Text Similarity Checker

Using Python

Mini Project report submitted

in partial fulfilment of the requirement for the degree of

Bachelor of Technology

In

Electronics & Telecommunication Engineering

By

Kishor Kumar Das (180610326047)

Aashis Kumar Chittawat (180610026001)

Under the guidance

of

Kamini Kumari, Assistant Professor, ETE Dept

Ankur Jyoti Sarma, Assistant Professor, ETE Dept

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING

ASSAM ENGINEERING COLLEGE

JALUKBARI- 781013, GUWAHATI

July, 2021

ASSAM ENGINEERING COLLEGE, GUWAHATI

CERTIFICATE

This is to certify that the report entitled “Text Similarity Checker using Python” submitted by Kishor Kumar Das (180610326047) and Aashis Kumar Chittawat (180610026001) of B.Tech. 6th semester, Electronics & Telecommunication Engineering Department of Assam Engineering college is an authentic work carried out by them under our supervision and guidance.

To the best of our knowledge, the matter embodied in the report has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Signature of Supervisor(s)
Kamini Kumari, Assistant Professor, ETE Dept
Ankur Jyoti Sarma, Assistant Professor, ETE Dept
Electronics and Telecommunication Engineering
Assam Engineering College
July, 2021

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)
Kishor Kumar Das
180610326047
Date:

(Signature)
Aashis Kumar Chittawat
180610026001
Date:

ACKNOWLEDGMENT

We would like to take this opportunity to thank our Head of the Department, Prof. Dinesh Shankar Pegu, HOD, ETE Dept for giving us the opportunity to do a mini project on “Text Similarity Checker using Python”. We would also like to thank our Guide Kamini Kumari, Assistant Professor, ETE Department along with our co-guide Ankur Jyoti Sarma, Assistant Professor, ETE Department for guiding us in the project, for providing valuable suggestions, for their ongoing support during the project, from initial advice and provision of contacts in the first stages through ongoing advice and encouragement, which led to the final report of this mini project.

We are also grateful to other teachers of the department for contribution their inputs in this project

Kishor Kumar Das (180610326047)
Aashis Kumar Chittawat (180610026001)

ABSTRACT

In a world where the Internet has become a part of everyday life and an integral part of education with its massive amount of freely available information, it has become easier than ever to steal text and pass it as your own without being detected. Students seldom use books when researching and writing text. It is far more common to use the Internet for these purposes. Plagiarists can simply copy the text from some unknown source from the Internet to save themselves hours of work, and in many cases this goes undetected. Detecting textual plagiarism is a tedious and almost impossible process when done manually.

Text similarity is a process where two texts are compared to find the degree of similarity between them. And text similarity checker is a tool or software used to measure the degree of similarity. Measuring the similarity between words, sentences, paragraphs and documents is an important component in various tasks such as information retrieval, document clustering, word-sense disambiguation, automatic essay scoring, short answer grading, machine translation and text summarization. Or in simple words it plays an important role in our daily life. There are numerous ways to check similarity such as Block distance, Cosine similarity, Dice's coefficient, Jaccard similarity, etc.

Here we used the cosine similarity using TF-IDF algorithm. Our Methodology involves 3 steps: Data Preparation, Pre-processing, Calculation of TF-IDF (using cosine similarity).

In data preparation, all the raw data is cleared and reformatting of data is done. After that combining of data on the basis of analysis.

In Pre-processing of data, all uppercases are converted into lowercases. Some grammatical changes also done like removing of punctuations, removing of stop words, etc. Stemming (reducing word to its root level) is also done in this level.

In mathematical calculation of the project, the mathematical calculation of the method used is done. In the present project of text similarity we have used cosine similarity method and in this method we need dot product of two vectors, graph theory, cosine formula and etc., all these are discussed in this third part.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

Chapter1

Introduction

In this section we are going to give a brief over view of the chapter. Firstly, deal with what is text similarity and what is text similarity checker? in brief. Text similarity is a process where two texts are compared to find the degree of similarity between them. And text similarity checker is a tool or software used to measure the degree of similarity. Secondly, what is the use of it in our day to day life? Measuring the similarity between words, sentences, paragraphs and documents is an important component in various tasks such as information retrieval, document clustering, word-sense disambiguation, automatic essay scoring, short answer grading, machine translation and text summarization. Or in simple words it plays an important role in our daily life. Thirdly, what are the various methods in this field which is used to text check similarity?. There are numerous ways to check similarity such as Block distance, Cosine similarity, Dice's coefficient, Jaccard similarity, etc. Among these we used a specific method due to its uniqueness and high efficiency. Fourthly, what is the future of text similarity checking?. As we all know as the day to day life is drastically changing towards digital world so it become more easier from early days to steal one's data and use it in another one's work, but it is illegal to do so and there are many more such illegal work of copying one's data and pasting it as one's own in this digitalised world, is possible. So, with that in respect the use of text similarity checker or any other type of similarity checker in more demanding in the near future.

1.1 Introduction to the area of work:

This project we are basically dealing with the text similarity checking as there are many more type of similarity such as String based similarity , Corpus based & Knowledge based similarity (these similarities are further divided into sub groups i.e, shown in the fig1,fig2 and fig3 respectively), but our topic is text similarity. In our project we are going to use python programming language to build our project, and we use jupyter IDE due to its user friendly interface and easily accessed from any device using internet.

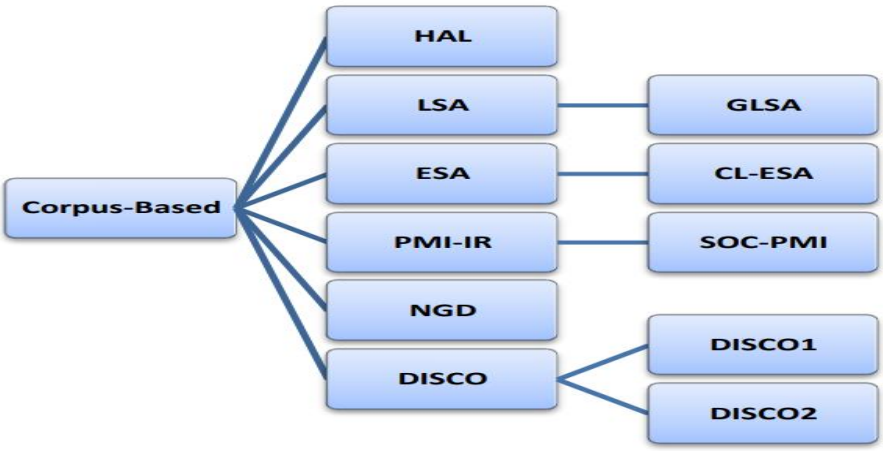

Figure 1.0 Corpus Based Similarity

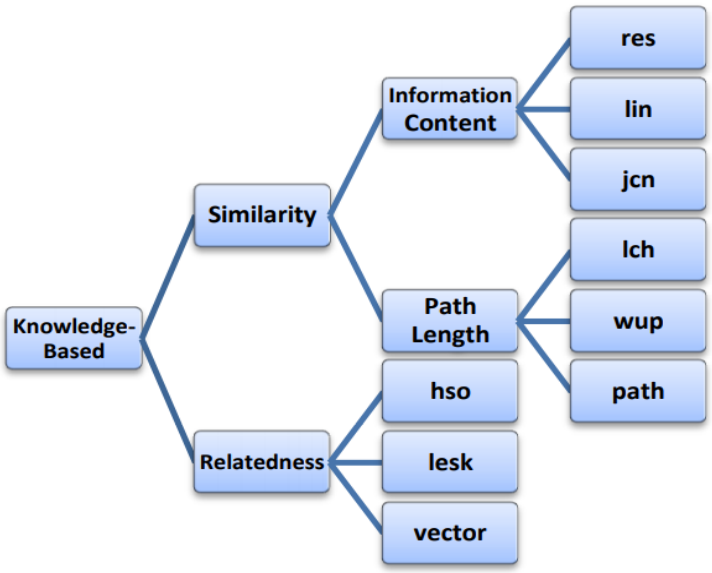

Figure 1.1 String Based Similarity

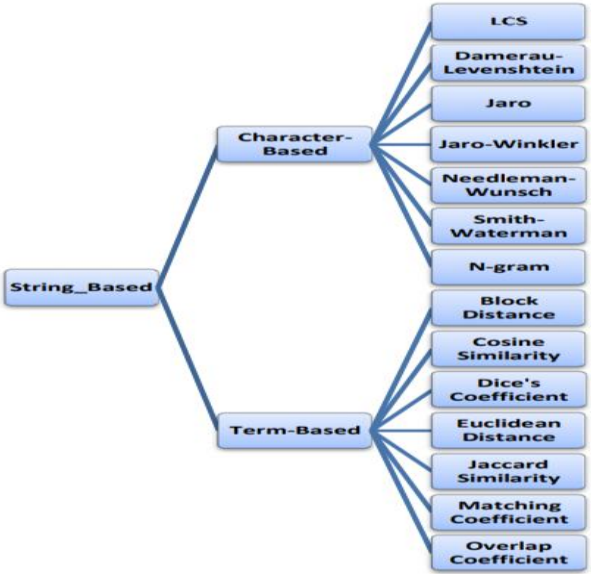

Figure 1.2 Knowledge Based Similarity

1.2 Brief present day scenario with regard to the work of text similarity checking:

In today's world it is used in many fields such as

- *Biomedical Informatics:* Basically compare the genes.
- *Geo-Informatics:* Find similarity between geological features.
- *Natural Language Processing:* It is already being used in this field (IT) from early age, but now a days its efficiency in this field is increased.

1.3 Motivation to do the project work:

- *Uniqueness of the methodology that will be adopted:* Regarding the similarity of text there are different methods to check the similarity of words, sentences and paragraph depending upon the type of similarity needs to be checked. For example, if we want to check the similarity use different method instead of the method choosed for similarity in the meaning.

- *Significance of the possible end result:* The result we get in this project is the most accurate and authentic result. A brief reason of it is that the method (cosine similarity) that we preferred to use in this project is based on document orientation as well as on document distance, it is unlike the other method those are only based on document distances.

- *Brief importance of the Text similarity in the present context:* In present world of digitalisation it is very easy to steal data of others and use it as your own in any social platform, but it is a illegal work. So, with this in regard the government of each country come up with a solution and i.e., to build a software to check plagiarism (similarity). And there are many more uses of it such as in educational institutes, bio-medical etc.

1.4 Objective of the text similarity checker:

In a world where the Internet has become a part of everyday life and an integral part of education with its massive amount of freely available information, it has become easier than ever to steal text and pass it as your own without being detected. Students seldom use books when researching and writing text. It is far more common to use the Internet for these purposes. Plagiarists can simply copy the text from some unknown source from the Internet to save themselves hours of work, and in many cases this goes undetected. Detecting textual plagiarism is a tedious and almost impossible process when done manually. With millions of possible sources online, this entire process benefits from being automated. A long line of commercial, automatic plagiarism detection tools have been made available to both academic institutions and individuals. The inner workings of these are seldom public due to the commercial nature of the products. This is an active field of research, but the accuracy of these tools falters as the plagiarism becomes more complex. Verbatim copies are easy to detect, but as words are replaced with synonyms and shuffled around, detecting plagiarism becomes harder. Many different approaches have been designed attempting to handle these challenges, but few have yet to reach a very high level of accuracy in complex plagiarism methods. Different levels in the education system require different amounts of your own work in order to be legitimate hand-ins. Children in high school or lower grades could potentially have more lenient plagiarism detection systems. These could allow for summaries and simple rewrites of passages as long as it is shown that some work has been done on the text, and it is not a simple verbatim copy, their text should be passable as their own. Some classes may focus on purely the research and not the writing, while others may be vice versa, and should be very strict. In higher levels of education even summaries are often not acceptable, but a level of verbatim copying should perhaps be without triggering an alarm to allow for quotes. Finding the perfect settings for each of these scenarios is tedious and seldom produce optimal results. Tuning it also requires knowledge of the algorithm, which few users have.

1.5 Target specification:

After performing the trails on our project we got little bit change each time due little change in the pattern of words and with that it show the limit of accuracy it has. The end result is calculated by using cosine formula and dot product of mathematics. From the end result one can see that it is easy to check plagiarism in between words in this digital world. And the main importance of the end result is that it shows the extend or percent of similarity the two words have in between each other.

1.6 Organisation of project report:

In the first chapter a brief introduction to the text similarity is provided along with its present day scenario. It also contain about the methods used in this project and also the objective of this project.

In the second chapter there is a literature review of the project. It also have all the mathematical derivations and theoretical part of project. And it also contains the total outcome of the literature that have been used here.

In the third chapter the complete methodology is discussed in detailed.

It contains all the assumptions made in the method to do this project along with the description of the tool used in this project.

In the fourth chapter it give complete review of the project in brief and also provide the significance of the end result.

Chapter2

Introduction

In this chapter the review of the literatures, used in the project, is given. In this field there are various experiments done to come up with a solution to plagiarism in different sectors of digital world and a brief discussion is done in this regard. Here a specific discussion on the project title i.e., text similarity is done. About the brief history of the text similarity is provided in this chapter. Apart from history part it also contain the theory part. Whatever mathematics topic used is discussed in this chapter along with some derivations of them.

2.2 Introduction to the Text similarity:

As already discussed in the chapter1 that text similarity means the degree of similarity present in between the two compared documents. For checking similarity a method is needs to be followed and the method consists of three steps:

- ✓ Firstly, data preparation.
- ✓ Secondly, Preprocessing,
- ✓ Thirdly, Mathematical calculation of the project.

In data preparation, all the raw data is cleared and reformatting of data is done. After that combing of data on the basis of analysis.

In Preprocessing of data, all uppercases are converted into lowercases. Some grammatical changes also done like removing of punctuations, removing of stop words, etc. Stemming (reducing word to its root level) is also done in this level.

In mathematical calculation of the project, the mathematical calculation of the method used is done. In the present project of text similarity we have used cosine similarity method and in this method we need dot product of two vectors, graph theory, cosine formula and etc., all these are discussed in this third part.

2.3 Literature Review:

2.3.1 Present state / recent developments in the work area:

The issue of plagiarism is raised in early 70's and from them onwards till the present day the researchers are working in this field. They developed tools for checking plagiarism in different languages in both text detection as well as in source code detection. These developments were done on the basis of

1. *Monolingual Plagiarism Detection:* This detection is done in homogeneous language settings such as English-English. They are further sub-divided into two part, and they are :
    a) Intrinsic Plagiarism Detection: This detection is based on the writing style as well as on the basis of owner conformity with the available local data.
    b) Extrinsic Plagiarism Detection: This approach uses an external source to compare the submitted data.
2. *Cross-Lingual Plagiarism Detection:* This detection is done in heterogeneous language settings such as English-Chinese.

But still there are some development needs to be done, and those are,

1. A proof of completeness and proof of correctness is still missing even after checking the similarity between the data.
2. A common solution or measure that completely detect the plagiarized text segment in both intrinsic (compared depending upon itself) and extrinsic (compared with an external source) is still not available.
3. And there is need to develop a text similarity checker that accepts a data narrated by a user and find similarity of it with correct source.

Rather than developments needed in this field there were many developments done in this field. Some of the tools developed till now are as follows,

1. SafeAssignment: This checker is based on proprietary (owner) searching and ranking algorithms.
2. Urkund: This is a checker which check plagiarism in server side. It uses email system for submitting and viewing result.
3. Copycatch: It is most authenticated checker in the field of plagiarism. It has two types of it, one is client based and second one is web based. In client based it simply use the local database for comparison and in web based user need personal Google API.
4. Turnitin: This checker is web based provided by iParadigms. It is the most widely used plagiarism checker due to its user friendly interface and an organised report generation.

And there are many more such type of tools available.

2.3.2 Brief background theory:

In our project of text similarity checker we have analysed the topic and found that it comes under string based similarity. After going through many research it is known that there are many methods to solve the similarity problem such as Block distance similarity, Cosine similarity, Jaccard similarity, etc. Among this solutions we found that cosine similarity is the effective one. Then we build the algorithm for it which includes dot product of two vectors, Term frequency, Inverse document frequency, document length, etc.

- *Term frequency*: How frequently a term occurs in a document.
- *Inverse document frequency*:- $\log\{\frac{(\text{total no. of docs})}{(\text{no. of docs with the term in it})}\}$

2.3.3 Literature survey:

We have gone through different websites and registered sources for this project and links of those are given in the reference section. After the survey done by us then it is seen that a number of developments were made on the basis of monolingual and cross-lingual. But even after that there are still development is needed.

2.4 Summarized outcome of the literature review:

Summarising the outcome of literature part we get that every methods have some cons, not single method is fully effective. But even after that, going through the survey we found that cosine similarity is the effective one. This similarity method is based upon the cosine dot product, frequency of each term occurred. It also uses logarithmic methods in it. If the dot product if the two document vector is 1 then, they are fully similar otherwise there is some dissimilarity between them.

2.5 Theoretical discussions:

Now the theory of text similarity is that it calculated by calculating text distance. In this case each word is treated as a vector and angle between them is calculated. And to start with the process first data preparation is done, after that pre-processing is done and at last mathematical calculation is done.

To understand Cosine similarity let's take an example, consider following four documents,

d1: The best Italian restaurant enjoy the best pasta.

d2: American restaurant enjoy the best hamburger.

d3: Korean restaurant enjoy the best bibimbap.

d4: The best the best American restaurant.

Now, vector representation of document 1, 2, 3 and 4 are:-

d1: [1,1,1,2,2,1,0,0,0,0]

d2: [0,1,1,1,1,0,1,1,0,0]

d3: [0,1,1,1,1,0,0,0,1,1]

d4: [0,1,0,2,2,0,1,0,0,0]

Now applying cosine-similarity we get,

Cosine similarity = $\frac{A.B}{||A||.||B||}$.

- ✓ For d1and d4,

Cosine similarity = 0.821.

- ✓ For d2 and d4,

Cosine similarity=0.77.

- ✓ For d3 and d4,

Cosine similarity= 0.64.

- ✓ For d4 and d4,

	Italian	restaurant	enjoy	the	best	Pasta	American	hamburger	Korean	bibimbap
d1	1	1	1	2	2	1	0	0	0	0
d2	0	1	1	1	1	0	1	1	0	0
d3	0	1	1	1	1	0	0	0	1	1
d4	0	1	0	2	2	0	1	0	0	0

Table 2.0: Showing the no. of times text occurred in document

✓ Cosine similarity=1.

✓ For d2 and d4,

Cosine similarity=0.77.

✓ For d3 and d4,

Cosine similarity= 0.64.

✓ For d4 and d4,

Cosine similarity=1.

✓ For d3 and d4,

Cosine similarity= 0.64.

✓ For d4 and d4,

Cosine similarity=1.

Document Id	Cosine similarity with d4
d1	0.82
d2	0.77
d3	0.65
d4	1

Table 2.1: Cosine similarity value of documents with document4

words	TF				IDF	TF.IDF			
	d1	d2	d3	d4		d1	d2	d3	d4
Italian	1/8	0	0	0	Log(4/1) = 0.602	0.075	0	0	0
Restaurant	1/8	1/6	1/6	1/6	Log(4/4) = 0	0	0	0	0
enjoy	1/8	1/6	1/6	0	Log(4/3) =0.125	0.015	0.020	0.020	0
the	2/8	1/6	1/6	2/6	Log(4/4) = 0	0	0	0	0
best	2/8	1/6	1/6	2/6	Log(4/4) = 0	0	0	0	0
pasta	1/8	0	0	0	Log(4/1) = 0.602	0.075	0	0	0
American	0	1/6	0	0	Log(4/2) = 0.301	0	0.050	0	0.050
hamburger	0	1/6	0	0	Log(4/1) = 0.602	0	0.99	0	0
Korean	0	0	1/6	0	Log(4/1) = 0.602	0	0	0.99	0
bibimbap	0	0	1/6	0	Log(4/1) = 0.602	0	0	0.99	0

Table 2.3: term-frequency (TF) & Inverse Document Frequency (IDF)

Document id	TF-IDF bag of words	Cosine similarity with d4
d1	[0.075,0,0.075,0,0.075,0,0,0,0]	0
d2	[0,0,0.2,0,0,0,0.05,0.1,0,0]	0.44
d3	[0,0,0.02,0,0,0,0,0,0.1,0.1]	0
d4	[0,0,0,0,0,0,0,0.05,0,0,0]	1

Table 2.4: Cosine similarity value of documents with document4 using TF-IDF

2.6 General Analysis

From the above context it can be seen that how a cosine similarity work.

TF-IDF stands for “Term Frequency — Inverse Document Frequency”.

This is a technique to quantify a word in documents, we generally compute a weight to each word which signifies the importance of the word in the document and corpus.

Term Frequency also known as TF measures the number of times a term (word) occurs in a document.TF-IDF is the multiplication of term frequency and inverse document frequency.

2.7 Mathematical derivations:

- Cosine similarity:-

Cosine similarity = $\frac{\text{A.B}}{||\text{A}||.||\text{B}||}$.

- $IDF = 1 + log\left(\frac{\text{Total Number Of Documents}}{\text{Number Of Documents with required term in it}}\right)$
- A.B = |A||B| cosθ

CHAPTER 3

METHODOLOGY

In this chapter we will discuss about the Methodology/ algorithm we used build our project. Also we will discuss about various tools/ software that help us to complete this project.

3.1 Methodology

Below we discuss about the detailed methodology which we have accepted in order to complete our project. Here we used the cosine similarity using TF-IDF algorithm. Our Methodology involves 3 steps:

3.1.1 Data Preparation:

3.1.2 Pre-processing

3.1.3 Calculation of TF-IDF (using cosine similarity)

3.1.1 Data Preparation: As each dataset is different and highly specific to the project, Data preparation become a most difficult steps in any data driven project. Data preparation involves steps like cleaning the raw data, Access the data, ingest (or fetch) the data, format the data, combine the data and finally, analyse the data. In our case we need to convert the given document file (pdf, docx, ppt, html) to .txt file by using external software.
After that we store the .txt files in a folder in our system. Then we read all the files looping through all of them and store all the files in a list. After that we will proceed to the next step which is pre-processing.

3.1.2 Pre-processing: Pre-processing is one of the major steps when we are dealing with any kind of text models. During this stage we have to look at the distribution of our data, what techniques are needed and how deep we should clean. Few mandatory pre-processing are converting to lowercase, removing punctuation, removing stop words and lemmatization/stemming.

- *Convert Lower Case:* During the text processing each sentence is split to words and each word is considered as a token after pre-processing. Programming languages consider textual data as sensitive, which means that 'The' is different from 'the'. We humans know that those both belong to same token but due to the character encoding those are considered as different tokens. Converting to lowercase is a very mandatory pre-processing step. As we have all our data in list, numpy has a method which can convert the list of lists to lowercase at once.

```
In [7]: import numpy as np
        doc_1 = "CONVERT to LowerCase"
        np.char.lower(doc_1)

Out[7]: array('convert to lowercase', dtype='<U20')
```

Figure 2.0: code snippet-convert to lower case

- *Remove punctuation:* Punctuation are the unnecessary symbols. We are going to store all our symbols in a variable and iterate that variable removing that particular symbol in the whole dataset. To do so we used numpy library.

```
In [15]: import numpy as np
         doc_2 = "!@#$ Remove *() punctuation @#"
         symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
         for i in symbols:
             doc_2 = np.char.replace(doc_2, i, '')
         doc_2
Out[15]: array(' Remove  punctuation ', dtype='<U21')
```

Figure 3.1: code snippet-remove punctuation

- *Remove apostrophe:* Like punctuation, apostrophes are also the unnecessary symbols. So we remove it. To remove the apostrophe we replace each apostrophe with blank space using np.char.replace () method.

```
In [23]: import numpy as np
         doc_2 = "Remove ' Apostrophe"
         doc_2 = np.char.replace(doc_2, "'","")
         doc_2
Out[23]: array('Remove  Apostrophe', dtype='<U18')
```

Figure 4.2: code snippet-Remove apostrophe

- *Remove single characters:* Single characters are not much useful in knowing the importance of the document and few final single characters might be irrelevant symbols, so it is always good be remove the single characters.

```
In [33]: doc_3 = "Remove a single character"
         new_text = ""
         for w in doc_3.split(" "):
             if len(w) > 1:
                 new_text = new_text + " " + w
         print(new_text)
          Remove single character
```

Figure 5.3: code snippet-remove single character

- *Convert numbers:* As we know that in Programming Language textual data is sensitive. So if the user search for 100 or hundred we should return both the terms are same. Here we used num2words library which convert a given number into its word form.

```
In [34]: from num2words import num2words
         num2words(100)
Out[34]: 'one hundred'
```

Figure 6.4: code snippet-convert numbers

- *Remove stop words:* Stop words are the most commonly occurring words which don't give any additional value to the document vector. In-fact removing these will increase computation and space efficiency. nltk

library has a method to download the stopwords, so instead of explicitly mentioning all the stopwords ourselves we can just use the nltk library and iterate over all the words and remove the stop words.

- *Stemming:* This is the final and most important part of the pre-processing. Stemming converts words to its stem. For example 'playing' and 'played' are the same type of words which basically indicate an action play. We are going to use a library called "porter-stemmer" which is a rule based stemmer. Porter-Stemmer identifies and removes the suffix or affix of a word. The words given by the stemmer need note be meaningful few times, but it will be identified as the same for the model.

```
In [44]: from nltk.stem import PorterStemmer
         stemmer = PorterStemmer()
         stemmer.stem("Playing")

Out[44]: 'play'
```

Figure 7.5: code snippet-stemming

3.*1.3 Calculation of TF-IDF:* TF-IDF stands for "Term Frequency-Inverse Document Frequency". This is a technique to quantify a word in documents, we generally compute a weight to each word which signifies the importance of the word in the document and corpus.

- *Term Frequency (TF):* Term Frequency also known as TF measures the number of times a term (word) occurs in a document.
- *Normalized Term Frequency*: In reality each document will be of different size. On a large document the frequency of the terms will be much higher than the smaller ones. Hence we need to normalize the document based on its size. A simple trick is to divide the term frequency by the total number of terms.

- *Inverse Document Frequency (IDF):* In Term Frequency all terms are considered equally important. In fact certain terms that occur too frequently have little power in determining the relevance. We need a way to weigh down the effects of too frequently occurring terms. Also the terms that occur less in the document can be more relevant. We need a way to weigh up the effects of less frequently occurring terms. Logarithms helps us to solve this problem. Logarithms helps us to solve this problem.

$$IDF = 1 + \left(\frac{The\ number\ of\ documents}{number\ of\ documents\ with\ required\ item\ in\ it} \right)$$

- *TF-IDF score across all docs for the query string:* TF-IDF is the multiplication of term frequency and inverse document frequency.

- *Cosine similarity:* Cosine similarity is one of the metric to measure the text-similarity between two documents irrespective of their size in Natural language Processing. A word is represented into a vector

form. The text documents are represented in n-dimensional vector space. Mathematically, Cosine similarity metric measures the cosine of the angle between two n-dimensional vectors projected in a multi-dimensional space. The Cosine similarity of two documents will range from 0 to 1. If the Cosine similarity score is 1, it means two vectors have the same orientation. The value closer to 0 indicates that the two documents have less similarity.

$$Cosine\ Similarity = \frac{A.B}{\|A\|.\|B\|}$$

- *Programming of TF-IDF Calculation:* To calculate cosine similarity using cosine similarity we use a python library named scikit-learn (sklearn). Using CountVectorizer module we vectorise our documents. Then we calculate the term-frequency, normalize term-frequency and inverse document frequency.
  We import cosine_similarity module from scikit-learn library to calculate cosine similarity.

3.2 Tools used

To build our project we choose Python as our programming language and Jupyter notebook as IDE. There are several reasons for choosing python. Python is a widely used language now a days. Some of the advantages of using Python are listed below:

- Has all the libraries required for our project.
- Easy to use , Fewer Line Code
- Python is simple, approachable, versatile and complete.
- Object Oriented

Some of the advantages of using Jupyter notebook are listed below:

- Language Independent
- Data Visualization
- Documenting code samples is easy.
- Easy Caching In Built-In Cell

CHAPTER 4
RESULT ANALYSIS

In this chapter we will discuss about our result which we obtained from text similarity checker. We will also discuss how our program behaves with real world data.

4.1 Result Analysis:

Our main objective was to compare 2 documents and check how much they are similar to each other. To do that we use cosine similarity algorithm. If cosine similarity of the two documents is near to 1, then both the documents are similar. On the other hand if cosine similarity of both the documents is near 0, then both the documents are dissimilar. We extend our project to compare n number of documents. We take the inputs from the user and display their cosine similarity in graphical form.

Out[127]:

	doc_1	doc_2	doc_3	query
doc_1	1.000000	0.393119	0.367059	0.389667
doc_2	0.393119	1.000000	0.345401	0.673160
doc_3	0.367059	0.345401	1.000000	0.293531
doc_4	0.389667	0.673160	0.293531	1.000000

Figure 4.0 graphical representation of cosine similarity

Above figure shows the graphical representation of cosine similarity. From the figure it is clear that doc_1 is perfectly similar with doc_1, so their cosine similarity is 1. On the other hand doc_1 and doc_2 is 39.3119% similar.

4.2 Significance of the result obtained:

We are happy with the obtained result, as we have successfully able to compare 2 or multiple files using our text similarity checker. It displays the similarities in a tabular form. From the result table we can easily tell how the documents are similar with each other.

CHAPTER 5
CONCLUSION AND FUTURE SCOPE OF WORK

5.1 Brief summary of the work:

We have seen that using text similarity checker we have successfully compute the similarity between 2 documents. Our objective was to build (Code) a program to compare 2 text documents to check how they are similar to each other. We used cosine similarity algorithm to complete our project. Cosine similarity algorithm works on the principle - ‘dot product of 2 similar vectors is 1’.So if cosine similarity of 2 given documents is 1, then we consider both the documents as similar. Our work methodology involves 3 steps mainly preparing of data, pre-processing and calculation of cosine similarity. In the data preparation, we prepare required data from raw data by cleaning the data, format the data etc. Pre-processing of data involves converting to lowercase, removing punctuation, removing stop words, stemming etc. To calculate cosine similarity we use a python library called sklearn.

5.2 Conclusions:

Past two month was a new experience for us as we are dealing with a project for the first time. We have studied a lots of new things, searches lots of queries, stuck on problem for many days then found the solution, all are new experience for us. This mini-project will surely help us to handle the pressure of final year project. We have studied various research paper and adopt cosine similarity as our algorithm and then using various python libraries we build a program which tells us how two given documents is similar with each other.

5.3 Future scope of work:

Our text similarity checker is doing well for .txt input files and it gave us desired output. But in the real world data set there are various types of data format like docx, pdf, ppt etc. Our text similarity checker is failed to give output if we provide input file other than .txt file. So in future we are looking forward to add this feature to work with any kind of data input. Also we want to compare 2 html page just by their URL. As we have seen our text similarity checker is just a terminal/IDE program, so we are looking forward to work with python GUI to give it a user friendly look. In future we are also looking forward to add some of machine learning to our project to compare diagrams as it is excluded for now. Then after it will become a complete project to use in any organization to check plagiarism. For now we are happy with our project as we have discovered a lots of new text similarity checking algorithms also completing a whole project gives lots of satisfaction.

REFERENCES

Journal / Conference Papers

[1] The Implementation of Cosine Similarity to Calculate Text Relevance between Two Documents.
--BY: D.Gunawan , C A Sembiring , M A Budiman Department of Information Technology, Universities Sumatera Utara.

[2] Corpus-based and Knowledge-based Measures of Text Semantic Similarity.
--BY: Rada Mihalcea and Courtney Corley Department of Computer Science University of North Texas

[3] Comparing sets of patterns with the Jaccard index Sam Fletcher.
--BY: Md Zahidul Islam School of Computing and Mathematics, Charles Sturt University, Bathurst, Australia.

Web

[1] Text Similarities: Estimate the degree of similarity between two texts
https://medium.com/@adriensieg/text-similarities-da019229c894