Basics of NLP

Human language as it is spoken and written is referred to as natural language. Natural language processing (NLP) is the ability of a computer program to understand natural language. NLP is a component of artificial intelligence (AI). Businesses deal with massive volumes of unstructured, text-heavy data, and they need an effective way to process it efficiently. A lot of the information generated online and stored in databases is in the form of natural human language. Until recently, businesses could not analyze this data effectively. This is where natural language processing has proved to be extremely useful.

Common Interview Questions

1. What are the applications of NLP?
2. What is lexical processing?
3. What are stop words, and why do we need to remove them?
4. What is TF-IDF, and how does it differ from Count Vectorizer?
5. What do you mean by stemming in NLP?
6. What do you mean by lemmatization in NLP? How does it differ from stemming?
7. What is bag-of-words representation, and what are its drawbacks?
8. What is edit distance, and why is it used?
9. What is the application of pointwise mutual information?
10. What is canonicalization?

Applications of NLP

- Social media analytics
- Banking and loan processing
- Insurance claim processing

Components of NLP

- **Lexical processing:** It involves converting raw text into words depending on an application's needs.
- **Syntactic processing:** It involves deriving more meanings from the syntax of the sentence.
- **Semantic processing:** It involves building advanced NLP applications such as language translation and chatbots.

Regular expressions: Regular expressions help immensely in the text preprocessing stage. You learned about various characters that have special meaning in the regular expressions' engine.

Here are some of these characters:

1. * - This matches the preceding character or character set zero or more times.
2. + - This matches the preceding character or character set one or more times.
3. ? - This matches the preceding character or character set zero or one time.
4. {m, n} - This matches the preceding character or character set if it is present from "m" times to "n" times.
5. ^ - This marks the start of a string. When this character is used inside a character set, it acts as a negation set.
6. $ - This marks the end of a string.
7. . - This is a wildcard character (used to match any character).
8. [a-z0-9] - This is a character set. It matches the character present inside the set.
9. \w - This meta-sequence is used to match all alphanumeric characters.
10. \W - This meta-sequence is used to match all characters except alphanumeric characters.
11. \d - This meta-sequence is used to match all numeric digits.
12. \D - This meta-sequence is used to match all characters except numeric digits.
13. \s - This meta-sequence is used to match all whitespace characters.
14. \S - This meta-sequence is used to match all characters except whitespace characters.
15. *? This non-greedy search stops as soon as the specified pattern is found. "?" can be used after any quantifiers to make a search non-greedy.
16. | - This is used as an OR operator inside character sets or groups.
17. () - This is used to group parts of regex to extract characters separately.

To use regex in Python, you need to know the workings of certain functions. The most useful functions are listed below:

1. re.match(pattern ,string) - This function tries to look for a pattern in a string from the beginning. It returns "None" if a match is not found at the start of the string.

2 re.search(pattern, string) - This function tries to look for a pattern in a given string. It returns "None" if a match is not found in the entire string.

3 re.finall(pattern, string) - This tries to look for each occurrence of a pattern in a given string and returns all of them in a list. It returns "None" if a match is not found.

4 re.finditer(pattern, string) - This is similar to re.findall(), but it is iterated over matches one by one.

5 re.sub(pattern, replacement, string) - This function tries to look for each occurrence of a pattern in a given string and replaces all of them with the replacement string. It returns the original string if a match is not found.

Tokenization: Tokenization is the process of breaking down a text corpus into different words or sentences or paragraphs based on the end application in mind. The NLTK library has various tokenizers, some of which are as follows:

