125914              0.044844

   PercChangeRevenues  DroppedCalls  BlockedCalls  ...  HandsetModels  \
0           -0.449987     -0.587303     -0.309284  ...       0.487071
1            0.030120     -0.631532     -0.373230  ...      -0.616775
2            0.030120     -0.664703     -0.373230  ...      -0.616775
3            0.234797      4.012499      0.330172  ...       2.694763
4            0.025066     -0.664703     -0.373230  ...       1.590917

   CurrentEquipmentDays    AgeHH1    AgeHH2  RetentionCalls  \
0             -0.077013  1.387766 -0.883541        4.662897
1              3.019920  0.392039  0.871495       -0.180167
2              3.019920 -0.241605  0.202910       -0.180167
3              0.305179 -0.060564 -0.883541       -0.180167
4              1.857585  0.663601  1.372934       -0.180167

   RetentionOffersAccepted  ReferralsMadeBySubscriber  IncomeGroup  \
0                  -0.1283                  -0.169283    -0.103411
1                  -0.1283                  -0.169283     0.215243
2                  -0.1283                  -0.169283     0.533896
3                  -0.1283                  -0.169283     0.533896
4                  -0.1283                  -0.169283     1.489856

   AdjustmentsToCreditRating  HandsetPrice
0                  -0.140707     -0.864858
1                  -0.140707     -0.864858
2                  -0.140707     -0.368174
3                  -0.140707     -1.195980
4                   2.469282     -1.195980

[5 rows x 35 columns]
```

We will compute the Euclidean distance between two examples in our data. In other words, our vectors A and B will be two distinct *rows* of our DataFrame `df_numerical` (which we filtered to include only numerical columns).

The code cell below randomly samples two rows from the `df_numerical` dataset and stores each in new DataFrame objects named `A` and `B`, respectively.

[17]:
```
A = df_numerical.sample(replace=False)
B = df_numerical.sample(replace=False)
```

[18]:
```
A
```

[18]:
```
       MonthlyRevenue  MonthlyMinutes  TotalRecurringCharge  \
31056        0.450391        0.304505              0.552229
```

```
       DirectorAssistedCalls  OverageMinutes  RoamingCalls  PercChangeMinutes  \
31056               0.154708        0.299959     -0.125914          -1.135682

       PercChangeRevenues  DroppedCalls  BlockedCalls  ...  HandsetModels  \
31056           -0.591492        2.0222      0.019579  ...       0.487071

       CurrentEquipmentDays    AgeHH1     AgeHH2  RetentionCalls  \
31056             -0.735013  0.48256  0.787922       -0.180167

       RetentionOffersAccepted  ReferralsMadeBySubscriber  IncomeGroup  \
31056                  -0.1283                  -0.169283     0.852549

       AdjustmentsToCreditRating  HandsetPrice
31056                  -0.140707     -0.864858

[1 rows x 35 columns]
```

[19]:
```
B
```

[19]:
```
       MonthlyRevenue  MonthlyMinutes  TotalRecurringCharge  \
42903       -0.569226       -0.491923             -0.286393

       DirectorAssistedCalls  OverageMinutes  RoamingCalls  PercChangeMinutes  \
42903              -0.401714       -0.414422      0.200012          -0.626969

       PercChangeRevenues  DroppedCalls  BlockedCalls  ...  HandsetModels  \
42903            0.123614      0.330446     -0.217933  ...      -0.616775

       CurrentEquipmentDays     AgeHH1     AgeHH2  RetentionCalls  \
42903             -0.794114  0.754122  1.122214       -0.180167

       RetentionOffersAccepted  ReferralsMadeBySubscriber  IncomeGroup  \
42903                  -0.1283                  -0.169283     1.489856

       AdjustmentsToCreditRating  HandsetPrice
42903                  -0.140707      0.790755

[1 rows x 35 columns]
```

1.3 Step 2: Compute the Euclidean Distance Between Two Vectors Using Python

We will first implement a function that finds the Euclidean distance in Python. Since we will be working with Python, let us convert DataFrames `A` and `B` into Python lists.

[20]:
```
list_A = A.values.flatten().tolist()
list_B = B.values.flatten().tolist()
```

```
list_A
```

[20]:
```
[0.4503910267294967,
 0.3045045869325048,
 0.5522291485247021,
 0.15470821146877115,
 0.2999593842618582,
 -0.1259135548660268,
 -1.1356823309402944,
 -0.5914920040818883,
 2.022200331268707,
 0.019579057214878026,
 0.6099169936193746,
 0.4770327624039685,
 -0.2557966673525969,
 1.0943969852329347,
 -0.4850410632895274,
 -0.4307108741459736,
 0.3377959655294201,
 -0.06847081268218727,
 1.2113042357031023,
 -0.020662564533263195,
 -0.3295397355456733,
 -0.6894117553078604,
 0.38242109864799056,
 -0.5245829903418487,
 0.14600359624956538,
 0.4870710798513511,
 -0.7350126526839595,
 0.482559657577787,
 0.7879217820917835,
 -0.18016687925557376,
 -0.12830030819753901,
 -0.16928338528385997,
 0.8525494747501462,
 -0.14070742066769315,
 -0.8648576341987015]
```

Using the definition above, complete the function below that returns the Euclidean distance between its two list inputs.

You will use a traditional `for` loop to handle the computation for each pair of i-th coordinates of the two input lists (You can think of each pair as a 'column' in a DataFrame with just two rows -- *A* and *B*.).

Tip: to compute the square root, use the Python `math.sqrt()` function.

1.3.1 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[21]:
```python
def euclidean_distance(vector1 , vector2):
    ## the sum_squares variable will contain the current value of the sum of␣
↪squares of each i-th coordinate pair
    sum_squares = 0

    numberOfIterations = len(vector1)

    ## TODO: Complete loop below ##

    # The number of times the loop will be executed is the length of the␣
↪vectors.
    #
    # At each loop iteration, you will:
    #  Step 1. index into each vector and find the difference between the ith␣
↪element in vector2 and vector1
    #  Step 2. square the difference
    #  Step 3. update the value of the 'sum_squares' variable by adding the␣
↪result in Step 2 to
    #          the existing value of sum_squares


    for i in range(numberOfIterations):

        # Inside this loop follow steps 1-3 to update the value of the␣
↪'sum_squares' variable by
        # adding the squared difference of the i'th coordinate pair to the sum.


        # YOUR CODE HERE
        difference=vector2[i]-vector1[i]
        square=difference**2
        sum_squares+=square


    ### TODO: Compute the Distance ###

    # Compute the square root of the variable 'sum_squares' and assign
    # that result to a new variable named 'distance'

    # YOUR CODE HERE
    distance=math.sqrt(sum_squares)

    # return the Euclidean distance
    return distance
```

1.3.2 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell.

[22]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testFunction

try:
    p, err = testFunction(euclidean_distance)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

The code cell below tests your function. Run the cell to view the results.

[23]:
```
euclidean_distance(list_A, list_B)
```

[23]: 4.345778982532897

1.4 Step 3: Compute the Euclidean Distance Between Two Vectors Using NumPy

The NumPy package provides an easy way to compute the Euclidean distance between two vectors. NumPy has a `norm()` function, which is part of a linear algebra module called `linalg`. You can call the function using this syntax: `np.linalg.norm()`. The `norm([vector_name])` finds a vector norm. A vector has both magnitude and direction, and calculating the vector norm finds the magnitude.

By default, the `norm()` function calculates the L2 norm, also known as the Euclidean norm since it calculates the Euclidean distance. We can therefore use the `norm()` function to calculate the distance between two vectors.

The `norm()` function requires that its input vectors be of type NumPy array. The code cell below converts DataFrame `A` and `B` to NumPy arrays and uses the `norm()` function to find the Euclidean distance.

Run the cell below and compare the results. Is the Euclidean distance the same value as what your function `euclidean_distance` produces? Try using the `norm()` function to find the Euclidean distance between the vectors `vector_A` and `vector_B` as well.

[24]:
```
array1 = np.array(A)
array2 = np.array(B)
np.linalg.norm(array2-array1)
```

[24]: 4.345778982532897

You can see how easy it is to find the Euclidean distance between two vectors, or examples using NumPy! For more information about the `norm()` function, consult the online documentation.