SENTIMENT ANALYSIS

PROLOGUE:

Machine Learning is one of the greatest modern-day development in the field of Data Science. It has numerous applications all over the world in different domains, be it in Digital Media and Entertainment, Healthcare, Marketing, Banking and Financial, Insurance, Transportation, Automobile and the list goes on.

However, observing a similarity in the given applications of machine learning we learn that it mostly deals in numbers and values. With growing applications of Machine Learning we surely see an improved professional experience around us, but can this technology help us with our personal life as well? Can we ask our google assistant *"Hey Google! How are my emotions today?"* or *"Hey Google! How am I feeling today?"* and expect a sensible reply?

Well, yes! With the on-going improvements in the field of **Machine Learning** and **Sentiment Analysis** we can surely expect something sensible out of these seemingly ridiculous questions.

Now let's go through a bunch of questions to get a better understanding of Sentimental Analysis

Q 1) What is Sentiment Analysis?

A) Sentiment Analysis is basically a way to computationally determine the sentiment or emotions of the speaker or writer. A computer can read only numerical or binary language so the text is first converted into suitable form and then analysed to derive results or the sentiments of the writer.

Q 2) What are the python libraries that can be put to use for Sentimental Analysis?

A) textblob and nltk are the most frequently used python libraries for Sentimental Analysis. *Sentiment function of the textblob library has a couple of properties that are used to determine the sentiment or emotions of the writer.*

Q 3) What are the different properties of the Sentiment function of the textblob library?

A) Sentiment Function has two properties namely, Polarity and Subjectivity.

Q 4) What is Polarity?

- It is the expression that determines the sentimental aspect of an opinion.
- The sentiment Polarity can be expressed as **positive (+1), negative (-1)** or **neutral (0).**
- Polarity basically focusses on the literal meaning of text and gives result accordingly, taking into consideration all the positive and negative words of the text.

Q 5) What is Subjectivity?

- In natural language Subjectivity refers to **expressions of opinions, evaluations, feelings** and speculations and thus incorporates sentiment.
- It is between 0 to 1 where, 1 refers to very Subjective statement and 0 refers to very Objective statement.
- Subjectivity tries to capture the essence of what the writer wants to convey through his text.

Q 6) What is the relation between Subjectivity and Polarity?

A) There is as such no specific relation between Subjectivity and Polarity. At a particular value of Polarity there can be very different values of Subjectivity and at a given value of Subjectivity it is possible to obtain different values of Polarity. They mostly depend on the text and no specific relation or trend can be seen in their values.

Q 7) How can we individually determine the polarity and subjectivity of a review?

A) We can individually determine the polarity and subjectivity of the text by ***'sentiment.polarity'*** and ***'sentiment.subjectivity'***.

Now, let's understand the concept of sentimental analysis through a few lines of code.

```
In [3]: from textblob import TextBlob

In [5]: review1 = 'My tour to Gangtok was amazing'
        review2 = 'My tour to Gangtok was nice'
        review3 = 'My tour to Gangtok was not good'
        review4 = 'My tour to Gangtok was disasterous'

        blob1 = TextBlob(review1)
        blob2 = TextBlob(review2)
        blob3 = TextBlob(review3)
        blob4 = TextBlob(review4)

        print(blob1.sentiment)
        print(blob2.sentiment)
        print(blob3.sentiment)
        print(blob4.sentiment)

Sentiment(polarity=0.6000000000000001, subjectivity=0.9)
Sentiment(polarity=0.6, subjectivity=1.0)
Sentiment(polarity=-0.35, subjectivity=0.6000000000000001)
Sentiment(polarity=0.0, subjectivity=0.0)
```

- Here, in the first line we have imported the **textblob** library.
- If not already available you can install the textblob library through **'pip install textblob'**.
- Now, we have reviews of 4 different people for a trip to Gangtok.
- We have assigned four different variables (say blob1, blob2, blob3, blob4) and have applied the sentiment function on each of those.
- Further, we print the output.

RESULTS:

- The first statement that seems quite a positive review has high value of polarity (0.6) and even a higher value of subjectivity (0.9)
- The second statement shows the same value of polarity as the first one (0.6) and an even higher value of subjectivity (1.0, i.e., max subjectivity)
- The third one is not a good review so it shows negative polarity and a very less value of Subjectivity.
- The fourth statement is a bit vague leading to a value of zero for both subjectivity and polarity.

The above mentioned process seems pretty easy and effective for a smaller data set but however in large MNCs, hotel Management and Consultancies people deal with humongous data sets and reviews and we can't apply this method of assigning a variable to each review and finding its sentiment. So, how to deal with big data sets and extract some useful information out of it? Let's see how to do this by a problem statement,

PROBLEM STATEMENT #1

You are planning an outing for the weekend with your family and are a bit confused for your bookings. You are seeing a few hotels, reading their reviews but actually cannot make your mind of the place you want to go.

Let's take a view of the reviews of different hotels by different people and see if we can figure out a solution through sentiment analysis.

Review No.	Hotel 1	Hotel 2	Hotel 3
1	Good	Nice and clean rooms	Worst experience
2	Loved the ambience	Everything nice except the food	Service not upto the mark
3	hated the food but the hospitality w	Food didn't meet the expectations	Good place to celebrate occasions
4	bad	Pleasant Experience	Cost of food items was not justified by the taste of it
5	food seemed nice , could have been	Amazing food and view	Nice ambience
6	Delightful	Need the improve the service of food	Loved the atmosphere
7	A memorable time spent and good etiquettes by the staff	Lovable ambience	Slow service led to a bad experience
8	Food was tasty	Very expensive	Rooms were very clean
9	Food was served quite late leading to a bad experience	Excellent	Waste of money
10	Never going to visit again	Prices not justified	A lot of scope for improvement

Here we can observe that each hotel has very varied reviews and on all aspects of the Hotel, be it ambience or food or cleanliness or any other attribute.

Let's get to our python notebooks and try to get some results out of it.

SOLUTION

Step 1) The data is stored in an excel workbook so we use the **pandas** library to extract it and then further read it. *(Make sure you are in the same directory as of where the excel workbook is stored)*

```
In [1]: import pandas as pd

In [2]: data=pd.read_excel('Customer Reviews.xlsx')
```

Step 2) Setting the Review No. column as the index and seeing the top 5 rows of dataset with the help of **head** function.

```
In [5]: data.set_index('Review No.').head()
```

Out[5]:

Review No.	Hotel 1	Hotel 2	Hotel 3
1	Good	Nice and clean rooms	Worst experience
2	Loved the ambience	Everything nice except the food	Service not upto the mark
3	hated the food but the hospitality was good	Food didn't meet the expectations	Good place to celebrate occasions
4	bad	Pleasant Experience	Cost of food items was not justified by the ta...
5	food seemed nice , could have been better	Amazing food and view	Nice ambience

Step 3) Next, we can import the **textblob library** for our sentiment analysis. Further, we create a **function named, get_polarity** and input an **argument text** in it with an aim to get the output polarity of the text we supply in. We create a variable textblob for our text to be passed through the TextBlob library. *(utf-8 is the Unicode used for encoding)*. Further we create a **variable pol** for getting our polarity value of each of the reviews.

```
In [9]: from textblob import TextBlob

In [12]: def get_polarity(text):
             textblob=TextBlob(str(text.encode('utf-8')))
             pol=textblob.sentiment.polarity
             return pol
```

Step 4) Now, we create three new columns in the dataset namely pol1, pol2 and pol3. We store the polarity values of reviews of each hotel in them.

```
In [13]: data['pol1']=data['Hotel 1'].apply(get_polarity)

In [14]: data['pol2']=data['Hotel 2'].apply(get_polarity)

In [15]: data['pol3']=data['Hotel 3'].apply(get_polarity)
```

Step 5) Similar to step number three we create a function to get the values of subjectivity of reviews of each of the hotels.

```
In [16]: def get_subjectivity(text):
             textblob=TextBlob(str(text.encode('utf-8')))
             subj=textblob.sentiment.subjectivity
             return subj
```

Step 6) Similar to step number 4 we create three new columns namely subj1, subj2 and subj3 to store the subjectivity values of the reviews.

```
In [17]: data['subj1']=data['Hotel 1'].apply(get_subjectivity)

In [18]: data['subj2']=data['Hotel 2'].apply(get_subjectivity)

In [19]: data['subj3']=data['Hotel 3'].apply(get_subjectivity)
```

Step 7) Here, we try to get a description of the new columns that we have created and get out important results out of it.

- Mean Polarity and Mean Subjectivity of Hotel 2 is better than that of any other Hotel.
- Worst review by any of the customers for any hotel is given to hotel 3 (it has the least minimum polarity value that is -1, through the minimum subjectivity value is zero for each of the hotel)
- We can observe the quantiles as well for the polarity and subjectivity values of each of the hotel reviews.
- All the hotel reviews reach the max value of subjectivity in at least one of their reviews, but in case of polarity hotel 1 and hotel 2 reach their maximum values while hotel 3 is at a max polarity value of 0.7.

RESULTS:
According to the outcomes obtained **Hotel 2 seems an ideal choice** for the outing. It has the **best value of subjectivity plus has decent quantile values as well**.

```
In [21]: data[['pol1','pol2','pol3','subj1','subj2','subj3']].describe()
```

Out[21]:

	pol1	pol2	pol3	subj1	subj2	subj3
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	0.201667	0.306667	0.057667	0.566667	0.702667	0.574333
std	0.553800	0.501492	0.564108	0.323274	0.396882	0.424782
min	-0.700000	-0.650000	-1.000000	0.000000	0.000000	0.000000
25%	-0.075000	0.000000	-0.200000	0.575000	0.587500	0.133333
50%	0.183333	0.491667	0.000000	0.641667	0.900000	0.700000
75%	0.662500	0.600000	0.569167	0.729167	0.952500	0.907500
max	1.000000	1.000000	0.700000	1.000000	1.000000	1.000000

Step 8) We can see the whole dataset along with the new columns with the help of head function.

```
In [23]: data.set_index('Review No.').head()
```

Out[23]:

Review No.	Hotel 1	Hotel 2	Hotel 3	pol1	pol2	pol3	subj1	subj2	subj3
1	Good	Nice and clean rooms	Worst experience	0.70	0.483333	-1.0	0.600000	0.850000	1.0
2	Loved the ambience	Everything nice except the food	Service not upto the mark	0.70	0.600000	0.0	0.800000	1.000000	0.0
3	hated the food but the hospitality was good	Food didn't meet the expectations	Good place to celebrate occasions	-0.10	0.000000	0.7	0.650000	0.000000	0.6
4	bad	Pleasant Experience	Cost of food items was not justified by the ta...	-0.70	0.733333	-0.2	0.666667	0.966667	0.9
5	food seemed nice , could have been better	Amazing food and view	Nice ambience	0.55	0.600000	0.6	0.750000	0.900000	1.0

Q 8) What is NLP?

A) NLP is **Natural Language processing** which is used to make a computer understand text. Computers only understand Binary codes or numbers so to make sense out of certain paragraph or sentence NPL is used.

Q 9) What is NLTK?

A) NLTK stands for **Natural Language Toolkit** and is a python module that helps to deal with Natural Language Processing.

Q 9) What is the step-by-step procedure for sentimental analysis?

A) The following steps can be followed for sentimental analysis:

1. Tokenization – Splitting the whole sentence into individual characters.
2. Removing the symbols.
3. Removing the text that doesn't contribute to analytics (he, she, the etc.)
4. Classifying the leftover words as positive, negative or neutral.
5. Calculation to finally determine the sentiment.

Q 10) What is Tokenizing?

A) Tokenizing is the segregation of a paragraph or an article in individual words or sentences. It is mostly the very **first step of Sentimental Analysis**.

Q 11) How can we implement Tokenization?

A) We can implement tokenization with the help of **nltk (Natural Language Toolkit)** library. Let's understand tokenization with the help of few lines of code.

```
In [1]: import nltk

In [ ]: nltk.download()

In [2]: paragraph = ''' Students can select any paragraph on Education according to their particular requirement. Education helps in
        enlightening the minds of people. It must be given top priority. Education must be provided free of cost so that all children
        are educated.'''
```

- We import the **nltk library** in order to perform tokenization.
- Download all the sub libraries within nltk using **'nltk.download()'** if not already been downloaded.
- And obviously, we need a paragraph to tokenize.

```
In [3]: sentences=nltk.sent_tokenize(paragraph)

In [4]: sentences

Out[4]: [' Students can select any paragraph on Education according to their particular requirement.',
         'Education helps in \nenlightening the minds of people.',
         'It must be given top priority.',
         'Education must be provided free of cost so that all children \nare educated.']
```

- We define a variable 'sentences' and using the **'sent_tokenize' function** with argument being the paragraph we want to tokenize to dissolve the paragraph in its individual sentences.
- We print the variable sentences and observe that the paragraph has been reduced to its individual sentences.

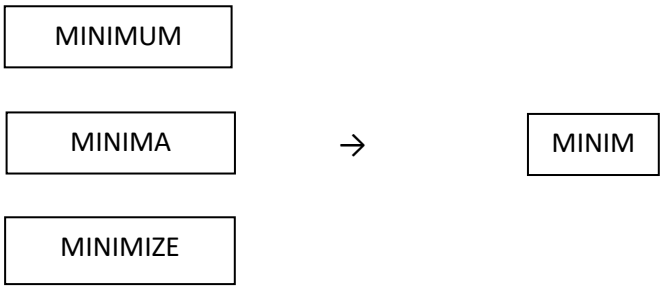
```
In [5]: words=nltk.word_tokenize(paragraph)
```

```
In [6]: words
Out[6]: ['Students',
         'can',
         'select',
         'any',
         'paragraph',
         'on',
         'Education',
         'according',
         'to',
         'their',
         'particular',
         'requirement',
         '.',
         'Education',
         'helps',
         'in',
         'enlightening',
         'the',
         'minds',
         'of',
         'people',
         '.',
         'It',
```

- We define another variable named 'words' and using the **'word_tokenize' function** with argument passed again being the paragraph dissolve the paragraph in form of its individual words.
- We print the variable word and notice that the paragraph has been reduced to individual words.
  # Also notice that the word_tokenize function assumes the symbols (.,! @&) as individual words and prints them independently as a single unit.

Q 12) What do you mean by Stemming?

A) Stemming is a process of conversion of a set of words into their stem. It can be implemented using 'nltk' library in python. Let us take an example to see what it actually means to convert a set of words into their stem.



Here, **MINIMUM, MINIMA** and **MINIMIZE** are a set of three words whose stem word is **MINIM**.

Q 13) What is the significance of stemming?

A) Generally, in Sentimental Analysis one can predict whether a word is having a positive influence or a negative influence in a review just by seeing the stem of that word. This is where stemming comes into play and saves some valuable time and memory.

Q 14) What do you mean by lemmatization?

A) Lemmatization is much similar to stemming, a major difference being that lemmatization always reduces the set of words in something (a word) that is meaningful, unlike stemming that just returns the stem of the words (like minim, that has no meaning).

Q 15) What are the differences between stemming and lemmatization?

A)

Difference in case of	**Stemming**	**Lemmatization**
Meaning of the output	The output may or may not have any meaning	The output must have some meaning
Time	Takes less time than lemmatization	Takes more time because it has to analyse all the words and come up with a sensible meaningful output
Application	It is used in Sentimental Analysis, Gmail spam detection and many more.	It is used in Chat Bot, any Q&A application etc.

Q 16) What are stopwords?

A) Stopwords are words like he, she, the, is, that, it and many other words of the same kind that don't contribute much to the meaning or sentiment of a sentence. They can't be judged as a positive or a negative remark. *We can remove stopwords from a paragraph by using the*

'stopwords' library *from* ***'nltk.corpus'***. We can get the list of stopwords in a particular language through the given lines of code.

```
In [7]: from nltk.corpus import stopwords

In [8]: stopwords.words('english')

Out[8]: ['i',
         'me',
         'my',
         'myself',
         'we',
         'our',
         'ours',
         'ourselves',
         'you',
         "you're",
         "you've",
         "you'll",
         "you'd",
         'your',
         'yours',
         'yourself',
         'yourselves',
         'he',
         'him',
```

To get the stopwords of any other language we just have to replace 'english' with the language we require stopwords of.

Q 17) How can one apply stemming on a python notebook?

A) Let's see how can we apply stemming on a python notebook.

Step 1) First of all we import all the important libraries as shown.

```
In [1]: import nltk

In [2]: from nltk.stem import PorterStemmer

In [3]: from nltk.corpus import stopwords
```

Step 2)

- Obviously, a paragraph is required for our stemming operation. So, we have our paragraph variable that contains the text we hope to get stemmed.
- Next, we tokenize the paragraph in sentences (using the same method we discussed earlier) and observe the output.

```
In [4]: paragraph = ''' The history teacher talked about the historical significance of the monuments. The monumental structure of
        various new buildings confuses students with the monuments built during the ancient times. '''

In [5]: sentences = nltk.sent_tokenize(paragraph)

In [6]: sentences

Out[6]: [' The history teacher talked about the historical significance of the monuments.',
         'The monumental structure of \nvarious new buildings confuses students with the monuments built during the ancient times.']
```

Step 3)

- This is a pretty important part of the code where stemming is actually done. First of all we have created a object named stemmer out of the **PorterStemmer library** that is obviously used for stemming.
- Next, we have applied a for loop in each sentence with the **range being the length of a sentence**.
- We tokenize all the words in each sentence and store and store them in a list named words.
- The third line of for loop implies that we stem each word in the words list if it is not in the set of stopwords in the english language.
- Further, we join the words in their respective sentences.

```
In [7]: stemmer = PorterStemmer()

In [8]: for i in range(len(sentences)):
            words=nltk.word_tokenize(sentences[i])
            words=[stemmer.stem(word) for word in words if word not in set(stopwords.words('english'))]
            sentences[i]=' '.join(words)
```

Step 4)

- We observe the output sentence and here are the significant changes or the stemmed words we are able to locate. **** histori, talk, histor, signific, monument, structur, variou, build, confus, student and time ****
- Also, all the **stopwords have been removed.**

```
In [9]: sentences

Out[9]: ['the histori teacher talk histor signific monument .',
         'the monument structur variou new build confus student monument built ancient time .']
```

Q 18) How can one apply Lemmatization on a python notebook?

A) Let's see how can we apply stemming on a python notebook.

Step 1) The first part of code is almost the same as that of stemming. The only difference being the library that we are importing for lemmatization, i.e., **'WordNetLemmatizer'** from **'nltk.stem'**.

```
In [1]: import nltk

In [2]: from nltk.stem import WordNetLemmatizer

In [3]: from nltk.corpus import stopwords

In [4]: paragraph = ''' The history teacher talked about the historical significance of the monuments. The monumental structure of
        various new buildings confuses students with the monuments built during the ancient times. '''

In [5]: sentences = nltk.sent_tokenize(paragraph)

In [6]: sentences
Out[6]: [' The history teacher talked about the historical significance of the monuments.',
         'The monumental structure of \nvarious new buildings confuses students with the monuments built during the ancient times.']
```

Step 2) Again almost the same lines of code as that of stemming. The only difference being the object named lemmatizer created out of the WordNetLemmatizer library. But the lemmatized words are quite different, that are, ***monument, building, student and time*** and as in above case the **stopwords are removed** from here as well.

```
In [7]: lemmatizer=WordNetLemmatizer()

In [8]: for i in range(len(sentences)):
            words=nltk.word_tokenize(sentences[i])
            words=[lemmatizer.lemmatize(word) for word in words if word not in set(stopwords.words('english'))]
            sentences[i]=' '.join(words)

In [9]: sentences
Out[9]: ['The history teacher talked historical significance monument .',
         'The monumental structure various new building confuses student monument built ancient time .']
```

Q 19) What is Bag of words?

A) Bag of words is another process used in **Natural Language Processing (NLP)** where the main aim is to convert texts in vectors (or in numerical form that can be analysed by the computer). In Bag of words a sentence is represented by a group of important words in that sentence and further vectorization is done.

Q 20) What are the steps involved in the process of Bag of Words?

A) One might want to follow the series of steps mentioned below in order to implement Bag of Words:

Step 1) Remove all the stopwords from the sentences. As stopwords don't contribute towards the sentiment analysis or have any significant meaning of their own that might play a vital role in any form of analysis, they better be removed and not included in our bag (group) of words.

Step 2) Bring all the remaining words of the sentences in the same case (preferably lower, as it is mostly the standard operating procedure). This process is however important because in python same words with different cases will be considered as two different words and we might not want to include a single word two or three times in our bag of words with different cases. Let's see this through an example:

"He likes chocolates and he also Likes cakes" → Here, 'likes' and 'Likes' will be considered as two different words and if applied bag of words without getting them in the same case can lead to the addition of both 'likes' and 'Likes' in the bag of words leading to absurd or incorrect results.

Step 3) We can further implement stemming and lemmatization as per the requirement.

Step 4) We make a histogram of the important words. (In simpler terms, a table consisting of the important words and the frequency of that word in the whole group of sentences we are provided with).

Step 5) We further convert it into vectors for further processing and this step or the end result is called the **Bag of words**.

Let us understand these steps with the help of an **example** for more clarification:

Let's say we have three sentences,

Sentence 1) John likes Tennis.

Sentence 2) Mary likes tennis.

Sentence 3) John and Mary like Tennis.

Now let's follow our given set of steps,

Step 1) Removing all the Stopwords. We end up with the following sentences.

Only 'and' removed from the third sentence

1) John likes Tennis.
2) Mary likes tennis.
3) John Mary like Tennis.

Step 2) Bring all of words in the same case (lower). We end up with following sentences.

tennis and Tennis won't be considered different words now

1) john likes tennis.
2) mary likes tennis.
3) john mary like tennis.

Step 3) Implementing stemming and lemmatization. We end up with following sentences.

likes converted to like

1) john like tennis.
2) mary like tennis.
3) john mary like tennis.

Step 4) The table obtained is shown below, (table should be preferably in descending order of frequency)

Word	**Frequency**
like	3
tennis	3
john	2
mary	2

Step 5) The main step of creating the bag of words, here's how it follows,

Words	**john**	**mary**	**like**	**tennis**
Sentence 1	1	0	1	1
Sentence 2	0	1	1	1
Sentence 3	1	1	1	1

We set 0 if the word is not present in a particular sentence and 1 if the word is present in that sentence.

This is how we create our Bag of words. We can further use this as our input to train the model and get suitable or required output.

Q 21) What are the different types of Bag of Words?

A) There are majorly two types of Bag Words:

1) Binary bag of words – It consists of only zeros and ones. Like the example shown above.

Words	john	mary	like	tennis
Sentence 1	1	0	1	1
Sentence 2	0	1	1	1
Sentence 3	1	1	1	1

2) Normal bag of words – If any cell contains any value other than zero and one implying the presence of that word more than once in that sentence then that bag of words is classified as Normal bag of words.

Words	john	mary	like	tennis
Sentence 1	1	0	2	1
Sentence 2	0	1	1	4
Sentence 3	1	1	3	1

Here, as we can see there are values other than zero and one so it is an example of normal bag of words.

Q 22) What are the disadvantages of Bag of words?

A) A major disadvantage of the bag of words is that it gives the **same emphasis** on each of the word present in the sentence. In sentiment analysis one should know which word to give more emphasis on or to derive or conclude more information or outcome from. However, Bag of words fail to deliver this and thus the essence of sentiment analysis is lost in big datasets due to bag of words.

Let's see this through and example,

In the sentence **'John likes Tennis'**, for sentiment analysis more emphasis should be given on the word **'likes'** rather than **'John'** and **'Tennis'**, but using bag of words will give equal emphasis on all the three words.

Q 23) How will you implement bag of words in python notebook?

A) Follow the series of steps to implement bag of words in python notebook:

Step 1) Firstly, we import various important libraries, that are,

- **'nltk' or 'Natural Language Toolkit'** for various important operations.
- **'re' or 'Regular Expression'** for various text cleaning purposes. (re provides fast operations on strings)
- **'stopwords'** for identifying the stopwords in the text.
- **'WordNetLemmatizer'** for lemmatization.

```
In [1]: import nltk

In [2]: paragraph = ''' The history teacher talked about the historical significance of the monuments. The monumental structure of
        various new buildings confuses students with the monuments built during the ancient times. '''

In [3]: import re

In [4]: from nltk.corpus import stopwords

In [5]: from nltk.stem import WordNetLemmatizer
```

Step 2)

- Here, we have introduced the object lemmatizer for lemmatizing the text.
- We have tokenized the paragraph in form of sentences and stored it under a variable named **'sentences'**.

- We created an empty list namely **'cleaned'** for storing the sentences after cleaning them.

```
In [6]: lemmatizer=WordNetLemmatizer()

In [7]: sentences=nltk.sent_tokenize(paragraph)

In [8]: cleanedset = []
```

Step 3) This is a pretty important part of code and we have done the following operations with the help of for loop in each sentence,

- Removed all the miscellaneous characters from each sentence (removed all characters excluding a-z and A-Z)
- Brought the remaining set of words in a sentence in **lower case**.
- Applied the split operation and **lemmatized** the sentences.
- Added the cleaned set of sentences in the '**cleanedset**' list.

```
In [9]: for i in range(len(sentences)):
            review = re.sub('[^a-zA-Z]',' ',sentences[i])
            review = review.lower()
            review = review.split()
            review = [lemmatizer.lemmatize(word) for word in review if not word in set(stopwords.words('english'))]
            review = ' '.join(review)
            cleanedset.append(review)

In [10]: cleanedset
Out[10]: ['history teacher talked historical significance monument',
          'monumental structure various new building confuses student monument built ancient time']
```

Step 4)

- Here, first of all we imported the **CountVectorizer library** that is majorly used for the creation of Bag of Words.
- Created an object named cv for the operation of CountVectorizer.
- Fitted the data (input features/ Bag of words) in an array named **'X'**.

```
In [11]: from sklearn.feature_extraction.text import CountVectorizer
         cv = CountVectorizer(max_features = 1500)
         X = cv.fit_transform(cleanedset).toarray()

In [12]: X
Out[12]: array([[0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0],
                [1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1]], dtype=int64)
```

So, this is how we obtain our Bag of Words or the input features and use them to create a model.

Q 24) What is TF-IDF?

A) TF-IDF stands for **Term Frequency – Inverse Document Frequency**. It is another method of converting text in vectors but one major thing to point out is that it covers the drawback of Bag of Words, i.e., it gives more emphasis on words that seem more important and thus, can be applied to large datasets for sentimental analysis and any other required application.

Q 25) How is TF-IDF calculated?

A) TF-IDF is basically the product of two values, that are, **'term frequency'** and **'inverse document frequency'.**

TERM FREQUENCY = $\frac{NO.\ OF\ REPETITIONS\ OF\ A\ WORD\ IN\ A\ SENTENCE}{NO.\ OF\ WORDS\ IN\ A\ SENTENCE}$

INVERSE DOCUMENT FREQUENCY = log ($\frac{TOTAL\ NUMBER\ OF\ SENTENCES}{NUMBER\ OF\ SENTENCES\ HAVING\ A\ PARTICULAR\ WORD}$**)**

Let's calculate term frequency and inverse document frequency of an example for a better understanding.

We have the following sentences (same as above example of Bag of words), obviously the stopwords have been removed and lemmatization has been done.

1) john like tennis.
2) mary like tennis.
3) john mary like tennis.

Now, let's create our **term frequency table:**

	Sentence 1	**Sentence 2**	**Sentence 3**
like	1/3	1/3	1/4
tennis	1/3	1/3	1/4
john	1/3	0	1/4
mary	0	1/3	1/4

So, here we have created our term frequency table, let's see how to create the inverse document frequency table with the same set of sentences.

So, **inverse document frequency table:**

Words	**IDF**
like	Log (3/3) = 0
tennis	Log (3/3) = 0
john	Log (3/2)
mary	Log (3/2)

Q 26) How can we use the term frequency and inverse document frequency to get the input features to build a model of?

A) We multiply the term frequency and inverse document frequency in order to get the required input features for our model.

Let's implement this in our example and observe the results. (Term frequency and Inverse Document Frequency tables are as of above question)

	like	tennis	john	mary
Sentence 1	(1/3)*0=0	(1/3)*0=0	(1/3)*log (3/2)	0*log (3/2) = 0
Sentence 2	(1/3)*0=0	(1/3)*0=0	0*log (3/2) = 0	(1/3)*log (3/2)
Sentence 3	(1/4)*0=0	(1/4)*0=0	(1/4)*log (3/2)	(1/4)*log (3/2)

So, here are our input features through **TF-IDF** which we can further use to build our model.

Q 27) How can we implement Tf-idf in a python notebook?

A) Follow the series of steps to implement Tf-idf in python notebook:

Step 1) The first part of code is almost same as that of Bag of Words. Here also,

- We import the necessary libraries.
- Create an object named lemmatize for lemmatization of sentences.
- Tokenize the paragraph in its sentences.
- Create an empty list to store the cleaned text.
- Clean the sentences with the help of a for loop. (Remove miscellaneous characters, lower the case, split then lemmatize and then append the cleaned sentences in the empty list)
- Print the cleaned set and observe the sentences.

```
In [1]: import nltk

In [2]: paragraph = ''' The history teacher talked about the historical significance of the monuments. The monumental structure of
        various new buildings confuses students with the monuments built during the ancient times. '''

In [3]: import re

In [4]: from nltk.corpus import stopwords

In [5]: from nltk.stem import WordNetLemmatizer

In [6]: lemmatizer=WordNetLemmatizer()

In [7]: sentences=nltk.sent_tokenize(paragraph)

In [8]: cleanedset = []

In [9]: for i in range(len(sentences)):
            review = re.sub('[^a-zA-Z]',' ',sentences[i])
            review = review.lower()
            review = review.split()
            review = [lemmatizer.lemmatize(word) for word in review if not word in set(stopwords.words('english'))]
            review = ' '.join(review)
            cleanedset.append(review)

In [10]: cleanedset

Out[10]: ['history teacher talked historical significance monument',
          'monumental structure various new building confuses student monument built ancient time']
```

Step 2) Here's the important part,

- We import the library **TfidfVectorizer** to perform **Tf-idf.**
- Create an object cv to perform the operation.
- Fit the data (input features) in an array named X.

```
In [12]: from sklearn.feature_extraction.text import TfidfVectorizer
         cv = TfidfVectorizer()
         X = cv.fit_transform(cleanedset).toarray()

In [13]: X

Out[13]: array([[0.        , 0.        , 0.        , 0.        , 0.4261596 ,
                 0.4261596 , 0.30321606, 0.        , 0.        , 0.4261596 ,
                 0.        , 0.        , 0.4261596 , 0.4261596 , 0.        ,
                 0.        ],
                [0.30851498, 0.30851498, 0.30851498, 0.30851498, 0.        ,
                 0.        , 0.21951095, 0.30851498, 0.30851498, 0.        ,
                 0.30851498, 0.30851498, 0.        , 0.        , 0.30851498,
                 0.30851498]])

In [ ]:
```

Here, we observe that unlike Bag of Words there are numerical values assigned that are not zero or 1. The values are assigned on the basis of the emphasis of that word on sentimental analysis. And hence, the drawback of Bag of Words is covered and surely, we have got a method to deal to big datasets with more precision.

Q 28) What are language models?

A) Language models basically help us to complete or form sentences in a way that it will make sense. It does so by computing the probability distribution of the words or a sentence formed. Let's understand this with a couple of examples:

Example 1) He likes to play _____.

We are supposed to fill this blank, and the options we have are,

1) Badminton
2) They
3) Hawaii
4) Eat

So, here obviously the probability of Badminton filling the blank is much higher than any other option doing so. We, as humans know this but to computationally come up with the right answer, Language Models are used.

Example 2) To find out which of the following sentences,

1) He likes to play Badminton, and

2) Play he badminton to like

makes sense, Language Models are used.

Q 29) What are the types of Language Models?

A) There are basically two types of Language Models,

1) Statistical Language Models – They use methods like n-grams, Hidden Markov Model (HMM) and a few other features to get the probability distribution of words.

2) Neural Language Models – They use different kinds of neural networks to create Language Models.

Q 30) What is a n-gram model?

A) A n-gram language model predicts the probability of occurrence of a word on the basis of n-grams (Unigrams, Bigrams, Trigrams etc.). To be a bit specific we can get the probability a given word on the basis of the previous words *(except in case of Unigrams).*

Say, we have an incomplete sentence with (n-1) words, so using n-gram model we can predict the n^{th} word of the sentence.

Q 31) What is a Unigram?

A) A Unigram is a n-gram model where n=1, or it can be said as the **single word sequence**. The probability of occurrence of the word is determined by its previous occurrences. Let's see what a unigram is through an example,

"He has a very arrogant attitude" – In this sentence the unigrams will be, 'He' , 'has' , 'a' , 'very' , 'arrogant' , 'attitude'.

Q 32) What is a Bigram?

A) A Bigram is also a n-gram model where n=2, that is, it is a **sequence of two words** and the probability distribution of the next word is predicted on the basis of the previous word. Let's see this with an example,

In the sentence "He has a very arrogant attitude", the following will be the Bigrams,

1) He has
2) Has a
3) A very
4) Very arrogant
5) Arrogant attitude

The set of two words in a bigram are always corresponding to each other

Q 33) What is a trigram or 4-gram?

A) A trigram and 4-gram are again n-gram models where n=3 and n=4 respectively, that is a sequence of three or four words with the probability distribution of the next word being predicted on the basis of previous two or three words respectively. In the above sentence,

“He has a very arrogant attitude”

The trigrams are as follows:

1) He has a
2) Has a very
3) A very arrogant
4) Very arrogant attitude

And the 4-grams are as follows:

1) He has a very
2) Has a very arrogant
3) A very arrogant attitude.

And similarly, the n-grams follow.

One must understand conditional probability in order to use or understand n-gram model

Q 34) What are the drawbacks of n-gram modelling?

A) n-gram has a couple of drawbacks:

- n-gram modelling takes into consideration only a **limited number of sample set**, that is, it can predict only those words that it has encountered previously, there is almost zero probability of prediction of a new word.
- With an increase in ‘n’ the probability of correct prediction also increases but it brings along an **increase in computational time and memory**.

Q 35) What is the chain rule of probability?

A) According to the chain rule of probability, if w = (w1w2w3w4….wk) is a set of words then, the probability of the occurrence of words in a particular order will be P(w), where,

P(w) = P(w1).P(w2|w1).P(w3|w1w2)…P(wk|w1w2…w(k-1))

Where, P(w1) is the probability of occurrence of word w1 and P(w2|w1) is the probability of occurrence of w2 when w1 has already occurred *(Conditional Probability)*. Same goes for P(w3|w1w2) that is the occurrence of word w3 when w1 and w2 have already occurred. And it goes on till wk.

However, this process of calculating the probability for a given sequence seems a bit lengthy and complex. Hence, we use the **Markov Assumption**.

Q 36) What is Markov Assumption?

A) According to Markov Assumption,

P(wk|w1w2…w(k-1)) = P(wk|w(k-1)),

Here, as we can see the probability of occurring of word wk is determined directly on the basis of word just preceding it rather than all the words preceding it *(Bigram Model).*

One can vary the number of preceding words to take in account as per requirement

Q 37) What do you mean by Word2Vec?

A) Word2Vec is a Neural Language Model that basically assigns each word a position in a 3D plane and the level of similarity of words is judged on the basis of the distance between them. Words after Word2Vec or being on a 3D plane are referred to as Word Vectors. Let us understand this through an example,

If the words, 'He', 'She' ,'Delhi' ,'India' is represented on a 3D plane the, he and she will appear close relatively as compared to their distances with Delhi and India. Similarly, Delhi and India will appear close when compared to their distances with he and she. This is obviously because he and she are similar in some obvious literature aspects and so is Delhi and India.

One should have a good understanding of Neural Networks in order to implement or completely understand Neural Language Models

EPILOGUE:

Natural Language Processing or Sentiment Analysis however has various applications and continue to serve major industries and companies for their growth and betterment. It is a modern-day innovation and still has a major scope of growth and development in itself.

Sentiment Analysis can't be restricted to the concepts we discussed in this article. It continues to grow and has several other methods to understand text computationally and get results out of it.

For as much as we know is that Sentiment Analysis is a big breakthrough in human civilization.

SUMMARY:

In this article we discussed about Sentiment Analysis and Natural Language Processing, a modern-day innovation that helps to under the sentiment or emotions of writer through computational means.

We started with the exploration of the textblob library and properties related to it, that is, Polarity and Subjectivity. We understand Polarity and Subjectivity through a couple of examples or problem statements and get familiar with the lines of code required to perform Sentiment Analysis through textblob library in smaller and a relatively bigger dataset.

After exploring textblob library, we move towards a few concepts of NLP (Natural Language Processing) and are introduced to nltk (Natural Language Toolkit) library. We study the chronological order in which one should proceed towards Sentiment Analysis where we got introduced to words like Tokenization, Stopwords, Stemming and Lemmatization. We understood the importance and requirement of each one of in Natural Language Processing or Sentimental Analysis and further deep dive into our python notebooks to understand the important libraries and lines of code related to each of the following terms. We also discussed the types of tokenization and the points of differences between stemming and lemmatization.

All of this is followed by a major discussion about Bag of Words and TF-IDF which are methods to finally convert our reduced text (after the application of tokenization, stopwords removal, stemming, lemmatization and other methods of text cleaning) in its vector or numerical form. We understand Bag of words and TF-IDF with various physical illustrations. After getting the essence of Bag of Words and TF-IDF we further look into the lines of code and enhance our understanding by observing the way of implementation as well as the output obtained.

After having a complete hold on the concepts of converting the text in its vector form, we jump to a very fresh concept of n-grams, that are basically used to predict a word or order a sentence computationally in a particular order to make sense out of it. We look in various sub divisions of n-grams (Unigrams, Bigrams etc.) and try to understand their way of working.

n-grams are based on the concepts of Conditional Probability, i.e., they predict words on the basis of their probability distribution. So, we take a sneak-peak into the concepts of Conditional Probability, Chain rule or Probability and Markov's Assumption. Finally, we end our article with a small introduction to one of the Neural Language Models, Word2Vec.