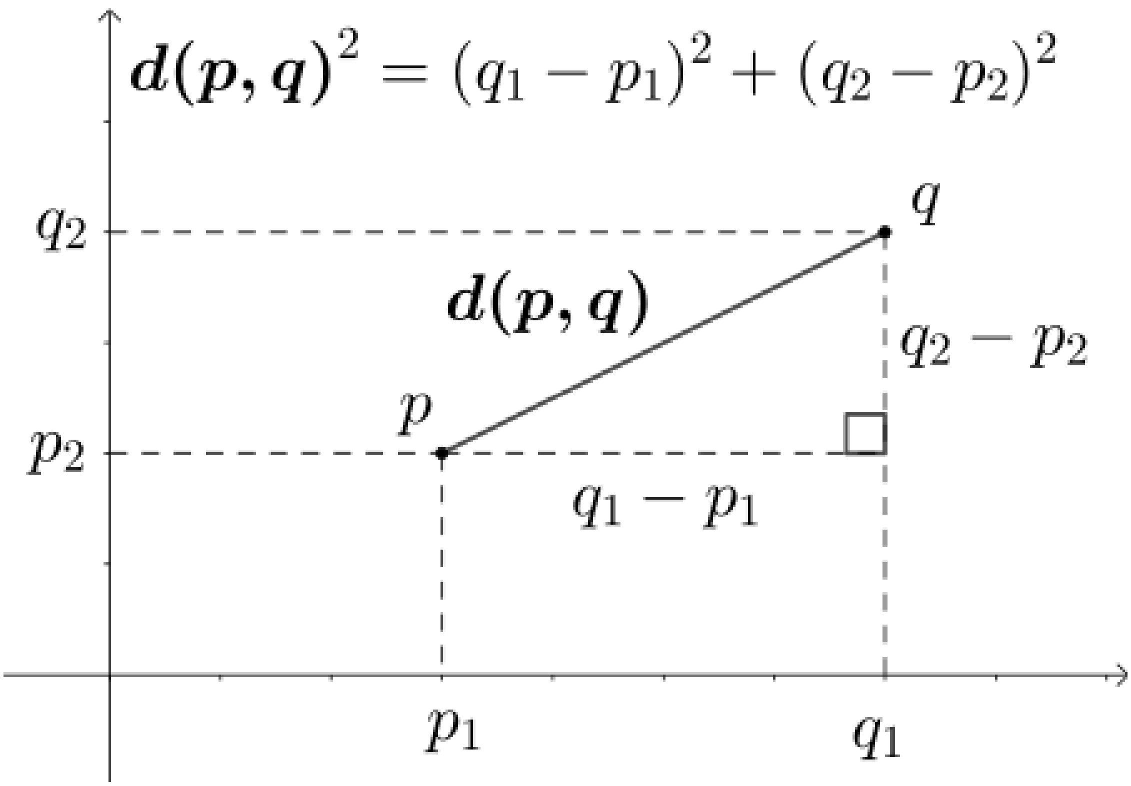
# Untitled

September 1, 2021

## 1 Euclidean distance

the Euclidean distance between two points in Euclidean space is the length of a line segment between the two points.




[16]:
```
import math

# higher dimensional formula
# d = sqrt( (y1 - x1)^2  + (y2 - x1)^2 + (y3 - x3)^2 +       + (yn - xn)^2 )

def euclideanDistance(x , y):

    distance = 0

    for i,j in zip( range(len(x)) , range(len(y)) ):
```

```
        distance = distance + ((x[i] - y[j]) ** 2)

    distance = math.sqrt(distance)

    return distance


euclideanDistance([1,5,3,4,1] , [3,4,2,1,1])
```

[16]: 3.872983346207417

## 2 hamming distance

the Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different.

[8]:
```
def hammingDistance(x , y):

    distance = 0

    for i,j in zip(x,y):
        if(i != j):
            distance = distance + 1

    return distance


hammingDistance("hello world" , "helle wordl")
```

[8]: 3

## 3 cosine similarity

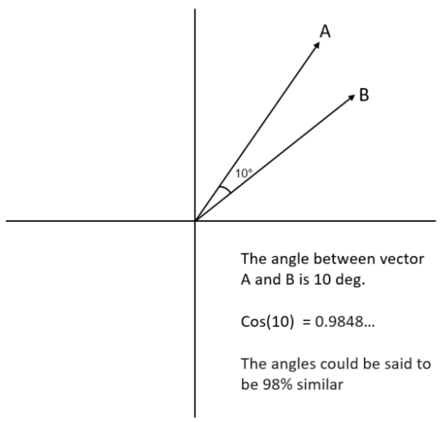
Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors I am talking about are arrays containing the word counts of two documents.

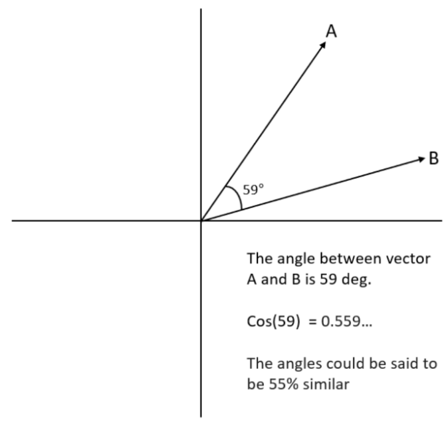
When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude. If you want the magnitude, compute the Euclidean distance instead.

The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size (like, the word 'cricket' appeared 50 times in one document and 10 times in another) they could still have a smaller angle between them. Smaller the angle, higher the similarity.

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \, \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ is the dot product of the two vectors.




It is widely used in document matching

example -

Document 1: Deep Learning can be hard

Document 2: Deep Learning can be simple

### Vectorised Representation

| Aa Word | Document 1 | Document 2 |
|---|---|---|
| Deep | 1 | 1 |
| Learning | 1 | 1 |
| Can | 1 | 1 |
| Be | 1 | 1 |
| Hard | 1 | 0 |
| Simple | 0 | 1 |

*Document 1: [1, 1, 1, 1, 1, 0] let's refer to this as A*

*Document 2: [1, 1, 1, 1, 0, 1] let's refer to this as B*

[11]:
```python
# formula for cosine similarity

# a.b / mag(a) * mag(b)

def cosine_similarity(x , y):

    aDotb = 0
    a = 0
    b = 0

    for i in range(len(x)):
        aDotb = aDotb + (x[i] * y[i])
        a = a + (x[i] * x[i])
        b = b + (y[i] * y[i])

    cosTheta = aDotb / (math.sqrt(a) * math.sqrt(b))

    return cosTheta

cosine_similarity([1,3,4,1] , [3,2,1,1])
```

[11]: 0.6956655929999346

# 4 jaccard similarity

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

[23]:
```
# formula = len(intersection) / len(union)

def jacardSimilarity(x , y):

    setX = set(x)
    setY = set(y)

    print(setX.intersection(setY))
    print(setX.union(setY))

    jaccardSim = len(setX.intersection(setY)) / len(setX.union(setY))

    return jaccardSim

jacardSimilarity([0,1,2,4,5,6],[0,2,3,4,5,1,7,9])
```

```
{0, 1, 2, 4, 5}
{0, 1, 2, 3, 4, 5, 6, 7, 9}
```

[23]: 0.5555555555555556

[29]:
```
# without using sets

# x intersection y
def listIntersection(x , y):

    result = []

    for i in x:
        if(i in y):
            result.append(i)

    return result
```

```
def listUnion(x , y):
    result = x

    for i in y:
        if(i not in result):
            result.append(i)

    return result



def jacardSimilarity(x , y):

    listX = []
    listY = []

    for i in x:
        if(i not in listX):
            listX.append(i)

    for j in y:
        if(j not in listY):
            listY.append(j)


    intersection = listIntersection(listX , listY)
    union = listUnion(listX , listY)

    jaccardSim = len(intersection) / len(union)

    return jaccardSim



jacardSimilarity([0,1,2,4,5,6],[0,2,3,4,5,1,7,9])
```

[29]: 0.5555555555555556

[ ]: