A

Report

On

TEXT summarization

By

Amit Yadav

ML group 33.o

List of chapters

1.What is Natural Language Processing (NLP)?

The field of study that concentrate on the interactions between human language and computers is called Natural Language Processing or NLP. Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence which deals with interactions of computers and human language. The objective is a computer should capable of "understanding" the contents of documents, including the contextual tones of the language within them. The technology can then accurately extract information and perceptions contained in the documents as well as categorize and unify the documents themselves. NLP draws from many modifications, including computer science and computational linguistics, its pursuit to cover the gap between human communication and computer understanding.

To deal with humans, a program must understand composition (grammar), semantics (word meaning), and morphology (tense), pragmatics (conversation). In NLP, some of these tasks translate to tokenization, chunking, part of speech tagging, parsing, machine translation, speech recognition, and most of them are still the toughest challenges for computers. Currently, NLP is one of the rarest skill sets that is required in the industry. After the advent of big data, the major challenge is that we need more people who are good with not just structured, but also with semi or unstructured data.

2.Natural language Interfaces

It important that the interface between human and computer be user-friendly, in the sense of being easy to learn, use and understand. The natural language interfaces are not necessarily the best solution for the all applications. A database management system enables the information in a database to be altered, sorted and retrieved at the command of user. There are innumerable questions and request that a user might want to issue from such a system.

To achieve some of the above applications and other basic NLP pre-processing, there are many open-source tools available. Some of them are developed by organizations to build their own NLP applications, while some of them are open-sourced. Here is a small list of available NLP tools:

• GATE
• Mallet
• Open NLP
• UIMA
• Stanford toolkit
• Genism
• **Natural Language Tool Kit** (**NLTK**)

Most of the tools are written in Java and have similar functionalities. However NLTK is also a very good learning kit because the learning curve of Python (on which NLTK is written) is very fast. NLTK has incorporated most of the NLP tasks, it's very elegant and easy to work with. For all these reasons, NLTK has become one of the most popular libraries in the NLP community.

3.What is Natural language tool kit (NLTK)

NLTK defines a basic infrastructure that can be used to build NLP programs in Python. It provides basic classes for representing data relevant to natural language processing, standard interfaces for performing tasks such as tokenization, tagging, and parsing, standard implementations for each task which can be combined to solve complex problems and extensive documentation including tutorials and reference documentation.

NLTK was designed with six requirements in mind:

Ease of use:

The primary purpose of the toolkit is to allow students to concentrate on building natural language processing systems. The more time students must spend learning to use the toolkit, the less useful it is. We have provided software distributions for several platforms, along with platform-specific instructions, to make the toolkit easy to install.

Consistency:

We have made a significant effort to ensure that all the data structures and interfaces are consistent, making it easy to carry out a variety of tasks using a uniform framework.

Extensibility:

The toolkit easily accommodates new components, whether those components replicate or extend existing functionality. Moreover, the toolkit is organized so that it is usually obvious where extensions would fit into the toolkit's infrastructure.

Simplicity:

We have tried to provide an intuitive and appealing framework along with substantial building blocks, for students to gain a practical knowledge of NLP without getting bogged down in the tedious house-keeping usually associated with processing annotated language data.

Modularity:

The interaction between different components of the toolkit uses simple, well-defined interfaces. It is possible to complete individual projects using small parts of the toolkit, without needing to understand how they interact with the rest of the toolkit. This allows students to learn how to use the toolkit incrementally throughout a course. Modularity also makes it easier to change and extend the toolkit.

Well-Documented:

The toolkit comes with substantial documentation, including nomenclature, data structures, and implementations.

The most fundamental NLTK components are for identifying and manipulating individual words of text. These include: **tokenize**, for breaking up strings of characters into word tokens; **tag**, for adding part-of-speech tags, including regular-expression taggers, n-gram taggers and Brill taggers; and the Porter stemmer.

The second kind of module is for creating and manipulating structured linguistic information. These components include: **tree**, for representing and processing parse trees; **feature structure**, for building and unifying nested feature structures (or attribute-value matrices), for specifying free grammars; and **parse**, for creating parse trees over input text, including chart parsers, chunk parsers and probabilistic parsers.

4.Steps involved in NLP

There is no right way to do NLP analysis as one can explore multiple ways and take different approaches to handle text data. However, from a Machine Learning standpoint, there are five major steps that one should take to make the text data ready for analysis. The five major steps involved in NLP are:

1. Reading the corpus
2. Sentence splitter
3. Tokenization
4. Cleaning /Stopword removal
5. Stemming
6. Converting into Numerical Form
7.

5.Corpus

A corpus is known as the entire collection of text documents. For example, suppose we have thousands of pages in a collection that we need to process and analyse for our use. This group of pages is known as a corpus as it contains all the text documents. The next step in text processing is tokenization.

6.Sentence Splitter

Some of the NLP applications require splitting a large raw text into sentences to get more meaningful information out. Intuitively, a sentence is an acceptable unit of conversation. When it comes to computers, it is a harder task than it looks. A typical sentence splitter can be something as simple as splitting the string on (.), to something as complex as a predictive classifier to identify sentence boundaries.

For example: -

input string: -

' This is an example sent. The sentence splitter will split on sent markers. Ohh really !!'
Output: -

' This is an example sent', …….1
'The sentence splitter will split on sent markers. '…….2
'Ohh really !!'……...3

7.Tokenization

A word (*Token*) is the minimal unit that a machine can understand and process. So any text string cannot be further processed without going through tokenization. The method of dividing the given sentence or collection of words of a text document into separate individual words is known as tokenization. It removes the unnecessary characters such as punctuation. For example, if we have a sentence such as:

Input: He really liked the London City. He is there for two more days.

Tokens:
He, really, liked, the, London, City, He, is, there, for, two, more, days

We end up with 13 tokens for the above input sentence. In the preceding code we have used various tokenizers. The word_tokenize method is a generic and more robust method of tokenization for any kind of text corpus. The word_tokenize method comes pre-built with NLTK. The other is regex_tokenize, which is more of a customized tokenizer for the specific needs of the user. Most of the other tokenizers can be derived from regex tokenizers. You can also build a very specific tokenizer using a different pattern.

8.Stopwords Removal

Stop word removal is one of the most commonly used pre-processing steps across different NLP applications. The idea is simply removing the words that occur commonly across all the documents in the corpus. Typically, articles and pronouns are generally classified as stop words. These words have no significance in some of the NLP tasks like information retrieval and classification, which means these words are not very discriminative. On the contrary, in some NLP applications stop word removal will have very little impact. Most of the time, the stop word list for the given language is a well hand-curated list of words that occur most commonly across corpuses As you can observe, the tokens column contains very common words such as ‘this’, ‘the’, ‘to’,‘was’, ‘that’, etc. These words are known as stopwords and they seem to add very little value to the analysis. If they are to be used in analysis, it increases the computation overhead without adding too much value or insight. Hence, it's always considered a good idea to drop these stopwords from the tokens.

For Example: -

Input = "This is just a test"

Output = “test”

9.Stemming

Stemming, in literal terms, is the process of cutting down the branches of a tree to its stem. So effectively, with the use of some basic rules, any token can be cut down to its stem. Stemming is more of a crude rule-based process by which we want to club together different variations of the token. For example, the word *eat* will have variations like eating, eaten, eats, and so on. In some applications, as it does not make sense to differentiate between eat and eaten, we typically use stemming to club both grammatical variances to the root of the word. While stemming is used most of the time for its simplicity, there are cases of complex language or complex NLP tasks where it's necessary to use lemmatization instead. Lemmatization is a more robust and methodical way of combining grammatical variations to the root of a word.

A basic rule-based stemmer, like removing *–s/es* or *-ing* or *-ed* can give you a precision of more than 70 percent, while **Porter stemmer** also uses more rules and can achieve very good accuracies.

10.Converting into Numerical Form

This is the methodology through which we can represent the text data into numerical form for it to be used by Machine Learning or any other analysis. Text data is generally unstructured and varies in its length. BOW (Bag of Words) allows us to convert the text form into a numerical vector form by considering the occurrence of the words in text documents.

For example,

Doc 1: The best thing in life is to travel

Doc 2: Travel is the best medicine

Doc 3: One should travel more often

Vocabulary:

The list of unique words appearing in all the documents in known as vocabulary. In the above example, we have 13 unique words that are part of the vocabulary. Each document can be represented by this vector of fixed size 13.

The best thing in life is to travel medicine one should more often

---

Another element is the representation of the word in the particular
document using a Boolean value.
(1 or 0).
Doc 1:

The best thing in life is to travel medicine one should more often

---

1 1 1 1 1 1 1 1 0 0 0 0 0

Doc 2:

The	best	thing	in	life	is	to	travel	medicine	one	should	more	often
1	1	0	0	0	1	0	1	1	0	0	0	0

The BOW does not consider the order of words in the document and the semantic meaning of the word and hence is the most baseline method to represent the text data into numerical form. There are other ways by which we can convert the textual data into numerical form, which are mentioned in the next section. We will use PySpark to go through each one of these methods.

10.1Count Vectorizer

In BOW, we saw the representation of occurrence of words by simply 1 or 0 and did not consider the frequency of the words. The count vectorizer instead takes count of the word appearing in the particular document. We will use the same text documents that we created earlier while using tokenization. We first import the Count Vectorizer.

[In]: from pyspark.ml.feature import CountVectorizer
[In]: count_vec=CountVectorizer(inputCol='refined_tokens',
outputCol='features')
[In]: cv_df=count_vec.fit(refined_df).transform(refined_df)
[In]: cv_df.select(['user_id','refined_tokens','features']).
show(4,False)
[Out]:

```
+-------+---------------------------------+-------------------------------+
|user_id|refined_tokens                   |features                       |
+-------+---------------------------------+-------------------------------+
|1      |[really, liked, movie]           |(11,[0,4,9],[1.0,1.0,1.0])     |
|2      |[recommend, movie, friends]      |(11,[0,6,10],[1.0,1.0,1.0])    |
|3      |[movie, alright, acting, horrible]|(11,[0,2,3,5],[1.0,1.0,1.0,1.0])|
|4      |[never, watching, movie, ever]   |(11,[0,1,7,8],[1.0,1.0,1.0,1.0])|
+-------+---------------------------------+-------------------------------+
```

As we can observe, each sentence is represented as a dense vector. It shows that the vector length is 11 and the first sentence contains 3 values at the 0th, 4th, and 9th indexes.

To validate the vocabulary of the count vectorizer, we can simply use the vocabulary function.

```
[In]: count_vec.fit(refined_df).vocabulary
[Out]:
['movie',
'horrible',
'really',
'alright',
'liked',
'friends',
'recommend',
'never',
'ever',
'acting',
'watching']
```

So, the vocabulary size for the above sentences is 11 and if you look at the features carefully, they are similar to the input feature vector that we have been using for Machine Learning in PySpark. The drawback of using the Count Vectorizer method is that it doesn't consider the co-occurrences of words in other documents. In simple terms, the words appearing more often would have a larger impact on the feature vector. Hence, another approach to convert text data into numerical form is known as Term Frequency – inverse Document Frequency (TF-IDF).

10.2TF-IDF

This method tries to normalize the frequency of word occurrence based on other documents. The whole idea is to give more weight to the word if appearing a high number of times in the same document but penalize if it is appearing a higher number of times in other documents as well. This indicates that a word is common across the corpus and is not as important as its frequency in the current document indicates. Term Frequency: Score based on the frequency of word in current document. Inverse Document Frequency: Scoring based on

frequency of documents that contains the current word. Now, we create features based on TF-IDF in PySpark using the same refined df dataframe.

[In]: from pyspark.ml.feature import HashingTF,IDF

[In]: hashing_
vec=HashingTF(inputCol='refined_tokens',
outputCol='tf_features')

[In]: hashing_df=hashing_vec.transform(refined_df)

[In]: hashing_df.select(['user_id','refined_tokens',
'tf_features']).show(4,False)

[Out]:

```
+-------+---------------------------------+-----------------------------------------------------+
|user_id|refined_tokens                   |tf_features                                          |
+-------+---------------------------------+-----------------------------------------------------+
|1      |[really, liked, movie]           |(262144,[14,32675,155321],[1.0,1.0,1.0])             |
|2      |[recommend, movie, friends]      |(262144,[129613,155321,222394],[1.0,1.0,1.0])        |
|3      |[movie, alright, acting, horrible]|(262144,[80824,155321,236263,240286],[1.0,1.0,1.0,1.0])|
|4      |[never, watching, movie, ever]   |(262144,[63139,155321,203802,245806],[1.0,1.0,1.0,1.0])|
+-------+---------------------------------+-----------------------------------------------------+
```

[In]: tf_idf_vec=IDF(inputCol='tf_features',outputCol='tf_idf_
features')

[In]: tf_idf_df=tf_idf_vec.fit(hashing_df).transform(hashing_df)

