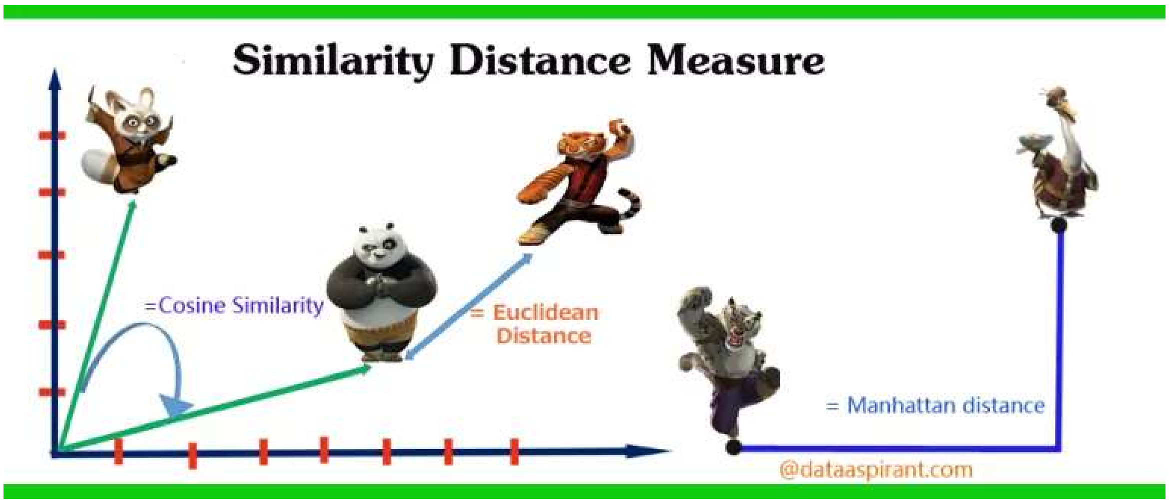
FIVE MOST POPULAR SIMILARITY MEASURES IMPLEMENTATION IN PYTHON

April 11, 2015 Saimadhu Polamuri 29 Comments DATAMINING, python, Recommendation Engine



The buzz term similarity distance measure has got a wide variety of definitions among the math and data mining practitioners. As a result, those terms, concepts and their usage went way beyond the head for the beginner, Who started to understand them for the very first time. So today I write this post to give more clear and very intuitive definitions for similarity, and I will drive to Five most popular similarity measures and implementation of them.

Before going to explain different similarity distance measures let me explain the effective key term similarity in datamining. This similarity is the very basic building block for activities such as Recommendation engines, clustering, classification and anomaly detection.

Similarity:

The similarity measure is the measure of how much alike two data objects are. Similarity measure in a data mining context is a distance with dimensions representing features of the objects. If this distance is small, it will be the high degree of similarity where large distance will be the low degree of similarity.

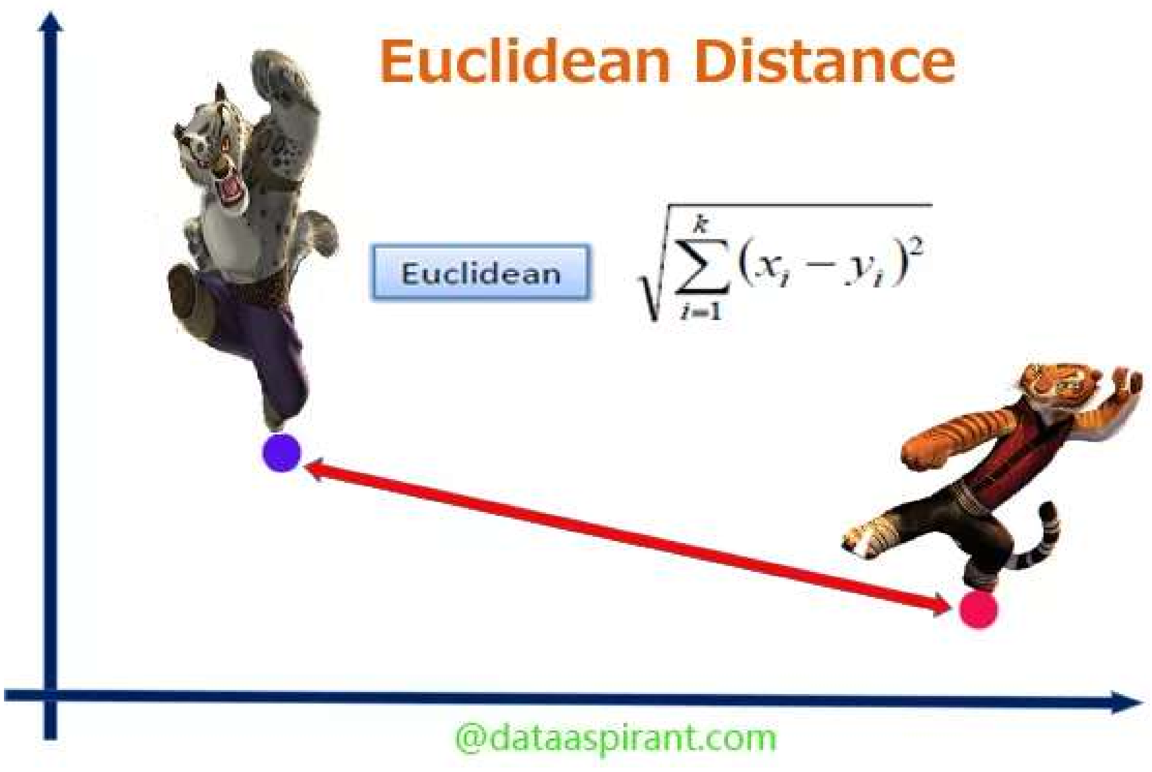
The similarity is subjective and is highly dependent on the domain and application. For example, two fruits are similar because of color or size or taste. Care should be taken when calculating distance across dimensions/features that are unrelated. The relative values of each element must be normalized, or one feature could end up dominating the distance calculation. **Similarity are measured in the range 0 to 1 [0,1].**

Two main consideration about similarity:

- Similarity = 1 if X = Y (Where X, Y are two objects)
- Similarity = 0 if X ≠ Y

That's all about similarity let's drive to five most popular similarity distance measures.

Euclidean distance:



Euclidean distance is the most common use of distance. In most cases when people said about distance, they will refer to Euclidean distance. Euclidean distance is also known as simply distance. When data is dense or continuous, this is the best proximity measure.

The Euclidean distance between two points is the length of the path connecting them.The Pythagorean theorem gives this distance between two points.

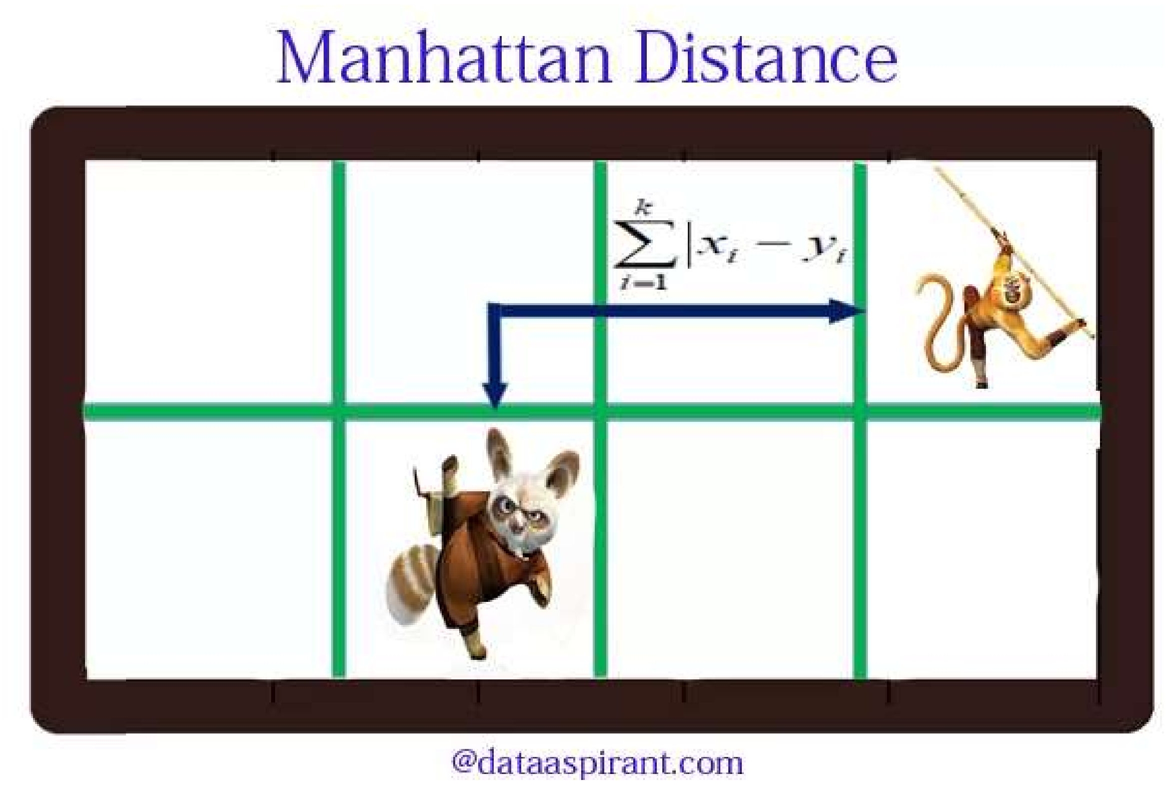
Euclidean distance implementation in python:

```
#!/usr/bin/env python


from math import*

def euclidean_distance(x,y):

    return sqrt(sum(pow(a-b,2) for a, b in zip(x, y)))

print euclidean_distance([0,3,4,5],[7,6,3,-1])
```

Script Output:

```
9.74679434481
[Finished in 0.0s]
```

Manhattan distance:



Manhattan distance is a metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. In a simple way of saying it is the total suzm of the difference between the x-coordinates and y-coordinates.

Suppose we have two points A and B if we want to find the Manhattan distance between them, just we have, to sum up, the absolute x-axis and y – axis variation means we have to find how these two points A and B are varying in X-axis and Y- axis. In a more mathematical way of saying Manhattan distance between two points measured along axes at right angles.

In a plane with p1 at (x1, y1) and p2 at (x2, y2).

Manhattan distance = |x1 – x2| + |y1 – y2|

This Manhattan distance metric is also known as Manhattan length, rectilinear distance, L1 distance or L1 norm, city block distance, Minkowski's L1 distance, taxi-cab metric, or city block distance.

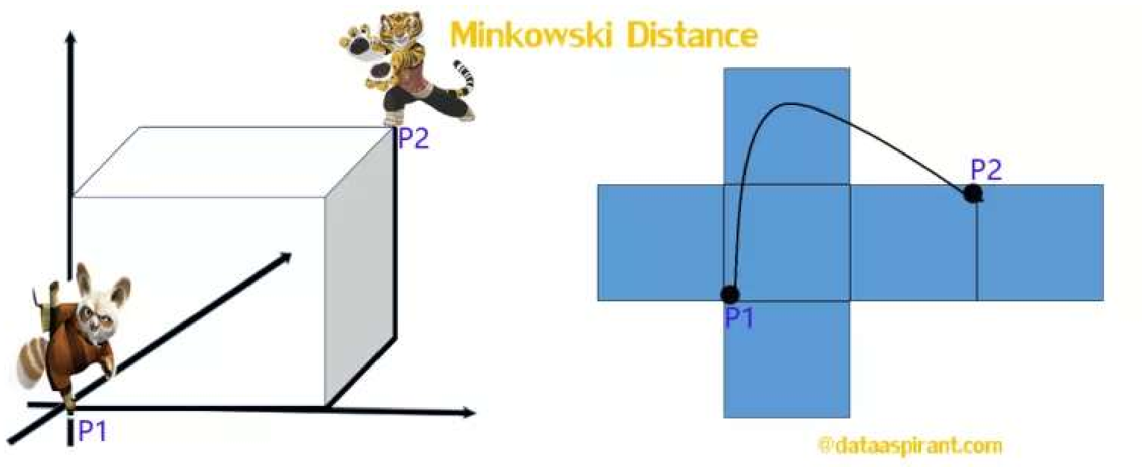
Manhattan distance implementation in python:

```
#!/usr/bin/env python

from math import*

def manhattan_distance(x,y):

    return sum(abs(a-b) for a,b in zip(x,y))

print manhattan_distance([10,20,10],[10,20,20])
```

Script Output:

```
10
[Finished in 0.0s]
```

Minkowski distance:



The Minkowski distance is a generalized metric form of Euclidean distance and Manhattan distance.

$$d^{MKD}(i,j) = \sqrt[\lambda]{\sum_{k=0}^{n-1} \left|y_{i,k} - y_{j,k}\right|^{\lambda}}$$

In the equation, d^MKD is the Minkowski distance between the data record i and j, k the index of a variable, n the total number of variables y and λ the order of the Minkowski metric. Although it is defined for any λ > 0, it is rarely used for values other than 1, 2 and ∞.

The way distances are measured by the Minkowski metric of different orders between two objects with three variables (In the image it displayed in a coordinate system with x, y ,z-axes).

Synonyms of Minkowski:
Different names for the Minkowski distance or Minkowski metric arise from the order:

- λ = 1 is the Manhattan distance. Synonyms are L1-Norm, Taxicab or City-Block distance. For two vectors of ranked ordinal variables, the Manhattan distance is sometimes called Foot-ruler distance.
- λ = 2 is the Euclidean distance. Synonyms are L2-Norm or Ruler distance. For two vectors of ranked ordinal variables, the Euclidean distance is sometimes called Spear-man distance.
- λ = ∞ is the Chebyshev distance. Synonyms are Lmax-Norm or Chessboard distance.
  reference.

Minkowski distance implementation in python:

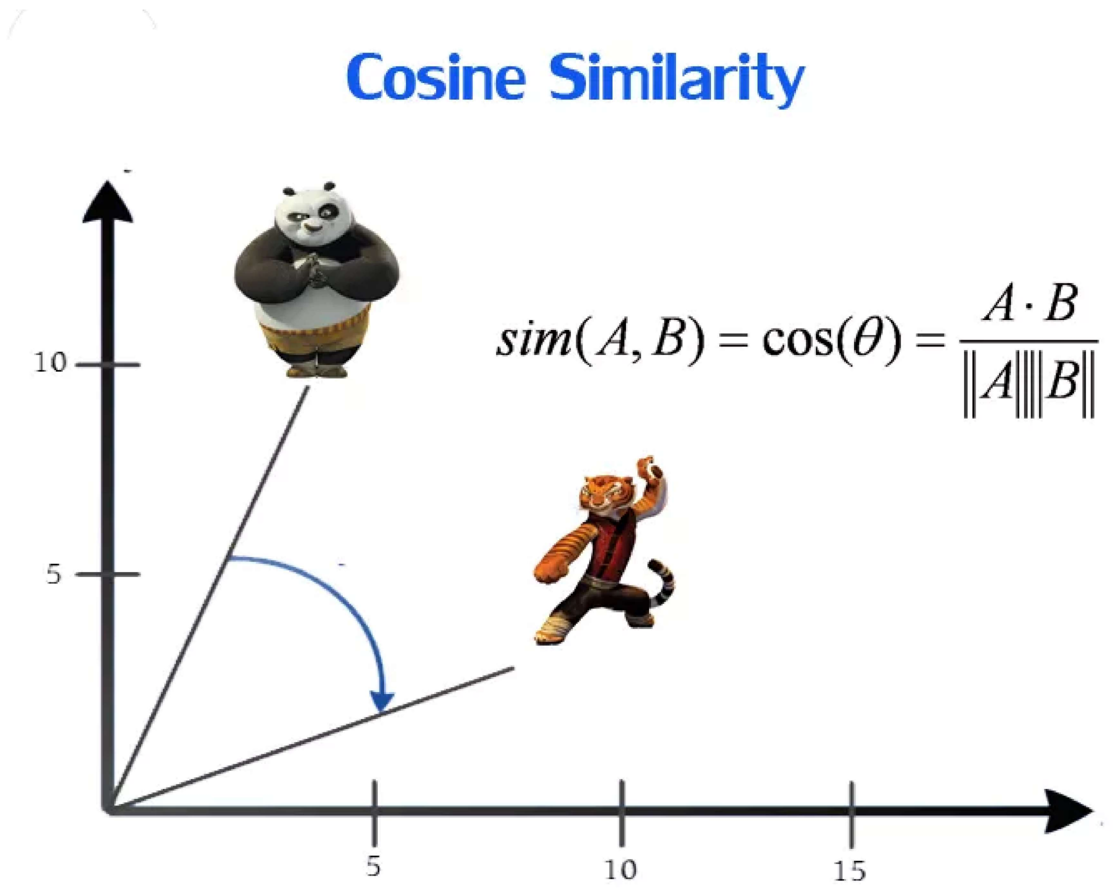
```
#!/usr/bin/env python


from math import*
from decimal import Decimal

def nth_root(value, n_root):

    root_value = 1/float(n_root)
    return round (Decimal(value) ** Decimal(root_value),3)

def minkowski_distance(x,y,p_value):

    return nth_root(sum(pow(abs(a-b),p_value) for a,b in zip(x, y)),p_value)

print minkowski_distance([0,3,4,5],[7,6,3,-1],3)
```

