**Experiment 7**

**Aim**  
Plagiarism Detection Using NLP

**Theory**

Natural Language Processing technologies can be used to effectively to detect plagiarism in texts. NLP distance measures can be applied to detect external plagiarism, i.e., when both the original text as well as the suspicious text are available.

**Steps**

1. Before using other NLP techniques, we first apply pre-processing techniques to the text. 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.
2. Next, we read the original and the suspicious (possibly plagiarized) documents.
3. The plagiarism content between the two texts is found by calculating the Jaccard similarity coefficient.
4. Another method is finding the Longest Common Subsequence (LCS) in the texts.
5. Evaluate all the scores on the documents in the dataset. There are three types of documents: near copy, lightly revised and heavily revised.

**Code**

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

from nltk.corpus import wordnet as wn

doc1 = "When your focus is to improve employee performance, it's essential to encourage ongoing dialogue between managers and their direct reports. Some companies encourage supervisors to hold one-on-one meetings with employees as a way to facilitate two-way communication."

doc2 = "When your focus is to improve employee performance, ongoing dialogue between managers and their direct reports is essential. While performance management often involves conducting annual performance evaluations, it does involve more than just that."

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

stop\_words = set(stopwords.words("english"))

word\_tokens1 = word\_tokenize(doc1)

word\_tokens2 = word\_tokenize(doc2)

filtered\_sentence1 = [w for w in word\_tokens1 if not w.lower() in stop\_words]

filtered\_sentence2 = [w for w in word\_tokens2 if not w.lower() in stop\_words]

filtered\_sentence1 = []

for w in word\_tokens1:

    if w not in stop\_words:

        filtered\_sentence1.append(w)

filtered\_sentence2 = []

for w in word\_tokens2:

    if w not in stop\_words:

        filtered\_sentence2.append(w)

print(word\_tokens1)

print(filtered\_sentence1)

print(word\_tokens2)

print(filtered\_sentence2)

s1 = " ".join(filtered\_sentence1)

s2 = " ".join(filtered\_sentence2)

from nltk.tokenize import RegexpTokenizer

tokenizer = RegexpTokenizer(r"\w+")

s1 = tokenizer.tokenize(s1)

s2 = tokenizer.tokenize(s2)

s1 = " ".join(s1)

s2 = " ".join(s2)

jd\_sent\_1\_2 = nltk.jaccard\_distance(set(s1), set(s2))

print(f"Similarity using Jaccard Similarity {(1 - jd\_sent\_1\_2)\*100}%")

def lcs(*l1*, *l2*):

    s1 = word\_tokenize(l1)

    s2 = word\_tokenize(l2)

    # storing the dp values

    dp = [[None] \* (len(s1) + 1) for i in range(len(s2) + 1)]

    for i in range(len(s2) + 1):

        for j in range(len(s1) + 1):

            if i == 0 or j == 0:

                dp[i][j] = 0

            elif s2[i - 1] == s1[j - 1]:

                dp[i][j] = dp[i - 1][j - 1] + 1

            else:

                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    return dp[len(s2)][len(s1)]

from nltk.tokenize import sent\_tokenize, word\_tokenize

tokens\_o = word\_tokenize(doc1)

tokens\_p = word\_tokenize(doc2)

sent\_o = sent\_tokenize(doc1)

sent\_p = sent\_tokenize(doc2)

# maximum length of LCS for a sentence in suspicious text

max\_lcs = 0

sum\_lcs = 0

for i in sent\_p:

    for j in sent\_o:

        l = lcs(i, j)

        max\_lcs = max(max\_lcs, l)

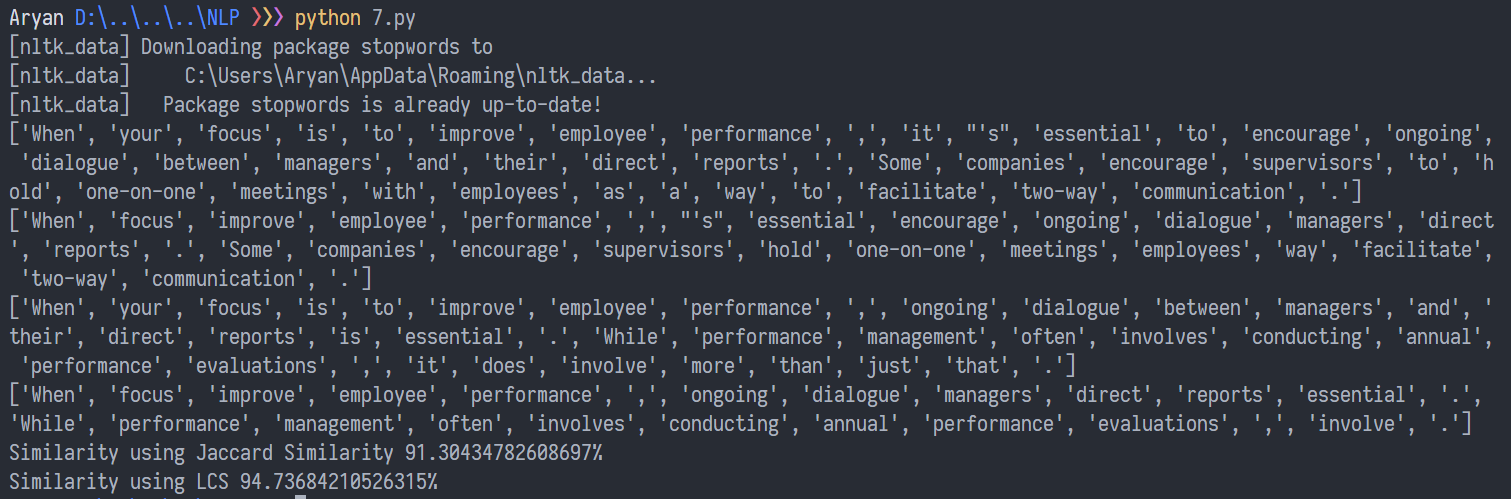
    sum\_lcs += max\_lcs

    smax\_lcs = 0

score = sum\_lcs / len(tokens\_p)

print(f"Similarity using LCS {score\*100}%")

**Output**

****

**Conclusion**Hence, plagiarism checker has been performed using Jaccard Similarity and LCS.