NLP PRACTICALS

(TY AI A& B)

PRACTICAL 1

Title: Use of named entity recognition information extraction technique.

PROBLEM STATEMENT:

Use of named entity recognition information extraction technique

OBJECTIVE:

To understand the concept of named entity recognition in Natural Language Processing

Introduction :

Natural Language Processing (NLP) is a process of manipulating or understanding the text or speech by any software or machine. An analogy is that humans interact and understand each other's views and respond with the appropriate answer. In NLP, this interaction, understanding, and response are made by a computer instead of a human.

The named entity recognition (NER) is one of the most popular data preprocessing task. It involves the identification of key information in the text and classification into a set of predefined categories. An entity is basically the thing that is consistently talked about or refers to in the text.

NER is the form of NLP.

At its core, NLP is just a two-step process, below are the two steps that are involved:

- Detecting the entities from the text
- Classifying them into different categories

Some of the categories that are the most important architecture in NER such that:

- Person
- Organization
- Place/ location

Other common tasks include classifying of the following:

- date/time.

- expression
- Numeral measurement (money, percent, weight, etc)
- E-mail address

Ambiguity in NE

- For a person, the category definition is intuitively quite clear, but for computers, there is some ambiguity in classification. Let's look at some ambiguous example:
    - *England (Organisation)* won the 2019 world cup vs The 2019 world cup happened in *England(Location).*
    - *Washington(Location)* is the capital of the US vs The first president of the US was *Washington(Person)*.

Methods of NER

- One way is to train the model for multi-class classification using different machine learning algorithms, but it requires a lot of labelling. In addition to labelling the model also requires a deep understanding of context to deal with the ambiguity of the sentences. This makes it a challenging task for a simple machine learning algorithm.
- Another way is that Conditional random field that is implemented by both NLP Speech Tagger and NLTK. It is a probabilistic model that can be used to model sequential data such as words. The CRF can capture a deep understanding of the context of the sentence.
- **Deep Learning Based NER:** deep learning NER is much more accurate than previous method, as it is capable to assemble words. This is due to the fact that it used a method called word embedding, that is capable of understanding the semantic and syntactic relationship between various words. It is also able to learn analyzes topic-specific as well as high level words automatically. This makes deep learning NER applicable for performing multiple tasks.

How does Named Entity Recognition work?

As we can simple observed that after reading a particular text, naturally we can recognize named entities such as people, values, locations, and so on.For Example, Consider the following sentence:

Sentence: Sundar Pichai, the CEO of Google Inc. is walking in the streets of California.

From the above sentence, we can identify three types of entities: (Named Entities)

- ("person": "Sundar Pichai"),
- ("org": "Google Inc."),
- ("location": "California").

But to do the same thing with the help of computers, we need to help them recognize entities first so that they can categorize them. So, to do so we can take the help of machine learning and Natural Language Processing (NLP).

The role of both these things while implementing NER using computers:

- NLP: It studies the structure and rules of language and forms intelligent systems that are capable of deriving meaning from text and speech.
- Machine Learning: It helps machines learn and improve over time.

To learn what an entity is, a NER model needs to be able to detect a word or string of words that form an entity (e.g. California) and decide which entity category it belongs to.

So, as a concluding step we can say that the heart of any NER model is a two-step process:

- Detect a named entity
- Categorize the entity

So first, we need to create entity categories, like Name, Location, Event, Organization, etc., and feed a NER model relevant training data. Then, by tagging some samples of words and phrases with their corresponding entities, we'll eventually teach our NER model to detect the entities and categorize them.

Use-Cases of Named Entity Recognition

The Named entity recognition (NER) will help us to easily identify the key components in a text, such as names of people, places, brands, monetary values, and more. And extracting the main entities from a text helps us to sort the unstructured data and detect the important information, which is crucial if you have to deal with large datasets.

Various NLP Libraries

NLP Library	Description
NLTK	This is one of the most usable and mother of all NLP libraries.

spaCy	This is a completely optimized and highly accurate library widely used in deep learning
Stanford CoreNLP Python	For client-server-based architecture, this is a good library in NLTK. This is written in JAVA, but it provides modularity to use it in Python.
TextBlob	This is an NLP library which works in Pyhton2 and python3. This is used for processing textual data and provide mainly all type of operation in the form of API.
Gensim	Genism is a robust open source NLP library support in Python. This library is highly efficient and scalable.

Tokenizing

By **tokenizing**, you can conveniently split up text by word or by sentence. This will allow you to work with smaller pieces of text that are still relatively coherent and meaningful even outside of the context of the rest of the text. It's your first step in turning unstructured data into structured data, which is easier to analyze.

When you're analyzing text, you'll be tokenizing by word and tokenizing by sentence. Here's what both types of tokenization bring to the table:

- **Tokenizing by word:** Words are like the atoms of natural language. They're the smallest unit of meaning that still makes sense on its own. Tokenizing your text by word allows you to identify words that come up particularly often. For example, if you were analyzing a group of job ads, then you might find that the word "Python" comes up often. That could suggest high demand for Python knowledge, but you'd need to look deeper to know more.
- **Tokenizing by sentence:** When you tokenize by sentence, you can analyze how those words relate to one another and see more context. Are there a lot of negative words around the word "Python" because the hiring manager doesn't like Python? Are there more terms from the domain of herpetology than the domain of software development, suggesting that you may be dealing with an entirely different kind of python than you were expecting?

Here's how to import the relevant parts of NLTK so you can tokenize by word and by sentence:

```
>>>
>>> from nltk.tokenize import sent_tokenize, word_tokenize
```

Now that you've imported what you need, you can create a string to tokenize. Here's a quote from Dune that you can use:

```
>>>
>>> example_string = """
... Muad'Dib learned rapidly because his first training was in how to learn.
... And the first lesson of all was the basic trust that he could learn.
... It's shocking to find how many people do not believe they can learn,
... and how many more believe learning to be difficult."""
```

You can use sent_tokenize() to split up example_string into sentences:

```
>>>
>>> sent_tokenize(example_string)
["Muad'Dib learned rapidly because his first training was in how to learn.",
'And the first lesson of all was the basic trust that he could learn.',
"It's shocking to find how many people do not believe they can learn, and how many more believe learning to be difficult."]
```

Tokenizing example_string by sentence gives you a list of three strings that are sentences:

1. "Muad'Dib learned rapidly because his first training was in how to learn."
2. 'And the first lesson of all was the basic trust that he could learn.'
3. "It's shocking to find how many people do not believe they can learn, and how many more believe learning to be difficult."

Now try tokenizing example_string by word:

```
>>>
>>> word_tokenize(example_string)
["Muad'Dib",
 'learned',
 'rapidly',
 'because',
 'his',
 'first',
 'training',
 'was',
 'in',
```

'how',
'to',
'learn',
'.',
'And',
'the',
'first',
'lesson',
'of',
'all',
'was',
'the',
'basic',
'trust',
'that',
'he',
'could',
'learn',
'.',
'It',
"'s",
'shocking',
'to',
'find',
'how',
'many',
'people',
'do',
'not',
'believe',
'they',
'can',
'learn',
',',
'and',
'how',

```
'many',
'more',
'believe',
'learning',
'to',
'be',
'difficult',
'.']
```

You got a list of strings that NLTK considers to be words, such as:

- "Muad'Dib"
- 'training'
- 'how'

But the following strings were also considered to be words:

- "'s"
- ','
- '.'

See how "It's" was split at the apostrophe to give you 'It' and "'s", but "Muad'Dib" was left whole? This happened because NLTK knows that 'It' and "'s" (a contraction of "is") are two distinct words, so it counted them separately. But "Muad'Dib" isn't an accepted contraction like "It's", so it wasn't read as two separate words and was left intact.

Filtering Stop Words

Stop words are words that you want to ignore, so you filter them out of your text when you're processing it. Very common words like 'in', 'is', and 'an' are often used as stop words since they don't add a lot of meaning to a text in and of themselves.

Here's how to import the relevant parts of NLTK in order to filter out stop words:

```
>>>
>>> nltk.download("stopwords")
>>> from nltk.corpus import stopwords
>>> from nltk.tokenize import word_tokenize
```

Here's a quote from Worf that you can filter:

```
>>>
```

```
>>> worf_quote = "Sir, I protest. I am not a merry man!"
```
Now tokenize worf_quote by word and store the resulting list in words_in_quote:

```
>>>
>>> words_in_quote = word_tokenize(worf_quote)
>>> words_in_quote
['Sir', ',', 'protest', '.', 'merry', 'man', '!']
```
You have a list of the words in worf_quote, so the next step is to create a set of stop words to filter words_in_quote. For this example, you'll need to focus on stop words in "english":

```
>>>
>>> stop_words = set(stopwords.words("english"))
```
Next, create an empty list to hold the words that make it past the filter:

```
>>>
>>> filtered_list = []
```
You created an empty list, filtered_list, to hold all the words in words_in_quote that aren't stop words. Now you can use stop_words to filter words_in_quote:

```
>>>
>>> for word in words_in_quote:
...     if word.casefold() not in stop_words:
...         filtered_list.append(word)
```
You iterated over words_in_quote with a for loop and added all the words that weren't stop words to filtered_list. You used .casefold() on word so you could ignore whether the letters in word were uppercase or lowercase. This is worth doing because stopwords.words('english') includes only lowercase versions of stop words.

Alternatively, you could use a list comprehension to make a list of all the words in your text that aren't stop words:

```
>>>
>>> filtered_list = [
...     word for word in words_in_quote if word.casefold() not in stop_words
... ]
```
When you use a list comprehension, you don't create an empty list and then add items to the end of it. Instead, you define the list and its contents at the same time. Using a list comprehension is often seen as more Pythonic.

Take a look at the words that ended up in filtered_list:

```
>>>
```

```
>>> filtered_list
['Sir', ',', 'protest', '.', 'merry', 'man', '!']
```

You filtered out a few words like 'am' and 'a', but you also filtered out 'not', which does affect the overall meaning of the sentence. (Worf won't be happy about this.)

Words like 'I' and 'not' may seem too important to filter out, and depending on what kind of analysis you want to do, they can be. Here's why:

- '**I**' is a pronoun, which are context words rather than content words:
  - **Content words** give you information about the topics covered in the text or the sentiment that the author has about those topics.
  - **Context words** give you information about writing style. You can observe patterns in how authors use context words in order to quantify their writing style. Once you've quantified their writing style, you can analyze a text written by an unknown author to see how closely it follows a particular writing style so you can try to identify who the author is.
- '**not**' is technically an adverb but has still been included in NLTK's list of stop words for English. If you want to edit the list of stop words to exclude 'not' or make other changes, then you can download it.

So, 'I' and 'not' can be important parts of a sentence, but it depends on what you're trying to learn from that sentence.

Conclusion:

In the above program we can learn how to use the named entity recognisation by tokenization and various other ways.

Natural Language Processing

ASSIGNMENT NO.2 :

TITLE:

Implement sentiment analysis technique for classifying the data in to positive, negative or neutral class

PROBLEM STATEMENT:

Create Python program for Sentiment Analysis

OBJECTIVE:

To understand the different concepts of Natural Language Processing

THEORY:

What is Sentiment Analysis?

Sentiment analysis is the process of classifying whether a block of text is positive, negative, or, neutral. Sentiment analysis is contextual mining of words which indicates the social sentiment of a brand and also helps the business to determine whether the product which they are manufacturing is going to make a demand in the market or not. The goal which Sentiment analysis tries to gain is to analyze people's opinion in a way that it can help the businesses expand. It focuses not only on polarity (positive, negative & neutral) but also on emotions (happy, sad, angry, etc.). It uses various Natural Language Processing algorithms such as Rule-based, Automatic, and Hybrid.

Types of Sentiment Analysis

- **Fine-grained sentiment analysis:** This depends on the polarity based. This category can be designed as very positive, positive, neutral, negative, very negative. The rating is done on the scale 1 to 5. If the rating is 5 then it is very positive, 2 then negative and 3 then neutral.

- **Emotion detection:** The sentiment happy, sad, anger, upset, jolly, pleasant, and so on come under emotion detection. It is also known as a lexicon method of sentiment analysis.
- **Aspect based sentiment analysis:** It focuses on a particular aspect like for instance, if a person wants to check the feature of the cell phone then it checks the aspect such as battery, screen, camera quality then aspect based is used.
- **Multilingual sentiment analysis:** Multilingual consists of different languages where the classification needs to be done as positive, negative, and neutral. This is highly challenging and comparatively difficult.

How does Sentiment Analysis work?

There are three approaches used:

- **Rule-based approach:** Over here, the lexicon method, tokenization, parsing comes in the rule-based. The approach is that counts the number of positive and negative words in the given dataset. If the number of positive words is greater than the negative words then the sentiment is positive else vice-versa.

- **Automatic Approach:** This approach works on the machine learning technique. Firstly, the datasets are trained and predictive analysis is done. The next process is the extraction of words from the text is done. This text extraction can be done using different techniques such as Naive Bayes, Linear Regression, Support Vector, Deep Learning like this machine learning techniques are used.
- **Hybrid Approach:** It is the combination of both the above approaches i.e. rule-based and automatic approach. The surplus is that the accuracy is high compared to the other two approaches.

Applications

Sentiment Analysis has a wide range of applications as:

- **Social Media:** If for instance the comments on social media side as Instagram, over here all the reviews are analyzed and categorized as positive, negative, and neutral.
- **Customer Service:** In the play store, all the comments in the form of 1 to 5 are done with the help of sentiment analysis

approaches.

- **Marketing Sector:** In the marketing area where a particular product needs to be reviewed as good or bad.
- **Reviewer side:** All the reviewers will have a look at the comments and will check and give the overall review of the product.

Code :

1. **Textblob**

TextBlob is a Python (2 and 3) library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

https://textblob.readthedocs.io/en/dev/

```
from textblob import TextBlob

import nltk

nltk.download('movie_reviews')

nltk.download('punkt')

from textblob.sentiments import NaiveBayesAnalyzer, PatternAnalyzer
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...

[nltk_data]   Package movie_reviews is already up-to-date!

[nltk_data] Downloading package punkt to /root/nltk_data...

[nltk_data]   Package punkt is already up-to-date!
```

```
text = 'Apple is a very succcessful company due to its visionary leadership team. They have a very bright future'
```

```
#Default Method

blob = TextBlob(text)

blob.sentiment
```

```
Sentiment(polarity=0.24625000000000002, subjectivity=0.45)
```

```
blob = TextBlob(text, analyzer=NaiveBayesAnalyzer())

blob.sentiment
```

```
Sentiment(classification='pos', p_pos=0.8980227173699891, p_neg=0.10197728263001006)
```

#example

```
text = 'There is a lockdown in Delhi due to covid19 pandemic'
```

```
blob = TextBlob(text)

blob.sentiment
```

```
Sentiment(polarity=-0.125, subjectivity=0.375)
```

2. Word-Dictionary based

Create Positive Negative word dictionary created from University of Pittsburgh mpqa corpus (link) http://mpqa.cs.pitt.edu/lexicons/subj_lexicon/

The MPQA Opinion Corpus contains news articles from a wide variety of news sources manually annotated for opinions and other private states (i.e., beliefs, emotions, sentiments, speculations, etc.)

```
pos_words = []

neg_words = []


#Ignoring neutral, both, and other polarities

with open('subjclueslen1-HLTEMNLP05.tff') as file:

    for line in file:

        line_attrib = line.split()

        word = line_attrib[2].split('=')[1] #2nd column in the file

        polarity = line_attrib[-1].split('=')[1] #last column in the file

        if polarity =='positive':

            pos_words.append(word)

        elif polarity=='negative':

            neg_words.append(word)


print('Total positive words found: ',len(pos_words))

print('Total negative words found: ',len(neg_words))


#Write results to file for future use

with open('pos_words.txt', mode='wt', encoding='utf-8') as myfile:

    myfile.write('\n'.join(pos_words))

with open('neg_words.txt', mode='wt', encoding='utf-8') as myfile:

    myfile.write('\n'.join(neg_words))
```

Total positive words found: 2718

Total negative words found: 4911

Append your own domain specific words to positive or negative words

```
pos_word_add = ['lifted']
for term in pos_word_add:
    pos_words.append(term)

neg_word_add = ['coronavirus','covid','corona','covid19','pandemic','lockdown']
for term in neg_word_add:
    neg_words.append(term)

print('Total positive words found: ',len(pos_words))
print('Total negative words found: ',len(neg_words))
```

Total positive words found: 2720

Total negative words found: 4923

Function to calculate sentiment based on word-dictionary

```
import nltk
def calc_sentiment_based_on_word_dict(text):
    sentiment_score = 0
    words = nltk.word_tokenize(text)
    for word in words:
        if word in pos_words:
```

```
            print('pos:',word)
            sentiment_score=sentiment_score+1
        if word in neg_words:
            print('neg:',word)
            sentiment_score=sentiment_score-1
    return sentiment_score/len(words)
```