Script Output:

```
8.373
```

```
[Finished in 0.0s]
```

Cosine similarity:



Cosine similarity metric finds the normalized dot product of the two attributes. By determining the cosine similarity, we would effectively try to find the cosine of the angle between the two objects. The cosine of 0° is 1, and it is less than 1 for any other angle.

It is thus a judgement of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.

Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in [0,1]. One of the reasons for the popularity of cosine similarity is that it is very efficient to evaluate, especially for sparse vectors.

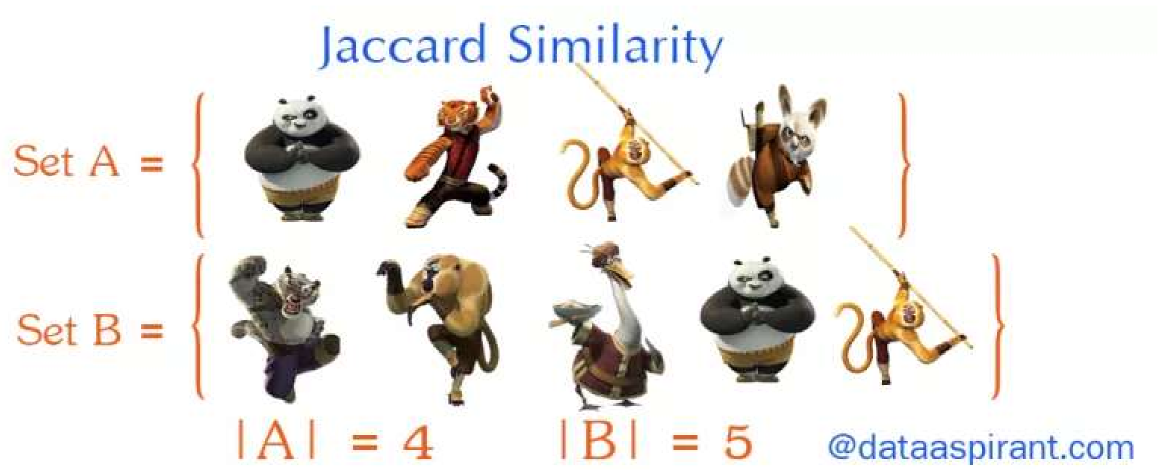
Cosine similarity implementation in python:

```
#!/usr/bin/env python


from math import*

def square_rooted(x):

     return round(sqrt(sum([a*a for a in x])),3)

def cosine_similarity(x,y):

     numerator = sum(a*b for a,b in zip(x,y))
    denominator = square_rooted(x)*square_rooted(y)
    return round(numerator/float(denominator),3)

print cosine_similarity([3, 45, 7, 2], [2, 54, 13, 15])
```

Script Output:

```
0.972
[Finished in 0.1s]
```

Jaccard similarity:



We so far discussed some metrics to find the similarity between objects. where the objects are points or vectors .When we consider about Jaccard similarity this objects will be sets. So first let's learn some very basic about sets.

