**SIMILARITY SEARCH FOR A DOCUMENT**

A Term Paper

Report

# Submitted in partial fulfilment of the Requirements for the award of the Degree of Bachelor of Technology

In

Computer science and Engineering Under the esteemed guidance of

**D.KOMALI**

(Assistant Professor)

**Done By Batch - 12**

|  |  |
| --- | --- |
| **STUDENT ID** | **STUDENT NAME** |
| 170030882 | N.Navya |
| 170030860 | M.V.S. PavanKalyan |

**(DST-FIST Sponsored Department) K L EDUCATION FOUNDATION**

**Green Fields, Vaddeswaram, Guntur District-522 502 2019-2020**

**K L EDUCATION FOUNDATION**

**DEPT OF COMPUTER SCINCE AND ENGINEERING**

**(DST-FIST Sponsored Department)**



**DECLARATION**

We hereby declare that thisTerm Paper reportentitled **“SIMILARITY SEARCH FOR A DOCUMENT”** has been prepared by us in partial fulfilment of the requirement for the award of degree bachelor of technology in **COMPUTERSCIENCEANDENGINEERING** in KL University. We also declare that this report is of our own effort and it has not been submitted to any other university for the award of anydegree.

N.Navya (170030882)

M.V.S. PavanKalyan (170030860)

**K L EDUCATION FOUNDATION DEPARTMENT OF COMPUTER SCIENCE ANDENGINEERING**

**(DST-FIST Sponsored Department)**

**K L EDUCATION FOUNDATION DEPARTMENT OF COMPUTER SCIENCE ANDENGINEERING**

**(DST-FIST Sponsored Department)**



**CERTIFICATE**

This is to certify that this Term Paper report entitled **“SIMILARITY SEARCH FOR A DOCUMENT”** is a bonafide work done by N.Navya (170030882) and M.V.S. PavanKalyan (170030860) submitted in partial fulfilment of the requirements for the award of Degree in Bachelor of Technology in **COMPUTER SCIENCE AND ENGNEERING** to K L University is a record of bonafide work carried out under efficient guidance and supervision. The results embodied in this report have not been copied from any other departments/University/Institute.

# Facultyin Charge Head of theDepartment

**D. KOMALI Dr. V. HARI KIRAN**

**(Assistant Professor)**

**ACKNOWLEDGEMENT**

Our sincere thanks to **D.KOMALI** in the research for her outstanding support throughout the project for the successful completion of the work.

We express our gratitude to **Dr.V. HARIKIRAN,** Head of the Department for computer science and Engineering for providing us with adequate facilities, ways and means by which we can complete this Research work.

We would like to place on record the deep sense of gratitude to the honourable Vice Chancellor, K L University for providing the necessary facilities to carry the work.

Last but not the least, we thank all Teaching and Non-Teaching Staff of our department and especially our classmates and our friends for their support in the completion of our work.

**N. Navya(170030882)**

**M.V.S. PavanKalyan (170030860)**

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **CHAPTERS** | **PAGE NO** |
| ABSTRACT | 6 |
| CHAPTER 1: INTRODUCTION |  |
| 1.1 INTRODUCTION | 8 |
| 1.2 PROBLEM DEFINITION | 8 |
| 1.3 PURPOSE | 8 |
| CHAPTER 2: LITERATURE SURVEY  CHAPTER 3: STATEMENT OF PURPOSE | 9 |
| 3.1 PLATFORM REQUIREMENTS | 11 |
| 3.2 MODULE DESCRIPTION | 11 |
| CHAPTER 4: BASIC DESCRIPTION |  |
| 4.1 ALGORITHMS | 13 |
| 4.2 PSEUDO CODE | 19 |
| CHAPTER 5: OUTPUTS AND SCREENSHOTS | 24 |
| CHAPTER 6: CONCLUSION | 27 |

|  |  |
| --- | --- |
| CHAPTER 7: REFERENCES | 28 |

**ABSTRACT**

Measuring similarity between texts is an important task for several applications. Available approaches to measure document similarity are inadequate for document pairs that have non- comparable lengths, such as a long document and its summary. This is because of the lexical, contextual and the abstraction gaps between a long document of rich details and its concise summary of abstract information. In this paper, we present a document matching approach to bridge this gap, by comparing the texts in a common space of hidden topics. We evaluate the matching algorithm on two matching tasks and find that it consistently and widely outperforms strong baselines. We also highlight the benefits of the incorporation of domain knowledge to text matching. The recent considerable growth in the amount of easily available on-line text has brought tothe foreground the need for large-scale natural language processing tools for text data mining.In this paper we address the problem of organizing documents into meaningful groupsaccording to their content and to visualize a text collection, providing an overview the range of documents and of their relationships, so that they can be browsed more easily. Great efficiency challenges arise in creating thesemaps. We study linguistically-motivated ways of reducing the representation of a document toincrease efficiency and ways to disambiguate the words in the documents.

**LIST OF FIGURES AND TABLES**

|  |  |
| --- | --- |
| **FIGURES AND TABLES** | **PAGE NO** |
| Tab3.1. requirements | 11 |
| Fig 4.1.1 cosine similarity | 13 |
| Fig 4.1.1.1 cosine similarity formula  Fig 4.1.2 Documents Comparision  Fig 4.1.2.8 Average similarity | 14  17  18 |

# Chapter-1: INTRODUCTION

**1.1. Introduction**

Similarity has to determine how ‘close’ two pieces of text are both in surface closeness [lexical similarity] and meaning [semantic similarity]. A commonly used approach to match similar documents is based on counting the maximum number of common words between the documents. We will use cosine similarity to find the similarity between two documents. 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. We will take two text documents as input for this project and on that dataset we will perform cosine similarity method in order to obtain similarity between two documents. And we will convert those documents into vectors.Thesevectorsarethearrayscontainingthewordcountsofthedocuments.

Smaller the angle, higher thesimilarity.

The cosine similarity helps overcome this fundamental flaw in the ‘count-the-common- words’ or Euclidean distanceapproach.

# 1.2. ProblemDefinition

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.

# 1.3. Purpose

We will use cosine similarity to find the similarity between two documents.

**Chapter-2: LITERATURE SURVEY**

This literature review shows lexical similarity measure. It is classified in two types of similarities:

1) character based similarity and 2) statement based similarity.

2.1 Character based similarity:

In character based similarity four algorithms are used namely:

1. LCS similarity
2. N-gram similarity
3. Levenshtein similarity
4. Jaro similarity.

2.1.1 LCS (Longest Common Subsequence) similarity:

It is a frequently used technique to compute the comparison between two strings. It calculates the longest substring from all the matched substring among two strings.

2.1.2 N-gram similarity:

Using N-gram similarity method we can find the similarity of sub-sequence of n objects. In this similarity we compute the similarity on the basis of distance between each character in two strings.

2.1.3 Levenshtein similarity:

It is a technique which uses the distance factor to calculate the similarity between given two strings.

2.1.4 Jaro similarity:

This technique defines the comparison between two strings on the basis of common character and it is used in duplicate detection.

2.2 Statement based similarity:

In statement based similarity three different algorithms are described namely:

1. Cosine Similarity
2. Centroid based similarity
3. Web Jaccard Similarity

2.2.1 Cosine Similarity:

It is broadly used method to find the similarity between two documents. Each text document is denoted in the form of vector.

2.2.2 Centroid based similarity:

It is a statement based similarity in which each statement is considered as vector form of a document.

2.2.3 Web Jaccard Similarity:

It is a count based co-occurrence measure technique. This technique is used to find the similarity between words.

# Chapter-3: STATEMENT OF PURPOSE

**3.1. Platform Requirements**

|  |  |  |
| --- | --- | --- |
| Hardware/ Software | Hardware / Software element | Specification /version |
| Hardware | Processor | Intel Core i5 |
| RAM | 8GB |
| Hard Disk | 10GB |
| Software | OS | Windows10  Jupyterlab |
| Python IDE |

**Tab 3.1. Platform Requirements**

**3.2 Module Description**

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).To compute the cosine similarity, you need the word count of the words in each document.

The Count Vectorizer or the TfidfVectorizer from scikit learn lets us compute this. The output on this, and optionally converting it to a pandas data frame to see the word freq. Even better, I could have used the TfidfVectorizer() instead of CountVectorizer(), because it would have down weighted words that occur frequently across documents. Then use cosine\_similarity() to get the final output. It can take the document term matrix as a data frame and also a sparse-matrixinputs.

**Chapter-4: BASIC DESCRIPTION**

* 1. **4.1. Algorithms**
  3. **4.1.1. Cosine Similarity**

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. 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.

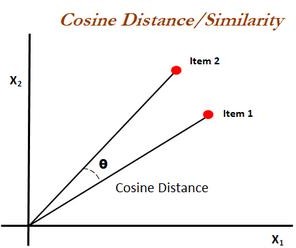


Fig 4.1.1 cosine similarity

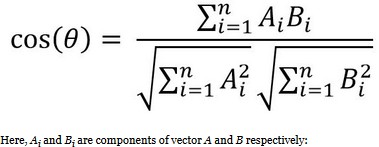


Fig 4.1.1.1 cosine similarity formula

**4.1.2 Natural Language Processing**

**Natural language processing** (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (**natural**) languages, in particular how to program computers to **process** and analyze large amounts of **natural language** data.

**Natural language processing** involves the reading and understanding of spoken or written **language** through the medium of a computer. This includes, for example, the automatic translation of one **language** into another, but also spoken word recognition, or the automatic answering of questions.

#### **4.1.2.1 NLTK**

**Natural language toolkit (NLTK)** is the most popular library for natural language processing (NLP) which was written in Python and has a big community behind it. NLTK also is very easy to learn, actually, it’ s the easiest natural language processing (NLP) library that we are going to use. It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.

**4.1.2.2 Genism**

**Gensim** is billed as a Natural Language Processing package that does ‘Topic Modeling for Humans’. But it is practically much more than that. It is a leading and a state-of-the-art package for processing texts, working with word vector models (such as Word2Vec, FastTextetc).

#### **4.1.2.3 Tokenization of words and sentences (NLTK)**

Tokenization is the process by which big quantity of text is divided into smaller parts called **tokens**.

Natural language processing is used for building applications such as Text classification, intelligent chatbot, sentimental analysis, language translation, etc. It becomes vital to understand the pattern in the text to achieve the above-stated purpose. **These tokens are very useful for finding such patterns as well as is considered as a base step for stemming and lemmatization.**

We use the method **word\_tokenize()** to split a sentence into words. The output of word tokenization can be converted to Data Frame for better text understanding in machine learning applications. It can also be provided as input for further text cleaning steps such as punctuation removal, numeric character removal or stemming. Machine learning models need numeric data to be trained and make a prediction. Word tokenization becomes a crucial part of the text (string) to numeric data conversion.

#### **4.1.2.4 Create a bag of words**

The **bag of words** algorithm uses **word** counts to represent the input text for your machine learning algorithm. It works like this: **Create** a bucket for each unique **word** you want represented (the vocabulary). Next go over the text and put a token in the right buckets for the **words** you encounter.

#### **4.1.2.5 TFIDF**

Term Frequency – Inverse Document Frequency(TF-IDF) is also a bag-of-words model but unlike the regular corpus, TFIDF down weights tokens (words) that appears frequently across documents.

Tf-Idf is calculated by multiplying a local component (TF) with a global component (IDF) and optionally normalizing the result to unit length. Term frequency is how often the word shows up in the document and inverse document frequency scales the value by how rare the word is in the corpus. In simple terms, words that occur more frequently across the documents get smaller weights.

#### **4.1.2.6 Creating similarity measure object**

Now, we are going to create similarity object. The main class is Similarity, which builds an index for a given set of documents.The Similarity class splits the index into several smaller sub-indexes, which are disk-based. Let's just create similarity object then you will understand how we can use it for comparing.We are storing index matrix in 'workdir' directory but you can name it whatever you want and of course you have to create it with same directory of your program.

#### **4.1.2.7 Create Query Document**

Once the index is built, we are going to calculate how similar is this query document to each document in the index. So, create second .txt file which will include query documents or sentences and tokenize them as we did before.

#### **4.1.2.8 Average Similarity**

I think it is better to calculate average similarity of query document. At this time, we are going to import numpy to calculate sum of these similarity outputs.Numpy will help us to calculate sum of these floats and output is: To calculate average similarity we have to divide this value with count of documents.

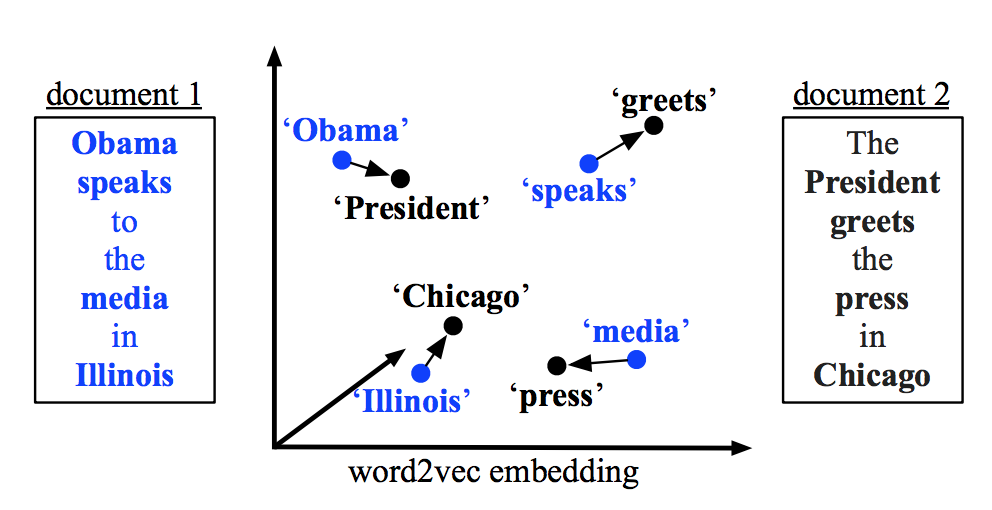


Fig 4.1.2 Documents Comparision

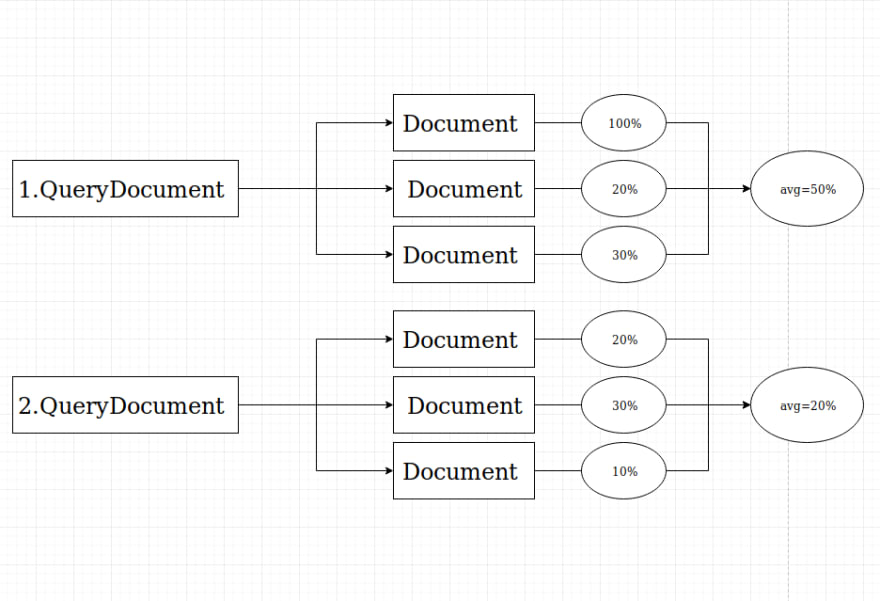


Fig 4.1.2.8 Average similarity

**4.2 Pseudo Code**

**4.2.1 Dataset**

Consider 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 witch hunt 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.

# 4.2.2 Source Code

# 4.2.2.1 Cosine Similarity

doc\_trump = "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\_election = "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 = "Post elections, Vladimir Putin became President of Russia. President Putin had served as the Prime Minister earlier in his political career"

documents = [doc\_trump, doc\_election, doc\_putin]

fromsklearn.feature\_extraction.text import CountVectorizer

import pandas as pd

count\_vectorizer = CountVectorizer(stop\_words='english')

count\_vectorizer = CountVectorizer()

sparse\_matrix = count\_vectorizer.fit\_transform(documents)

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

fromsklearn.metrics.pairwise import cosine\_similarity

print(cosine\_similarity(df, df))

**4.2.2.2 Natural Language Processing and Gensim**

pip install nltk

pip install gensim

importnltk

nltk.download('punkt')

fromnltk.tokenize import word\_tokenize

data = "Mars is approximately half the diameter of Earth."

print(word\_tokenize(data))

fromnltk.tokenize import sent\_tokenize

data = "Mars is a cold desert world. It is half the size of Earth. "

print(sent\_tokenize(data))

importnltk

fromnltk.tokenize import word\_tokenize, sent\_tokenize

file\_docs = []

with open ('demofile.txt') as f:

tokens = sent\_tokenize(f.read())

for line in tokens:

file\_docs.append(line)

print("Number of documents:",len(file\_docs))

gen\_docs = [[w.lower() for w in word\_tokenize(text)] for text in file\_docs]

dictionary = gensim.corpora.Dictionary(gen\_docs)

print(dictionary.token2id)

corpus = [dictionary.doc2bow(gen\_doc) for gen\_doc in gen\_docs]

tf\_idf = gensim.models.TfidfModel(corpus)

for doc in tfidf[corpus]:

print([[dictionary[id], np.around(freq, decimals=2)] for id, freq in doc])

sims = gensim.similarities.Similarity('C:/Users/M V S P K L/Desktop/file',tf\_idf[corpus],num\_features=len(dictionary))

file2\_docs = []

with open ('demofile2.txt') as f:

tokens = sent\_tokenize(f.read())

for line in tokens:

file2\_docs.append(line)

print("Number of documents:",len(file2\_docs))

for line in file2\_docs:

query\_doc = [w.lower() for w in word\_tokenize(line)]

query\_doc\_bow = dictionary.doc2bow(query\_doc) #update an existing dictionary and create bag of words

query\_doc\_tf\_idf = tf\_idf[query\_doc\_bow]

print('Comparing Result:', sims[query\_doc\_tf\_idf])

importnumpy as np

sum\_of\_sims=(np.sum(sims[query\_doc\_tf\_idf], dtype=np.float32))

print(sum\_of\_sims)

percentage\_of\_similarity = round(float((sum\_of\_sims / len(file\_docs)) \* 100))

print(f'Average similarity float: {float(sum\_of\_sims / len(file\_docs))}')

print(f'Average similarity percentage: {float(sum\_of\_sims / len(file\_docs)) \* 100}')

print(f'Average similarity rounded percentage: {percentage\_of\_similarity}')

avg\_sims = []

for line in file2\_docs:

query\_doc = [w.lower() for w in word\_tokenize(line)]

query\_doc\_bow = dictionary.doc2bow(query\_doc)

query\_doc\_tf\_idf = tf\_idf[query\_doc\_bow]

print('Comparing Result:', sims[query\_doc\_tf\_idf])

sum\_of\_sims=(np.sum(sims[query\_doc\_tf\_idf], dtype=np.float32))

avg = sum\_of\_sims / len(file\_docs)

print(f'avg: {sum\_of\_sims / len(file\_docs)}')

avg\_sims.append(avg)

total\_avg = np.sum(avg\_sims, dtype=np.float)

percentage\_of\_similarity = round(float(total\_avg) \* 100)

ifpercentage\_of\_similarity>= 100:

percentage\_of\_similarity = 100

# Chapter-5: OUTPUTS AND SCRENSHOTS

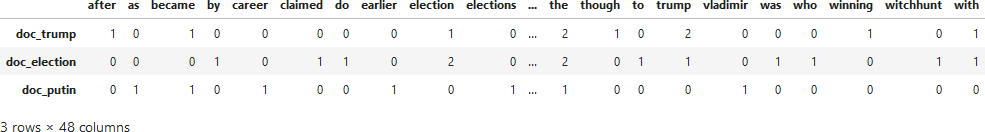
**5.1 Cosine Similarity Related Outputs and Screenshots**

**Output:**

[[1. 0.514804850.38890873]

[0.514804851. 0.38829014]

[0.388908730.38829014 1. ]]



**5.2 Natural Language Processing and Gensim Outputs and Screenshots**

**Output:**

**Document1:**

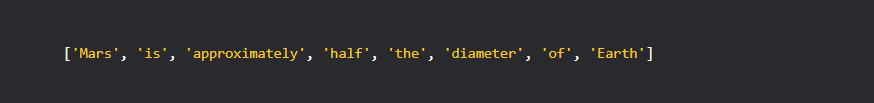
Mars is the fourth planet in our solar system.

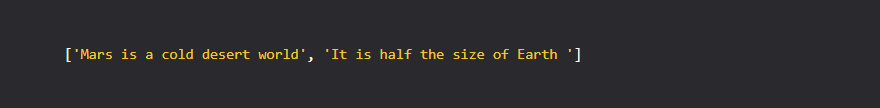
It is second-smallest planet in the Solar System after Mercury.

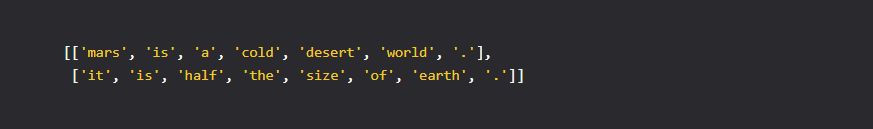
Saturn is yellow planet.

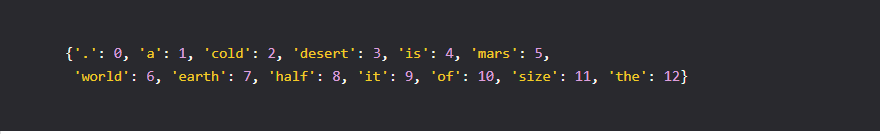
**Document 2:**

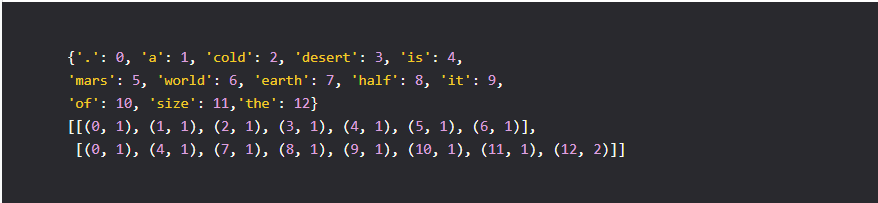
Saturn is the sixth planet from the Sun.







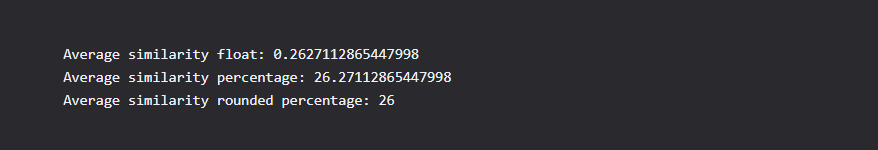


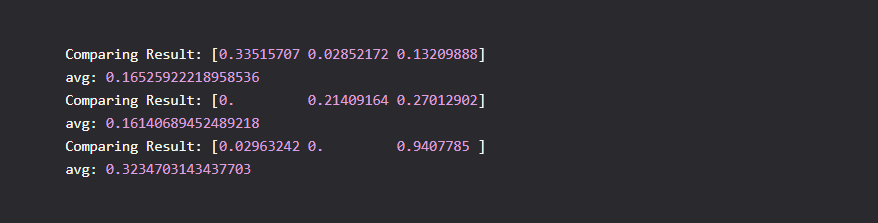














**Chapter-6: CONCLUSION**

Document similarity search is to find documents similar to a query document in a text corpus and return a ranked list of documents to users, which is widely used in recommender systems in library or web applications. The popular approach to similarity search is to calculate the similarities between the query document and documents in the corpus and then rank the documents. This project is very much useful to find how much similar are the documents. 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. This similarity search is more popularly used by the search engines like Google Chrome, Internet Explorer, Mozilla Firefox etc.

# Chapter-7: REFERENCES

# Aggarwal, C.C., Yu, P.S.: On effective conceptual indexing and similarity search in text data. In: Proceedings of the 2001 IEEE International Conference on Data Mining, San Jose, California, USA, pp. 3–10 (2001)

# Carbonell, J., Goldstein, J.: The Use of MMR, Diversity-based Reranking for Reordering Documents and Producing Summaries. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 335–336 (1998)

1. Graham L. Giller (2012). "The Statistical Properties of Random Bitstreams and the Sampling Distribution of Cosine Similarity"*.*
2. Sidorov, Grigori; Gelbukh, Alexander; Gómez-Adorno, Helena; Pinto, David (29 September 2014). ["Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model"](http://cys.cic.ipn.mx/ojs/index.php/CyS/article/view/2043).