
Group A

Assignment No: 7

Contents for Theory:

1. **Basic concepts of Text Analytics**
2. **Text Analysis Operations using natural language toolkit**
3. **Text Analysis Model using TF-IDF.**
4. **Bag of Words (BoW)**

---

1. Basic concepts of Text Analytics

One of the most frequent types of day-to-day conversion is text communication. In our everyday routine, we chat, message, tweet, share status, email, create blogs, and offer opinions and criticism. All of these actions lead to a substantial amount of unstructured text being produced. It is critical to examine huge amounts of data in this sector of the online world and social media to determine people's opinions.

Text mining is also referred to as text analytics. Text mining is a process of exploring sizable textual data and finding patterns. Text Mining processes the text itself, while NLP processes with the underlying metadata. Finding frequency counts of words, length of the sentence, presence/absence of specific words is known as text mining. Natural language processing is one of the components of text mining. NLP helps identify sentiment, finding entities in the sentence, and category of blog/article. Text mining is preprocessed data for text analytics. In Text Analytics, statistical and machine learning algorithms are used to classify information.

2. Text Analysis Operations using natural language toolkit

NLTK(natural language toolkit) is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces and lexical resources

such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning and many more.
Analysing movie reviews is one of the classic examples to demonstrate a simple NLP Bag-of-words model, on movie reviews.

2.1. Tokenization:

Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks such as words or sentences is called Tokenization. Token is a single entity that is the building blocks for a sentence or paragraph.

- Sentence tokenization : split a paragraph into **list of sentences** using **sent_tokenize()** method
- Word tokenization : split a sentence into **list of words** using **word_tokenize()** method

2.2. Stop words removal

Stopwords considered as noise in the text. Text may contain stop words such as is, am, are, this, a, an, the, etc. In NLTK for removing stopwords, you need to create a list of stopwords and filter out your list of tokens from these words.

2.3. Stemming and Lemmatization

Stemming is a normalization technique where lists of tokenized words are converted into shortened root words to remove redundancy. Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form.
A computer program that stems word may be called a stemmer.
E.g.
A stemmer reduces the words like fishing, fished, and fisher to the stem fish.
The stem need not be a word, for example the Porter algorithm reduces, argue, argued, argues, arguing, and argus to the stem argu .

Lemmatization in NLTK is the algorithmic process of finding the lemma of a word depending on its meaning and context. Lemmatization usually refers to the morphological analysis of words, which aims to remove inflectional endings. It helps in returning the base or dictionary form of a word known as the lemma.
Eg. Lemma for studies is study

1) Lemmatization Vs Stemming

Stemming algorithm works by cutting the suffix from the word. In a broader sense cuts either the beginning or end of the word.

On the contrary, Lemmatization is a more powerful operation, and it takes into consideration morphological analysis of the words. It returns the lemma which is the base form of all its inflectional forms. In-depth linguistic knowledge is required to create dictionaries and look for the proper form of the word. Stemming is a general operation while lemmatization is an intelligent operation where the proper form will be looked in the dictionary. Hence, lemmatization helps in forming better machine learning features.

2.4. POS Tagging

POS (Parts of Speech) tell us about grammatical information of words of the sentence by assigning specific token (Determiner, noun, adjective , adverb , verb,Personal Pronoun etc.) as tag (DT,NN ,JJ,RB,VB,PRP etc) to each words.
Word can have more than one POS depending upon the context where it is used. We can use POS tags as statistical NLP tasks. It distinguishes a sense of word which is very helpful in text realization and infer semantic information from text for sentiment analysis.

3. Text Analysis Model using TF-IDF.

Term frequency–inverse document frequency(TFIDF) , is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

- **Term Frequency (TF)**

It is a measure of the frequency of a word (w) in a document (d). TF is defined as the ratio of a word’s occurrence in a document to the total number of words in a document. The denominator term in the formula is to normalize since all the corpus documents are of different lengths.

$$TF(w,d) = \frac{occurences\ of\ w\ in\ document\ d}{total\ number\ of\ words\ in\ document\ d}$$

Example:

Documents	Text	Total number of words in a document
A	Jupiter is the largest planet	5
B	Mars is the fourth planet from the sun	8

The initial step is to make a vocabulary of unique words and calculate TF for each document. TF will be more for words that frequently appear in a document and less for rare words in a document.

- **Inverse Document Frequency (IDF)**

  It is the measure of the importance of a word. Term frequency (TF) does not consider the importance of words. Some words such as’ of’, ‘and’, etc. can be most frequently present but are of little significance. IDF provides weightage to each word based on its frequency in the corpus D.

$$IDF(w,D) = \ln(\frac{Total\ number\ of\ documents\ (N)\ in\ corpus\ D}{number\ of\ documents\ containing\ w})$$

In our example, since we have two documents in the corpus, N=2.

Words	TF (for A)	TF (for B)	IDF
Jupiter	1/5	0	ln(2/1) = 0.69
Is	1/5	1/8	ln(2/2) = 0
The	1/5	2/8	ln(2/2) = 0
largest	1/5	0	ln(2/1) = 0.69
Planet	1/5	1/8	ln(2/2) = 0
Mars	0	1/8	ln(2/1) = 0.69
Fourth	0	1/8	ln(2/1) = 0.69
From	0	1/8	ln(2/1) = 0.69
Sun	0	1/8	ln(2/1) = 0.69

- **Term Frequency — Inverse Document Frequency (TFIDF)**

  It is the product of TF and IDF.

  TFIDF gives more weightage to the word that is rare in the corpus (all the documents).

  TFIDF provides more importance to the word that is more frequent in the document.

$$TFIDF\ (w, d, D) = TF(w, d) * IDF(w, D)$$

Words	TF (for A)	TF (for B)	IDF	TFIDF (A)	TFIDF (B)
Jupiter	1/5	0	ln(2/1) = 0.69	0.138	0
Is	1/5	1/8	ln(2/2) = 0	0	0
The	1/5	2/8	ln(2/2) = 0	0	0
largest	1/5	0	ln(2/1) = 0.69	0.138	0
Planet	1/5	1/8	ln(2/2) = 0	0.138	0
Mars	0	1/8	ln(2/1) = 0.69	0	0.086
Fourth	0	1/8	ln(2/1) = 0.69	0	0.086
From	0	1/8	ln(2/1) = 0.69	0	0.086
Sun	0	1/8	ln(2/1) = 0.69	0	0.086

After applying TFIDF, text in A and B documents can be represented as a TFIDF vector of dimension equal to the vocabulary words. The value corresponding to each word represents the importance of that word in a particular document.

TFIDF is the product of TF with IDF. Since TF values lie between 0 and 1, not using ***ln*** can result in high IDF for some words, thereby dominating the TFIDF. We don't want that, and therefore, we use ***ln*** so that the IDF should not completely dominate the TFIDF.

- **Disadvantage of TFIDF**

  It is unable to capture the semantics. For example, funny and humorous are synonyms, but TFIDF does not capture that. Moreover, TFIDF can be computationally expensive if the vocabulary is vast.

4. **Bag of Words (BoW)**

   Machine learning algorithms cannot work with raw text directly. Rather, the text must be converted into vectors of numbers. In natural language processing, a common technique for extracting features from text is to place all of the words that occur in the text in a bucket. This approach is called a bag of words model or BoW for short. It's referred to as a "bag" of words because any information about the structure of the sentence is lost.

Algorithm for Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization:

Step 1: Download the required packages

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

Step 2: Initialize the text

```
text= "Tokenization is the first step in text analytics. The
process of breaking down a text paragraph into smaller chunks
such as words or sentences is called Tokenization."
```

Step 3: Perform Tokenization

```
#Sentence Tokenization
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
print(tokenized_text)

#Word  Tokenization
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

Step 4: Removing Punctuations and Stop Word

```
# print stop words of English
 from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)

text= "How to remove stop words with NLTK library in Python?"
text= re.sub('[^a-zA-Z]', ' ',text)
tokens = word_tokenize(text.lower())
filtered_text=[]
for w in tokens:
      if w not in stop_words:
            filtered_text.append(w)
print("Tokenized Sentence:",tokens)
print("Filterd  Sentence:",filtered_text)
```

Step 5 : Perform Stemming

```
from nltk.stem import PorterStemmer
e_words= ["wait", "waiting", "waited", "waits"]
ps =PorterStemmer()
for w in e_words:
       rootWord=ps.stem(w)
 print(rootWord)
```

Step 6: Perform Lemmatization

```
 from nltk.stem import   WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
      print("Lemma        for        {}        is        {}".format(w,
      wordnet_lemmatizer.lemmatize(w)))
```

Step 7: Apply POS Tagging to text

```
import nltk
from nltk.tokenize import word_tokenize
data="The pink sweater fit her perfectly"
words=word_tokenize(data)
 for word in words:
     print(nltk.pos_tag([word]))
```

Algorithm for Create representation of document by calculating TFIDF

Step 1: Import the necessary libraries.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

Step 2: Initialize the Documents.

```
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

Step 3: Create BagofWords (BoW) for Document A and B.

```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

Step 4: Create Collection of Unique words from Document A and B.

```
uniqueWords  =  set(bagOfWordsA).union(set(bagOfWordsB))
```

Step 5: Create a dictionary of words and their occurrence for each document in the corpus

```
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
    numOfWordsA[word] += 1
    numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
    numOfWordsB[word] += 1
```

Step 6: Compute the term frequency for each of our documents.

```
def computeTF(wordDict, bagOfWords):
tfDict = {}
bagOfWordsCount =  len(bagOfWords)
for word, count in wordDict.items():
    tfDict[word] = count / float(bagOfWordsCount)
return tfDict
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

Step 7: Compute the term Inverse Document Frequency.

```
def computeIDF(documents):
import math
N = len(documents)

idfDict = dict.fromkeys(documents[0].keys(), 0)
for document in documents:
    for word, val in document.items():
        if val > 0:
            idfDict[word] += 1

for word, val in idfDict.items():
    idfDict[word] = math.log(N / float(val))
return idfDict
idfs = computeIDF([numOfWordsA, numOfWordsB])
idfs
```

Step 8: Compute the term TF/IDF for all words.

```
    def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
    tfidfA = computeTFIDF(tfA, idfs)
    tfidfB = computeTFIDF(tfB, idfs)
    df = pd.DataFrame([tfidfA, tfidfB])
    df
```

Conclusion:

In this way we have done text data analysis using TF IDF algorithm

.