Sets:

A set is (unordered) collection of objects {a,b,c}. we use the notation as elements separated by commas inside curly brackets { }. They are unordered so {a,b} = { b,a }.

Cardinality:

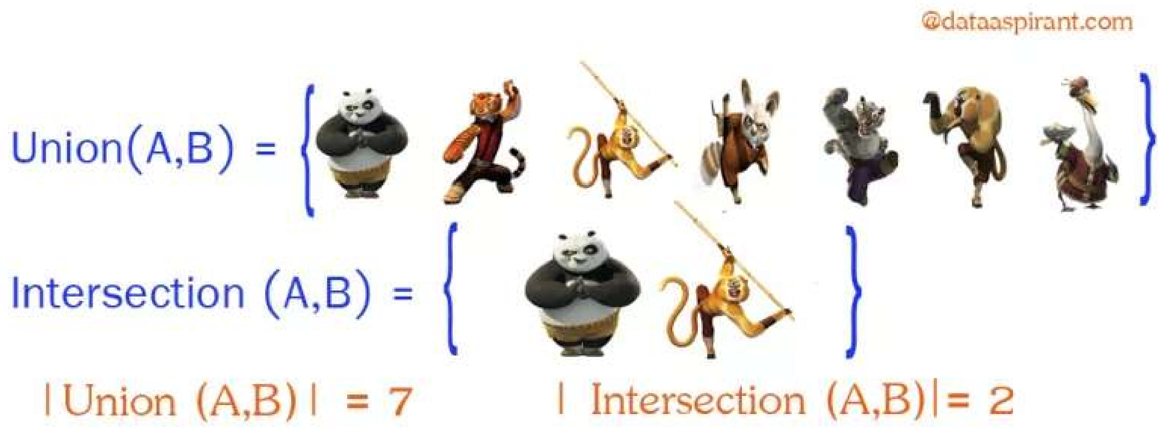
The cardinality of A denoted by **|A|** which counts how many elements are in A.

Intersection:

The intersection between two sets A and B is denoted **A ∩ B** and reveals all items which are in both sets A,B.

Union:

Union between two sets A and B is denoted **A ∪ B** and reveals all items which are in either set.



Now going back to Jaccard similarity. The Jaccard similarity measures the similarity between finite sample sets and is defined as the cardinality of the intersection of sets divided by the cardinality of the union of the sample sets. Suppose you want to find Jaccard similarity between two sets A and B it is the ration of cardinality of A ∩ B and A ∪ B

$$\text{Jaccard Similarity } J\ (A,B) = |\text{ Intersection } (A,B)\ | \ / \ |\text{ Union } (A,B)\ |$$

$$= 2\ /\ 7$$

$$= 0.286$$

Jaccard similarity implementation:

```
#!/usr/bin/env python


from math import*

def jaccard_similarity(x,y):

    intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
    union_cardinality = len(set.union(*[set(x), set(y)]))
    return intersection_cardinality/float(union_cardinality)

print jaccard_similarity([0,1,2,5,6],[0,2,3,5,7,9])
```

Script Output:

```
0.375
[Finished in 0.0s]
```

Implementaion of all 5 similarity measure into one Similarity class:

file_name : **similaritymeasures.py**

```
#!/usr/bin/env python


from math import*
from decimal import Decimal

class Similarity():

    """ Five similarity measures function """

    def euclidean_distance(self,x,y):

        """ return euclidean distance between two lists """

        return sqrt(sum(pow(a-b,2) for a, b in zip(x, y)))

    def manhattan_distance(self,x,y):

        """ return manhattan distance between two lists """

        return sum(abs(a-b) for a,b in zip(x,y))

    def minkowski_distance(self,x,y,p_value):

        """ return minkowski distance between two lists """

        return self.nth_root(sum(pow(abs(a-b),p_value) for a,b in zip(x, y))
           p_value)

    def nth_root(self,value, n_root):

        """ returns the n_root of an value """

        root_value = 1/float(n_root)
        return round (Decimal(value) ** Decimal(root_value),3)

    def cosine_similarity(self,x,y):

        """ return cosine similarity between two lists """

        numerator = sum(a*b for a,b in zip(x,y))
        denominator = self.square_rooted(x)*self.square_rooted(y)
        return round(numerator/float(denominator),3)

    def square_rooted(self,x):

        """ return 3 rounded square rooted value """

        return round(sqrt(sum([a*a for a in x])),3)

    def jaccard_similarity(self,x,y):

    """ returns the jaccard similarity between two lists """

        intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
        union_cardinality = len(set.union(*[set(x), set(y)]))
        return intersection_cardinality/float(union_cardinality)
```

Using Similarity class:

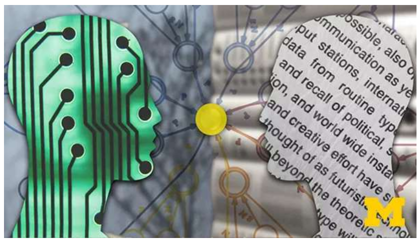
Using similarity measure class Python

```


from similaritymeasures import Similarity

def main():

    """ the main function to create Similarity class instance and get used t

    measures = Similarity()

    print measures.euclidean_distance([0,3,4,5],[7,6,3,-1])
    print measures.jaccard_similarity([0,1,2,5,6],[0,2,3,5,7,9])

if __name__ == "__main__":
    main()
```

You can get all the complete codes of dataaspirant at dataaspirant data science codes

Related Courses:

Do check out unlimited data science courses

Course	Course Link	What You will Lea
Introduction to natural language processing 	Introduction to Natural Language Processing	• Introduction t Natural Langu • Relevant back in Linguistics, Probabilities, Science. • Text Similarity Tagging, Parsi Question Ans Analysis, and Summarizatic

Applied Text mining in python	Applied Text Mining in Python	• Will learn the mining and te • Handling text concepts of N framework ar text with it. • working on cc manipulation regular expre for text), clear preparing tex learning proce • Implement yo classifier in py
Easy Natural Language Processing (NLP) in Python	Easy Natural Language Processing (NLP) in Python	• Will learn abo detection and implement sp application in • Learn about l sentiment ana going to imple sentiment ana python. • Learn the con semantic anal • Finally, you wi spinner in pyt

Follow us:

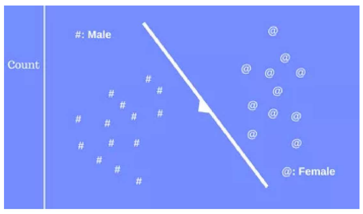
FACEBOOK| **QUORA** |**TWITTER**| **GOOGLE+** | **LINKEDIN**| **REDDIT** | **FLIPBOARD** | **MEDIUM** | **GITHUB**

I hope you like this post. If you have any questions then feel free to comment below. If you want me to write on one specific topic then do tell it to me in the comments below.

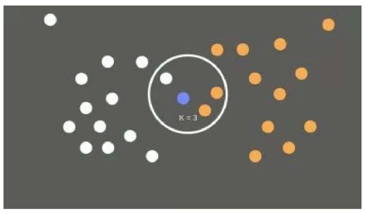
Share this:

Related

collaborative filtering recommendation engine implementation in python
May 25, 2015
In "Data Science"



classification and clustering algorithms
September 24, 2016
In "Data Science"



Knn Classifier, Introduction to K-Nearest Neighbor Algorithm
December 23, 2016
In "Data Science"

datamining python Recommendation_engine similarity_distance