**Assignment No 7**

**Text Analytics**

1. Extract Sample document and apply following document preprocessing methods:

a. Tokenization,

b. POS Tagging,

c. Stop words removal,

d. Stemming,

e. Lemmatization.

2. Create representation of document by calculating Term Frequency and Inverse Document Frequency.

**Step I:**

Extract Sample Document i.e Extracting text from Doc File

Here we will extract text from the doc file using docx module.

For installation:

pip install python-docx

Suppose we have **sample.docx** text file with us which is having this content “Hello this is class of TE student Div II ”

# Importing our library and reading the doc file

import docx

doc = docx.Document('sample.docx')

# Printing the title

print(doc.paragraphs[0].text)

Output:

Hello this is class of TE student Div II

1. **Tokenization**
2. **Tokenization with NLTK**

NLTK stands for Natural Language Toolkit. This is a suite of libraries and programs for statistical natural language processing for English written in Python.

NLTK contains a module called tokenize with a word\_tokenize() method that will help us split a text into tokens. Once you installed NLTK, write the following code to tokenize text.

import nltk

word\_data = "It originated from the idea that there are readers who prefer learning new skills from the comforts of their drawing rooms"

nltk\_tokens = nltk.word\_tokenize(word\_data)

print (nltk\_tokens)

Output:

['It', 'originated', 'from', 'the', 'idea', 'that', 'there', 'are', 'readers', 'who', 'prefer', 'learning', 'new', 'skills', 'from', 'the',

'comforts', 'of', 'their', 'drawing', 'rooms']

1. **Simple tokenization with .split**

This is the simplest method to perform tokenization in Python. If you type .split(), the text will be separated at each blank space.

text = “””It originated from the idea that there are readers who prefer learning new skills from the comforts of their drawing rooms.”””

text.split()

Output:

['It', 'originated', 'from', 'the', 'idea', 'that', 'there', 'are', 'readers', 'who', 'prefer', 'learning', 'new', 'skills', 'from', 'the',

'comforts', 'of', 'their', 'drawing', 'rooms']

**B) POS Tagging**

POS Tagging (Parts of Speech Tagging) is a process to mark up the words in text format for a particular part of a speech based on its definition and context. It is responsible for text reading in a language and assigning some specific token (Parts of Speech) to each word. It is also called grammatical tagging.

For Example :

Input: Everything to permit us.

Output: [(‘Everything’, NN),(‘to’, TO), (‘permit’, VB), (‘us’, PRP)]

**Steps Involved in the POS tagging example:**

1. Tokenize text (word\_tokenize)
2. apply pos\_tag to above step that is nltk.pos\_tag(tokenize\_text)

**NLTK POS Tags Examples are as below:**

| **Abbreviation** | **Meaning** |
| --- | --- |
| CC | coordinating conjunction |
| CD | cardinal digit |
| DT | determiner |
| EX | existential there |
| FW | foreign word |
| IN | preposition/subordinating conjunction |
| JJ | This NLTK POS Tag is an adjective (large) |
| JJR | adjective, comparative (larger) |
| JJS | adjective, superlative (largest) |
| LS | list market |
| MD | modal (could, will) |
| NN | noun, singular (cat, tree) |
| NNS | noun plural (desks) |
| NNP | proper noun, singular (sarah) |
| NNPS | proper noun, plural (indians or americans) |
| PDT | predeterminer (all, both, half) |
| POS | possessive ending (parent\ ‘s) |
| PRP | personal pronoun (hers, herself, him, himself) |
| PRP$ | possessive pronoun (her, his, mine, my, our ) |
| RB | adverb (occasionally, swiftly) |
| RBR | adverb, comparative (greater) |
| RBS | adverb, superlative (biggest) |
| RP | particle (about) |
| TO | infinite marker (to) |
| UH | interjection (goodbye) |
| VB | verb (ask) |
| VBG | verb gerund (judging) |
| VBD | verb past tense (pleaded) |
| VBN | verb past participle (reunified) |
| VBP | verb, present tense not 3rd person singular(wrap) |
| VBZ | verb, present tense with 3rd person singular (bases) |
| WDT | wh-determiner (that, what) |
| WP | wh- pronoun (who) |
| WRB | wh- adverb (how) |

The above NLTK POS tag list contains all the NLTK POS Tags. NLTK POS tagger is used to assign grammatical information of each word of the sentence. Installing, Importing and downloading all the packages of POS NLTK is complete.

Sample code:

from nltk import pos\_tag

from nltk import RegexpParser

text ="learn php from SKNCOE and make study easy".split()

print("After Split:",text)

tokens\_tag = pos\_tag(text)

print("After Token:",tokens\_tag)

OutPut:

After Split: ['learn', 'php', 'from', 'SKNCOE', 'and', 'make', 'study', 'easy']

After Token: [('learn', 'JJ'), ('php', 'NN'), ('from', 'IN'), ('SKNCOE', 'NN'), ('and', 'CC'), ('make', 'VB'), ('study', 'NN'), ('easy', 'JJ')]

**C) Stop words removal**

**Removing stop words with NLTK in Python**

The process of converting data to something a computer can understand is referred to as pre-processing. 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 to take 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 stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. You can find them in the nltk\_data directory.

**To check the list of stopwords you can type the following commands in the python shell.** 

import nltk

from nltk.corpus import stopwords

print(stopwords.words('english'))

Output:

{‘ourselves’, ‘hers’, ‘between’, ‘yourself’, ‘but’, ‘again’, ‘there’, ‘about’, ‘once’, ‘during’, ‘out’, ‘very’, ‘having’, ‘with’, ‘they’, ‘own’, ‘an’, ‘be’, ‘some’, ‘for’, ‘do’, ‘its’, ‘yours’, ‘such’, ‘into’, ‘of’, ‘most’, ‘itself’, ‘other’, ‘off’, ‘is’, ‘s’, ‘am’, ‘or’, ‘who’, ‘as’, ‘from’, ‘him’, ‘each’, ‘the’, ‘themselves’, ‘until’, ‘below’, ‘are’, ‘we’, ‘these’, ‘your’, ‘his’, ‘through’, ‘don’, ‘nor’, ‘me’, ‘were’, ‘her’, ‘more’, ‘himself’, ‘this’, ‘down’, ‘should’, ‘our’, ‘their’, ‘while’, ‘above’, ‘both’, ‘up’, ‘to’, ‘ours’, ‘had’, ‘she’, ‘all’, ‘no’, ‘when’, ‘at’, ‘any’, ‘before’, ‘them’, ‘same’, ‘and’, ‘been’, ‘have’, ‘in’, ‘will’, ‘on’, ‘does’, ‘yourselves’, ‘then’, ‘that’, ‘because’, ‘what’, ‘over’, ‘why’, ‘so’, ‘can’, ‘did’, ‘not’, ‘now’, ‘under’, ‘he’, ‘you’, ‘herself’, ‘has’, ‘just’, ‘where’, ‘too’, ‘only’, ‘myself’, ‘which’, ‘those’, ‘i’, ‘after’, ‘few’, ‘whom’, ‘t’, ‘being’, ‘if’, ‘theirs’, ‘my’, ‘against’, ‘a’, ‘by’, ‘doing’, ‘it’, ‘how’, ‘further’, ‘was’, ‘here’, ‘than’}

**Note:**You can even modify the list by adding words of your choice in the English .txt. File in the stop words directory.

**Stop word removal using Using Python's NLTK Library**

The NLTK library is one of the oldest and most commonly used Python libraries for Natural Language Processing. NLTK supports stop word removal, and you can find the list of stop words in the corpus module. To remove stop words from a sentence, you can divide your text into words and then remove the word if it exits in the list of stop words provided by NLTK.

Example:

from nltk.corpus import stopwords

nltk.download('stopwords')

from nltk.tokenize import word\_tokenize

text = "Nick likes to play football, however he is not too fond of tennis."

text\_tokens = word\_tokenize(text)

tokens\_without\_sw = [word for word in text\_tokens if not word in stopwords.words()]

print(tokens\_without\_sw)

Output:

['Nick', 'likes', 'play', 'football', ',', 'however', 'fond', 'tennis', '.']

**D) Stemming**

**Stemming words with NLTK**

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”.

Some more example of stemming for root word "like" include:

-> "likes"

-> "liked"

-> "likely"

-> "liking"

**Applications of stemming are:**

1) Stemming is used in information retrieval systems like search engines.

2) It is used to determine domain vocabularies in domain analysis.

Stemming is desirable as it may reduce redundancy as most of the time the word stem and their inflected/derived words mean the same.

Example:

# importing modules

from nltk.stem import PorterStemmer

from nltk.tokenize import word\_tokenize

ps = PorterStemmer()

sentence = "Programmers program with programming languages"

words = word\_tokenize(sentence)

for w in words:

    print(w, " : ", ps.stem(w))

**Output :**

Programmers: program

program: program

with: with

programming: program

languages: language

**E) Lemmatization**

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meanings 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 the same. Actually, lemmatization is preferred over Stemming because lemmatization does morphological analysis of the words.

**Applications of lemmatization are:**

1) Used in comprehensive retrieval systems like search engines.

2) Used in compact indexing

**Examples of lemmatization:**

-> rocks : rock

-> corpora : corpus

-> better : good

One major difference with stemming is that lemmatize takes a part of speech parameter, “pos” If not supplied, the default is “noun.”

Below is the implementation of lemmatization words using NLTK:

Sample Code:

# import these modules

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print("rocks :", lemmatizer.lemmatize("rocks"))

print("corpora :", lemmatizer.lemmatize("corpora"))

# a denotes adjective in "pos"

print("better :", lemmatizer.lemmatize("better", pos ="a"))

**Output :**

rocks: rock

corpora: corpus

better: good

**Part II of Assignment:**

**Create representation of document by calculating Term Frequency and Inverse Document Frequency**

**TF-IDF**

This technique is used to find meaning of sentences consisting of words and cancels out the incapabilities of Bag of Words technique which is good for text classification or for helping a machine read words in numbers.

**1. Term Frequency (TF):**

Suppose we have a set of English text documents and wish to rank which document is most relevant to the query , “Data Science is awesome !” A simple way to start out is by eliminating documents that do not contain all three words “Data”,”is”, “Science”, and “awesome”, but this still leaves many documents. To further distinguish them, we might count the number of times each term occurs in each document; the number of times a term occurs in a document is called its term frequency.

The weight of a term that occurs in a document is simply proportional to the term frequency.

Formula :

tf(t,d) = count of t in d / number of words in d

**2 -Document Frequency :**

Thismeasures the importance of document in whole set of corpus, this is very similar to TF. The only difference is that TF is frequency counter for a term t in document d, where as DF is the count of **occurrences** of term t in the document set N. In other words, DF is the number of documents in which the word is present. We consider one occurrence if the term consists in the document at least once, we do not need to know the number of times the term is present.

*df(t) = occurrence of t in documents*

**3 -Inverse Document Frequency(IDF):**

While computing TF, all terms are considered equally important. However it is known that certain terms, such as “is”, “of”, and “that”, may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing IDF, an *inverse document frequency* factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

IDF is the inverse of the document frequency which measures the informativeness of term t. When we calculate IDF, it will be very low for the most occurring words such as stop words (because stop words such as “is” is present in almost all of the documents, and N/df will give a very low value to that word). This finally gives what we want, a relative weightage.

*idf(t) = N/df*

Now there are few other problems with the IDF , in case of a large corpus,say 100,000,000 , the IDF value explodes , to avoid the effect we take the log of idf .

During the query time, when a word which is not in vocab occurs, the df will be 0. As we cannot divide by 0, we smoothen the value by adding 1 to the denominator.

that’s the final formula:

**Formula :**

*idf(t) = log(N/(df + 1))*

tf-idf now is a the right measure to evaluate how important a word is to a document in a collection or corpus.here are many different variations of TF-IDF but for now let us concentrate on the this basic version.

**Formula :**

*tf-idf(t, d) = tf(t, d) \* log(N/(df + 1))*

**Implementing TF-IDF in Python from Scratch:**

To make TF-IDF from scratch in python, let’s imagine those two sentences from different document:

first\_sentence : “Data Science is the sexiest job of the 21st century”.

second\_sentence : “machine learning is the key for data science”.

First step we have to create the TF function to calculate total word frequency for all documents. Here are the codes below:

first as usual we should import the necessary libraries :

import pandas as pd  
import sklearn as sk  
import math

so let’s load our sentences and combine them together in a single set :

first\_sentence = "Data Science is the sexiest job of the 21st century"  
second\_sentence = "machine learning is the key for data science"#split so each word have their own stringfirst\_sentence = first\_sentence.split(" ")  
second\_sentence = second\_sentence.split(" ")#join them to remove common duplicate words  
total= set(first\_sentence).union(set(second\_sentence))print(total)

Output :

{'data', 'Science', 'job', 'sexiest', 'the', 'for', 'science', 'machine', 'of', 'is', 'learning', '21st', 'key', 'Data', 'century'}

Now lets add a way to count the words using a dictionary key-value pairing for both sentences :

wordDictA = dict.fromkeys(total, 0)   
wordDictB = dict.fromkeys(total, 0)for word in first\_sentence:  
 wordDictA[word]+=1  
   
for word in second\_sentence:  
 wordDictB[word]+=1

Now we put them in a dataframe and then view the result:

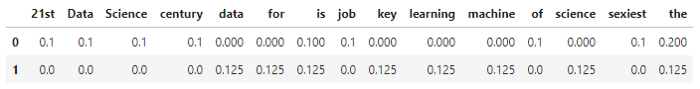
pd.DataFrame([wordDictA, wordDictB])



No let’s writing the TF Function :

def computeTF(wordDict, doc):  
 tfDict = {}  
 corpusCount = len(doc)  
 for word, count in wordDict.items():  
 tfDict[word] = count/float(corpusCount)  
 return(tfDict)#running our sentences through the tf function:tfFirst = computeTF(wordDictA, first\_sentence)  
tfSecond = computeTF(wordDictB, second\_sentence)#Converting to dataframe for visualizationtf = pd.DataFrame([tfFirst, tfSecond])

and this is the expected output :



And now that we finished the TF section, we move onto the IDF part:

def computeIDF(docList):  
 idfDict = {}  
 N = len(docList)  
   
 idfDict = dict.fromkeys(docList[0].keys(), 0)  
 for word, val in idfDict.items():  
 idfDict[word] = math.log10(N / (float(val) + 1))  
   
 return(idfDict)#inputing our sentences in the log file  
idfs = computeIDF([wordDictA, wordDictB])

and now we implement the idf formula , let’s finish with calculating the TFIDF

def computeTFIDF(tfBow, idfs):  
 tfidf = {}  
 for word, val in tfBow.items():  
 tfidf[word] = val\*idfs[word]  
 return(tfidf)  
#running our two sentences through the IDF:idfFirst = computeTFIDF(tfFirst, idfs)  
idfSecond = computeTFIDF(tfSecond, idfs)  
#putting it in a dataframe  
idf= pd.DataFrame([idfFirst, idfSecond])  
print(idf)

output :

