Search

Learn Free

Cosine Similarity FastText Gensim NLP Soft Cosine Similarity

# Cosine Similarity – Understanding the math and how it works (with python codes)

October 22, 2018 by Selva Prabhakaran

*Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.*

By the end of this tutorial you will know:

1. What is cosine similarity is and how it works?
2. How to compute cosine similarity of documents in python?
3. What is soft cosine similarity and how its different from cosine similarity?
4. When to use soft cosine similarity and how to compute it in python?

Cosine Similarity – Understanding the math and how it works. Photo by Matt Lamers

## 1. Introduction

A commonly used approach to match similar documents is based on counting the maximum number of common words between the documents.

But this approach has an inherent flaw. That is, as the size of the document increases, the number of common words tend to increase even if the documents talk about different topics.

The cosine similarity helps overcome this fundamental flaw in the 'count-the-common-words' or Euclidean distance approach.

## 2. What is Cosine Similarity and why is it advantageous?

Cosine similarity is a metric used to determine how similar the documents are irrespective of their size.

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.

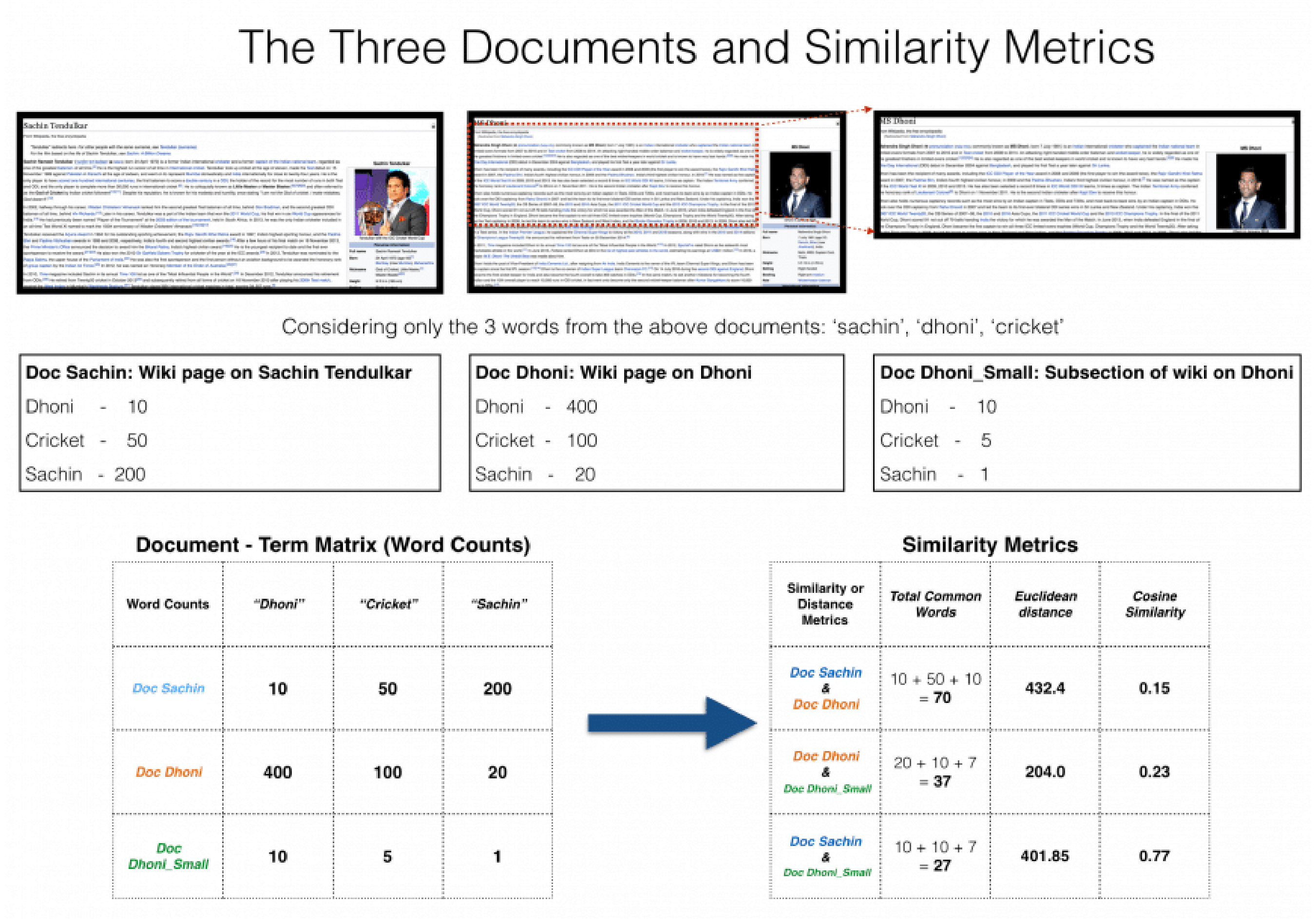
As a similarity metric, how does cosine similarity differ from the number of common words?

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.

## 3. Cosine Similarity Example

Let's suppose you have 3 documents based on a couple of star cricket players – Sachin Tendulkar and Dhoni. Two of the documents (A) and (B) are from the wikipedia pages on the respective players and the third document (C) is a smaller snippet from Dhoni's wikipedia page.



The Three Documents and Similarity Metrics

Considering only the 3 words from the above documents: 'sachin', 'dhoni', 'cricket'

**Doc Sachin: Wiki page on Sachin Tendulkar**

| | | |
|---|---|---|
| Dhoni | - | 10 |
| Cricket | - | 50 |
| Sachin | - | 200 |

**Doc Dhoni: Wiki page on Dhoni**

| | | |
|---|---|---|
| Dhoni | - | 400 |
| Cricket | - | 100 |
| Sachin | - | 20 |

**Doc Dhoni_Small: Subsection of wiki on Dhoni**

| | | |
|---|---|---|
| Dhoni | - | 10 |
| Cricket | - | 5 |
| Sachin | - | 1 |

**Document - Term Matrix (Word Counts)**

| Word Counts | *"Dhoni"* | *"Cricket"* | *"Sachin"* |
|---|---|---|---|
| *Doc Sachin* | **10** | **50** | **200** |
| *Doc Dhoni* | **400** | **100** | **20** |
| *Doc Dhoni_Small* | **10** | **5** | **1** |

**Similarity Metrics**

| Similarity or Distance Metrics | *Total Common Words* | *Euclidean distance* | *Cosine Similarity* |
|---|---|---|---|
| *Doc Sachin & Doc Dhoni* | 10 + 50 + 10 = **70** | **432.4** | **0.15** |
| *Doc Dhoni & Doc Dhoni_Small* | 20 + 10 + 7 = **37** | **204.0** | **0.23** |
| *Doc Sachin & Doc Dhoni_Small* | 10 + 10 + 7 = **27** | **401.85** | **0.77** |

The Three Documents

As you can see, all three documents are connected by a common theme – the game of Cricket.

Our objective is to quantitatively estimate the similarity between the documents.

For ease of understanding, let's consider only the top 3 common words between the documents: 'Dhoni', 'Sachin' and 'Cricket'.

You would expect `Doc B` and `Doc C`, that is the two documents on Dhoni would have a higher similarity over `Doc A` and `Doc B`, because, `Doc C` is essentially a snippet from `Doc B` itself.

However, if we go by the number of common words, the two larger documents will have the most common words and therefore will be judged as most similar, which is exactly what we want to avoid.

The results would be more congruent when we use the cosine similarity score to assess the similarity.

Let me explain.

Let’s project the documents in a 3-dimensional space, where each dimension is a frequency count of either: ‘Sachin’, ‘Dhoni’ or ‘Cricket’. When plotted on this space, the 3 documents would appear something like this.

3d Projection

As you can see, `Doc Dhoni_Small` and the main `Doc Dhoni` are oriented closer together in 3-D space, even though they are far apart by magnitiude.

It turns out, the closer the documents are by angle, the higher is the Cosine Similarity (Cos theta).

Cosine Similarity Formula

As you include more words from the document, it's harder to visualize a higher dimensional space. But you can directly compute the cosine similarity using this math formula.

Enough with the theory. Let's compute the cosine similarity with Python's scikit learn.

## 4. How to Compute Cosine Similarity in Python?

We have the following 3 texts:

Doc Trump (A) : Mr. Trump became president after winning the political election. Though he lost the support of some republican friends, Trump is friends with President Putin.

Doc Trump Election (B) : President Trump says Putin had no political interference is the election outcome. He says it was a witchhunt by political parties. He claimed President Putin is a friend who had nothing to do with the election.

Doc Putin (C) : Post elections, Vladimir Putin became President of Russia. President Putin had served as the Prime Minister earlier in his political career.

Since, Doc B has more in common with Doc A than with Doc C, I would expect the Cosine between A and B to be larger than (C and B).

```
 # Define the documents
doc_trump = "Mr. Trump became president after winning the political election. T
```

```
doc_election = "President Trump says Putin had no political interference is the

doc_putin = "Post elections, Vladimir Putin became President of Russia. Preside

documents = [doc_trump, doc_election, doc_putin]
```

To compute the cosine similarity, you need the word count of the words in each document. The `CountVectorizer` or the `TfidfVectorizer` from scikit learn lets us compute this. The output of this comes as a `sparse_matrix`.

On this, am optionally converting it to a pandas dataframe to see the word frequencies in a tabular format.

```
 # Scikit Learn
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

# Create the Document Term Matrix
count_vectorizer = CountVectorizer(stop_words='english')
count_vectorizer = CountVectorizer()
sparse_matrix = count_vectorizer.fit_transform(documents)

# OPTIONAL: Convert Sparse Matrix to Pandas Dataframe if you want to see the wo
doc_term_matrix = sparse_matrix.todense()
df = pd.DataFrame(doc_term_matrix,
                  columns=count_vectorizer.get_feature_names(),
                  index=['doc_trump', 'doc_election', 'doc_putin'])
df
```

Doc-Term Matrix

Even better, I could have used the `TfidfVectorizer()` instead of `CountVectorizer()` , because it would have downweighted words that occur frequently across docuemnts.

Then, use `cosine_similarity()` to get the final output. It can take the document term matri as a pandas dataframe as well as a sparse matrix as inputs.

```
 # 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.        ]]
```

## 5. Soft Cosine Similarity

Suppose if you have another set of documents on a completely different topic, say 'food', you want a similarity metric that gives higher scores for documents belonging to the same topic and lower scores when comparing docs from different topics.

In such case, we need to consider the semantic meaning should be considered. That is, words similar in meaning should be treated as similar. For Example, 'President' vs 'Prime minister', 'Food' vs 'Dish', 'Hi' vs 'Hello' should be considered similar. For this, converting the words into respective word vectors, and then, computing the similarities can address this problem.

Soft Cosines

Let's define 3 additional documents on food items.

```
 # Define the documents
doc_soup = "Soup is a primarily liquid food, generally served warm or hot (but

doc_noodles = "Noodles are a staple food in many cultures. They are made from u

doc_dosa = "Dosa is a type of pancake from the Indian subcontinent, made from a

documents = [doc_trump, doc_election, doc_putin, doc_soup, doc_noodles, doc_dos
```

To get the word vectors, you need a word embedding model. Let's download the `FastText` model using gensim's downloader api.

```
 import gensim
# upgrade gensim if you can't import softcossim
from gensim.matutils import softcossim
from gensim import corpora
```

```
import gensim.downloader as api
from gensim.utils import simple_preprocess
print(gensim.__version__)
#> '3.6.0'

# Download the FastText model
fasttext_model300 = api.load('fasttext-wiki-news-subwords-300')
```

To compute soft cosines, you need the dictionary (a map of word to unique id), the corpus (word counts) for each sentence and the similarity matrix.

```
 # Prepare a dictionary and a corpus.
dictionary = corpora.Dictionary([simple_preprocess(doc) for doc in documents])

# Prepare the similarity matrix
similarity_matrix = fasttext_model300.similarity_matrix(dictionary, tfidf=None,

# Convert the sentences into bag-of-words vectors.
sent_1 = dictionary.doc2bow(simple_preprocess(doc_trump))
sent_2 = dictionary.doc2bow(simple_preprocess(doc_election))
sent_3 = dictionary.doc2bow(simple_preprocess(doc_putin))
sent_4 = dictionary.doc2bow(simple_preprocess(doc_soup))
sent_5 = dictionary.doc2bow(simple_preprocess(doc_noodles))
sent_6 = dictionary.doc2bow(simple_preprocess(doc_dosa))

sentences = [sent_1, sent_2, sent_3, sent_4, sent_5, sent_6]
```

If you want the soft cosine similarity of 2 documents, you can just call the `softcossim()` function

```
 # Compute soft cosine similarity
print(softcossim(sent_1, sent_2, similarity_matrix))
#> 0.567228632589
```

But, I want to compare the soft cosines for all documents against each other. So, create the soft cosine similarity matrix.

```
 import numpy as np
import pandas as pd

def create_soft_cossim_matrix(sentences):
    len_array = np.arange(len(sentences))
    xx, yy = np.meshgrid(len_array, len_array)
    cossim_mat = pd.DataFrame([[round(softcossim(sentences[i],sentences[j], sim
    return cossim_mat

soft_cosine_similarity_matrix(sentences)
```

Soft cosine similarity matrix

As one might expect, the similarity scores amongst similar documents are higher (see the red boxes).

## 6. Conclusion

Now you should clearly understand the math behind the computation of cosine similarity and how it is advantageous over magnitude based metrics like Euclidean distance.

Soft cosines can be a great feature if you want to use a similarity metric that can help in clustering or classification of documents.

If you want to dig in further into natural language processing, the gensim tutorial is highly recommended.

## Selva Prabhakaran

Selva is the Chief Author and Editor of Machine Learning Plus, with 4 Million+ readership. He has authored courses and books with100K+ students, and is the Principal Data Scientist of a global firm.

Previous Article

**Gensim Tutorial – A Complete Beginners Guide**

←

Next Article

**Parallel Processing in Python – A Practical Guide with Examples**

→

ALSO ON **MACHINELEARNINGPLUS.COM**

3 months ago • 1 comment
**Subplots Python (Matplotlib)**

a year ago • 1 comment
**Portfolio Optimization with ...**

3 mont
**How spa Aut**

**What do you think?**

73 Responses

Upvote 52

Love 21

Comments and reactions for this thread are now closed. ×

**Comments** **Community** **Privacy Policy** 1 **Login**

**Recommend** 3 Tweet Share **Sort by Newest**

**Lisa Ann Yu** • 6 months ago

Great intro. For Section 4, don't you want to exclude the stop words and hence only use the first line?

` ` `

count_vectorizer = CountVectorizer(stop_words='english')
count_vectorizer = CountVectorizer()
` ` `

^ | ˅ • Share ›

**Shashank Suresh Shahi** • a year ago

What the value "7" stands for in similarity matrix ? Can you elaborate the number in similarity matrix according to their meanings ? I got confused because of number"7". Thanks Your article cleared my doubt about cosine similarity.

^ | ˅ • Share ›

**Selva Prabhakaran** Mod → Shashank Suresh Shahi • a year ago

The calculation is incorrect Shashank. Take the smaller of the two numbers. Instead of 20 + 10 + 7, it should be 10 + 5 + 1.

^ | ˅ 1 • Share ›

**Cefas Garcia Pereira** • a year ago

Is there anyway to make the api.load('fasttext-wiki-news-subwords-300') faster?

1 ^ | ˅ • Share ›

**fundabasic** ➦ Cefas Garcia Pereira • a year ago

How long is it taking to load?

^ | ˅ • Share ›

**Suhas Gajendra** • 2 years ago

Thanks for the clear explanation.

Am facing the below error,
DeprecationWarning: Call to deprecated `similarity_matrix` (Method will be removed in 4.0.0, use gensim.models.keyedvectors.WordEmbeddingSimilarityIndex instead). [ipykernel_launcher.py:1]

^ | ˅ • Share ›

**Selva Prabhakaran** Mod ➦ Suhas Gajendra
• a year ago

Looks like that

^ | ˅ • Share ›

**Ihor Konovalenko** • 2 years ago

Article is very interesting. But, author wrote

> You would expect Doc A and Doc C, that is the two documents on Dhoni would have a higher similarity over Doc A and Doc B, because, Doc C is essentially a snippet from Doc A itself.

May be it would be more correct to say "You would expect Doc **B** and Doc C, that is the two documents on Dhoni... ...because, Doc C is essentially a snippet from Doc **B** itself".

It based on author's previous explanation that

> Two of the documents (A) and (B) are from the wikipedia pages on the respective players and the third document (C) is a smaller snippet from Dhoni's wikipedia page.

1 ^ | ˅ • Share ›

**Selva Prabhakaran** Mod ➦ Ihor Konovalenko
• a year ago

Yes, thats a typo. Pretty bad one. Have fixed it. Thanks for pointing out.

^ | ˅ • Share ›

**Laveena** • 2 years ago

Very impressive explanation, can you please explain how you have calculated "total_common_words" in Similarity Metrics? It doesn't actually matches my calculation for e.g. Doc Sachin & Doc Dhoni = (Dhoni)10 + (Cricket)50 + (Sachin)20

^ | ˅ • Share ›

**Saikam Rama Krishna Reddy** • 2 years ago

can you explain how similarity matrix derived from word counts matrix?

^ | ˅ • Share ›

**Sandeep Hooda** • 2 years ago

Your explanation style is very impressive. You might need few corrections in documents I guess. For example consider this "You would expect Doc A and Doc C, that is the two documents on Dhoni would have a higher similarity over Doc A and Doc B". But in you example you have taken doc B and C on Dhoni and doc A on sachin. Similarly in Trump and putin example "Since, Doc B has more in common with Doc A than with Doc C, I would expect the Cosine between A and B to be larger " Cosine should be small for A and B , right?

^ | ˅ • Share ›

**Luca Nannini** • 2 years ago

```
-------------------------------------------------------------
--------------
MemoryError Traceback (most recent call last)
<ipython-input-12-e9b5f1dc9c0b> in <module>
----> 1 print(softcossim(sent_1, sent_2, similarity_matrix))

~\Anaconda3\lib\site-packages\gensim\matutils.py in
softcossim(vec1, vec2, similarity_matrix)
814 vec1 = np.array([vec1[i] if i in vec1 else 0 for i in
word_indices], dtype=dtype)
815 vec2 = np.array([vec2[i] if i in vec2 else 0 for i in
word_indices], dtype=dtype)
--> 816 dense_matrix = similarity_matrix[[[i] for i in
word_indices], word_indices].todense()
817 vec1len = vec1.T.dot(dense_matrix).dot(vec1)[0, 0]
818 vec2len = vec2.T.dot(dense_matrix).dot(vec2)[0, 0]
```

```
~\Anaconda3\lib\site-packages\scipy\sparse\csc.py in
```

**see more**

• Share ›

## More Articles

NLP

### What is Tokenization in Natural Language Processing (NLP)?

Feb 01, 2021

NLP

### Text Summarization Approaches for NLP – Practical Guide with Generative Examples

Oct 24, 2020

NLP

**Complete Guide to Natural Language Processing (NLP) – with Practical Examples**

Oct 14, 2020

Enter Email*

Join

Subscribe to Machine Learning Plus for high value data science content

Resources

Blogs

Courses

Project Bluebook

Time Series Template

About us

Terms of Use

Privacy Policy

Contact Us

Refund Policy