Abstract

Plagiarism is one of the most serious crimes in academia and research fields. In this modern era, where access to information has become much easier, the act of plagiarism is rapidly increasing. This paper aligns on external plagiarism detection method, where the source documents is available against which the suspicious documents are compared. Primary focus is to detect intelligent plagiarism cases where semantics and linguistic variations play an important role. The paper explores the different preprocessing methods based on Natural Language Processing (NLP) techniques. It further explores Cosine similarity measures for document comparisons.

Plagiarism in free text has become a common occurrence due to the wide availability of voluminous information resources. Automatic plagiarism detection systems aim to identify plagiarized content present in large repositories. This task is rendered difficult by the use of sophisticated plagiarism techniques such as paraphrasing and summarization, which mask the occurrence of plagiarism. In this work, a semantic based plagiarism detection technique has been developed to tackle cases of plagiarism. A cosine similarity based recognition system, which works by extracting lexical, syntactic, and semantic features from input text has been used. Both sentence-level and passage-level approaches have been investigated.

The project entitled **"Plagiarism detection in text documents"** proposed in this paper deals with detection of intelligent plagiarism cases using an extrinsic plagiarism detection system. The system uses different pre-processing methods based on NLP techniques and considers Cosine similarity measures for document comparisons.

Contents

Chapter 1

INTRODUTION

To plagiarize is to steal and pass off the ideas or words of another as one's own: use a created production without crediting the source . Plagiarism is defined as appropriating someone else's words or ideas without acknowledgment. It is defined in dictionaries as the "wrongful appropriation," "close imitation," or "purloining and publication of another author's language, thoughts, ideas, or expressions and the representation of them as one's own original work"

There are different types of text plagiarism, which are broadly categorized as literal and intelligent plagiarism. In the former, the plagiarists do not spend much time in hiding the academic crime they committed. For example, they simply copy and paste the text from the Internet . In intelligent plagiarism, the plagiarists try to betray readers by changing the contributions of others in an intelligent way and make it appear as their own work. Intelligent plagiarists try to hide, obfuscate, and change the original work in various intelligent ways, including text manipulation, translation, and idea adoption. Text manipulation is related to the switching of the words in the original text and making a fake one. In idea adoption the whole idea from a text is copied. Thus intelligent plagiarism cases are more complex to detect and hence challenging to work with.

In plagiarism detection there are mainly two methods, extrinsic/ external plagiarism detection and intrinsic/ internal plagiarism detection. Extrinsic compares a suspicious document with a reference collection, which is a set of documents assumed to be genuine . Intrinsic solely analyzes the text to be evaluated without performing comparisons to external documents. This approach aims to recognize changes in the unique writing style of an author as an indicator for potential plagiarism.

The work proposed in this paper deals with detection of intelligent plagiarism cases using an extrinsic plagiarism detection system. The system uses different pre-processing methods based on NLP techniques and considers Cosine similarity measures for document comparisons.

Chapter 2

PROOF OF CONCEPT

The project called “Detection of Plagiarism in Text Documents” is based on two papers. These papers has used different technologies for implementing this. Deepa Gupta ,Vani K,Charan Kamal Singh [1] in their paper'' Using Natural Language Processing Techniques and Fuzzy-Semantic Similarity for Automatic External Plagiarism Detection " presents the work proposed in this paper deals with detection of intelligent plagiarism cases using an extrinsic plagiarism detection system. The system uses different pre-processing methods based on NLP techniques and considers fuzzysemantic similarity measures for document comparisons.All the approaches available in external plagiarism detection mainly concentrate on improving efficiency of detection system. Most of the approaches present today follow a common methodology, only a few deviates. The general approach includes the following steps: pre-processing, candidate document selection, document comparisons, passage boundary detection and evaluation.

Waqar Ali,Tanveer ahmed [2] in their paper describes a novel approach for "Detection of Plagiarism in Urdu Text Documents" proposes an approach based on a distance measuring method, structural alignment algorithm, and vector space model. The system performance is evaluated using machine learning classifiers i.e. Support Vector Machine and Naïve Bayes. The experimental results demon-

strated that performance of the proposed method is improved as compared to other existing model i.e. simple Jaccard measure.

Also referred website [3] "www.machinelearningplus.com/nlp/cosine-similarity/" for correcting the errors and more documentation.

2.1 OBJECTIVES

The project “Plagiarism detection in Text Documents” have tried to implement concepts similar in the above papers. It used the methods similar these papers to run the project. The objectives of the project “plagiarism detection of in Text Documents” are as follows.In this project the input to the system is a reference document and a suspecious document which is compared against. We are needed to implement the following aspects:

- 1. Given two documents, finding out if there are any similarities with other document that it is compared with.
- 2. Text is preprocessed.
- 3. converting text or to vector.
- 4. finding cosine similarity.
- 5. finding matching score in percentage.

Chapter 3

IMPLEMENTATION

This project aligns on external plagiarism detection method, where the source document is available against which the suspicious document are compared. Primary focus is to detect intelligent plagiarism cases where semantics and linguistic variations play an important role. The project explores the different preprocessing methods based on Natural Language Processing (NLP) techniques. It further explores cosine similarity measures for document comparisons. The project is implemented using Python3.7. Python is a widely used high-level programming language for general-purpose programming.Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python interpreters are available for many operating systems.

Intelligent plagiarists try to hide, obfuscate, and change the original work in various intelligent ways, including text manipulation, translation, and idea adoption. Text manipulation is related to the switching of the words in the original text and making a fake one. In idea adoption the whole idea from a text is copied. Thus intelligent plagiarism cases are more complex to detect and hence challenging to work with.

The basic approach in proposed system, since the focus is on detection of intelligent plagiarism,It uses sentence based comparisons instead of N-gram comparison, which is the commonly used approach.

Preprocessing of the given text files is the first module in this project. To preprocess the text simply means to bring the text into a form that is predictable and analyzable for the task. A task here is a combination of approach and domain .In natural language processing the preprocessing of the text data is done using various methods.Raw data contain numerical value, punctuation,special character etc as shown in Figure 1 and Figure 2 . These value can hamper the performance of model so before applying any text featurization first we need to convert raw data into meaningful data which is also called as text preprocessing . This can be done by following ways:
Step 1 : Data Preprocessing

- Tokenization-convert sentences to words
- Removing unnecessary- punctuation, tags
- Removing stop words-frequent words such as ”the”, ”is”, etc. that do not have specific semantic
- Stemming-words are reduced to a root by removing inflection through dropping unnecessary characters, usually a suffix.
- Lemmatization-Another approach to remove inflection by determining the part of speech and utilizing detailed database of the language.

Stemming and lemmatization help reduce words like ‘studies’, ‘studying’ to a common base form or root word ‘study’.Not all the steps are mandatory and is based on the application use case. For Spam Filtering we may follow all the above steps but may not for language translation problem. We can use python to do many text preprocessing operations.

- NLTK - The Natural Language ToolKit is one of the best-known and most-used NLP libraries, useful for all sorts of tasks from tokenization, stemming, tagging, parsing, and beyond.

After completeing the text preprocessing the next module in Plagiarism detection project is to convert string features into numerical features(convert text to vector).Machine learning algorithms operate on a numeric feature space, expecting input as a two-dimensional array where rows are instances and columns are features. In order to perform machine learning on text, we need to transform our documents into vector representations such that we can apply numeric machine learning. This process is called feature extraction or more simply, vectorization, and is an essential first step toward language-aware analysis.
Representing documents numerically gives us the ability to perform meaningful analytics and also creates the instances on which machine learning algorithms operate. In text analysis, instances are entire documents or utterances, which can vary in length from quotes or tweets to entire books, but whose vectors are always of a uniform length. Each property of the vector representation is a feature.
The simplest encoding of semantic space is the bag-of-words model, whose primary insight is that meaning and similarity are encoded in vocabulary.The idea behind this method is straightforward, though very powerful. First, we define a fixed length vector where each entry corresponds to a word in our pre-defined dictionary of words. The size of the vector equals the size of the dictionary. Then, for representing a text using this vector, we count how many times each word of our dictionary appears in the text and we put this number in the corresponding vector entry.

The next stage in the project is to calculate the similarity between the source document and the suspecious document.For this the cosine similarity algorithm is used. 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.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.

In the perspective of the final result, the result is viewed as amount of plagiarism detected in percentage.

3.1 System architecture

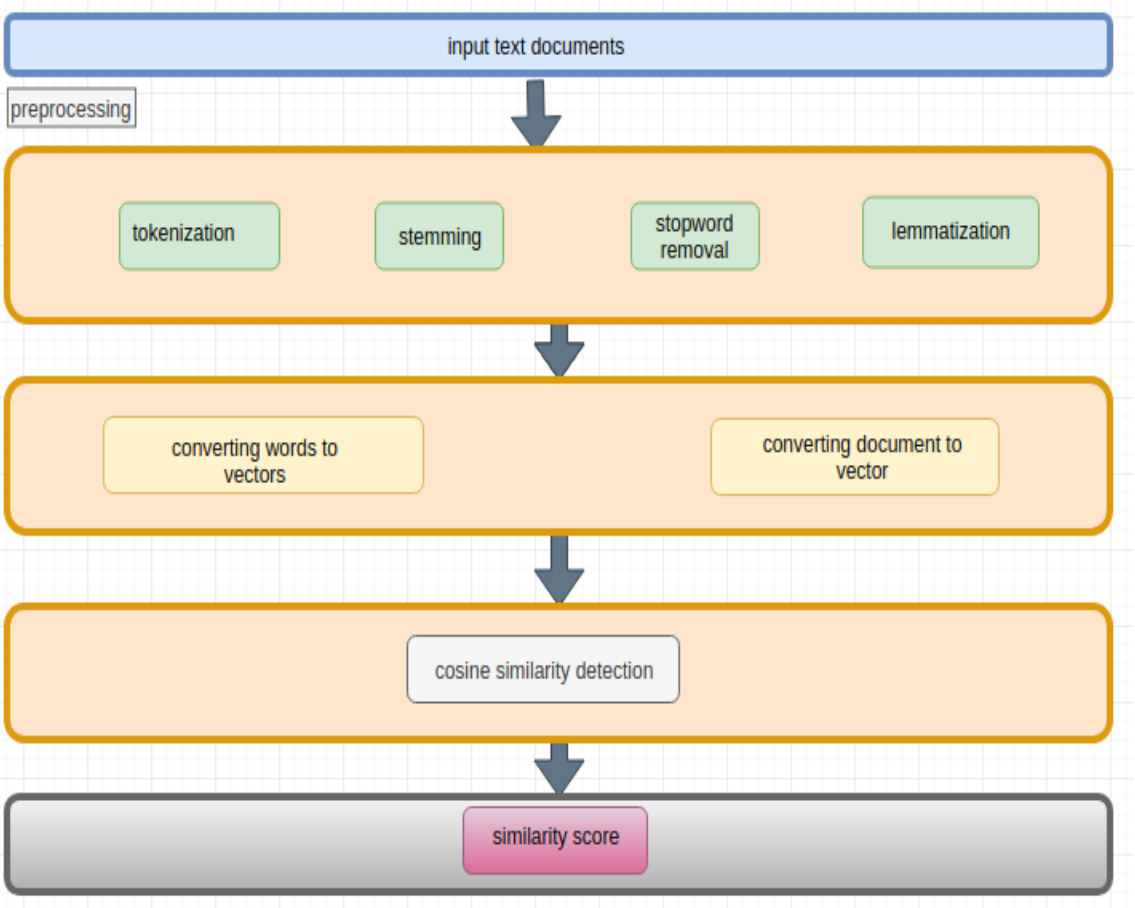


Figure 3.1: System Architecture

The figure 3.1 shows the overall working of the system.The user interact through the user interface.The architecture describes the operation of the entire system .The input to the system is Text documents, Which is being preprocessed using various NLP techniques including Tokenization, stop words removal,stemming, Lemmatization. This preprocessed text files are converted to the corresponding vector forms called vectorization.These corresponding vectors are being used to find the cosine similarity between the text files.The result is a similarity score

between documents represented in percentage.

Chapter 4

ALGORITHMS USED

4.1 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.

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.

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.

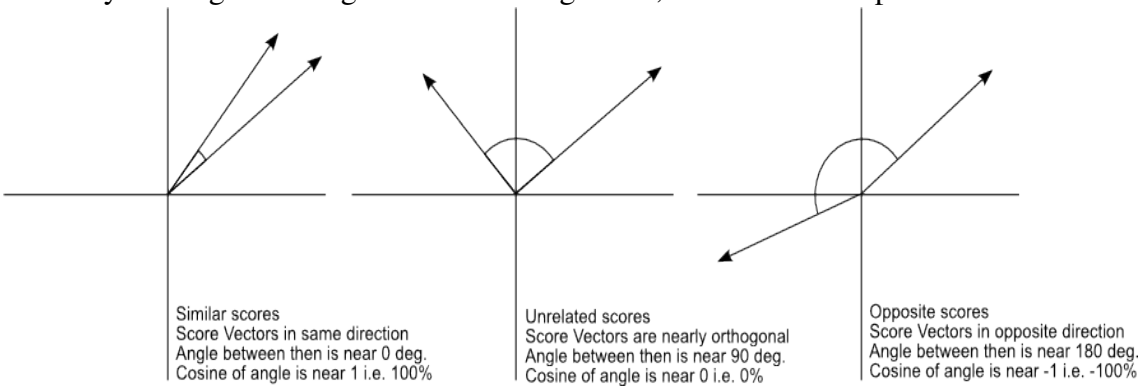
The cosine similarity between two vectors (or two documents on the Vector Space) is a measure that calculates the cosine of the angle between them. This metric is a measurement of orientation and not magnitude, it can be seen as a comparison between documents on a normalized space because we're not taking into the consideration only the magnitude of each word count (tf-idf) of each document, but the angle between the documents. What we have to do to build the cosine similarity equation is to solve the equation of the dot product for the $\cos\theta$:

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

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.

Putting all those ideas together Let a and b be vectors. Similarity formulation of these two vectors can be generalized as mentioned below. cosine similarity = (a1b1 + a2b2 + . . . + anbn) / ($\sqrt{}$(∑ (i = 1 to n) ai2) $\sqrt{}$(∑ (i = 1 to n) bi2)) or we can apply vectorization to find cosine similarity cosine similarity = (aT b) / ($\sqrt{}$(aT a) $\sqrt{}$(bT b)) In this way, similar vectors will produce high results.

Cosine Similarity will generate a metric that says how related are two documents by looking at the angle instead of magnitude, like in the examples below:



4.2 Porter Stemming Algorithm

The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval systems.

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words "chocolates", "chocolatey", "choco" to the root word, "chocolate" and "retrieval", "retrieved", "retrieves" reduce to the stem "retrieve".

Chapter 5

MODULES

5.1 Modules

5.1.1 Preprocessing

Text Preprocessing is the first module in this project.The process of converting data to something a computer can understand is referred to as pre-processing.Text Processing is one of the most common task in many ML applications.To deal with huge amount of text to perform classification or translation and involves a lot of work on the back end. Transforming text into something an algorithm can digest is a complicated process. Preprocessing algorithms is nothing but data conditioning algorithm which provide data for feature extraction process.In this project, the steps involved in text processing are:

Step 1 : Data Preprocessing

• Tokenization-convert sentences to words

In Python tokenization basically refers to splitting up a larger body of text into smaller lines, words or even creating words for a non-English language. The various tokenization functions in-built into the nltk module itself and can be used in programs.

• Removing unnecessary punctuation, tags.

• Removing stop words-frequent words such as ”the”, ”is”, etc. that do not have specific semantic.

One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words.

What are Stop words?

Stop Words: A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

We would not want these words taking up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to be stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. we can find them in the nltk$_{d}atadirectory$.

• Stemming-words are reduced to a root by removing inflection through dropping unnecessary characters, usually a suffix.

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”.Here in this project "porter stemmer" in python NLTK library used for stemming.

• Lemmatization-Another approach to remove inflection by determining the part of speech and utilizing detailed database of the language.

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meaning to one word.

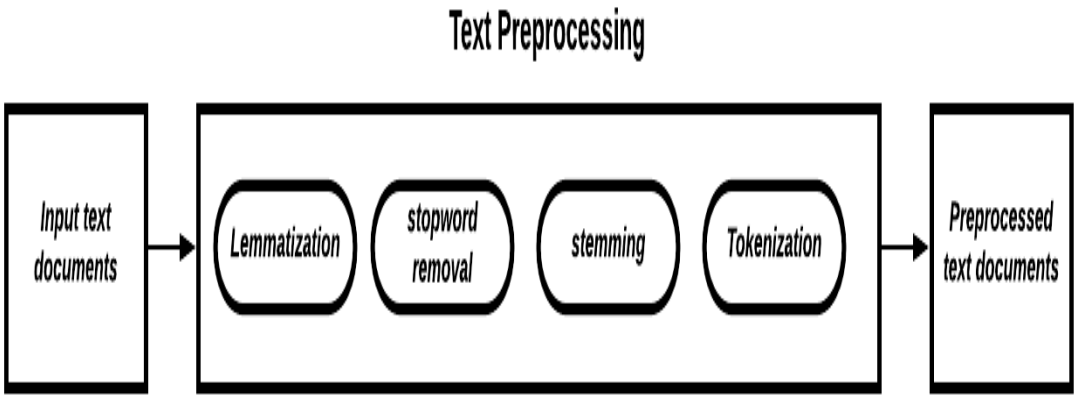
Text preprocessing includes both Stemming as well as Lemmatization. Many times people find these two terms confusing. Some treat these two as same. Actually, lemmatization is preferred over Stemming because lemmatization does

morphological analysis of the words. Lemmatization in the project is done with python NLTK library.
Step 2: Feature Extraction

In text processing, words of the text represent discrete, categorical features. How do we encode such data in a way which is ready to be used by the algorithms? The mapping from textual data to real valued vectors is called feature extraction. One of the simplest techniques to numerically represent text is Bag of Words..

Bag of Words (BOW): We make the list of unique words in the text corpus called vocabulary. Then we can represent each sentence or document as a vector with each word represented as 1 for present and 0 for absent from the vocabulary. Another representation can be count the number of times each word appears in a document.

Block Diagram



Algorithm

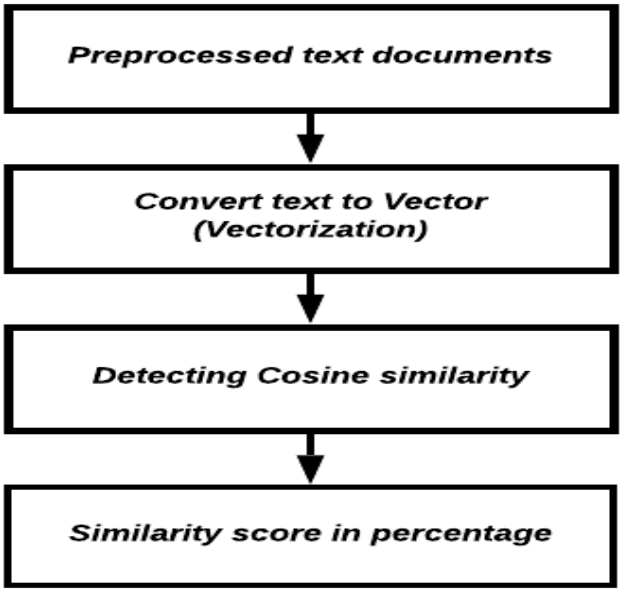
Algorithm 1 Detection of Plagiarism in Text Documents - Preprocessing
Input : Upload text documents
Output : Normalised text

Step1 : Start
Step2 : Read the text documents.
Step3 : Text documents are normalised using various NLP preprocessing strategies.
Step4 : Convert the raw Texts to normalised text.
Step5 : stop

5.1.2 Plagiarism detection using cosine similarity

A reference document and a suspecious document is uploaded. The documents are preprocessed by various NLP preprocessing strategies and the features are extracted . Finally the amount of plagiarism detected in the text document is genuinely displayed.

Block Diagram



Algorithm

Algorithm 2 Similarity calculation
Input : A reference text document and a suspecious text document is uploaded.
Output : How much plagiarised is the suspecious text document from the reference text document

Step1 : Start
Step2 : upload the Text files.
Step3 : Preprocess the text files by NLP techniques.
Step4 : Extract the features and convert text to vector(vectorisation).
Step5 : Similarity calculation using the cosine similarity
Step6 : Amount of plagiarism is genuninely displayed.
Step7 : stop

5.1.3 GUI development

A GUI for uploading the text documents for similarity calculation is build using python PyQt5.PyQt5 is a module that can be used to create graphical user interfaces (GUI).PyQt is a library that lets us use the Qt GUI framework from Python. Qt itself is written in C++. By using it from Python, we can build applications much more quickly while not sacrificing much of the speed of C++.PyQt5 refers to the most recent version 5 of Qt.

Chapter 6

ISSUES FACED AND REMEDIES TAKEN

- **Issue 1 :**Initially the project is started using dataset of english novels.But later it makes inconvenience to the system that it takes too long time for the system to respond for the events.
- **Remedy :** Changed the idea from dataset to comparision of two or more documents.
- **Issue 2 :** There was as an issue while running the code , showing a NameError.
- **Remedy :** Installed required libraries.
- **Issue 3:** It was difficult to find out an exact result while jaccard similarity is used.
- **Remedy :** Changed the idea to cosine similarity measure.

Chapter 7

RESULT ANALYSIS

The semantics-based method, is considered as one of the important method for plagiarism detection, focuses on detecting the similarities between documents by using the vector space model. It also can calculate and count the redundancy of the word in the document, and then they use the fingerprints for each document for matching it with fingerprints in other documents and find out the similarity. The semantic-based method is suitable for non partial plagiarism as mentioned before use the whole document and use vector space to match between the documents, but if the document has been partially plagiarized it cannot achieve good results, and this is considered as one of the limitations of this method, because it's difficult to fix the place of copied text in the original document.

7.1 Result

The project "Plagiarism detection in text documents" provides a fine grained result by using the sentence-by-sentence comparison.The initial stage of the project is about the preprocessing of the text documents.Various NLP stratigies commonly used in text preprocessing and normalisation is used before hand in the project.Since the project overlook on the semantics-based approach , In order to perform machine learning on text, we need to transform our documents into vector

representations such that we can apply numeric machine learning. This process is called feature extraction or more simply, vectorization, and is an essential first step toward language-aware analysis.

Representing documents numerically gives us the ability to perform meaningful analytics and also creates the instances on which machine learning algorithms operate. In text analysis, instances are entire documents or utterances, which can vary in length from quotes or tweets to entire books, but whose vectors are always of a uniform length. Each property of the vector representation is a feature. For text, features represent attributes and properties of documents—including its content as well as meta attributes, such as document length, author, source, and publication date.

The simplest encoding of semantic space is the bag-of-words model, whose primary insight is that meaning and similarity are encoded in vocabulary.

Final result of the works are calculated by finding the cosine similarity between the documents.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 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.Finally the result of the project calculated by the sentence-by-sentence comparision depicted in percentage format proves to be a result from an fine grained system.

Chapter 8

CONCLUSION AND FUTURE SCOPE

8.1 Conclusion

Intelligent plagiarism detection using cosine similarity on external plagiarism detection is explored in this paper. It uses the different pre-processing methods based on NLP techniques. Here mainly lemmatization, stop word removal and Vectorization are explored.It also throws light on how similarity calculation can be improved using cosine similarity measures. It introduces an improved semantic measure that can provide a significant improvement in the efficiency of the system compared to the other methods.The experimental results in terms (discussed in Sections) show that compared to the other methods, this system provides results accrately and efficiently. According to the analysis and discussions done, it can be observed that BOW method integrated with improved semantic similarity measure surpass the other methods in terms of efficiency.

8.2 Future Scope

In future, more efficient NLP techniques can be used to improve the performance of detection system. Results can be improved by using efficient passage boundary detection conditions. Evaluation can be performed using multiple files for proper analysis and comparisons. Advanced soft computing methods and optimization techniques can be used for enhancing the system performance.

Chapter 9

APPENDIX

9.1 Source Code

Plagiarism.py

```
import math
import re
import sys
import time
from collections import Counter

import nltk
import numpy as np
from PyQt5 import QtWidgets, QtGui, QtCore
from PyQt5.QtCore import QSize
from PyQt5.QtGui import QImage, QPalette, QBrush
from PyQt5.QtWidgets import QMainWindow, QPushButton, QFileDialog,
    QLineEdit, QPlainTextEdit, QLabel
from nltk import sent_tokenize, word_tokenize, PorterStemmer
from nltk.corpus import wordnet, stopwords
from nltk.stem import WordNetLemmatizer
```

```
from nltk.tag import pos_tag
from nltk.tokenize import PunktSentenceTokenizer

nltk.download('stopwords')


class MainWindow(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)
        self.font = QtGui.QFont("Times", 12)
        self.font2 = QtGui.QFont("Times", 20, QtGui.QFont.Bold)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("icon.png"),
            QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.setWindowIcon(icon)
        self.setWindowTitle("Document Plagarism Checker")
        self.setFixedSize(900, 600)

        oImage = QImage("plag.png")
        sImage = oImage.scaled(QSize(900, 600))
        palette = QPalette()
        palette.setBrush(10, QBrush(sImage))
        self.setPalette(palette)

        self.startButton = QPushButton('Check Matching', self)
        self.startButton.clicked.connect(self.startService)
        self.startButton.setCursor(QtGui.QCursor
        (QtCore.Qt.PointingHandCursor))
        self.startButton.move(180, 250)
        self.startButton.resize(150, 30)

        self.sample1_in = QLineEdit(self)
```

```
        self.sample1_in.move(20, 70)
        self.sample1_in.resize(400, 30)
        self.sample2_in = QLineEdit(self)
        self.sample2_in.move(20, 150)
        self.sample2_in.resize(400, 30)
        self.sample1_in.setDisabled(True)
        self.sample2_in.setDisabled(True)

        self.stopButton = QPushButton('Reset', self)
        self.stopButton.clicked.connect(self.reset)
        self.stopButton.setCursor(QtGui.QCursor
        (QtCore.Qt.PointingHandCursor))
        self.stopButton.move(700, 20)
        self.stopButton.resize(150, 30)

        self.chooseButton1 = QPushButton('Choose File 1', self)
        self.chooseButton1.clicked.connect
        (self.OpenFileNamesDialog1)
        self.chooseButton1.resize(100, 32)
        self.chooseButton1.move(450, 69)

        self.chooseButton2 = QPushButton('Choose File 2', self)
        self.chooseButton2.clicked.connect
        (self.OpenFileNamesDialog2)
        self.chooseButton2.resize(100, 32)
        self.chooseButton2.move(450, 150)

    def startService(self):
        input_text = open(self.sample1_in.text(), 'r')
        input_text1 = open(self.sample2_in.text(), 'r')
```

```
text = input_text.read()
text1 = input_text1.read()

input_text.close()
input_text1.close()

self.sentences = sent_tokenize(text)
self.sentences1 = sent_tokenize(text1)

self.N = len(self.sentences)
self.N1 = len(self.sentences1)

self.ps = PorterStemmer()
self.lemmatizer = WordNetLemmatizer()
self.stop_words = stopwords.words('english')
self.special = ['.', ',', '\'', '"', '-', '/', '*', '+',
    '=', '!', '@', '$', '%', '^', '&', '``', '\'\'', 'We',
          'The', 'This']

mat = self.f_s_to_s(self.sentences1)
print (mat)
length = len(mat)-2


point1= sum(mat)
percent = point1*100
percent_round = round(percent,2)
percent_text = str(percent_round)+" %"
print (point1)

self.result = QLabel(self)
```

```
        self.result.setText('score')
        self.result.resize(100, 30)
        self.result.move(380, 250)
        self.result.setFont(self.font2)
        self.result.show()

        self.sample3_in = QLineEdit(self)
        self.sample3_in.move(350, 300)
        self.sample3_in.resize(120, 30)
        self.sample3_in.setText(percent_text)
        self.sample3_in.setFont(self.font)
        self.sample3_in.show()

    def get_cosine(self,vec1, vec2):
         intersection = set(vec1) & set(vec2.keys())
         numerator = sum([vec1[x] * vec2[x] for x in intersection])

         sum1 = sum([vec1[x]**2 for x in vec1.keys()])
         sum2 = sum([vec2[x]**2 for x in vec2.keys()])
         denominator = math.sqrt(sum1) * math.sqrt(sum2)

         if not denominator:
            return 0.0
         else:
            return numerator / denominator


    def text_to_vector(self,text):
        words = word_tokenize(text)
        vec=[]
        for word in words:
         if(word not in self.stop_words):
```

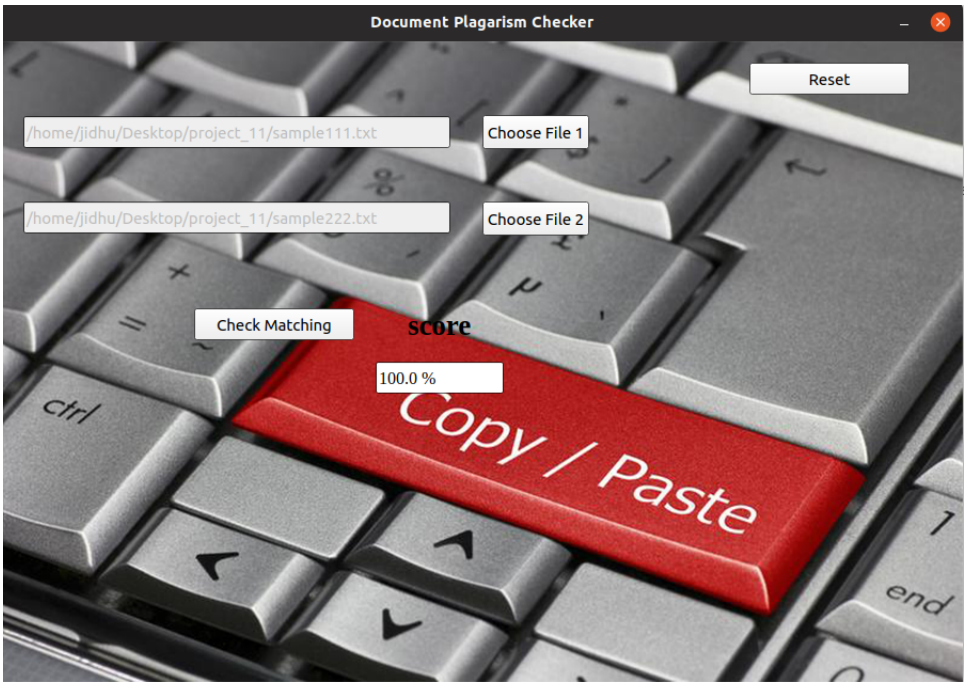
```
            if(word not in self.special):
                w=self.normalise(word);
                vec.append(w);
     #print Counter(vec)
        return Counter(vec)


    def docu_to_vector(self,sent):
        vec=[]
        for text in sent:
         words = word_tokenize(text)
         for word in words:
            if(word not in self.stop_words):
                if(word not in special):
                    w=normalise(word);
                    vec.append(w);
     #print Counter(vec)
        return Counter(vec)

    def f_s_to_s(self,sent):
        length = self.N+1
        cosine_mat=np.zeros(length)

        row=0
        for text in self.sentences:
            maxi=0
            vector1 = self.text_to_vector(text)
        for text1 in sent:
            vector2 = self.text_to_vector(text1)
            cosine = self.get_cosine(vector1, vector2)

        for text2 in sent:
```

```
            vector3 = self.text_to_vector(text2)
            cosine = self.get_cosine(vector1, vector3)

        for text3 in sent:
            vector4 = self.text_to_vector(text3)
            cosine = self.get_cosine(vector1, vector4)


        for text4 in sent:
            vector5 = self.text_to_vector(text4)
            cosine = self.get_cosine(vector1, vector5)

        for text5 in sent:
            vector6 = self.text_to_vector(text5)
            cosine = self.get_cosine(vector1, vector6)

        for text6 in sent:
            vector7 = self.text_to_vector(text6)
            cosine = self.get_cosine(vector1, vector7)

        for text7 in sent:
            vector8 = self.text_to_vector(text7)
            cosine = self.get_cosine(vector1, vector8)


        for text8 in sent:
            vector9 = self.text_to_vector(text4)
            cosine = self.get_cosine(vector1, vector9)

        for text9 in sent:
            vector10 = self.text_to_vector(text9)
            cosine = self.get_cosine(vector1, vector10)
```

```
            if(maxi<cosine):
                maxi=cosine

        cosine_mat[row]=maxi

        row+=1

        return cosine_mat



    def normalise(self,word):
        word = word.lower()
        word = self.ps.stem(word)
        return word

    def reset(self):
        self.sample1_in.setText("")
        self.sample2_in.setText("")
        self.sample3_in.setText("")
        self.sample3_in.hide()
        self.result.hide()

    def OpenFileNamesDialog1(self):
        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog
        fileName, _ = QFileDialog.getOpenFileName(self, "Select
            File","","Text Files (*.txt)", options=options)
        if fileName:
            self.sample1_in.setText(fileName)
```

```
    def OpenFileNamesDialog2(self):
        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog
        fileName, _ = QFileDialog.getOpenFileName(self, "Select
            File", " ","Text Files (*.txt)", options=options)
        if fileName:
            self.sample2_in.setText(fileName)


if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    mainWin = MainWindow()
    mainWin.show()
    sys.exit(app.exec_()
```

9.2 Screenshot



Chapter 10

REFERENCES

1. Miranda Chong, Lucia Specia and Ruslan Mitkov, " Using natural language processing for automatic plagiarism detection", 4 International Plagiarism Conference, Northrumbia University, 2010.

2. Martin Potthast, Benno Stein, Alberto Barron Cedeno and Paolo Rosso, "An evaluation framework for plagiarism detection", In Proc. of 23 International Conference on Computational Linguistics, COLING 2010,Beijing, China, 2010.

3. YuriiPalkovskii, Alexei Belov, IrynaMuzyka,"Using WordNet based semantic similarity measurement in external plagiarism detection", Notebook Papers of CLEF , 2011.

4. http://www.machinelearningplus.com/nlp/cosine-similarity/.

5. url http://stackoverflow.com/.