[In]: tf_idf_df.select(['user_id','tf_idf_features']).show(4,False)

[Out]:

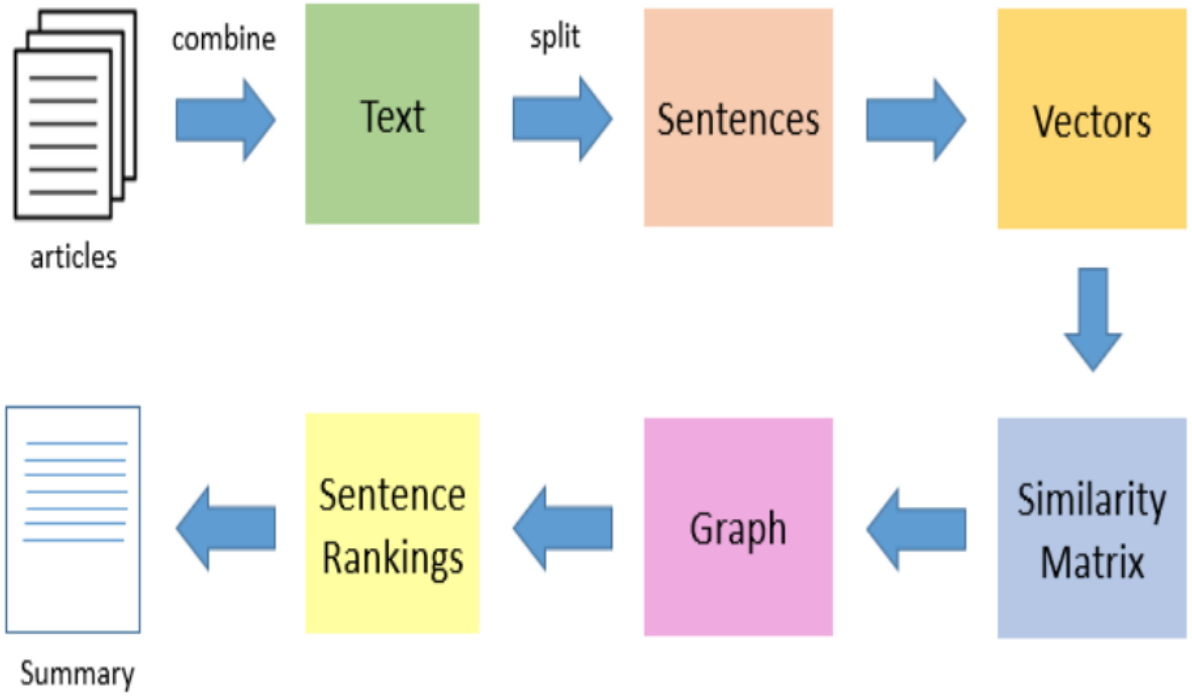
```
+-------+-----------------------------------------------------------------------------------------------------+
|user_id|tf_idf_features                                                                                      |
+-------+-----------------------------------------------------------------------------------------------------+
|1      |(262144,[14,32675,155321],[0.9162907318741551,0.9162907318741551,0.0])                               |
|2      |(262144,[129613,155321,222394],[0.9162907318741551,0.0,0.9162907318741551])                          |
|3      |(262144,[80824,155321,236263,240286],[0.9162907318741551,0.0,0.9162907318741551,0.9162907318741551])|
|4      |(262144,[63139,155321,203802,245806],[0.9162907318741551,0.0,0.9162907318741551,0.9162907318741551])|
+-------+-----------------------------------------------------------------------------------------------------+
```

11.Text Summarization: -

Text summarization is the problem of reducing the number of sentences and words of a document without changing its meaning. Text summarization is a very useful and important part of Natural Language Processing (NLP).

NLP (Natural Language Processing) is the field of artificial intelligence that studies the interactions between computers and human languages, in particular how to program computers to process and analyse large amounts of natural language data. The hardest NLP tasks are the ones where the output isn't a single label or value (like Classification and Regression), but a full new text (like Translation, Summarization and Conversation).

In this approach we build algorithms or programs which will reduce the text size and create a summary of our text data. This is called automatic text summarization in machine learning. Text summarization is the process of creating shorter text without removing the semantic structure of text.



1. Text summarization Working

12.List of text summarization methods

12.1Cluster Based Method

Some automatic summarization systems use clusters to generate a significant summary approaching the various topics of the document. The documents are represented using term frequency-inverse document frequency (TF-IDF). In this context the term frequency (TF) is the average number of occurrence (by document) in the cluster. The topic is represented by words of which the value TF-IDF is higher in the cluster. The selection of the relevant sentences is based on the similarity of the sentences with the topic of cluster .The measurement of similarity between the sentences is calculated according to the similarity of the words between two sentences and the semantic similarity of words. Then, the K-means method is used to gather the sentences of the document in the clusters.
A method based on the following steps: terms selection, terms weighting and sentences selection. In the first step, one of the three models of the text is extracted: Bag-of-words model, n-grams model and Maximal frequent Sequence (MFS) model.
In the second stage, the terms are weighted by using the Boolean method, TF, IDF or TF-IDF. In the third steps, the Expectation-maximization algorithm (EM) is used to form similar groups of sentences in order to obtain a sentence representing each cluster to be included in the summary.

12.2. Topic-Based Method

An approach which combines the automatic topics identification technique with the terms frequency method. This methodology consists of calculating initially the similarity between

the sentences, then carry out the identification of the subject covered by gathering similar sentences in clusters.
In a second stage, and based on terms frequency, the projecting sentences are selected starting from the local topics already identified. Kuo and Chen use not only the frequency of terms to detect relevant information in a text, the authors also use informative words and event-driven. This type of words indicates concepts and the important relations which can be used to detect important sentences in the text.

12.3. Method Based on Lexical Chains

The automatic text summarization by lexical chains was introduced in Barzilay and Elhadad . This method uses the WordNet database knowledge to determine the relations of cohesion between terms then composes chains based on these terms. Scores are given based on the number and type of relation in the chains. The final summary contains the sentences where the strongest chains are very concentrated.
A similar method with a graphs using the knowledge bases of WordNet and Wikipedia was presented in Pourvali and Abadeh Mohammad .
This method consists initially in finding the exact meaning of each word in the text using WordNet, then builds the lexical chains and removes those which have a weak score compared with the others. The structure of the lexical cohesion of the text can be exploited to determine the importance of a sentence.

12.4. Discourse Based Method

New techniques were born to solve the problem of automatic summarization; these techniques are based on the analysis of discourse and its structure. Among these techniques we quote the Rhetorical Structure Theory (RST). Moreover, Khan, et al. combines the RST with a generic summarizer to add linguistic knowledge to the process of automatic summarization. But this mixed approach could not improve the results obtained by the generic summarizer.
In other words, the disadvantage of this approach is found at the analyzer level which could not detect all RST relations, in fact, a good analysis and languages knowledge could have improved the output of the summary system. In the paper published by Li Chengcheng ,

the system extracts the rhetorical structure of the text and the components of the rhetorical relations between the sentences, then calculates the weight of each sentence of the text according to its utility and removes the least important parts of the structure having a weak weight.

12.5. Graphs Based Method

LexRank and TextRank are the most important algorithms used in automatic summarization system based on graph method. In the same context, proposed a method based on graphs algorithm for automatic texts summarization. This method consists in building a graph from the text. Nodes of the graph are represented by the text sentences, for each sentence there is a node. The edge of the graph represent connection (lexical or semantic) between the sentences, this connection is evaluated by calculating the similarity between the sentences.

The weight of each node is calculated by using the function COS. After that the summary is made up by taking the shortest way which starts with the first sentence of the original text and finishes with the last sentence. In addition, SUMGRAPH and Time stamped Graph are two automatic summarization systems based on graphs.

12.6. Latent Semantic Analysis (LSA) Based Method

LSA is an algebraic-statistical method that extracts and represents semantic knowledge of the text based on the observation of the co-occurrence of words. This technique aims to builds a semantic space with very large dimension from the statistical analysis of the whole co-occurrences in a corpus of texts. The starting point of LSA consists of a lexical table which contains the number of occurrences of each word in each document.
Gong and Liu proposed an automatic summarization system of news text with the use of LSA as a way to identify the important topics in the documents without using lexical resources like WordNet. In this way, the SVD is applied to matrix A to decompose into three new matrices as follows:

$$A = UWVT.$$

The suggested that the row of the matrix VT can be considered as various topics covered in the original text, while each column represents a sentence in the document. And finally, in order to produce an extractive summary, they consider each row of matrix VT consecutively, and select the sentence with the highest value.

