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

by Selva Prabhakaran (https://www.machinelearningplus.com/author/selva86/) | Posted on October 22, 2018 (https://www.machinelearningplus.com/nlp/cosine-similarity/)

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?

Comparing Lemmatization Approaches in Python. Photo by Matt Lamers

[container] [columnize] 1. Introduction

Contents

[/columnize] [/container]

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.

Matplotlib Histogram – How to Visualize Distributions in Python (https://www.machinelearningplus.com/plots/matplot histogram-python-examples/)

ARIMA Model – Complete Guide to Time Series Forecasting in Python (https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/)

Time Series Analysis in Python – A Comprehensive Guide with Examples (https://www.machinelearningplus.com/time-series/time-series-analysis-python/)

Matplotlib Tutorial – A Complete Guide to Python Plot w/ Examples (https://www.machinelearningplus.com/plots/matplot tutorial-complete-guide-python-plot-examples/)

Topic modeling visualization – How to present the results of LDA models? (https://www.machinelearningplus.com/nlp/topic-modeling-visualization-how-to-present-results-lda-models/)

Top 50 matplotlib Visualizations – The Master Plots (with full python code) (https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/)

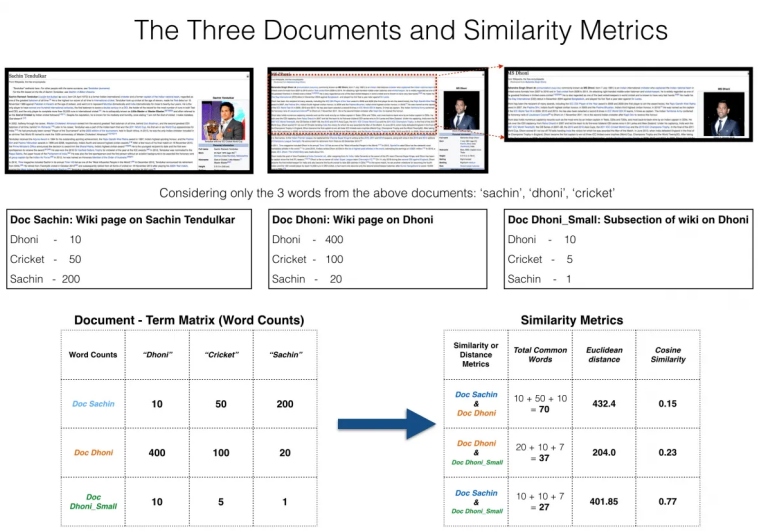
List Comprehensions in Python – My Simplified Guide (https://www.machinelearningplus.com/python/list-comprehensions-in-python/)

Python @Property Explained – How to Use and When? (Full Examples) (https://www.machinelearningplus.com/python/pytho property/)

How Naive Bayes Algorithm Works? (with example and full code) (https://www.machinelearningplus.com/predictive-modeling/how-naive-bayes-algorithm-works-with-example-and-full-code/)

Tags

Bigrams (https://www.machinelearningplus.com/tag/bigrams/) Classification (https://www.machinelearningplus.com/tag/cla Corpus (https://www.machinelearningplus.com/tag/corpus/) Cosine Similarity (https://www.machinelearningplus.com/tag/cosine-similarity/) data.table (https://www.machinelearningplus.com/tag/data-table/) Data Manipulation (https://www.machinelearningplus.com/tag manipulation/) Debugging (https://www.machinelearningplus.com/tag/debugging/) Doc2Vec (https://www.machinelearningplus.com/tag/doc2vec/)



(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/10/t/h/s/the_three_documents.png)

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 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.

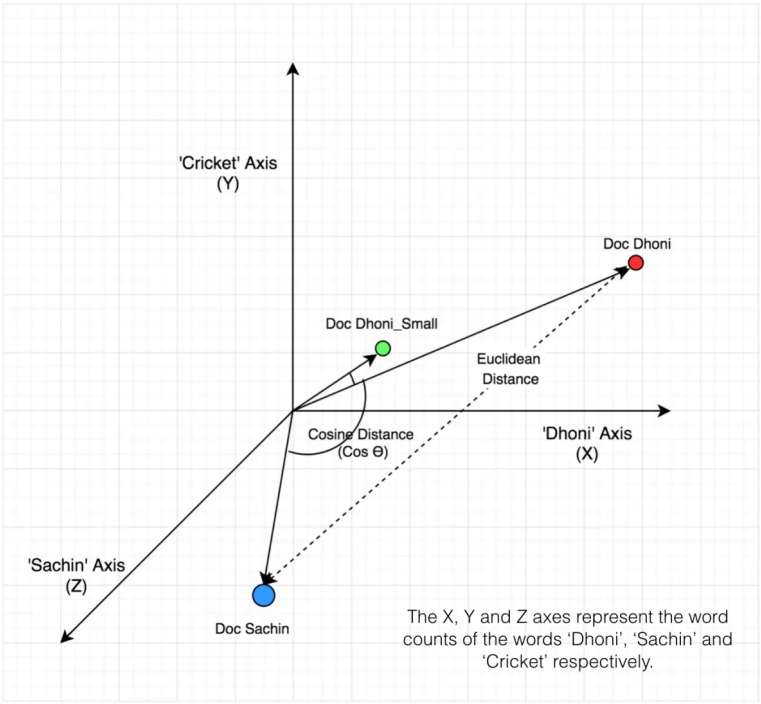
However, if we go by the number of common words, the two larger documents will have the most common words and therefore will be judged as most similar, which is exactly what we want to avoid.

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.

series/) Topic Modeling (https://www.machinelearningplus.com/tag/topic-modeling/) Visualization (https://www.machinelearningplus.com/tag/visualiz Word2Vec (https://www.machinelearningplus.com/tag/word2vec/)



(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/10/3/d/n/3d_projection.png)

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).

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

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.

(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/10/C/o/1/Cosine-Similarity-Formula-1.png)

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. Though he lost th

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

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

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 word frequencies.
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
```

	after	as	became	by	career	claimed	do	earlier	election	elections	...	the	though	to	trump	vladimir
doc_trump	1	0	1	0	0	0	0	0	1	0	...	1	1	0	2	0
doc_election	0	0	0	1	0	1	1	0	2	0	...	2	0	1	1	0
doc_putin	0	1	1	0	1	0	0	1	0	1	...	1	0	0	0	1

3 rows × 48 columns

(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/10/d/o/x/doc-term-matrix.png)

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.

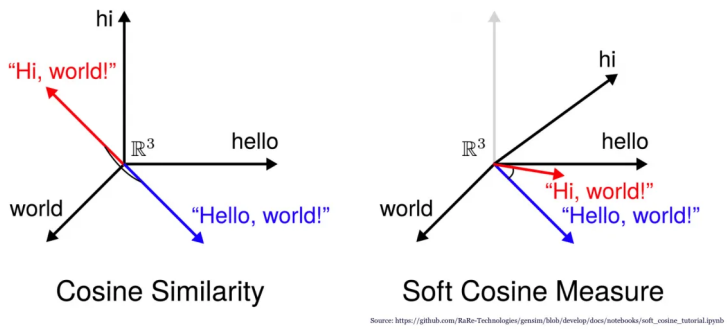
Then, use `cosine_similarity()` to get the final output. It can take the document term matri as a pandas dataframe as well as a sparse matrix as inputs.

```
# 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.



(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/10/s/o/e/soft-cosine.png)

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 may be cool or c

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

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

documents = [doc_trump, doc_election, doc_putin, doc_soup, doc_noodles, doc_dosa]
```

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, threshold=0.0,

# 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], similarity_matrix)
    return cossim_mat

soft_cosine_similarity_matrix(sentences)
```

	0	1	2	3	4	5
0	1.00	0.57	0.51	0.26	0.31	0.33
1	0.57	1.00	0.54	0.25	0.31	0.43
2	0.51	0.54	1.00	0.19	0.25	0.36
3	0.26	0.25	0.19	1.00	0.50	0.38
4	0.31	0.31	0.25	0.50	1.00	0.56
5	0.33	0.43	0.36	0.38	0.56	1.00

(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/10/s/o/x/soft-cosine-similarity-matrix.png)

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 (https://www.machinelearningplus.com/nlp/gensim-tutorial/) is highly recommended.