S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 9

Aim: Load the demotext2.txt text file into a variable , Do vectorization using TFID Vectorizer for Comprehension Feature Extraction

Name of Student: Shrutika Pradeep Bagdi

Roll No.: CS22130

Semester/Year: IV/VII

Academic Session: 2024-2025

Date of Performance: __________

Date of Submission: __________

AIM: Load the demotext2.txt text file into a variable , Do vectorization using TFID Vectorizer for Comprehension Feature Extraction

OBJECTIVE/EXPECTED LEARNING OUTCOME:

- Understanding Vectorization.
- Understanding TFID

HARDWARE AND SOFTWARE REQUIRMENTS:

Hardware Requirement:

Moniter, CPU, Keyboard

Software Requirement:

Google Colab

THEORY:

In Natural language Processing (NLP), we have to convert text into numerical representation to apply machine learning techniques to process the text. This is called Vectorization. TF-IDF is a popular vectorization technique used in NLP. One way to convert words (in a document)in to numbers is by calculating how many times a term (t) appears in document(D). This is called Term Frequency(TF).

TF(t) = (count(t) in D / Total words in D)

TF gives a measure of important words in a single document.

The problem with Term Frequency is that, it considers all words equally, stop words like ‘a’, ‘the’ etc will occur more in any document but are not significant. To solve this issue, we discuss IDF.

Inverse Document Frequency(IDF):

Before discussing IDF, let’s see what is Document Frequency(DF). Also remember now we are going to talk about a set of documents as in a document similarity, or search ranking application.

Document Frequency (DF) : *says how many documents in a set of documents contain a particular term.*

This will decide the informativeness of a word. If a term ‘t’ is present in all documents of the set, chances are less that the word is informative. *Eg: stop words.* So, we can say informativeness is inversely proportional to Document Frequency(DF).

And that is why **Inverse Document Frequency(IDF)** is significant. It is defined as

IDF(w) = log(Total no. of Docs / No. of docs containing the term ‘t’)

Let’s see with an example,

If there are 1000 documents and a word appears in all 1000,

log(1000/1000) = 0

(I take base 10 to keep it simple).

This means it is not an important word (can be a stop word).

But if only 10 documents have this word,

log(1000/10) = 2

This means word may be an important word and that document can be relevant in case of ranking /information retrieval.

TF-IDF

Now we use both TF and IDF to create a meaningful numerical representation of text.

TF-IDF = TF(t) * IDF(t)

This product will be high if both TF and IDF are high for a term. A term,'t' that repeats many times in a document, say **D1** and it occurs only in a few documents in the corpus will have a high TF-IDF value for **D1**. So if this is a search for 't' (like Google search) **D1** will appear first.

TF -IDF is frequencies weighted by uniqueness.

CODE:

```
from sklearn.feature_extraction.text import TfidfVectorizer
try:
    with open('/content/demotext2.txt', 'r') as file:
        text_content = file.read()
    print("File loaded successfully.")
    if text_content:
        tfidf_vectorizer = TfidfVectorizer()
        tfidf_features = tfidf_vectorizer.fit_transform([text_content])
        print("TF-IDF vectorization complete.")
        print("\nTF-IDF Feature Vectors:")
        # print(tfidf_features)
        # Get the feature names (words)
        feature_names = tfidf_vectorizer.get_feature_names_out()
        # Convert the sparse matrix to a dense array for easier iteration
        dense_features = tfidf_features.todense()
        # Print the words and their TF-IDF scores
        print("\nWords and their TF-IDF Scores:")
        for doc_index, doc_features in enumerate(dense_features):
            print(f"Document {doc_index + 1}:")
```

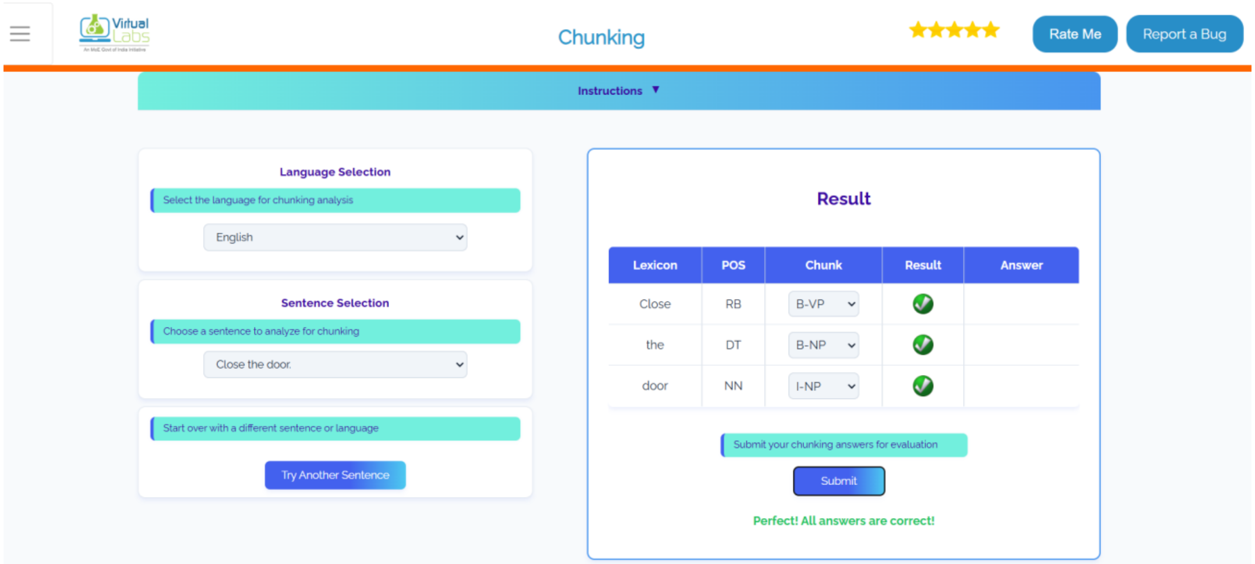
```
            for word_index, score in enumerate(doc_features.tolist()[0]):
                if score > 0: # Only print words with a score greater than 0
                    print(f"  {feature_names[word_index]}: {score}")
        else:
            print("Text content is empty. Cannot perform TF-IDF vectorization.")
except FileNotFoundError:
    text_content = None
    print("Error: demotext2.txt not found. Please upload the file.")
except Exception as e:
    print(f"An error occurred: {e}")
```

OUTPUT (SCREENSHOT):

```
File loaded successfully.
TF-IDF vectorization complete.

TF-IDF Feature Vectors:

Words and their TF-IDF Scores:
Document 1:
  16j7: 0.028629916715693413
  9th: 0.028629916715693413
  admirable: 0.028629916715693413
  admirer: 0.028629916715693413
  after: 0.057259833431386825
  alike: 0.028629916715693413
  also: 0.028629916715693413
  an: 0.028629916715693413
  and: 0.486708584166788
  any: 0.057259833431386825
  arabic: 0.028629916715693413
  arts: 0.028629916715693413
  as: 0.028629916715693413
  at: 0.028629916715693413
  attended: 0.028629916715693413
  beer: 0.028629916715693413
  bra: 0.028629916715693413
  brought: 0.028629916715693413
  but: 0.028629916715693413
  by: 0.057259833431386825
```



CONCLUSION:
Thus, successfully perform load the demotext2.txt text file into a variable , Do vectorization using TFID Vectorizer for Comprehension Feature Extraction.

DISCUSSION AND VIVA VOCE:

- What is TFIDF?
- Why we need TFIDF?

REFERENCE:

- https://nlp-iiith.vlabs.ac.in/exp/chunking/simulation.html
- https://www.geeksforgeeks.org/data-analysis/how-to-chunk-text-data-a-comparative-analysis/