Assignment No: 7

Aim/Problem Statement:- Text Analytics

Extract sample document and apply following document preprocessing methods:

1) Tokenization, POS tagging, stop words removal, stemming and lammetization
2) Create representation of document by calculating term frequency and inverse document frequency

Theory:-

Natural language processing is one of the fields in programming where the natural language is processed by the software. This has many applications like sentiment analysis, language translation, fake news detection, grammatical error detection etc.

The input in natural language processing is text. The data collection for this text happens from a lot of sources. This requires a lot of cleaning and processing before the data can be used for analysis.

These are some of the methods of processing the data in NLP:

- Tokenization
- Stop words removal
- Stemming
- Normalization
- Lemmatization
- Parts of speech tagging

Tokenization:

Tokenization is breaking the raw text into small chunks. Tokenization breaks the raw text into words, sentences called tokens. These tokens help in understanding the context or developing the model for the NLP. The tokenization helps in interpreting the meaning of the text by analyzing the sequence of the words.

For example, the text “It is raining” can be tokenized into ‘It’, ‘is’, ‘raining’

There are different methods and libraries available to perform tokenization. NLTK, Gensim, Keras are some of the libraries that can be used to accomplish the task.

Tokenization can be done to either separate words or sentences. If the text is split into words using some separation technique it is called word tokenization and same separation done for sentences is called sentence tokenization.

There are various tokenization techniques available which can be applicable based on the language and purpose of modeling. Below are a few of the tokenization techniques used in NLP.

White Space Tokenization

This is the simplest tokenization technique. Given a sentence or paragraph it tokenizes into words by splitting the input whenever a white space in encountered. This is the fastest tokenization technique but will work for languages in which the white space breaks apart the sentence into meaningful words. Example: English.

Dictionary Based Tokenization

In this method the tokens are found based on the tokens already existing in the dictionary. If the token is not found, then special rules are used to tokenize it. It is an advanced technique compared to whitespace tokenizer.

Rule Based Tokenization

In this technique a set of rules are created for the specific problem. The tokenization is done based on the rules. For example creating rules bases on grammar for particular language.

Regular Expression Tokenizer

This technique uses regular expression to control the tokenization of text into tokens. Regular expression can be simple to complex and sometimes difficult to comprehend. This technique should be preferred when the above methods does not serve the required purpose. It is a rule based tokenizer.

Penn TreeBank Tokenization

Tree bank is a corpus created which gives the semantic and syntactical annotation of language. Penn Treebank is one of the largest treebanks which was published. This technique of tokenization separates the punctuation, clitics (words that occur along with other words like I'm, don't) and hyphenated words together.

Spacy Tokenizer

This is a modern technique of tokenization which faster and easily customizable. It provides the flexibility to specify special tokens that need not be segmented or need to be segmented using special rules. Suppose you want to keep $ as a separate token, it takes precedence over other tokenization operations.

Moses Tokenizer

This is a tokenizer which is advanced and is available before Spacy was introduced. It is basically a collection of complex normalization and segmentation logic which works very well for structured language like English.

Subword Tokenization

This tokenization is very useful for specific application where sub words make significance. In this technique the most frequently used words are given unique ids and less frequent words are split into sub words and they best represent the meaning independently. For example if the word few is appearing frequently in the text it will be assigned a unique id, where fewer and fewest which are rare words and are less frequent in the text will be split into sub words like few, er, and est. This helps the language model not to learn fewer and fewest as two separate words. This allows to identify the unknown words in the data set during training. There are different types of subword tokenization and they are given below and Byte-Pair Encoding and WordPiece will be discussed briefly.

- Byte-Pair Encoding (BPE)
- WordPiece
- Unigram Language Model
- SentencePiece

Byte-Pair Encoding (BPE)

This technique is based on the concepts in information theory and compression. BPE uses Huffman encoding for tokenization meaning it uses more embedding or symbols for representing less frequent words and less symbols or embedding for more frequently used words.

The BPE tokenization is bottom up sub word tokenization technique. The steps involved in BPE algorithm is given below.

1. Starts with splitting the input words into single unicode characters and each of them corresponds to a symbol in the final vocabulary.
2. Find the most frequent occurring pair of symbols from the current vocabulary.
3. Add this to the vocabulary and size of vocabulary increases by one.
4. Repeat steps ii and iii till the defined number of tokens are built or no new combination of symbols exist with required frequency.

WordPiece

WordPiece is similar to BPE techniques expect the way the new token is added to the vocabulary. BPE considers the token with most frequent occurring pair of symbols to merge into the vocabulary. While WordPiece considers the frequency of individual symbols also and based on below count it merges into the vocabulary.

Count (x, y) = frequency of (x, y) / frequency (x) * frequency (y)

The pair of symbols with maximum count will be considered to merge into vocabulary. So it allows rare tokens to be included into vocabulary as compared to BPE.

Tokenization with NLTK

NLTK (natural language toolkit) is a python library developed by Microsoft to aid in NLP.

```
#Tokenization

#Splitting a sentece into words(tokes)
```

```
#Learn tokenize with nltk

# tokenization with nltk

print("\ntokenize with nltk: ",nlp.word_tokenize(data))
```

```
Input: Text can be sentences, strings, words, characters and large documents.
Now lets create a sentence to understand basics of text mining methods.
Our sentence is "no woman no cry" from Bob Marley
playing swimming dancing played danced

Output: tokenize with nltk:  ['Text', 'can', 'be', 'sentences', ',', 'strings',
',', 'words', ',', 'characters', 'and', 'large', 'documents', '.', 'Now', 'lets
', 'create', 'a', 'sentence', 'to', 'understand', 'basics', 'of', 'text', 'mini
ng', 'methods', '.', 'Our', 'sentence', 'is', '``', 'no', 'woman', 'no', 'cry',
 "''", 'from', 'Bob', 'Marley', 'playing', 'swimming', 'dancing', 'played', 'da
nced']
```

Stemming:

Stemming is a natural language processing technique that lowers inflection in words to their root forms, hence aiding in the preprocessing of text, words, and documents for text normalization.
According to Wikipedia, inflection is the process through which a word is modified to communicate many grammatical categories, including tense, case, voice, aspect, person, number, gender, and mood. Thus, although a word may exist in several inflected forms, having multiple inflected forms inside the same text adds redundancy to the NLP process.
As a result, we employ stemming to reduce words to their basic form or stem, which may or may not be a legitimate word in the language.
For instance, the stem of these three words, connections, connected, connects, is “connect”. On the other hand, the root of trouble, troubled, and troubles is “troubl,” which is not a recognized word.

Why Stemming is Important?
As previously stated, the English language has several variants of a single term. The presence of these variances in a text corpus results in data redundancy when developing NLP or machine learning models. Such models may be ineffective.
To build a robust model, it is essential to normalize text by removing repetition and transforming words to their base form through stemming.

Application of Stemming
In information retrieval, text mining SEOs, Web search results, indexing, tagging systems, and word analysis, stemming is employed. For instance, a Google search for prediction and predicted returns comparable results.

Martin Porter invented the Porter Stemmer or Porter algorithm in 1980. Five steps of word reduction are used in the method, each with its own set of mapping rules. Porter Stemmer is the original stemmer and

is renowned for its ease of use and rapidity. Frequently, the resultant stem is a shorter word with the same root meaning.
PorterStemmer() is a module in NLTK that implements the Porter Stemming technique. Let us examine this with the aid of an example.

Example of PorterStemmer()

In the example below, we construct an instance of PorterStemmer() and use the Porter algorithm to stem the list of words.

```
from nltk.stem import PorterStemmer

porter = PorterStemmer()

words =
['Connects','Connecting','Connections','Connected','Connection','Connectings','Connect']

for word in words:

    print(word,"--->",porter.stem(word))
```

```
Connects ---> connect

Connecting ---> connect

Connections ---> connect

Connected ---> connect

Connection ---> connect

Connectings ---> connect

Connect ---> connect
```

Lemmatization:

Lemmatization is another technique which is used to reduce words to a **normalized form**. In lemmatization, the transformation uses a **dictionary** to map different variants of a word back to its root format. So, with this approach, we are able to reduce non trivial inflections such as “is”, “was”, “were” back to the root “be”.

import the **WordNetLemmatizer** and try it out by the following example!

```
# lemmatization

lemma = nlp.WordNetLemmatizer()

lemma_roots = [lemma.lemmatize(each) for each in words_list]

print("result of lemmatization: ",lemma_roots)
```

```
result of lemmatization:  ['text', 'can', 'be', 'sentences,', 'strings,', 'word
s,', 'character', 'and', 'large', 'documents.\nnow', 'let', 'create', 'a', 'sen
tence', 'to', 'understand', 'basic', 'of', 'text', 'mining', 'methods.', '\nour
', 'sentence', 'is', '"no', 'woman', 'no', 'cry"', 'from', 'bob', 'marley\nplay
ing', 'swimming', 'dancing', 'played', 'danced']
```

Lemmatization is similar to stemming with one difference i.e. the final form is also a **meaningful word**. Thus, stemming operation does not need a dictionary like lemmatization.

Stopword:

There different techniques for removing stop words from strings in Python. Stop words are those words in natural language that have a very little meaning, such as "is", "an", "the", etc. Search engines and other enterprise indexing platforms often filter the stop words while fetching results from the database against the user queries.

Stop words are often removed from the text before training deep learning and machine learning models since stop words occur in abundance, hence providing little to no unique information that can be used for classification or clustering.

Removing Stop Words with Python

With the Python programming language, we have a myriad of options to use in order to remove stop words from strings. we can either use one of the several natural language processing libraries such as NLTK, SpaCy, Gensim, TextBlob, etc.,

There are number of different approaches, depending on the NLP library in use

- Stop Words with NLTK
- Stop Words with Gensim
- Stop Words with SpaCy

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 we can find the list of stop words in the `corpus` module. To remove stop words from a sentence, we can divide your text into words and then remove the word if it exits in the list of stop words provided by NLTK.

Steps Involved in the POS tagging

- Tokenize text (word_tokenize)
- 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)

Abbreviation	Meaning
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)

Abbreviation	Meaning
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.

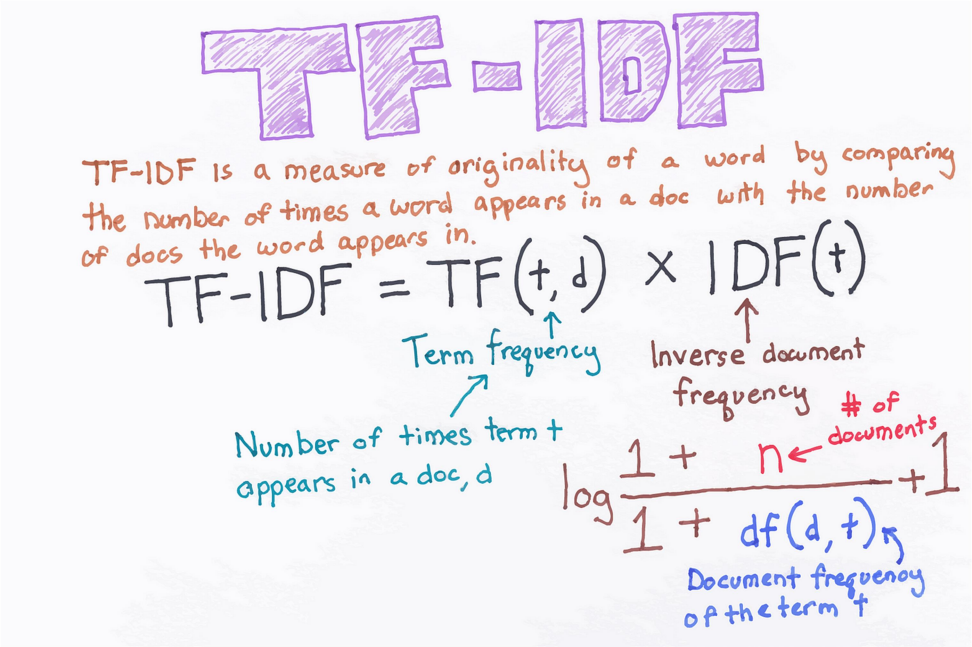
import nltk

nltk.download('averaged_perceptron_tagger')

print("\nPOS tagging:",nltk.pos_tag(text_tokens))

```
POS tagging: [('Text', 'NN'), ('can', 'MD'), ('be', 'VB'), ('sentences', 'NNS')
, (',', ','), ('strings', 'NNS'), (',', ','), ('words', 'NNS'), (',', ','), ('c
haracters', 'NNS'), ('and', 'CC'), ('large', 'JJ'), ('documents', 'NNS'), ('.',
 '.'), ('Now', 'RB'), ('lets', 'VBZ'), ('create', 'VB'), ('a', 'DT'), ('sentenc
e', 'NN'), ('to', 'TO'), ('understand', 'VB'), ('basics', 'NNS'), ('of', 'IN'),
 ('text', 'NN'), ('mining', 'NN'), ('methods', 'NNS'), ('.', '.'), ('Our', 'PRP
$'), ('sentence', 'NN'), ('is', 'VBZ'), ('``', '``'), ('no', 'DT'), ('woman', '
NN'), ('no', 'DT'), ('cry', 'NN'), ("''", "''"), ('from', 'IN'), ('Bob', 'NNP')
, ('Marley', 'NNP'), ('playing', 'VBG'), ('swimming', 'VBG'), ('dancing', 'VBG'
), ('played', 'VBN'), ('danced', 'VBD')]
```

What is Feature Engineering of text data?
The procedure of converting raw text data into machine understandable format(numbers) is called feature engineering of text data. Machine learning and deep learning algorithm performance and accuracy is fundamentally dependent on the type of feature engineering techniques used.
we are going to see Feature Engineering technique using TF-IDF and mathematical calculation of TF, IDF and TF-IDF to understand the insights of mathematical logic behind libraries such as *TfidfTransformer* from *sklearn.feature_extraction* package in python.



TF (Term Frequency) :

- Term frequency is simply the count of a word present in a sentence
- TF is basically capturing the importance of the word irrespective of the length of the document.
- A word with the frequency of 3 with the length of sentence being 10 is not the same as when the word length of sentence being 100 words. It should get more importance in the first scenario; that is what TF does.

IDF (Inverse Document Frequency):

- IDF of each word is the log of the ratio of the total number of rows to the number of rows in a particular document in which that word is present.
- IDF will measure the rareness of a term. word like ‘a’ and ‘the’ show up in all the documents of corpus, but the rare words is not in all the documents.

TF-IDF:

It is the simplest product of TF and IDF so that both of the drawbacks are addressed above, which makes predictions and information retrieval relevant.

Now import TfidfTransformer and CountVectorizer from sklearn.feature_extraction module.

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
```

CountVectorizer is used to find the word count in each document of a dataset. Also known as to calculate Term Frequency

```
docs=['the              cat              see              the              mouse',
'the        house        has        a        tiny        little        mouse',
'the        mouse        ran        away        from        the        house',
'the          cat          finally          ate          the          mouse',
'the           end           of          the          mouse           story'
]
```

```
#Extracting features by using TfidfTransformer from sklearn.feature_extraction package
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

#fit all the sentences using count-vectorizer and get an array of word counts of each document
import pandas as pd
cv = CountVectorizer()
word_count_vector = cv.fit_transform(docs)
tf = pd.DataFrame(word_count_vector.toarray(), columns=cv.get_feature_names())
print(tf)
```

```
ate  away  cat  end  finally  from  has  house  little  mouse  of  ran  \
0    0     0    1    0        0     0    0      0       0      1   0    0
1    0     0    0    0        0     0    1      1       1      1   0    0
2    0     1    0    0        0     1    0      1       0      1   0    1
3    1     0    1    0        1     0    0      0       0      1   0    0
4    0     0    0    1        0     0    0      0       0      1   1    0

   see  story  the  tiny
0    1      0    2     0
1    0      0    1     1
2    0      0    2     0
3    0      0    2     0
4    0      1    2     0
```

```
#declare TfidfTransformer() instance and fit it with the above word_count_vector to get IDF and final normalized features
tfidf_transformer = TfidfTransformer()
X = tfidf_transformer.fit_transform(word_count_vector)
idf = pd.DataFrame({'feature_name':cv.get_feature_names(), 'idf_weights':tfidf_transformer.idf_})
print(idf)
```

```
feature_name  idf_weights
0            ate     2.098612
1           away     2.098612
2            cat     1.693147
3            end     2.098612
4        finally     2.098612
5           from     2.098612
6            has     2.098612
7          house     1.693147
8         little     2.098612
9          mouse     1.000000
10            of     2.098612
11           ran     2.098612
12           see     2.098612
13         story     2.098612
```

```
14          the     1.000000
15         tiny     2.098612
```

#Final feature vectors are
tf_idf = pd.DataFrame(X.toarray() ,columns=cv.get_feature_names())
print(tf_idf)

```
ate      away       cat       end   finally      from       has  \
0  0.000000  0.000000  0.483344  0.000000  0.000000  0.000000  0.000000
1  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.493562
2  0.000000  0.457093  0.000000  0.000000  0.000000  0.457093  0.000000
3  0.513923  0.000000  0.414630  0.000000  0.513923  0.000000  0.000000
4  0.000000  0.000000  0.000000  0.491753  0.000000  0.000000  0.000000

      house    little     mouse        of       ran       see     story  \
0  0.000000  0.000000  0.285471  0.000000  0.000000  0.599092  0.000000
1  0.398203  0.493562  0.235185  0.000000  0.000000  0.000000  0.000000
2  0.368780  0.000000  0.217807  0.000000  0.457093  0.000000  0.000000
3  0.000000  0.000000  0.244887  0.000000  0.000000  0.000000  0.000000
4  0.000000  0.000000  0.234323  0.491753  0.000000  0.000000  0.491753

        the      tiny
0  0.570941  0.000000
1  0.235185  0.493562
2  0.435614  0.000000
3  0.489774  0.000000
4  0.468646  0.000000
```

df = pd.DataFrame({'docs': ["the cat see the mouse",
 "the house has a tiny little mouse",
 "the mouse ran away from the house",
 "the cat finally ate the mouse",
 "the end of the mouse story"
 ]})
print(df)

```
         docs
0             the cat see the mouse
1  the house has a tiny little mouse
2  the mouse ran away from the house
3      the cat finally ate the mouse
4         the end of the mouse story
```

Input: text document

Output: list of tokens,POS tags, lammetization and stemming output , TF,IDF and TF-IDF

Conclusion: In this way we have applied document pre-processing methods for tokenization, Stop word removal, stemming, lemmatization, PoS tagging on input document and created document representation b calculating TF-IDF.

Questions:

1) What is tokenization explain with example
2) Differentiate between stemming and lemmatization
3) Explain TF-IDF with an example
4) What is NLTK?
5) Explain different methods of stopword removal