**TEXT SUMMARIZATION BY NLP**

Natural Language Processing is the process of making machines understand and respond to text or voice data and even respond with text or speech. NLP combines computational linguistics- rule based modeling of human language- with statistical, machine learning and deep learning models. These technologies enable computers to process human language in the form of text or voice data and understand its full meaning.

Python provides a wide range of tools and libraries for solving NLP problems. The Natural Language Toolkit (NLTK) is an open source collection of libraries, programs and educational resources to build NLP programs.

Text summarization: Text summarization uses NLP techniques to create summaries of huge texts so as to help those who do not have time to read the full text.

There are two approaches to text summarization:

* Extractive approach – This approach makes use of the traditional algorithms. For example the frequency method which checks the frequency of words and includes them in the summary if they are significant. Here the summary contains sentences which are present in the text.
* Abstractive approach – This approach uses deep learning. The summary here contains new sentences and words which are different from the text.

Pre Processing of text:

It is the cleaning and preparing of the text data. It includes splitting the text into sentences, lowering the case of all words, removing stopwords and punctuations. This helps to get rid of the data or noise. The NLTK library acts as translator between the machine and humans. Pre processing reduces the size of the text data and retains only the relevant information. This gives a better efficiency. Steps in pre processing include-

* Removing HTML tags – HTML tags don’t add much value towards understanding and analyzing text. ‘Beautiful Soup’ package is used to remove HTML tags.
* Removing URLs – URLs in text data do not provide any additional information. So these are removed using the library ‘re’ which provides regular expression matching operations.
* Lowercasing- lowercasing the text data creates uniformity and helps with consistency of the output. If the text data contains the same word multiple times but in different casing then the machine treats each word uniquely, which is not desirable.
* Tokenization- Tokenization is splitting the text data into tokens. Paragraphs are split into sentences and then sentences are splits into words. This generates a list of all the words in the text data.
* import nltk
* from nltk import tokenize
* text = "NLP is a systematic process that help to do various task. It is used to analyze, organize and summarize the data"
* #decomposing the paragraph into lines
* lines=tokenize.sent\_tokenize(text)
* print(lines)

['NLP is a systematic process that help to do various task.', 'It is used to analyze, organize and summarize the data']

#decomposing the text into words

words=tokenize.word\_tokenize(text)

print(words)

['NLP', 'is', 'a', 'systematic', 'proces', 'that', 'help', 'to', 'do', 'various', 'task', '.', 'It', 'is', 'used', 'to', 'analyze', ',', 'organize', 'and', 'summarize', 'the', 'data']

* Lemmatization- Dataset may contain words which are made from a single word by adding some suffix or prefix. This causes redundancy in the dataset and does not give better output. In lemmatization the word generated after chopping off the suffix has some meaning and doesn’t produce any incorrect word. The generated word is known as lemma.
* from nltk.stem import WordNetLemmatizer
* wml = WordNetLemmatizer()
* words\_orig=["cries","crys","cried"]
* print('Original words-', words\_orig)
* lemma\_words=[]
* for word in words\_orig:
* tokens = wml.lemmatize(word)
* lemma\_words.append(tokens)
* print("After lemmatization", lemma\_words)

Original words- ['cries', 'crys', 'cried']

After lemmatization ['cry', 'cry', 'cried']

* Removing stop-words – stop words are those words which help to combine a sentence and make it sensible. ‘I, am,are..’ are examples of stop words. However stop words aren’t important to the machine and need to removed. NLTK contains a list of stop words in the English language which has to be imported and these cannot be modified. The words in the text data are compared with the list elements and removed if they are the same.

Code:

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

sentence = “machine learning is cool”

stop\_words=set(stopwords.words(‘english’))

word\_tokens=word\_tokenize(sentence)

filtered\_sentence=[w for w in word\_tokens if not w in stop\_words]

print(filtered\_sentence)

output: [‘machine’,’learning’,’cool’]

ALGORITHM FOR TEXT SUMMARIZATION:

The Textrank algorithm is a widely used extractive method based on Google’s page rank algorithm. The page rank algorithm is used to rank the web pages in its search engine results. It determines the similarity between sentences based upon number of common words and dividing by length of the sentences. It models the document as a graph using sentences as nodes and the similarity between the sentences as edges between the nodes.

The similarity function between two sentences is given by the number of words belonging to both the sentences divided by the sum of the log of the length of the two sentences.

The result of this is a dense graph representing the document. Then page rank is used to compute the importance of each vertex.

Page rank gives a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at a particular page. However it is assumed that the distribution is evenly divided among all documents in the collection at the beginning of computational process.

The page rank theory holds that an imaginary surfer is randomly clicking on links and will eventually stop clicking. The probability that at any step he stops clicking is damping factor d and its taken to be 0.85. A transition matrix is created with information of the linking of the web pages. Initially the probability is same but then gets updated for every iteration and then becomes constant. After some iterations the new probability of the web pages is obtained and sorted in descending order of probability. A high probability means that particular page had lot of incoming links and less outgoing links, indicating that it is important.

In textrank algorithm the Sentence Similarity Matrix is the equivalent of transition matrix. The similarity between sentences is computed using number of common words, cosine distance, BM25 or BM25+. When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation of the documents and not the magnitude. If cosine similarity is 1, it means the sentences are exactly alike and 0 means they are completely different.

When page rank is applied on the sentence similarity matrix, probabilities of the sentences is obtained and then sorted based on their values. Depending on the number of sentences wanted in the summary, the ones with higher probability are selected.

CONCLUSION:

Thus the textrank algorithm is an extractive approach which is used extensively by lot of people. But it is still being improved. It is an unsupervised model.

Textrank implementations tend to be lightweight and can run fast even with limited memory resources. Some textrank implementations can be directed by adding semantic relations, which can be used to enrich the graph – or incorporating human-in-the-loop approaches. This can provide advantage over supervised learning models which are trained purely on data. Text rank is more transparent which becomes important in context of model bias and data ethics. Textrank algorithm is a fast and simple technique for text summarization.