Example 1

```
text = 'Apple is a very succcessful company due to its visionary leadership team. They have a very bright future'
sentiment = calc_sentiment_based_on_word_dict(text)
print('The sentiment score of this text is: {:.2f}'.format(sentiment) )
```

pos: visionary

pos: bright

The sentiment score of this text is: 0.11

Example2

```
text = 'There is a lockdown in Chicago due to covid19 pandemic'
sentiment = calc_sentiment_based_on_word_dict(text)
print('The sentiment score of this text is: {:.2f}'.format(sentiment) )
```

neg: lockdown

neg: covid19

neg: pandemic

The sentiment score of this text is: -0.30

3. Named Entity based Sentiment Analysis (Targetted)

Spacy based Named Entity recognition

```
pip install spacy
```

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/

Requirement already satisfied: spacy in /usr/local/lib/python3.7/dist-packages (3.4.1)

Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from spacy) (57.4.0)

Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.7/dist-packages (from spacy) (1.0.8)

Requirement already satisfied: thinc<8.2.0,>=8.1.0 in /usr/local/lib/python3.7/dist-packages (from spacy) (8.1.0)

Requirement already satisfied: pathy>=0.3.5 in /usr/local/lib/python3.7/dist-packages (from spacy) (0.6.2)

Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy) (3.0.7)

Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from spacy) (1.0.3)

Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from spacy) (2.11.3)

Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.7/dist-packages (from spacy) (3.3.0)

Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.7/dist-packages (from spacy) (2.0.8)

Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.10.0,>=1.7.4 in /usr/local/lib/python3.7/dist-packages (from spacy) (1.9.2)

Requirement already satisfied: typing-extensions<4.2.0,>=3.7.4 in /usr/local/lib/python3.7/dist-packages (from spacy) (4.1.1)

Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.9 in /usr/local/lib/python3.7/dist-packages (from spacy) (3.0.10)

Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-packages (from spacy) (1.21.6)

Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.7/dist-packages (from spacy) (2.23.0)

Requirement already satisfied: wasabi<1.1.0,>=0.9.1 in /usr/local/lib/python3.7/dist-packages (from spacy) (0.10.1)

Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from spacy) (21.3)

Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.7/dist-packages (from spacy) (4.64.1)

Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.7/dist-packages (from spacy) (2.4.4)

Requirement already satisfied: typer<0.5.0,>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from spacy) (0.4.2)

Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy) (2.0.6)

Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from catalogue<2.1.0,>=2.0.6->spacy) (3.8.1)

Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=20.0->spacy) (3.0.9)

Requirement already satisfied: smart-open<6.0.0,>=5.2.1 in /usr/local/lib/python3.7/dist-packages (from pathy>=0.3.5->spacy) (5.2.1)

Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (1.24.3)

Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.0.4)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (2022.6.15)

Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (2.10)

Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.7/dist-packages (from thinc<8.2.0,>=8.1.0->spacy) (0.7.8)

Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.7/dist-packages (from typer<0.5.0,>=0.3.0->spacy) (7.1.2)

Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2->spacy) (2.0.1)

```
import spacy
from spacy import displacy
```

```
nlp = spacy.load("en_core_web_sm")


def spacy_ner(text):
    text = text.replace('\n', ' ')
    doc = nlp(text)
    entities = []
    labels = []
    position_start = []
    position_end = []

    for ent in doc.ents:
        if ent.label_ in ['PERSON','ORG','GPE']:
            entities.append(ent)
            labels.append(ent.label_)
    return entities,labels
```

```
def fit_ner(df):
    """The dataframe should have a column named 'text'"""
    print('Fitting Spacy NER model...')
    ner = df['text'].apply(spacy_ner)
    ner_org = {}
    ner_per = {}
    ner_gpe = {}
```

```
    for x in ner:
        #print(list(x))
        for entity, label in zip(x[0],x[1]):
            #print(type(entity.text))
            if label =='ORG':
                ner_org[entity.text] = ner_org.get(entity.text,0) + 1
            elif label=='PERSON':
                ner_per[entity.text] = ner_per.get(entity.text,0) + 1
            else:
                ner_gpe[entity.text] = ner_gpe.get(entity.text,0) + 1

    return {'ORG':ner_org,'PER':ner_per,'GPE':ner_gpe}


named_entities = fit_ner(news_df)
```

Fitting Spacy NER model...

Organization Named Entities

```
named_entities['ORG']
```

{'Target': 3, 'solidarity': 1, 'Shawn Dromgoole': 1, 'Lego': 1, 'Chemical': 1, 'UNSW Sydney': 1, 'Advanced Energy Materials': 1, 'UNSW's School of Chemical Engineering': 1, 'the Research Institute of Horticulture': 1, 'Kamryn & Friends': 1, 'Justice': 1, 'NFL': 1, 'Senate': 3, 'Criminal Justice Commission': 1, 'Good News Network': 1, 'House': 1, 'Portland State University': 1, 'The Star Spangled Banner': 1, 'the Portland Opera': 1, 'GNN': 2, 'Henreid': 1, 'Oxford University': 1, 'BBC': 1, 'the Murghazar Zoo': 1, 'the Pakistan High Court': 1, 'Kaavan': 1, 'Islamabad Wildlife Management Board': 1, 'Capitol Hill': 2, 'the Great American Outdoors Act': 4, 'the National Parks Service': 2, 'the Forest Service': 2, 'the Bureau of Indian Education': 2, 'the Bureau of Land Management': 2, 'the Fish and Wildlife Service': 2, 'the Land and Water Conservation Fund': 2, 'the Natural Resources Management Act': 2, 'the US House of Representatives': 2, 'the Conservation Fund': 2, 'John Abbott College': 1, 'Consumer Reports': 2, 'Whole Foods': 3, 'FDA': 3, 'CNN Business': 1, 'Jayaraj': 2, 'First

Information Report': 2, 'Magistrate': 3, 'NDTV': 2, 'St. Bernard's Hospital': 1, 'Trinity Hospital': 1, 'the University of Chicago Medical Center': 1, 'Mercy Hospital': 1, 'Deboni': 6, 'Nolan': 2, 'DUI': 2, 'THC': 1, 'Federman': 1, 'Chicago Police Supt': 1, 'Madurai': 1, 'High Court': 1, 'Human Rights': 1, "People's Watch": 1, 'the National Oceanic & Atmospheric Administration': 2, 'Sonoma Tech Meteorologist': 2, 'NOAA': 1, 'Comer Children's Hospital': 1, 'Deutsch': 1, 'Shades of Darkness Window Tinting': 1, 'Chrysler': 1, 'Palos Community Hospital': 1, 'the Phoolbagan Police Station': 1, 'DCP': 3, 'Muralidhar Agarwal': 1, 'the Kolkata Police': 1, 'the Mahadevpura Police': 1, 'Amit Agarwal': 2, 'Sooraj': 1, 'Kollam Rural Police': 1, 'Duke University': 2, 'Fahrenheit': 2, 'the National Climate Assessment': 1, 'Nasdaq': 2, 'Dow': 2, 'MUFG': 1, 'United Airlines': 1, 'Delta': 1, 'Bleakley Advisory Group': 1, 'Carnival': 1, 'Royal Caribbean': 1, 'Disney': 1, 'The Santa Fe Police Department': 1, 'SFR': 1, 'FBI': 2}

Person Named Entities

```
named_entities['PER']
```

{'Target': 1, 'Brian Cornell': 1, 'Shawn': 1, 'Max Melia': 1, 'Max': 2, 'Rahman Daiyan': 1, 'Dr Emma Lovell': 1, 'Rose Amal': 1, 'Wolfgang Palme': 1, '-11º Celsius': 1, 'Kamryn Johnson': 1, 'Kamryn': 1, 'Ron Johnson': 1, 'George Floyd': 3, 'Justin Amash': 1, 'Tim Scott': 1, 'Scott': 1, 'Madisen Hallberg': 1, 'Emmanuel Henreid': 1, 'Hallberg': 1, 'Peter Horby': 1, 'Martin Landry': 1, 'Horby': 1, 'Cher': 1, 'Kaavan': 1, 'April Judd': 1, 'Jody McDowell': 1, 'Tamil Nadu': 2, 'Pennis': 4, 'Santhankulam': 2, 'Arun Balagopalan': 2, 'Jessie D. Gregory': 1, 'Fernwood': 1, 'Gresham': 1, 'Stephen Nolan': 1, 'Kevin Deboni': 1, 'Nolan': 6, 'Caleb White': 1, 'Irving Federman': 1, 'David Brown': 1, 'Brown': 1, 'Stroger Hospital': 1, 'Stroger': 2, 'J. Jones': 1, 'E Palaniswami': 1, 'Kanimozhi': 1, 'Henri Tiphagne': 1, 'Saharan': 1, 'Jeff Beamish': 2, 'Wayne Deutsch': 1, 'Deutsch': 3, 'Jasean Francis': 1, 'Charles Riley': 1, 'Puma': 1, 'Kolkata': 3, 'Lalita Dhandhania': 1, 'Shilpi': 1, 'Mrs Shilpi Agarwal': 1, 'Mahadevpura PS': 1, 'Uthra': 2, 'Hari Sankar': 1, 'Complications': 1, 'Chris Rupkey': 1, 'Peter Boockvar': 1, 'Retailer Gap': 1, 'Paul Joye': 1, 'Joye': 1, 'Bajit Singh': 1}

Geopolitical Named Entities

```
named_entities['GPE']
```

{'U.S.': 7, 'Nashville': 1, 'Lower Austria': 1, 'Fahrenheit': 1, 'Palme': 2, 'Austria': 1, 'Minneapolis': 3, 'Kamryn': 2, 'Minnesota': 1, 'Colorado': 1, 'Kentucky': 1, 'Oregon': 1, 'UK': 2, 'Islamabad': 1, 'Sri Lanka': 1, 'Pakistan': 1, 'America': 4, 'Yellowstone': 2, 'Quebec': 2, 'Shawville': 1, 'Tennessee': 1, 'Baltimore': 1, 'Maryland': 1, 'Starkey': 1, 'Chicago': 8, 'Chatham': 1, 'Cook County': 3, 'Bridgeport': 1, 'Schaumburg': 2, 'Las Vegas': 1, 'Austin': 2, 'the United States': 2, 'Florida': 6, 'Texas': 9, 'Louisiana': 2, 'Little Village': 1, 'New Lenox': 1, 'Will County': 2, 'Orland Park': 1, 'Subhas': 1, 'Kolkata': 1, 'Bengaluru': 2, 'Whitefield': 2, 'India': 1, 'Sooraj': 1, 'Kerala': 2, 'Kollam': 1, 'California': 1, 'New York': 2, 'New Jersey': 2, 'Connecticut': 2, 'Arizona': 1, 'Orlando': 1}

Sentiment Analysis based on top named entities

Targetted by finding sentences containing the named entities and performing sentiment analysis only on those sentences one by one

```
from textblob import TextBlob

import nltk
```

```
def get_keyword_sentences(text,keyword):

    """Extract sentences containing the Named Entity/Keyword"""

    text = text.replace('\n', ' ')

    sentences = nltk.sent_tokenize(text)


    sent_of_interest = []

    for sent in sentences:

        if keyword in sent.lower():

            sent_of_interest.append(sent)

    return sent_of_interest if len(sent_of_interest)>0 else False
```

```
def get_sentiment(list_of_sent):

    """ Extract sentiment from a sentence using TextBlob Pattern Analyzer"""

    sentiment = 0

    count = 0


    if list_of_sent !=False:

        for sent in list_of_sent:

            blob = TextBlob(sent)

            sentiment += blob.sentiment.polarity

            count += len(sent)

    return sentiment/count if count!=0 else 0
```

```
def extract_sentiment(df,keywords,top=10):
    """df: data for the cluster, keywords: ner for that cluster"""
    print('Extracting Sentiments using TextBlob...')
    keywords = sorted(keywords.items(),key=lambda x: x[1],reverse=True)
    sentiment_dict = {}
    for keyword,count in keywords[:top]:
        df['sentences'] = df['text'].apply(get_keyword_sentences,keyword=keyword.lower())
        keyword_sentiment = df['sentences'].apply(get_sentiment).sum()
        sentiment_dict[keyword] = keyword_sentiment
    return sentiment_dict
```

Top 30 Organizations according to their sentiments

```
keywords = named_entities['ORG']
sentiment_result_org = extract_sentiment(news_df,keywords,top=30)
sentiment_result_org
sorted(sentiment_result_org.items(),key=lambda x:x[1], reverse=True)
```

```
Extracting Sentiments using TextBlob...
[('GNN', 0.004569909563803812),
 ('the US House of Representatives', 0.0019907100199071),
 ('the Conservation Fund', 0.0019907100199071),
 ('the National Parks Service', 0.0017921146953405018),
 ('the Land and Water Conservation Fund', 0.0013941535496305815),
 ('the Natural Resources Management Act', 0.0013941535496305815),
 ('Target', 0.0012131536833410769),
```

('the Great American Outdoors Act', 0.0011459492888064317),

('Capitol Hill', 0.000883481597767312),

('Senate', 0.0007037413380270522),

('FDA', 0.0005633802816901409),

('Whole Foods', 0.0004136029411764706),

('DUI', 0.0003423736143555005),

('Nolan', 0.00020554291972032683),

('Duke University', 0.0002017213555675094),

('the Forest Service', 0.0),

('the Bureau of Indian Education', 0.0),

('the Bureau of Land Management', 0.0),

('the Fish and Wildlife Service', 0.0),

('Jayaraj', 0.0),

('NDTV', 0.0),

('Sonoma Tech Meteorologist', 0.0),

('Consumer Reports', -5.604395604395601e-05),

('First Information Report', -0.00020449897750511242),

('Deboni', -0.0003628628628628629),

('the National Oceanic & Atmospheric Administration', -0.0005025125628140704),

('Amit Agarwal', -0.0006304347826086958),

('Magistrate', -0.0006684491978609625),

('DCP', -0.0010416666666666667),

('Fahrenheit', -0.0010437154242313098)]

Top 30 Persons according to their sentiments

```
keywords = named_entities['PER']

sentiment_result_org = extract_sentiment(news_df,keywords,top=30)
```

```
sentiment_result_org

sorted(sentiment_result_org.items(),key=lambda x:x[1], reverse=True)
```

```
Extracting Sentiments using TextBlob...
[('Brian Cornell', 0.002359882005899705),
 ('Kamryn', 0.0018972048066875655),
 ('Hallberg', 0.0014084507042253522),
 ('Ron Johnson', 0.0013736263736263737),
 ('Target', 0.0012131536833410769),
 ('Shawn', 0.001013856032443393),
 ('Rose Amal', 0.0009461358313817331),
 ('Max Melia', 0.0006799163179916319),
 ('Rahman Daiyan', 0.0006195786864931846),
 ('Dr Emma Lovell', 0.0006195786864931846),
 ('George Floyd', 0.0005177020708082832),
 ('Tim Scott', 0.0003012048192771084),
 ('Scott', 0.00023961661341853033),
 ('Nolan', 0.00020554291972032683),
 ('Wolfgang Palme', 5.8685446009389656e-05),
 ('Deutsch', 5.835667600373482e-05),
 ('Santhankulam', 0.0),
 ('Arun Balagopalan', 0.0),
 ('Jeff Beamish', 0.0),
 ('Madisen Hallberg', 0.0),
 ('Emmanuel Henreid', 0.0),
 ('Pennis', -0.00021367521367521362),
 ('Uthra', -0.0003479236812570146),
 ('Max', -0.0005604288499025341),
 ('Justin Amash', -0.0005889281507656066),
 ('-11º Celsius', -0.001097560975609756),
 ('Kolkata', -0.001114900314795383),
 ('Stroger', -0.002560975609756097),
 ('Tamil Nadu', -0.00257826887661141 8),
 ('Kamryn Johnson', -0.00641025641025641)]
```

Application: Analyzing Sentiment in a book

```
book = open('book_cab_and_goose.txt',encoding="utf8").read()
```

```
blob = TextBlob(book, analyzer=NaiveBayesAnalyzer())

blob.sentiment
```

```
Sentiment(classification='pos', p_pos=1.0, p_neg=6.5736053249909995e-257)
```

```
blob = TextBlob(book)

blob.sentiment
```

Sentiment per sentence

```
blob = TextBlob(book)


polarity = []

subjectivity = []

sentences = []

PatternAnalyzerDF = pd.DataFrame(columns=['sentence', 'polarity', 'subjectivity'])
```

```
for sentence in blob.sentences:

    polarity.append(sentence.sentiment.polarity)

    subjectivity.append(sentence.sentiment.subjectivity)

    sentences.append(str(sentence.raw))


PatternAnalyzerDF['sentence'] = sentences

PatternAnalyzerDF['polarity'] = polarity

PatternAnalyzerDF['subjectivity'] = subjectivity


PatternAnalyzerDF['sentence'] = PatternAnalyzerDF['sentence'].str.replace('\n', ' ')

PatternAnalyzerDF.head(10)
```

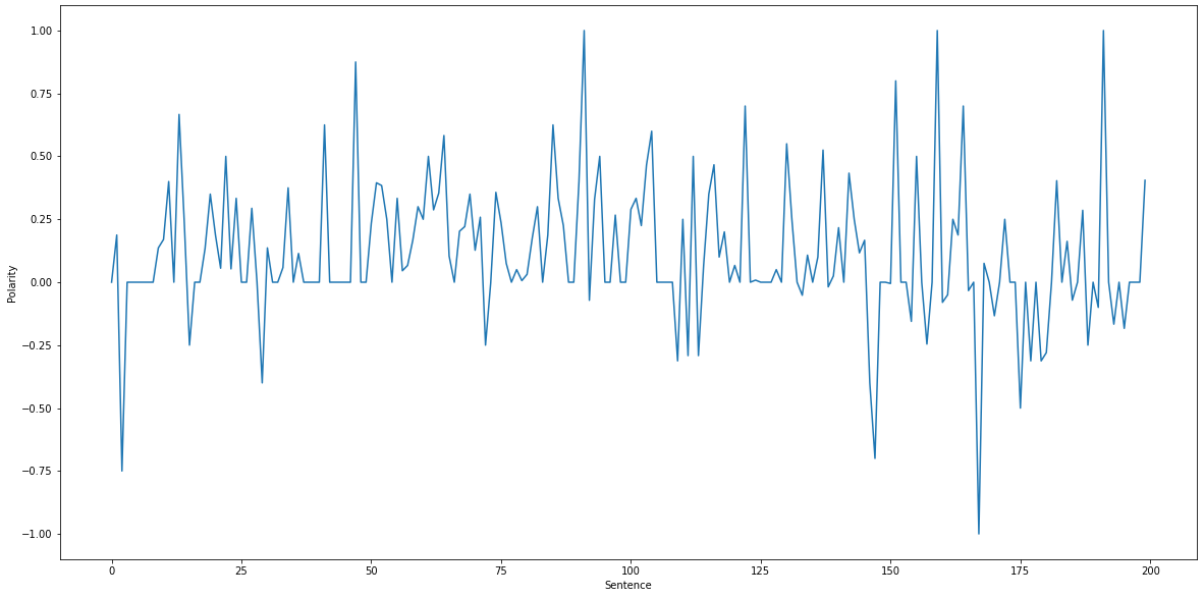
	sentence	polarity	subjectivity
0	The Project Gutenberg eBook, Cab and Caboose, by Kirk Munroe This eBook	0.000000	0.000000

	sentence	polarity	subjectivity
	is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever.		
1	You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this eBook or online at www.gutenberg.org Title: Cab and Caboose The Story of a Railroad Boy Author: Kirk Munroe Release Date: September 4, 2007 [eBook #22497] Language: English ***START OF THE PROJECT GUTENBERG EBOOK CAB AND CABOOSE*** E-text prepared by Mark C. Orton, Linda McKeown, Anne Storer, and the Project Gutenberg Online Distributed Proofreading Team (http://www.pgdp.net) Note: Project Gutenberg also has an HTML version of this file which includes the original illustrations.	0.187500	0.375000
2	See 22497-h.htm or 22497-h.zip: (http://www.gutenberg.net/dirs/2/2/4/9/22497/22497-h/22497-h.htm) or (http://www.gutenberg.net/dirs/2/2/4/9/22497/22497-h.zip) CAB AND CABOOSE The Story of a Railroad Boy by KIRK MUNROE OFFICERS OF THE NATIONAL COUNCIL Honorary President, THE HON.	-0.750000	1.000000
3	WOODROW WILSON Honorary Vice-President, HON.	0.000000	0.000000
4	WILLIAM H. TAFT Honorary Vice-President, COLONEL THEODORE ROOSEVELT President, COLIN H. LIVINGSTONE, Washington, D. C. Vice-President, B. L. DULANEY, Bristol, Tenn.	0.000000	0.000000
5	Vice-President, MILTON A. McRAE, Detroit.	0.000000	0.000000
6	Mich.	0.000000	0.000000
7	Vice-President, DAVID STARR JORDAN, Stanford University, Cal.	0.000000	0.000000
8	Vice-President, F. L. SEELY, Asheville, N. C. Vice-President, A. STAMFORD WHITE, Chicago, Ill. Chief Scout, ERNEST THOMPSON SETON, Greenwich, Connecticut National Scout Commissioner, DANIEL CARTER BEARD, Flushing, N. Y.	0.000000	0.000000
9	NATIONAL HEADQUARTERS BOY SCOUTS OF AMERICA THE FIFTH	0.136364	0.454545

	sentence	polarity	subjectivity
	AVENUE BUILDING, 200 FIFTH AVENUE TELEPHONE GRAMERCY 545 NEW YORK CITY FINANCE COMMITTEE John Sherman Hoyt, Chairman August Belmont George D. Pratt Mortimer L. Schiff H. Rogers Winthrop GEORGE D. PRATT, Treasurer JAMES E. WEST, Chief Scout Executive ADDITIONAL MEMBERS OF THE EXECUTIVE BOARD Ernest P. Bicknell Robert Garrett Lee F. Hanmer John Sherman Hoyt Charles C. Jackson Prof. Jeremiah W. Jenks William D. Murray Dr. Charles P. Neill George D. Porter Frank Presbrey Edgar M. Robinson Mortimer L. Schiff Lorillard Spencer Seth Sprague Terry July 31st, 1913.		

Plotting sentiment variation in the book as the story is unveiled

```
import matplotlib.pyplot as plt

PatternAnalyzerDF.sort_index(inplace=True)

sentiment_top_df = PatternAnalyzerDF.head(n=200)

pd.set_option('display.max_colwidth', 200)

plt.figure().set_size_inches(20, 10)

plt.plot(sentiment_top_df['polarity'])

plt.xlabel('Sentence')

plt.ylabel('Polarity')

plt.show()
```



Conclusion

In this Assignment we learned about 3 different ways of performing sentiment analysis

EXPERIMENT NUMBER 03

TITLE: -

Use of Natural Language Processing technique for text summarization

OBJECTIVE: -

To know/understand the concept of text summarization

THEORY: -

-What Is Text Summarization in NLP?

Text summarization is the practice of breaking down long publications into manageable paragraphs or sentences. The procedure extracts important information while also ensuring that the paragraph's sense is preserved. This shortens the time it takes to comprehend long materials like research articles while without omitting critical information.

The process of constructing a concise, cohesive, and fluent summary of a lengthier text document, which includes highlighting the text's important points, is known as text summarization.

-Requirement of text summarization

The amount of text data available from various sources has exploded in the big data age. This large volume of text has a wealth of information and expertise that must be adequately summarized to be useful**.**

Because of the growing availability of documents, much research in the field of natural language processing (NLP) for automatic text summarization is required. The job of creating a succinct and fluent summary without the assistance of a person while keeping the sense of the original text material is known as automatic text summarization.

-Different types of Text summarization

Text Summarization is classified on different bases. They are:

- *the basis of the Outcome*

There are mainly two types of text summarization in NLP:

1. Extraction-based summarization

   The extractive text summarizing approach entails extracting essential words from the source material and combining them to create a summary.

2. Abstractive Summarization

   Another way of text summarization is abstractive summarization. We create new sentences from the original content in this step.

- *On the basis of Context*

3. Domain-Specific

In domain-specific summarization, domain knowledge is applied. Specific context, knowledge, and language can be merged using domain-specific summarizers. For example, models can be combined with the terminology used in medical science so that they can better grasp and summarize scientific texts.

4. Query-based

Query-based summaries are primarily concerned with natural language questions. This is similar to the search results on Google.

5. Generic

Generic summarizers, unlike domain-specific or query-based summarizers, are not programmed to make any assumptions. The content from the source document is simply condensed or summarized.

#Code –

```
import spacy
nlp = spacy.load("en_core_web_sm")
```

```
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
```

```
# Build a List of Stopwords-->
stopwords = list(STOP_WORDS)
```

```
doc1="""Python is a high-level, general-
purpose programming language. Its design philosophy emphasizes code readability with the use of signifi
cant indentation.
Python is dynamically-typed and garbage-
collected. It supports multiple programming paradigms, including structured (particularly procedural), obj
ect-
oriented and functional programming. It is often described as a "batteries included" language due to its co
mprehensive standard library.
```

```
Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming la
nguage and first released it in 1991 as Python 0.9.0.[36] Python 2.0 was released in 2000 and introduced
new features such as list comprehensions, cycle-
detecting garbage collection, reference counting, and Unicode support.
Python 3.0, released in 2008, was a major revision that is not completely backward-
compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020."""
```

```
nlp =spacy.load('en_core_web_sm')
```

```
#Building an NLP object-->
docx = nlp(doc1)
```

```
# Tokenization of Text
mytokens = [token.text for token in docx]
```

```
#For calculating the frequencies of each word that are not the stopwords-->
word_freq={}
for word in docx:
    if word.text not in stopwords:
        if word.text not in word_freq.keys():
            word_freq[word.text]=1;

        else:
            word_freq[word.text]+=1;


#Here are all the word frequencies-->
word_freq
```

```
'paradigms': 1,
 'including': 1,
 'structured': 1,
 '(': 1,
 'particularly': 1,
 'procedural': 1,
 ')': 1,
 'object': 1,
 'oriented': 1,
 'functional': 1,
 'described': 1,
 '"': 2,
 'batteries': 1,
 'included': 1,
 'comprehensive': 1,
```

```
 'standard': 1,
 'library': 1,
 'Guido': 1,
 'van': 1,
 'Rossum': 1,
 'began': 1,
 'working': 1,
 'late': 1,
 '1980s': 1,
 'successor': 1,
 'ABC': 1,
 'released': 3,
 '1991': 1,
 '0.9.0.[36': 1,
 ']': 1,
 '2.0': 1,
 '2000': 1,
 'introduced': 1,
 'new': 1,
 'features': 1,
 'list': 1,
 'comprehensions': 1,
 'cycle': 1,
 'detecting': 1,
 'collection': 1,
 'reference': 1,
 'counting': 1,
 'Unicode': 1,
 'support': 1,
 '3.0': 1,
 '2008': 1,
 'major': 1,
 'revision': 1,
 'completely': 1,
 'backward': 1,
 'compatible': 1,
 'earlier': 1,
 'versions': 1,
 '2': 1,
 'discontinued': 1,
 'version': 1,
 '2.7.18': 1,
 '2020': 1}
```

```
max_freq=max(word_freq.values())

for word in word_freq:
   word_freq[word]=(word_freq[word]/max_freq)
```

```
print(word_freq)
```

{'Python': 0.875, 'high': 0.125, '-': 0.875, 'level': 0.125, ',': 1.0, 'general': 0.125, 'purpose': 0.125, 'programming': 0.5, 'language': 0.375, '.': 1.0, 'Its': 0.125, 'design': 0.125, 'philosophy': 0.125, 'emphasizes': 0.125, 'code': 0.125, 'readability': 0.125, 'use': 0.125, 'significant': 0.125, 'indentation': 0.125, '\n': 0.375, 'dynamically': 0.125, 'typed': 0.125, 'garbage': 0.25, 'collected': 0.125, 'It': 0.25, 'supports': 0.125, 'multiple': 0.125, 'paradigms': 0.125, 'including': 0.125, 'structured': 0.125, '(': 0.125, 'particularly': 0.125, 'procedural': 0.125, ')': 0.125, 'object': 0.125, 'oriented': 0.125, 'functional': 0.125, 'described': 0.125, '"': 0.25, 'batteries': 0.125, 'included': 0.125, 'comprehensive': 0.125, 'standard': 0.125, 'library': 0.125, 'Guido': 0.125, 'van': 0.125, 'Rossum': 0.125, 'began': 0.125, 'working': 0.125, 'late': 0.125, '1980s': 0.125, 'successor': 0.125, 'ABC': 0.125, 'released': 0.375, '1991': 0.125, '0.9.0.[36': 0.125, ']': 0.125, '2.0': 0.125, '2000': 0.125, 'introduced': 0.125, 'new': 0.125, 'features': 0.125, 'list': 0.125, 'comprehensions': 0.125, 'cycle': 0.125, 'detecting': 0.125, 'collection': 0.125, 'reference': 0.125, 'counting': 0.125, 'Unicode': 0.125, 'support': 0.125, '3.0': 0.125, '2008': 0.125, 'major': 0.125, 'revision': 0.125, 'completely': 0.125, 'backward': 0.125, 'compatible': 0.125, 'earlier': 0.125, 'versions': 0.125, '2': 0.125, 'discontinued': 0.125, 'version': 0.125, '2.7.18': 0.125, '2020': 0.125}

```
# Sentence Tokens-->
sentence_list = [ sentence for sentence in docx.sents]
```

```
#Lowering all the text-->
[w.text.lower() for t in sentence_list for w in t ]
```

['python', 'is', 'a', 'high', '-', 'level', ',', 'general', '-', 'purpose', 'programming', 'language', '.', 'its', 'design', 'philosophy', 'emphasizes', 'code', 'readability', 'with', 'the', 'use', 'of', 'significant', 'indentation', '.', '\n', 'python', 'is', 'dynamically', '-', 'typed', 'and', 'garbage', '-', 'collected', '.', 'it', 'supports', 'multiple', 'programming', 'paradigms', ',', 'including', 'structured', '(', 'particularly', 'procedural', ')', ',', 'object', '-', 'oriented', 'and', 'functional', 'programming', '.', 'it', 'is', 'often', 'described', 'as', 'a', '"', 'batteries', 'included', '"', 'language', 'due', 'to', 'its', 'comprehensive', 'standard', 'library', '.', '\n', 'guido', 'van', 'rossum', 'began', 'working', 'on', 'python', 'in', 'the', 'late', '1980s', 'as', 'a', 'successor', 'to', 'the', 'abc', 'programming', 'language', 'and', 'first', 'released', 'it', 'in', '1991', 'as', 'python', '0.9.0.[36', ']', 'python', '2.0', 'was', 'released', 'in', '2000', 'and', 'introduced', 'new', 'features', 'such', 'as', 'list', 'comprehensions', ',', 'cycle', '-', 'detecting', 'garbage', 'collection', ',', 'reference', 'counting', ',', 'and', 'unicode', 'support', '.', '\n', 'python', '3.0', ',', 'released', 'in', '2008', ',', 'was', 'a', 'major', 'revision', 'that', 'is', 'not', 'completely', 'backward', '-', 'compatible', 'with', 'earlier', 'versions', '.', 'python', '2', 'was', 'discontinued', 'with', 'version', '2.7.18', 'in', '2020', '.']

```
sentence_list
```

[Python is a high-level, general-purpose programming language., Its design philosophy emphasizes code readability with the use of significant indentation., Python is dynamically-typed and garbage-collected., It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming., It is often described as a "batteries included" language due to its comprehensive standard library., Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0.[36] Python 2.0 was released in 2000 and introduced new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. , Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions., Python 2 was discontinued with version 2.7.18 in 2020.]

```
#Calculating Sentence Scores-->
```

```
sentence_scores={}
for sentence in sentence_list:
   for word in sentence:
      if word.text in word_freq.keys():
         if len(sentence.text.split(' ')) < 30:
            if sentence not in sentence_scores.keys():
               sentence_scores[sentence] = word_freq[word.text]
            else:
               sentence_scores[sentence] += word_freq[word.text]
```

```
#Displaying All the Sentence Scores-->
sentence_scores
```

{Python is a high-level, general-purpose programming language.: 6.0, Its design philosophy emphasizes code readability with the use of significant indentation.: 2.5, Python is dynamically-typed and garbage-collected.: 4.25, It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming.: 6.625, It is often described as a "batteries included" language due to its comprehensive standard library.: 3.25, Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions.: 6.25, Python 2 was discontinued with version 2.7.18 in 2020.: 2.5}

```
#Finding Top N sentences using heapq-->
from heapq import nlargest
```

```
sum_sentences = nlargest(10, sentence_scores, key=sentence_scores.get)
```

```
sum_sentences
```

[It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming., Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions., Python is a high-level, general-purpose programming language., Python is dynamically-typed and garbage-collected., It is often described as a "batteries included" language due to its comprehensive standard library., Its design philosophy emphasizes code readability with the use of significant indentation., Python 2 was discontinued with version 2.7.18 in 2020.]

```
for w in sum_sentences:
   print(w.text)

# List Comprehension of Sentences Converted From Spacy.span to strings
final_sentences = [ w.text for w in sum_sentences]
final_sentences
```

It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming.
Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions.
Python is a high-level, general-purpose programming language.

Python is dynamically-typed and garbage-collected.
It is often described as a "batteries included" language due to its comprehensive standard library.

Its design philosophy emphasizes code readability with the use of significant indentation.

Python 2 was discontinued with version 2.7.18 in 2020.
['It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming.',
 'Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions.',
 'Python is a high-level, general-purpose programming language.',
 'Python is dynamically-typed and garbage-collected.',
 'It is often described as a "batteries included" language due to its comprehensive standard library.\n',
 'Its design philosophy emphasizes code readability with the use of significant indentation.\n',
 'Python 2 was discontinued with version 2.7.18 in 2020.']

```
summary = ' '.join(final_sentences)
summary
```

It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python is a high-level, general-purpose programming language. Python is dynamically-typed and garbage-collected. It is often described as a "batteries included" language due to its comprehensive standard library.\n Its design philosophy emphasizes code readability with the use of significant indentation.\n Python 2 was discontinued with version 2.7.18 in 2020.

```
len(summary)
```

613

```
len(doc1)
```

935

#Conclusion: -

Knowing the implementation of a technique for text summarization in natural language processing

Experiment No 4

TITLE: -

Implement Simple Machine translation from one language to another.

OBJECTIVE: -

To know/understand the concept of simple machine translation.

THEORY: -

What is a machine translation and how does it work?

Machine Translation or MT or robotized interpretation is simply a procedure when a computer software translates text from one language to another without human contribution. At its fundamental level, machine translation performs a straightforward replacement of atomic words in a single characteristic language for words in another.

Using corpus methods, more complicated translations can be conducted, taking into account better treatment of contrasts in phonetic typology, express acknowledgement, and translations of idioms, just as the seclusion of oddities. Currently, some systems are not able to perform just like a human translator, but in the coming future, it will also be possible.

In simple language, we can say that ***machine translation works by using computer software to translate the text from one source language to another target language.*** There are different types of machine translation and in the next section, we will discuss them in detail.

Different types of machine translation in NLP

There are four types of machine translation:

1. **Statistical Machine Translation or SMT**

   It works by alluding to statistical models that depend on the investigation of huge volumes of bilingual content. It expects to decide the correspondence between a word from the source language and a word from the objective language. ***A genuine illustration of this is Google Translate.***

   Presently, SMT is extraordinary for basic translation, however its most noteworthy disadvantage is that it doesn't factor in context, which implies translation can regularly be wrong or you can say, don't expect great quality translation. There are several types of statistical-based machine translation models which are: ***Hierarchical phrase-based translation, Syntax-based translation, Phrase-based translation, Word-based translation.***

Types of Machine Translation

2. **Rule-based Machine Translation or RBMT**

RBMT basically translates the basics of ***grammatical rules***. It directs a grammatical examination of the source language and the objective language to create the translated sentence. But, RBMT requires broad editing, and its substantial reliance on dictionaries implies that proficiency is accomplished after a significant period

3. **Hybrid Machine Translation or HMT**

HMT, as the term demonstrates, is a mix of RBMT and SMT. It uses a translation memory, making it unquestionably more successful regarding quality. Nevertheless, even HMT has a lot of downsides, the biggest of which is the requirement for enormous editing, and human translators will also be needed. There are several approaches to HMT like multi-engine, statistical rule generation, multi-pass, and confidence-based.

4. **Neural Machine Translation or NMT**

NMT is a type of machine translation that relies upon neural network models (based on the human brain) to build statistical models with the end goal of translation. The essential advantage of NMT is that it gives a solitary system that can be prepared to unravel the source and target text. Subsequently, it doesn't rely upon specific systems that are regular to other machine translation systems, particularly SMT.

What are the benefits of machine translation?

One of the crucial benefits of machine translation is speed as you have noticed that computer programs ***can translate a huge amount of text rapidly.*** Yes, the human translator does their work more accurately but they cannot match the speed of the computer.

If you especially train the machine to your requirements, machine translation gives the ideal blend of brisk and cost-effective translations as ***it is less expensive*** than using a human translator. With a specially trained machine, MT can catch the setting of full sentences before translating them, which gives you high quality and human-sounding yield. Another benefit of machine translation is its ***capability to learn important words and reuse them wherever they might fit.***

Applications of machine translation

Machine translation technology and products have been used in numerous application situations, for example, business travel, the travel industry, etc. In terms of the object of translation, there are composed language-oriented content text translation and spoken language.

- **Text translation**

  Automated text translation is broadly used in an assortment of sentence-level and text-level translation applications. Sentence-level translation applications

incorporate the translation of inquiry and recovery inputs and the translation of (OCR) outcomes of picture optical character acknowledgement. Text-level translation applications incorporate the translation of a wide range of unadulterated reports, and the translation of archives with organized data.

Organized data mostly incorporates the presentation configuration of text content, object type activity, and other data, for example, textual styles, colours, tables, structures, hyperlinks, etc. Presently, the translation objects of machine translation systems are mostly founded on the sentence level.

Most importantly, a sentence can completely communicate a subject substance, which normally frames an articulation unit, and the significance of each word in the sentence can be resolved to an enormous degree as per the restricted setting inside the sentence.

Also, the methods and nature of getting data at the sentence level granularity from the preparation corpus are more effective than that dependent on other morphological levels, for example, words, expressions, and text passages. Finally, the translation depends on sentence-level can be normally reached out to help translation at other morphological levels.

- **Speech translation**

With the fast advancement of mobile applications, voice input has become an advantageous method of human-computer cooperation, and discourse translation has become a significant application situation. The fundamental cycle of

discourse interpretation is "source language discourse source language text-target language text-target language discourse".

In this cycle, programmed text translation from source language text to target-language text is an important moderate module. What's more, the front end and back end likewise need programmed discourse recognition, ASR and text-to-speech, TTs.

- **Other applications**

Naturally, the task of machine translation is to change one source language word succession into another objective language word grouping which is semantically the same. Generally, it finishes a grouping transformation task, which changes over a succession object into another arrangement object as indicated by some information and rationale through model and algorithms.

All things considered, many undertaking situations total the change between grouping objects, and the language in the machine translation task is just one of the succession object types. In this manner, when the ideas of the source language and target language are stretched out from dialects to other arrangement object types, machine translation strategies and techniques can be applied to settle numerous comparable change undertakings.

Machine Translation vs Human translation

- Machine translation hits that sweet spot of ***cost and speed***, offering a truly snappy path for brands to translate their records at scale without much overhead. Yet, that

doesn't mean it's consistently relevant. On the other hand, human translation is incredible for those undertakings that require additional consideration and subtlety. Talented translators work on your image's substance to catch the first importance and pass on that feeling or message basically in another assortment of work.

- Leaning upon how much content should be translated, the machine translation can give translated content very quickly, though human translators will take additional time. Time spent finding, verifying, and dealing with a group of translators should likewise be considered.

- Numerous translation programming providers can give machine translations at practically zero cost, making it a reasonable answer for organizations who will be unable to manage the cost of expert translations.

- Machine Translation is the instant modification of text from one language to another utilizing artificial intelligence whereas a human translation, includes actual brainpower, in the form of one or more translators translating the text manually.

Conclusion

Machine translation is mainly equipment that assists marketers or translators to accomplish a motive. However, it is not a replacement for the old systems of translation, instead, it is a modification. Many machine-based translators considerably offer their services free of charge, making them much more alluring, especially to organizations and learners.

Machine translators analyze the structure of the first content, at that point, separate this content into single words or short expressions that can be easily translated. At last, they reconstruct those single words or short expressions using the very same structure in the picked target language.

Experiment No 5

TITLE: -

Implement a code for aspect mining and topic modeling.

OBJECTIVE: -

To know/understand the concept of aspect mining and topic mining.

THEORY: -

Sentiment Analysis (SA):

It is a technique to distinguish a person's feeling towards something or someone based on a piece of text they have written about it. It could be positive, negative or neutral. Let us consider a real-life example.

We see millions of tweets on Twitter on a daily basis. Here, we can build a sentiment analysis model to determine if their attitude towards a particular subject is happy, sad, angry or neutral. The current limitations of this technique are detecting sarcasm.

Aspect Modelling in Sentiment Analysis (ABSA):

Aspect modelling is an advanced text-analysis technique that refers to the process of breaking down the text input into aspect categories and its aspect terms and then identifying the sentiment behind each aspect in the whole text input. The two key terms in this model are:

- **Sentiments:** A positive or negative review about a particular aspect
- **Aspects:** the category, feature, or topic that is under observation.

Requirement

In the business world, there is always a major need to identify to observe the sentiment of the people towards a particular product or service to ensure their continuous interest in their business product. ABSA fulfils this purpose by identifying the sentiment behind each aspect category like food, location, ambience etc.

This helps businesses *keep track of the changing sentiment* of the customer in each field of their business.

ABSA Architecture

The ABSA model includes the following steps in order to obtain the desired output.

Step 1 - Consider the input text corpus and pre-process the dataset.

Step 2 - Create Word Embeddings of the text input. (i.e. vectorize the text input and create tokens.)

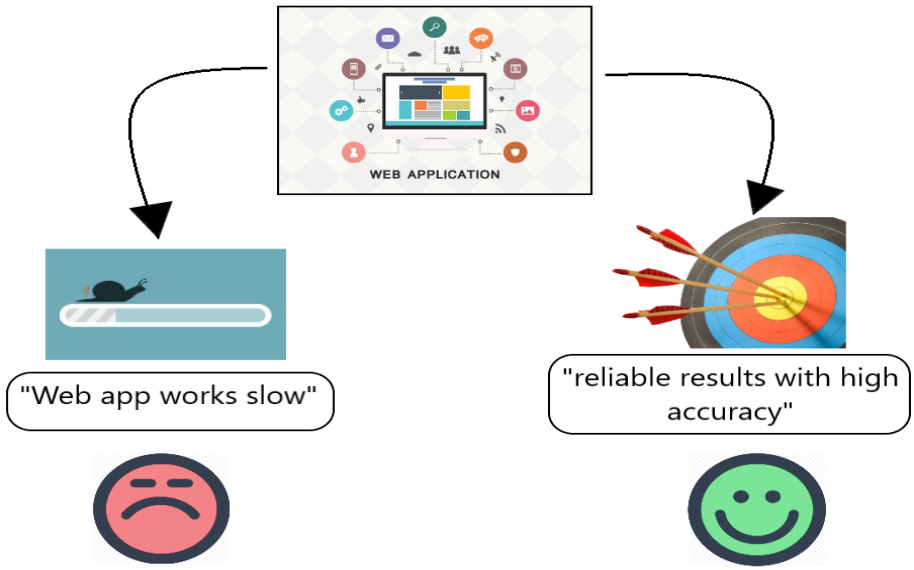
Step 3.a - Aspect Terms Extraction -> Aspect Categories Model

Step 3.b - Sentiment Extraction -> Sentiment Model

Step 4 - Combine 3.a and 3.b to create to get **Aspect Based Sentiment**.(OUTPUT)

Intuition:

Aspect: It is defined as a concept on which an opinion or a sentiment is based. Let us take an example for better understanding.
Suppose a company builds a web app that works slow but offers reliable results with high accuracy. Here, we break this text into two aspects. "*Web app works slow*" and "*reliable results with high accuracy*". On observing the two aspect categories, you can easily conclude that they have different sentiment associated with them. (As shown in Fig 1)



Tools & Dataset Used:

- spaCy (tokenization, sentence boundary detection, dependency parser, etc.)
- Scikit Learn & Scikit Multilearn (Label Powerset, MN Naive Bayes, Multilabel Binarizer, SGD classifier, Count Vectorizer & Tf-Idf, etc.)
- Word2Vec & vectors pre-trained on Google's News dataset
- Neural Coref v2.0 —pre-trained neural network model for recognizing and replacing pronouns
- Annotated Restaurant Reviews (SemEval-2014) — restaurant reviews that were manually labeled into categories
- Yelp Dataset (kaggle)
- Opinion lexicon (Minqing Hu and Bing Liu.)

Since Python is my go-to language, all of the tools and libraries I used are available for Python. If you're more comfortable with other languages, there are many other approaches you can take, i.e. using Stanford CoreNLP which normally runs on Java, instead of spaCy which runs on python.

```
# Importing the required libraries

import spacy

sp = spacy.load("en_core_web_sm")

from textblob import TextBlob

# Creating a list of positive and negative sentences.

mixed_sen = [

  'This chocolate truffle cake is really tasty',

  'This party is amazing!',

  'My mom is the best!',

  'App response is very slow!'

  'The trip to India was very enjoyable'

]

# An empty list for obtaining the extracted aspects

# from sentences.

ext_aspects = []

# Performing Aspect Extraction
```

```
for sen in mixed_sen:
  important = sp(sentence)
  descriptive_item = ''
  target = ''
  for token in important:
    if token.dep_ == 'nsubj' and token.pos_ == 'NOUN':
      target = token.text
    if token.pos_ == 'ADJ':
      added_terms = ''
      for mini_token in token.children:
        if mini_token.pos_ != 'ADV':
          continue
        added_terms += mini_token.text + ' '
      descriptive_item = added_terms + token.text
  ext_aspects.append({'aspect': target,
    'description': descriptive_item})


print("ASPECT EXTRACTION\n")
print(ext_aspects)
```

```
for aspect in ext_aspects:

  aspect['sentiment'] = TextBlob(aspect['description']).sentiment

print("\n")

print("SENTIMENT ASSOCIATION\n")

print(ext_aspects)
```

Output:

Topic Modeling:

Topic modeling is a machine learning technique that automatically analyzes text data to determine cluster words for a set of documents. This is known as 'unsupervised' machine learning because it doesn't require a predefined list of tags or training data that's been previously classified by humans. Since topic modeling doesn't require training, it's a quick and easy way to start analyzing your data. However, you can't guarantee you'll receive accurate results, which is why many businesses opt to invest time training a topic classification model. Since topic classification models require training, they're known as 'supervised' machine learning techniques. What does that mean? Well, as opposed to text modeling, topic classification needs to know the topics of a set of texts before analyzing them. Using these topics, data is tagged manually so that a topic classifier can learn and later make predictions by itself. For example, let's say you're a software company that's released a new data analysis feature, and you want to analyze what customers are saying about it. You'd first need to create a list of tags (topics) that are relevant to the new feature e.g. *Data Analysis*, *Features*, *User Experience*, then you'd need to use data samples to teach your topic classifier how to tag each text using these predefined topic tags. Although topic classification involves extra legwork, this topic analysis technique delivers more accurate results than unsupervised techniques, which means you'll get more valuable insights that help you make better, data-based decisions.

Topic Modeling

Topic Modeling refers to the process of dividing a corpus of documents in two:

1. A list of the topics covered by the documents in the corpus
2. Several sets of documents from the corpus grouped by the topics they cover.

The underlying assumption is that every document comprises a statistical mixture of topics, i.e. a statistical distribution of topics that can be obtained by "adding up" all of the distributions for all the topics covered.

Implementation of Topic Modelling using LDA:

```
# Parameters tuning using Grid Search

from sklearn.model_selection import GridSearchCV

from sklearn.decomposition import LatentDirichletAllocation

from sklearn.manifold import TSNE

grid_params = {'n_components' : list(range(5,10))}

# LDA model

lda = LatentDirichletAllocation()

lda_model = GridSearchCV(lda,param_grid=grid_params)

lda_model.fit(document_term_matrix)

# Estimators for LDA model
```

Topic Modeling in Python

Now, it's time to build a model for topic modeling! We'll be using the preprocessed data from the previous tutorial. Our weapon of choice this time around is Gensim, a simple library that's perfect for getting started with topic modeling.

So, as a first step, let's install Gensim in our local environment:

```
pip install gensim
```

Now, pick up from halfway through the classifier tutorial where we turned reviews into a list of stemmed words without any connectors:

```
['room extrem small practic bed',
 'room safe work',
 'mattress comfort',
 'uncomfort thin mattress plastic cover rustl everi time move',
 'bathroom room',
 ...
]
```

With this list of words, we can use Gensim to create a dictionary using the *bag of words* model:

```
from gensim import corpora
texts = [process_text(text) for text in texts]
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]
```

Next, we can use this dictionary to train an LDA model. We'll instruct Gensim to find three topics (clusters) in the data:

```
from gensim import models
model = models.ldamodel.LdaModel(corpus, num_topics=3, id2word=dictionary, passes=15)

topics = model.print_topics(num_words=3)
for topic in topics:
    print(topic)
```

And that's it! We have trained our first model for topic modeling! The code will return some words that represent the three topics:

```
0, '0.034*"room" + 0.021*"bathroom" + 0.013*"night"')
(1, '0.056*"bed" + 0.043*"room" + 0.024*"comfort"')
(2, '0.059*"room" + 0.033*"clean" + 0.023*"shower"')
```

For each topic cluster, we can see how the LDA algorithm surfaces words that look a lot like keywords for our original topics (*Facilities*, *Comfort*, and Cleanliness).

Since this is an example with just a few training samples we can't really understand the data, but we've illustrated the basics of how to do topic modeling using Gensim.

Conclusion: Here we have studied the two methods of NLP like aspect mining and topic modeling.