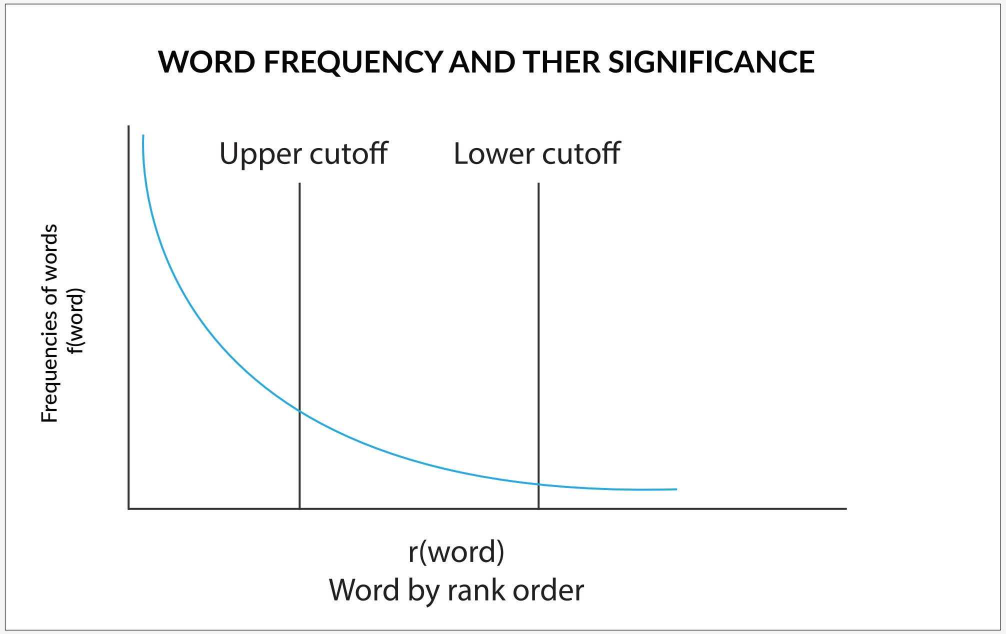
1. Word tokenizer: This tokenizer can be used to break down a text corpus into a list of words.
2. Sentence tokenizer: This tokenizer can be used to break down a text corpus into different sentences.
3. Tweet tokenizer: This tokenizer can be used to tokenize text into different words. Unlike a word tokenizer, this tokenizer tokenizes social media elements such as emojis and hashtags accurately.
4. Regexp tokenizer: This tokenizer can be used to tokenize text using a regular expression.

After tokenizing the required documents and removing stop words, the next step is to reduce words to their base form.

1. Stemming: This is a rule-based technique that chops off the suffix of a word to obtain its root form, which is called the "stem." For example, in the string "The driver is racing in his boss' car", the stem words "driver" and "racing" will be converted into their root form by chopping off the suffixes "er" and "ing." So, "driver" will be converted to "driv," and "racing" will be converted to "rac." Now you might think that the root forms (or stems) do not resemble the root words "drive" and "race." You need not worry about this because the stemmer will convert all the variants of "drive" and "racing" to the aforementioned root forms only. So, it will convert "drive," "driving," etc. to "driv," and "race," "racer," etc. to "rac."

2. Lemmatization: This is a sophisticated technique. It not only chops off the suffix of a word but takes an input word and searches for its base word by processing all the variations of dictionary words recursively.the variations of dictionary words. The base word, in this case, is called "lemma." Words such as "feet," "drove," "arose," and "bought" cannot be reduced to their correct base form using a stemmer. However, a lemmatizer can reduce them to their correct base form. WordNet, developed by a team of professors at Princeton University, is one of the most popular lemmatizers.

Word Frequency and Its Importance:



Stop words are removed from text for the following two reasons:

1. They provide no useful information in most applications.
2. They use a lot of memory because of the high frequency in which they are present.

Bag-of-word: After learning about all the preprocessing steps, su d how to change text from a textual form to a tabular form.

BAG-OF-WORDS MODEL

Document 1: "Gangs of Wasseypur is a great movie."
Document 2: "The success of a movie depends on the performance of the actors."
Document 3: "There are no new movies releasing this week."

	Actors	great	depends	gangs	movie	movie	new	performance	releasing	success	wasseypur	week
1	0	1	0	1	1	0	0	0	0	0	1	0
2	1	0	1	0	1	0	0	1	0	1	0	0
3	0	0	0	0	0	1	1	0	1	0	0	1

TF-IDF: To build the TF-IDF representation or the TF-IDF model, you need to calculate the TF-IDF score for each word in a document. The TF-IDF score comprises two terms – the TF (term frequency) and the IDF (inverse document frequency). The formula for calculating TF-IDF is given below.

$$tf_{t,d} = \frac{\text{frequency of term 't' in document 'd'}}{\text{total terms in document 'd'}}$$

$$idf_t = \log \frac{\text{total terms in document 'd'}}{\text{total documents that have the term 't'}}$$

$$tf — idf = tft,d * idft$$

Canonicalization: Canonicalization refers to the process of reducing a given word to its base word. Stemming and lemmatization are both techniques to canonicalize words to their base forms.

Phonetic hashing: Certain words have different pronunciations in different languages. As a result, they end up being spelled differently. Examples of such words include names of people, city names, names of dishes, etc. For example, New Delhi, the capital of India, is also pronounced as Dilli in Hindi.

ALGORITHM OF AMERICAN SOUNDEX:

For a given word, apply the following steps:

1. Retain the first letter of the word as is
2. Replace consonants with digits according to the following:
    - b, f, p ,v $\Rrightarrow$ 1
    - c, g, j, k, q, s, x, z $\Rrightarrow$ 2
    - d .t $\Rrightarrow$ 3
    - l $\Rrightarrow$ 4
    - m ,n $\Rrightarrow$ 5
    - r $\Rrightarrow$ 6
    - h, w ,y $\Rrightarrow$ unencoded
3. Remove all the vowels and the unencoded letters. And then merge any consecutive same numbers into a single number
4. Continue till the code has one letter and three numbers. If the length of Code is greater than four then truncate it to four letters. In case the code is shorter than four letters, pad it with zeroes

Edit distance: Edit distance is the number you get in the lower-right corner of a matrix. To calculate the edit distance between strings, you need to fill the entire matrix from right to left and top to bottom.

1. You determined whether the characters you were comparing are the same. If they are the same, you can copy the diagonal value (value present in the top-left cell) to the current cell.
2. If the characters are not the same, you consider the least value (in the top cell, the left cell, and the top-left cell), add one to this value, and fill the current cell with the value.

Spell corrector: The final technique to canonicalize words is correcting the spellings of incorrect words. You used four main functions to find the correct spelling of words:

1. **edits_one():** This function creates all the possible words that are one edit distance from the input word.
2. **edits_two():** Similar to edits_one(), this function is used to return all the possible spellings that are at a two-edit distance from the input word.
3. **known():** This function takes a list of words and outputs only valid English words.
4. **possible_corrections():** This function takes the input word and uses the above-mentioned three functions to return the possible correct spellings for the given input word. First, the function checks whether the spelling is correct or not. If it is correct, it returns the input word. If the spelling is incorrect, it looks for all the dictionary words that are one edit away from it and returns them. If there are no dictionary words that are one edit distance away from the given word, it looks for all the dictionary words that are two edits away and returns them. If no such words are at a two edit distance away, the original word is returned. This indicates that no correct spelling is found."
5. **prob():** The probability function takes multiple words and returns the word that returns the most frequently occurring words among the input words in the seed document. It returns returns the correctly spelled words.

Pointwise mutual information (PMI): PMI can be calculated using the following formula. Here, p(x,y) is the joint probability, whereas p(x) and p(y) are the marginal probabilities of the words "x" and "y."

$$\text{pmi}(x; y) = \log \frac{p(x, y)}{p(x)\ p(y)}$$

OCCURRENCE CONTEXT

S1 : Constraction of a new metro station state in New Delhi

S2: New Delhi is the capital of India

S3: The climate of Delhi is monsoon-influenced humid climate

$$p(\text{New Delhi}) = \frac{2}{3}$$

$$p(\text{New}) = \frac{2}{3}$$

$$p(\text{Delhi}) = \frac{2}{3}$$

$$\text{pmi}(\text{New Delhi}) = \frac{p(\text{New Delhi})}{p(\text{New})\ p(\text{Delhi})}$$

BiGram: For terms that comprise more than two words, you need to calculate the bigram approximation from the chain rule. Without the bigram approximation, calculating the PMI can become time-consuming. Consider the following formula used to calculate the PMI of a three-word term.

$$\text{pmi}(z, y, x) = \log \left(\frac{p(z \mid y, x)\ p(y \mid x)}{p(y)\ p(z)} \right)$$

As you can see, to calculate p(z|x,y), you need the joint probability of the words "x" and "y." Note that the complexity becomes more complex as you try to calculate the PMI of long terms.