In Yeh, et al. another method using LSA was proposed. It is a mixed approach between graphs based method and LSA based method. After using the SVD on a matrix of words per sentence and reduction of these dimensions, the corresponding matrix A'=U'Σ'V'T is built. Each column of A' denotes the sentence semantic representation which is used, instead of an occurrence frequency vector of keyword, in order to represent document as a graph of relations between sentences.

A ranking algorithm is then applied to the resulting graph. In the same context, a Non-negative Matrix Factorization (NMF) algorithm was proposed in Mashechkin, et al,instead of the SVD, to reduce the dimensions of the matrix. The idea is that from the matrix A whose columns are the n sentences of text and rows are the m terms, and since the elements of A are non-negative, so NMF can then decompose the matrix A into two positive matrices Wk and Hk. in order to approximate the matrix A in the decomposition form $Ak \approx WkHk$. Matrices Wk correspond to the mapping of space of k topics and the space of m terms, and Hk correspond to the representation of the sentences in the space of topics. Subsequently, we can find out what the terms of the text best characterize each topics associated with the columns of the matrix Wk.

12.7. Method Based on Fuzzy Logic

In Farshad, et al. , another approach to automatic summarization has been proposed, this time it is based on a fuzzy logic. This method takes into account every feature of the text such as word frequency, similarity to keywords, similarity to the title words, sentences position, statistics of co-occurrence of lexical chain, indicative expression etc. After extracting these features and depending on the results, a value of 0-1 is assigned to each sentence of the text according to the characteristics of sentences and rules available in the knowledge base.

The value obtained at the output determines the degree of importance of the sentence in the final summary. In Esther Hannah, et al. [18], different characteristics of each sentence were taken into account, such as title words, sentence length, term weight, Sentence to sentence similarity, etc. the values of these features are used by the inference engine to generate the score of each sentence of the text.

13.Text summarization using one method

One application of text analytics and NLP is Text Summarization. Text Summarization Python helps in summarizing and shortening the text in user feedback. It can be done with the help of an algorithm that can help reduce the text bodies while keeping their original meaning intact or by giving insights into their original text.

Two different approaches are used for Text Summarization

- Extractive Summarization
- Abstractive Summarization

13.1 Extractive Summarization

In Extractive Summarization, we identify essential phrases or sentences from the original text and extract only these phrases from the text. These extracted sentences would be the summary.

13.2Abstractive Summarization

We work on generating new sentences from the original text in the Abstractive Summarization approach. The abstractive method contrasts the approach described above, and the sentences generated through this approach might not even be present in the original text.

We are going to focus on using extractive methods. This method functions by identifying meaningful sentences or excerpts from the text and reproducing them as part of the summary. In this approach, no new text is generated; only the existing text is used in the summarization process.

Steps for Implementation

Step 1: The first step is to import the required libraries. Two NLTK libraries are necessary for building an efficient text summarizer.

```
1from nltk.corpus import stopwords
2from nltk.tokenize import word_tokenize, sent_tokenize
```

Terms Used:

- **Corpus**
  A collection of text is known as Corpus. This could be data sets such as bodies of work by an author, poems by a particular poet, etc. To explain this concept in the blog, we will use a data set of predetermined stop words.
- **Tokenizers**
  This divides a text into a series of tokens. Tokenizers have three

primary tokens – sentence, word, and regex tokenizer. We will be using only the word and the sentence tokenizer.

Step 2: Remove the Stop Words and store them in a separate array of words.

Stop Words

Words such as ***is, an, a, the,*** and '***for***' do not add value to the meaning of a sentence. For example, let us take a look at the following sentence:

GreatLearning is one of the most valuable websites for ArtificialIntelligence aspirants.

After removing the stop words in the above sentence, we can narrow the number of words and preserve the meaning as follows:

['GreatLearning', 'one', 'useful', 'website', 'ArtificialIntelligence', 'aspirants', '.']

Step 3: We can then create a frequency table of the words.

A Python Dictionary can record how many times each word will appear in the text after removing the stop words. We can use this dictionary over each sentence to know which sentences have the most relevant content in the overall text.

```
stopwords = set (stopwords.words("english"))
words = word_tokenize(text)
freqTable = dict()
```

Step 4: We will assign a score to each sentence depending on the words it contains and the frequency table.

Here, we will use the **sent_tokenize()** method that can be used to create the array of sentences. We will also need a dictionary to keep track of the score of each sentence, and we can later go through the dictionary to create a summary.

```
sentences = sent_tokenize(text)
sentenceValue = dict()
```

Step 5: Assign a score to compare the sentences within the text.

A straightforward approach that can be used to compare the scores is to find an average score of a particular sentence, which can be a reasonable threshold.

```
sumValues = 0
for sentence in sentenceValue:
```

```
3sumValues += sentenceValue[sentence]

4average = int(sumValues / len(sentenceValue))
```

Apply the threshold value and the stored sentences in order into the summary.

Output Summary

This brings us to the end of the blog on Text Summarization Python. We hope that you were able to learn more about the concept. If you wish to learn more about such concepts, take up the Python for Machine Learning free online course Great Learning Academy offers.

14.References

N. D. (2016). Automatic Text Summarization Using Supervised Machine Learning Technique for Hindi Langauge. *International Journal of Research in Engineering and Technology*, *05*(06), 361–367. https://doi.org/10.15623/ijret.2016.0506065
Ab, A., & Sunitha, C. (n.d.). *An Overview on Document Summarization Techniques*. 113–118.

Al-Radaideh, Q. A., & Bataineh, D. Q. (2018a). A Hybrid Approach for Arabic Text Summarization Using Domain Knowledge and Genetic Algorithms. *Cognitive Computation*, *10*(4), 651–669. https://doi.org/10.1007/s12559-018-9547-z

Al-Radaideh, Q. A., & Bataineh, D. Q. (2018b). A Hybrid Approach for Arabic Text Summarization Using Domain Knowledge and Genetic Algorithms. *Cognitive Computation*, *10*(4), 651–669. https://doi.org/10.1007/s12559-018-9547-z

Alami, N., Mallahi, M. El, Amakdouf, H., & Qjidaa, H. (2021). Hybrid method for text summarization based on statistical and semantic treatment. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-021-10613-9

Alami, N., Meknassi, M., & En-nahnahi, N. (2019). Enhancing unsupervised neural networks based text summarization with word embedding and ensemble learning. *Expert Systems with Applications*, *123*, 195–211. https://doi.org/10.1016/j.eswa.2019.01.037

Alami, N., Meknassi, M., En-nahnahi, N., El Adlouni, Y., & Ammor, O. (2021). Unsupervised neural networks for automatic Arabic text summarization using document clustering and topic modeling. *Expert Systems with Applications*, *172*. https://doi.org/10.1016/j.eswa.2021.114652

ALGULIEV, R., & ALIGULIYEV, R. (2009). Evolutionary Algorithm for Extractive Text Summarization. *Intelligent Information Management*, *01*(02), 128–138. https://doi.org/10.4236/iim.2009.12019

Alguliev, R. M., Aliguliyev, R. M., Hajirahimova, M. S., & Mehdiyev, C. A. (2011a). MCMR: Maximum coverage and minimum redundant text summarization model. *Expert Systems with Applications*, *38*(12), 14514–14522. https://doi.org/10.1016/j.eswa.2011.05.033

Alguliev, R. M., Aliguliyev, R. M., Hajirahimova, M. S., & Mehdiyev, C. A. (2011b). MCMR: Maximum coverage and minimum redundant text summarization model. *Expert Systems with Applications*, *38*(12), 14514–14522. https://doi.org/10.1016/j.eswa.2011.05.033

Alguliyev, R. M., Aliguliyev, R. M., Isazade, N. R., Abdi, A., & Idris, N. (2019a). COSUM: Text summarization based on clustering and optimization. *Expert Systems*, *36*(1), 1–17. https://doi.org/10.1111/exsy.12340

Alguliyev, R. M., Aliguliyev, R. M., Isazade, N. R., Abdi, A., & Idris, N. (2019b). COSUM: Text summarization based on clustering and optimization. *Expert Systems*, *36*(1). https://doi.org/10.1111/exsy.12340

Antiqueira, L., Oliveira, O. N., Costa, L. da F., & Nunes, M. das G. V. (2009). A complex network approach to text summarization. *Information Sciences*, *179*(5), 584–599. https://doi.org/10.1016/j.ins.2008.10.032

Aone, C., Okurowski, M. E., Gorlinsky, J., & Larsen, B. (1997). A Scalable Summarization System Using Robust NLP. *Proceedings of the Intelligent Scalable Text Summarization Workshop*, 66–73.

Banko, M., