TF IDF Cosine Similarity Formula Examples In

Table of Contents

What is Cosine similarity?

Cosine similarity is a measure to find the similarity between two files/documents.

Example of cosine similarity

What is the similarity between two files, file 1 and file 2?

Cosine similarity Formula

cos(file *1, file 2*) = (file *1* · file *2*) / ||file *1*|| ||file *2*|| ,

file 1 **=** (0, 3, 0, 0, 2, 0, 0, 2, 0, 5) *file 2* **=** (1, 2, 0, 0, 1, 1, 0, 1, 0, 3)

file 1 · file *2* = 0*1 + 3*2 + 0*0 + 0*0 + 2*1 + 0*1 + 0*0 + 2*1 + 0*0 + 5*3

= 25

||*d1*||= (0*0 + 3*3 + 0*0 + 0*0 + 2*2 + 0*0 + 0*0 + 2*2 + 0*0 + 5*5)**0.5**

=(42)**0.5** = 6.481

=(17)**0.5** = 4.12

cos(*d1* , *d2*) = 0.94

What is a good cosine similarity 0 or 1?

- Similarity 0 means no similarity
- Similarity 0 means identical
- A similarity above 0.5 might be a good starting point.

Is cosine similarity a metric?

Yes, Cosine similarity is a metric. This metric can be used to measure the similarity between two objects.

```
df = pd.DataFrame(doc_term_matrix,
                  columns=count_vectorizer.get_feature_names(),
                  index=['doc_Imran Khan', 'doc_election', 'doc_Nawaz Sharif'])
df
```

Doc-Term Matrix

It's more better to use the TfidfVectorizer() function instead of CountVectorizer() function, because it would have downweighted words. Here, we can see that it occurs frequently across the each document.

Finaly, we can write a cosine_similarity() function to get the final output. It can take the document term matri as a pandas dataframe as well as a sparse

```
# Compute Cosine Similarity
from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(df, df))
#> [[ 1.          0.48927489  0.37139068]
#>  [ 0.48927489  1.          0.38829014]
#>  [ 0.37139068  0.38829014  1.        ]]
```

Examples of TF IDF Cosine Similarity

Document 1: T4Tutorials website is a website and it is for professionals.

Document 2: T4Tutorials website is also for good students.

Document 3: i love T4Tutorials

Step 1:

Term Frequency (TF)

Term Frequency commonly known as TF measures the total number of times word appears in a selected document.

Term Frequency Matrix / Document-term matrix

Let's see some terms and their frequency on each of the document. In this example, there are three document.

TF for Document 1

Document1	**T4Tutorials**	**website**	**is**	**fantastic**	**and**	**It's**	**for**	**professio s**

When to use cosine similarity over Euclidean similarity?

In Cosine similarity our focus is at the angle between two vectors and in case of euclidian similarity our focus is at the distance between two points.

For example we want to analyse the data of a shop and the data is;

- User 1 bought 1x copy, 1x pencil and 1x rubber from the shop.
- User 2 bought 100x copy, 100x pencil and 100x rubber from the shop.
- User 3 bought 1x copy, 1x PEPSI and 1x Shoes Polish from the shop.

According to cosine similarity, user 1 and user 2 are more similar and in case of euclidean similarity, user 3 is more similar to user 1.

Cosine similarity python

Suppose we have text in the three documents;

Doc Imran Khan (A) : Mr. Imran Khan win the president seat after winning the National election 2020-2021. Though he lost the support of some republican friends, Imran Khan is friends with President Nawaz Sharif.

Doc Imran Khan Election (B) : President Imran Khan says Nawaz Sharif ha no political interference is the election outcome. He claimed President Nawaz

Term Frequency	1	2	2	1	1	1	1	1

TF for Document 2

Document2	T4Tutorials	website	is	also	for	Good	Students
Term Frequency	1	1	1	1	1	1	1

TF for Document 3

Term Frequency	1	1	1

For big data and for big values it's difficult to understand the data. So, its better to normalize the document based on its size. We can do this with different normalization techniques like min max, decimal scaling and Z-Score normalization. The simple is decimal scaling by dividing the term frequency by the total number of terms.

For example in Document 1 the term website occurs two times. The total number of terms in the document1 is 10. So, let;s normalized the term frequency by 2 / 10 = 0.2.

Now, let's see the normalized term frequency for all the document1, document2 and document3.

Normalized TF for Document 1

Document1	T4Tutorials	website	is	fantastic	and	It's	for	professionals
Normalized TF	0.1	0.2	0.2	0.1	0.1	0.1	0.1	0.1

Normalized TF for Document 2

Document2	T4Tutorials	website	is	also	for	Good	stu
Normalized TF	0.142857	0.142857	0.142857	0.142857	0.142857	0.142	0.

Sharif is a friend who had nothing to do with the election.

Doc Nawaz Sharif (C) : Post elections, Vladimir Nawaz Sharif win the president seat of Russia. President Nawaz Sharif had served as the Prime Minister earlier in his political career.

Here, we can see that, Doc B has more in common with Doc A than with Doc C, so here we can expect that the Cosine between the document A and B is larger than document B and document C.

```
# Define the documents
doc_Imran Khan = "Mr. Imran Khan win the president seat after winning the National elec

doc_election = "President Imran Khan says Nawaz Sharif had no political interference is

doc_Nawaz Sharif = "Post elections, Vladimir Nawaz Sharif win the president seat of Rus

documents = [doc_Imran Khan, doc_election, doc_Nawaz Sharif]
```

If we want to compute the cosine similarity, first of all we will count the total words in document A, B, and C. The CountVectorizer or the TfidfVectorizer from scikit learn lets us compute this. The output of this comes as a sparse_matrix.

Here, its not compulsory but let's convert it to a pandas dataframe to see the word frequencies in a tabular format.

```
# Let's begin with Scikit Learn
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Here, we are Creating the Document Term Matrix
count_vectorizer = CountVectorizer(stop_words='english')
count_vectorizer = CountVectorizer()
sparse_matrix = count_vectorizer.fit_transform(documents)

# Not compulsory and OPTIONAL: Converting the  Sparse Matrix to a Pandas Dataframe,
doc_term_matrix = sparse_matrix.todense()
```

Normalized TF for Document 3

Document3	i	love	T4Tutorials
Normalized TF	0.333333	0.333333	0.333333

Given below is the code in python which will do the normalized TF calculation.

```
def termFrequency(term, document):
    normalizeDocument = document.lower().split()
    return normalizeDocument.count(term.lower()) / float(len(normalizeDocument))
```

Step 2

Let us compute Inverse Document Frequency for the term **website**

```
IDF(website) = 1 + loge(Total Number Of Documents / Number Of Documents with term websi

Total documents are 3 = Document1, Document2, Document3
The term website appears in Document1

IDF(website) = 1 + loge(3 / 1)
             = 1 + 1.098726209
             = 2.098726209
```

Inverse Document Frequency (IDF) in Python

Here, I am sharing the python code to calculate the IDF.

```
def inverseDocumentFrequency(term, allDocuments):
    numDocumentsWithThisTerm = 0
    for doc in allDocuments:
        if term.lower() in allDocuments[doc].lower().split():
            numDocumentsWithThisTerm = numDocumentsWithThisTerm + 1

    if numDocumentsWithThisTerm > 0:
        return 1.0 + log(float(len(allDocuments)) / numDocumentsWithThisTerm)
    else:
        return 1.0
```

Step 3:

We need to multiply the Term Frequency with Document Frequency just like TF * IDF .

Step 4:

Cosine Similarity

```
Cosine Similarity (d1, d2) =  Dot product(d1, d2) / ||d1|| * ||d2||

Dot product (d1,d2) = d1[0] * d2[0] + d1[1] * d2[1] * … * d1[n] * d2[n]
||d1|| = square root(d1[0]2 + d1[1]2 + ... + d1[n]2)
||d2|| = square root(d2[0]2 + d2[1]2 + ... + d2[n]2)
```

26. Distance measure for asymmetric binary
27. Distance measure for symmetric binary
28. Euclidean distance
29. Classification
30. C4.5
31. Clustering
32. Association rule mining
33. Regression
34. MCQs
35. attribute selection measure

38. Major tasks of data pre-processing
39. Data Mining Primitives
40. Analytical Characterization in Data Mining
41. Data Generalization In Data Mining – Summarization Based Characterization
42. Binning Methods for Data Smoothing

Share to support

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment