

Natural Language Processing (CSD 350) – Project

Plagiarism Detector

SIDDHANT MITTAL

1910110388

What is Plagiarism?

Using someone else's words or ideas and passing them off as your own by not providing credit, either deliberately or accidentally.

AIM

To make an application that uses the dataset (a corpus of documents) which is used to detect plagiarism by Containment measure technique.

Prerequisites:

1. NLTK
2. Tkinter
3. Matplotlib

1. NLTK: pip install nltk

We will also need to download nltk data using nltk.download()

nltk.download('punkt')

nltk.download('stopwords')

Ref: Installing NLTK – [NLTK 3.5 documentation](#)

2. Matplotlib: pip install matplotlib

Ref: Installation Guide – [Matplotlib 3.3.2 documentation](#)

Steps followed for building this application:

1) Modules Imported: Natural language Toolkit (NLTK)

NLTK is an open-source tool which provides various tokenizers, lemmatizers, stemmers, taggers etc.

2) Reading the original/test document and the document to check:

After importing the modules, we read the original document set and the suspicious document.

3) Performing Pre-Processing on all the documents:

In the pre-processing step:

- Words are tokenized using word tokenizer
- After tokenizing, all words are converted to lowercase letters
- Then all stop words are removed from the set of lowercase letters.

4) Calculating the no of overlapping trigrams in the two texts:

The plagiarism content between the two texts is found by calculating the Jaccard similarity coefficient,

$$J(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

S(A)- set of trigrams in document to check

S(B)- set of trigrams in original document

The **containment measure** C is a better metric for or document pairs with varied document lengths. Here, we normalize by the trigrams in the document to be checked. C is given by,

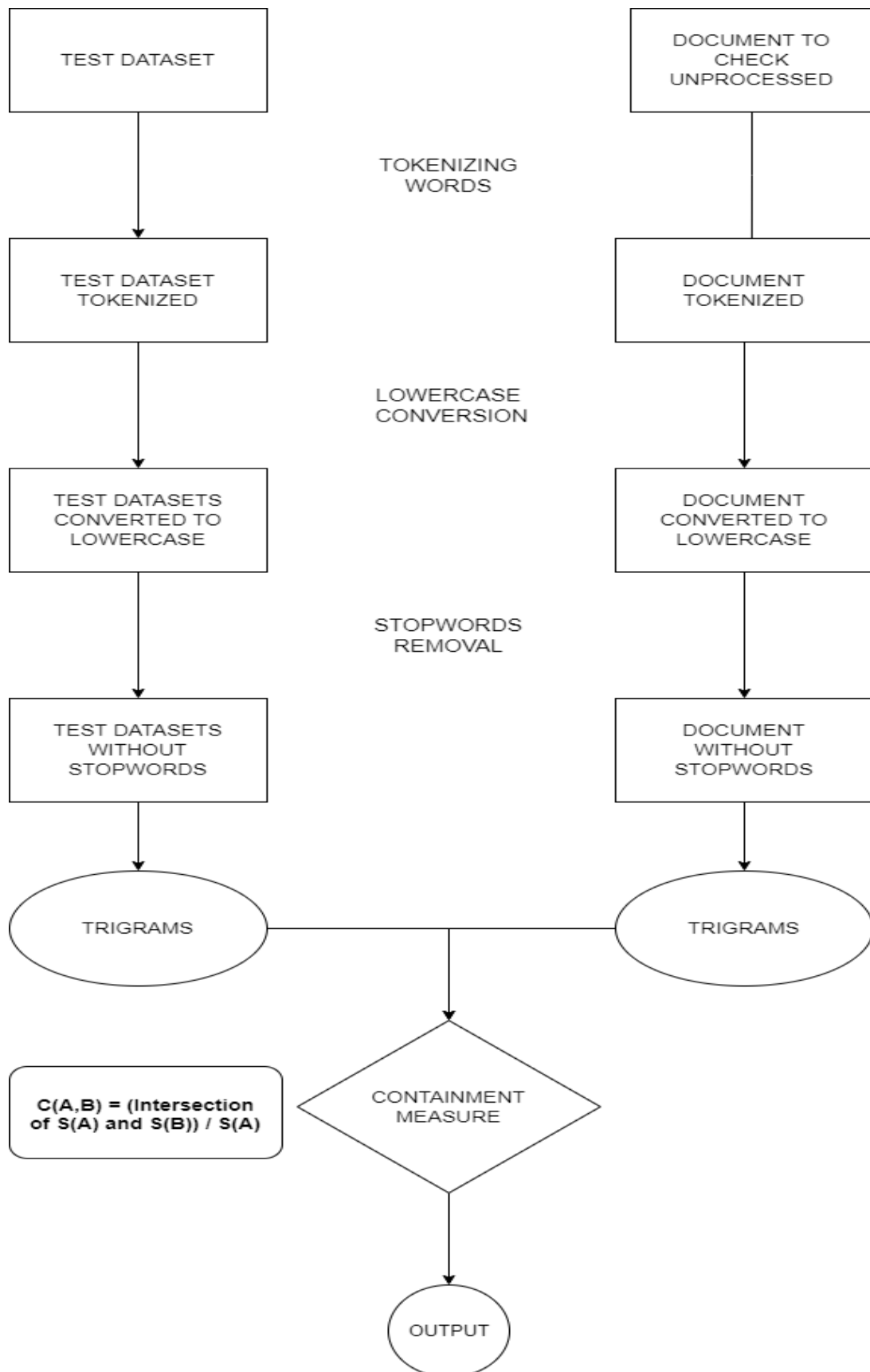
$$C(A, B) = \frac{|S(A) \cap S(B)|}{|S(A)|}$$

S(A)- set of trigrams in document to check

S(B)- set of trigrams in original document

We used containment measure to detect amount of similarity between the documents.

Architecture:



Run through of the code:

Importing all the required libraries and packages:

```
import os
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from queue import PriorityQueue
import re
import matplotlib.pyplot as plotGraph
from tkinter import *
```

Pre-processing the documents:

Opens the file and replaces the new line and special character to space and tokenizes the data.

We do the pre-processing of both the texts. Word tokenization splits the sentences into words. For example, *'I love books.'* tokenizes to *'I', 'love', 'books'* and *'.'*. We change all the uppercase alphabets to lowercase to generalize tokens across both the texts. Further, Stop-Words like *'or', 'the'* and *'in'* and punctuations are removed, as these are functional in nature and do not give any extra information about the document.

We find the number of overlapping trigrams in the two texts, i.e the number of continuous sequences of three words which are present in both texts.

```
def openFile(filename):
    f=open(filename,"r")
    orig=f.read().replace("\n"," ")
    orig = re.sub(r'^\w\s', '', orig)
    orig = re.sub(r'[0-9]+', '', orig)
    return word_tokenize(orig)
```

Converts the token into lower case, removes punctuation and stopwords using NLTK package

```
def clean(init_token):
    #lowercase
    tokens_o = [token.lower() for token in init_token]
    #stop word removal
    stop_words=set(stopwords.words('english'))
    #punctuation removal
    punctuations=['"', '.', '(', ')', ',', '?', ';', ':', '"', "'", '`']
    filtered_tokens = [w for w in tokens_o if not w in stop_words and not w in punctuations]
    return filtered_tokens
```

Takes the filtered tokens and make trigrams of continuous words

```
def makeTrig(clean_token):
    trigrams=[]
    for i in range(len(clean_token)-2):
        t=(clean_token[i],clean_token[i+1],clean_token[i+2])
        trigrams.append(t)
    return trigrams
```

Calls the above functions to pre-process the data and finally returns the list of trigrams for the particular document.

```
def Preprocess(filename):
    trig = []
    init_token = openFile(filename)
    clean_token = clean(init_token)
    trig_list = makeTrig(clean_token)
    trig = trig_list
    return trig
```

Pre-process the corpus taking all .txt files in directory and calls the PreProcess() function for each of them.

```
docList = {}
You, 20 hours ago • Initial Commit
def preProcessData():
    testDataPath = os.path.join(os.getcwd(), 'database')
    os.chdir(testDataPath)
    files = [doc for doc in os.listdir() if (doc.endswith(
        '.txt') and (doc != 'DataStore.txt' and doc != 'temp.txt'))]
    for doc in files:
        tempTrig = []
        tempTrig = Preprocess(doc)
        docList[doc] = tempTrig
```

Stores the pre-processed corpus in a text file to avoid the computational load of pre-processing every time and loads the pre-processed data.

```
def storeCorpusData():
    trg = open('DataStore.txt', "w")
    trg.write(str(docList))
    trg.close()

def loadCorpusData():
    temp = open('DataStore.txt', 'r')
    s = temp.read()
    temp.close()
    docList = eval(s)
    print('a', docList)
```

Calls the pre-processing functions

```
def PP():
    preProcessData()
    storeCorpusData()
```

Calculating the similarity and rank of the documents:

Calculate similarity between all the documents present in the dataset with the document to check, rank will contain number of trigrams matches per unit length for a given document. Document with higher priority (in the front) will have more similarity with the document being checked than a document with lower priority.

Priority queue with percentage plagiarism and doc name. Run a loop through all the documents, for each document if there is a trigram match increment the count (s in this case).

When all trigram is matched for a particular document add its value to the queue (count per unit length)

```
def CalculateRank(corpus,inp):
    rank = PriorityQueue()
    for doc in corpus:
        s = 0
        corp = corpus[doc]
        for tri in inp:
            if tri in corp:
                s+=1
        rank.put((s/len(inp)*100,doc))
    return rank
```

Creating the UI using tkinter and displaying graphs using matplotlib for showing the similarity analysis:

```
master = Tk()
master.title("Plagiarism Checker")
master.config(bg="#FADA5E")
Label(master, text="Enter file name", font=(
    "Helvetica", 12)).grid(row=0, column=1)

Label(master, text="Enter input text", font=(
    "Helvetica", 12)).grid(row=5, column=1)

master.geometry("750x780")
e1 = Entry(master)

e1.grid(row=3, column=1, pady=15)

t = Text(master, width=80, height=35)
t.grid(row=6, column=1, padx=50, pady=10)
```

Gets the data from input boxes and pre-processes it adds the file to the corpus and finally graphs the plagiarism percentage in form of bar graph:

```
def graphExecution(): # function that runs when we click the check button
    s = t.get("1.0", END)
    r = e1.get()
    r = r+'.txt'
    f = open(r, "w")
    f.write(s)
    f.close()
    inp = Preprocess(r)
    X = []
    Y = []
    doc_rank = CalculateRank(docList, inp)
    for i in range(len(docList)):
        item = doc_rank.get()
        Y.append(item[0])
        X.append(item[1])

    plotGraph.bar(X, Y, width=0.5, align='center')
    plotGraph.xlabel("Doc Name")
    plotGraph.ylabel("% Similarity")
    plotGraph.title("Similarity Analysis")
    plotGraph.xticks(rotation=90)
    plotGraph.tight_layout()
    plotGraph.show()
```

```

prep = Button(master, text='Process Data', bg='black',
               fg='white', width=24, command=PP)
prep.grid(row=8, column=1, pady=10)

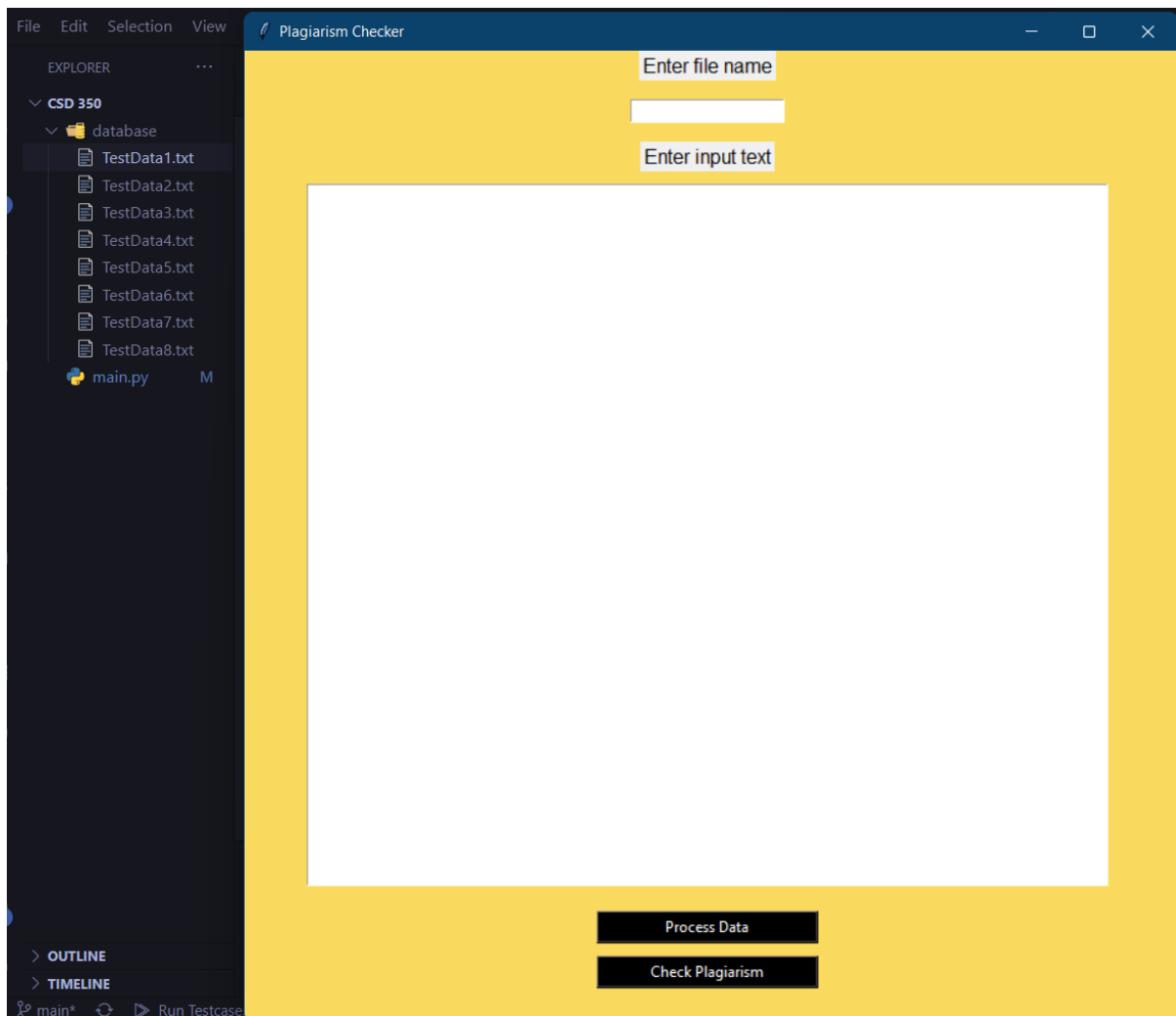
check = Button(master, text='Check Plagiarism', bg='black',
               fg='white', width=24, command=graphExecution)
check.grid(row=12, column=1)

master.mainloop()

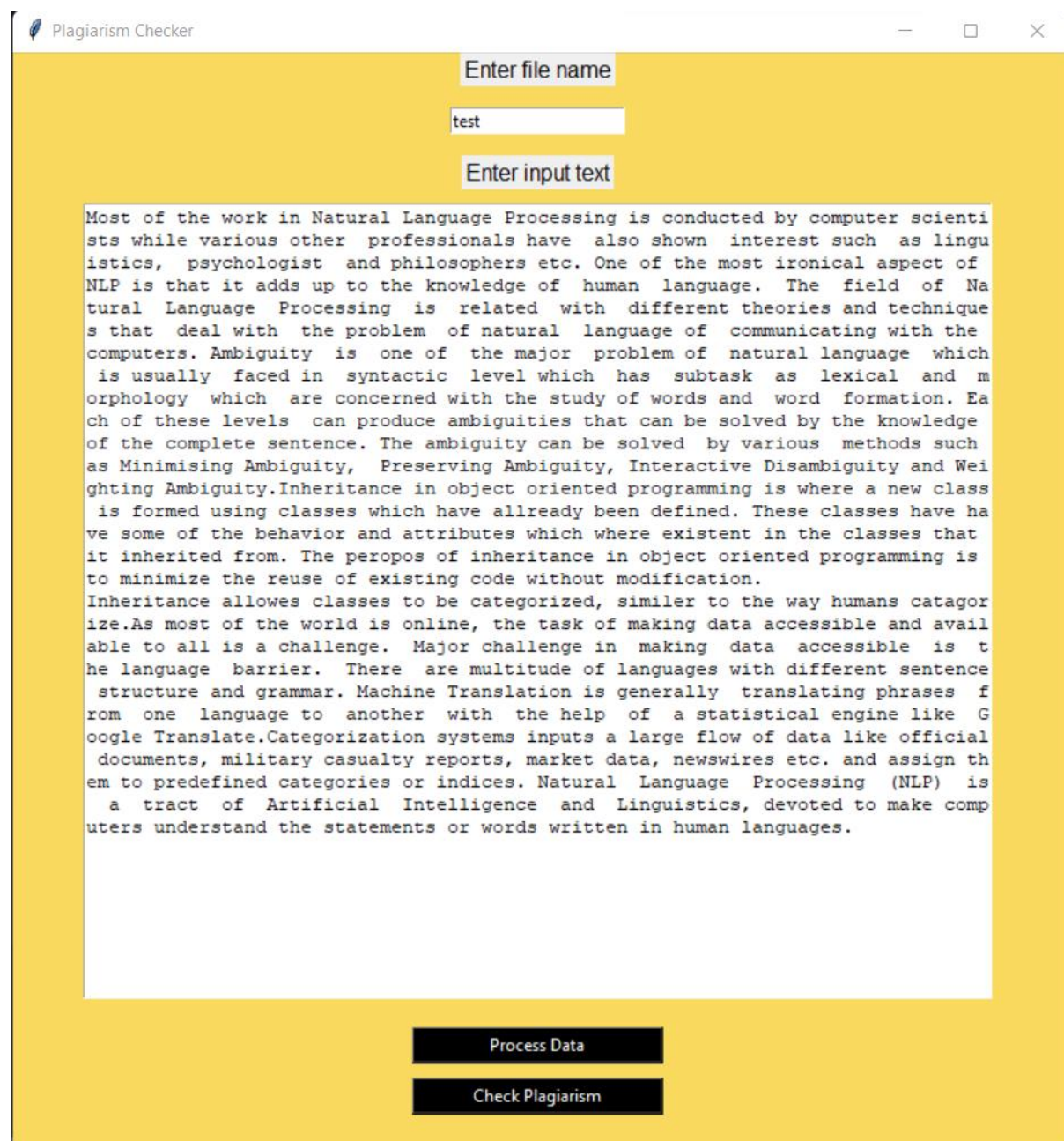
```

Screenshots of the application:

This screen shows up when the application is executed



Insert the text and give the file name



The screenshot shows a web application titled "Plagiarism Checker". It has a yellow background. At the top, there is a label "Enter file name" above a text input field containing the word "test". Below this is another label "Enter input text" above a large text area. The text area contains a paragraph about Natural Language Processing (NLP) and ambiguity. At the bottom of the interface, there are two black buttons with white text: "Process Data" and "Check Plagiarism".

Plagiarism Checker

Enter file name

test

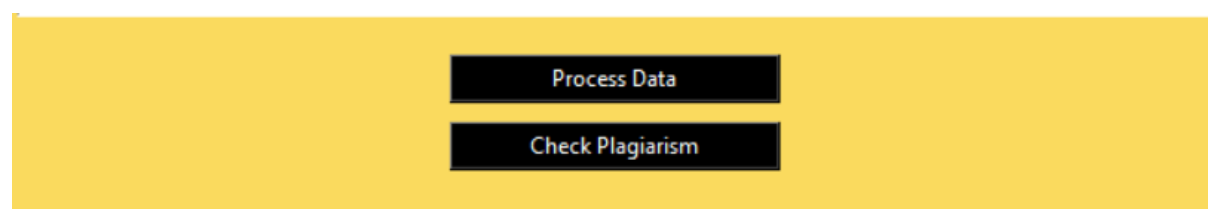
Enter input text

Most of the work in Natural Language Processing is conducted by computer scientists while various other professionals have also shown interest such as linguistics, psychologist and philosophers etc. One of the most ironical aspect of NLP is that it adds up to the knowledge of human language. The field of Natural Language Processing is related with different theories and techniques that deal with the problem of natural language of communicating with the computers. Ambiguity is one of the major problem of natural language which is usually faced in syntactic level which has subtask as lexical and morphology which are concerned with the study of words and word formation. Each of these levels can produce ambiguities that can be solved by the knowledge of the complete sentence. The ambiguity can be solved by various methods such as Minimising Ambiguity, Preserving Ambiguity, Interactive Disambiguity and Weighting Ambiguity. Inheritance in object oriented programming is where a new class is formed using classes which have already been defined. These classes have have some of the behavior and attributes which where existent in the classes that it inherited from. The peropos of inheritance in object oriented programming is to minimize the reuse of existing code without modification. Inheritance allows classes to be categorized, similar to the way humans categorize. As most of the world is online, the task of making data accessible and available to all is a challenge. Major challenge in making data accessible is the language barrier. There are multitude of languages with different sentence structure and grammar. Machine Translation is generally translating phrases from one language to another with the help of a statistical engine like Google Translate. Categorization systems inputs a large flow of data like official documents, military casualty reports, market data, newswires etc. and assign them to predefined categories or indices. Natural Language Processing (NLP) is a tract of Artificial Intelligence and Linguistics, devoted to make computers understand the statements or words written in human languages.

Process Data

Check Plagiarism

Click on the process data button to start the pre-processing

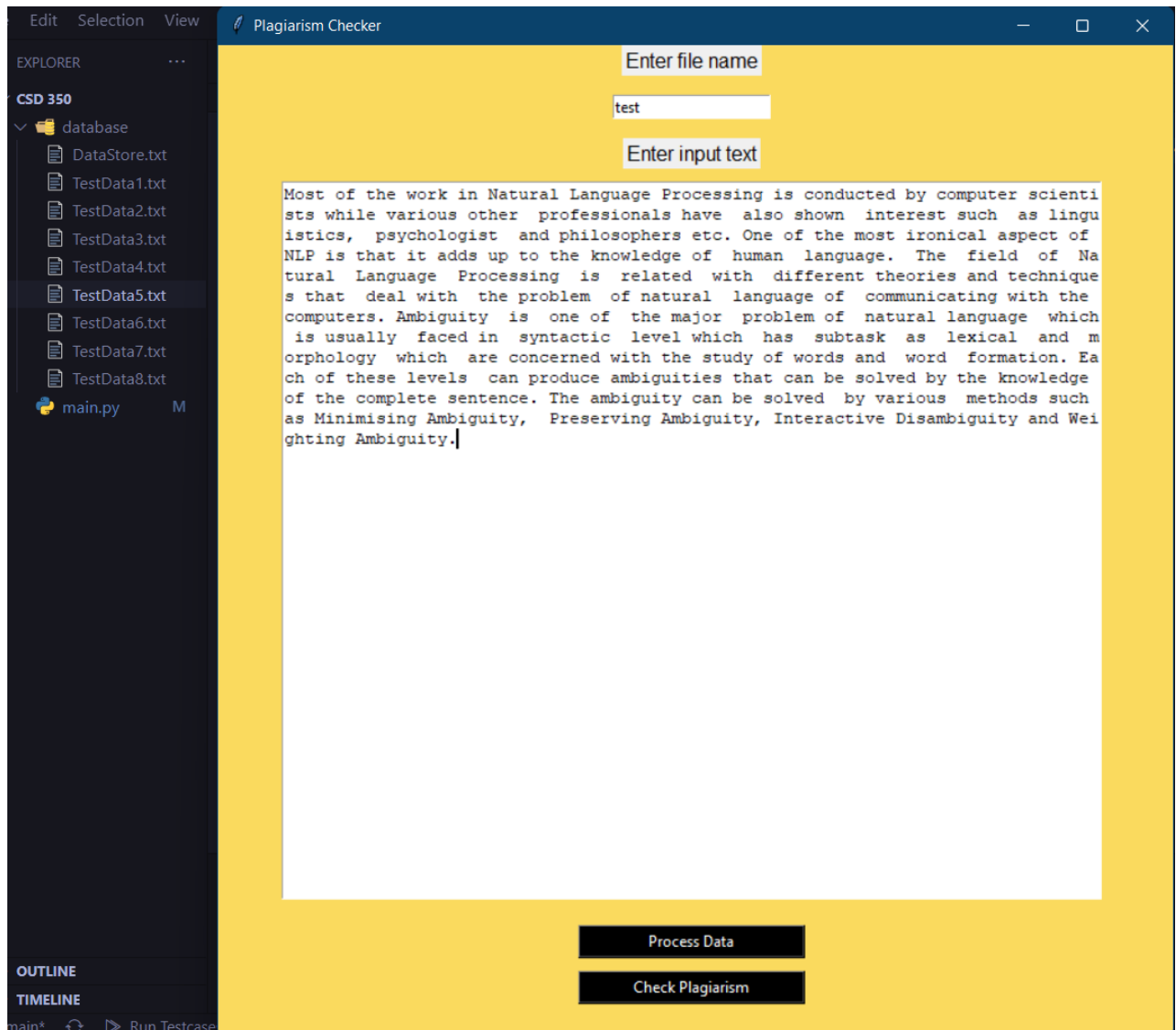


This screenshot shows the same yellow interface as before, but the text area is empty. The "Process Data" and "Check Plagiarism" buttons are still present at the bottom.

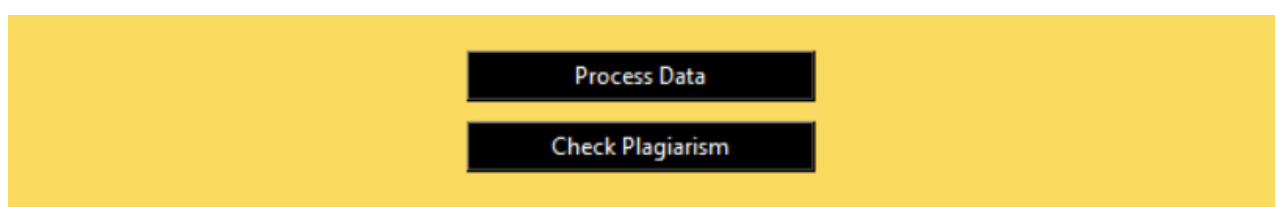
Process Data

Check Plagiarism

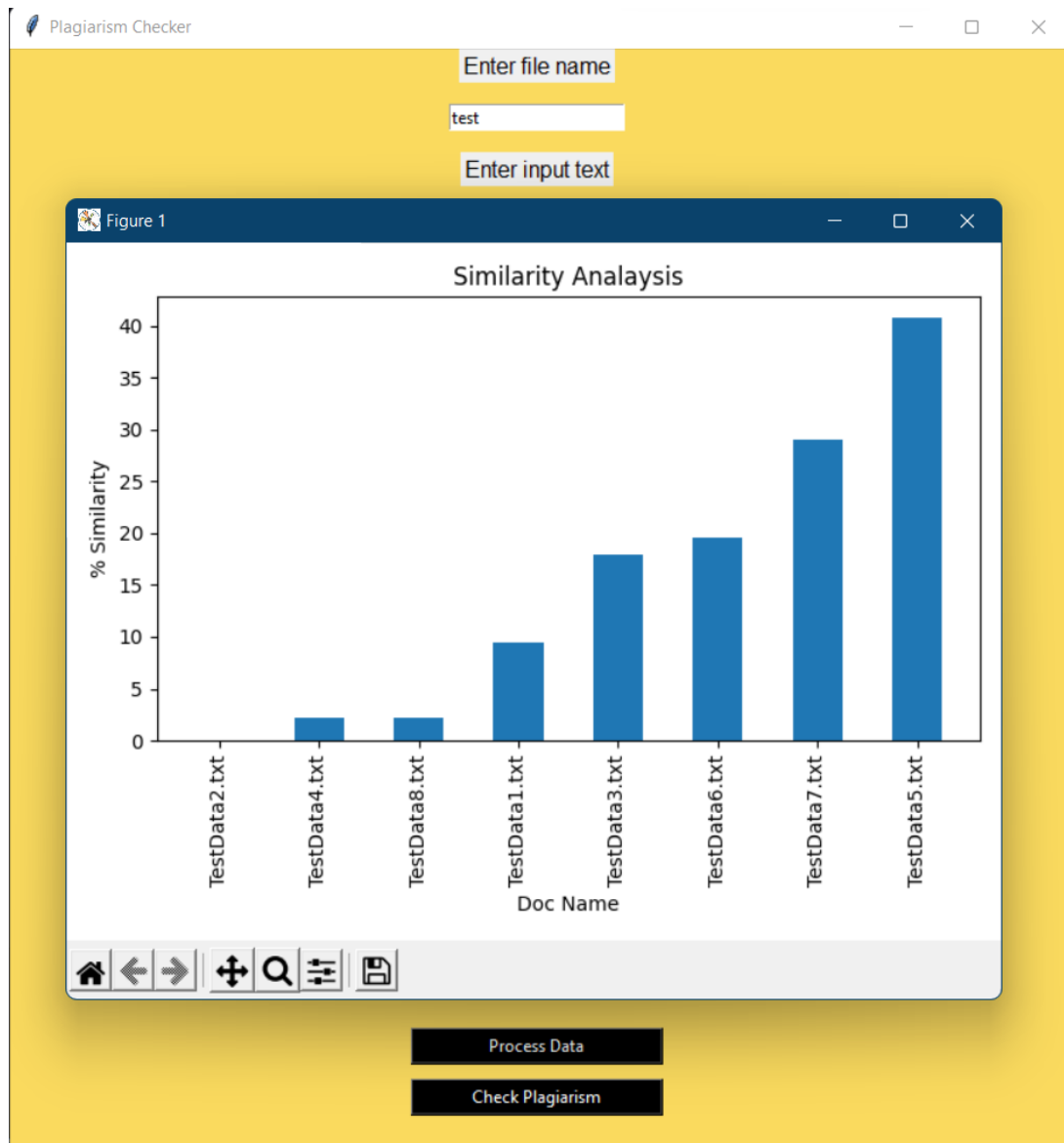
After we click the Process Data button, we see that a new file named DataStore.txt is created which contains all the trigrams of dataset file and newly created text file.



We then click on Check Plagiarism button



After clicking the check plagiarism, we see the similarity analysis using graph



We see a new file is created in the database folder with name and text we entered

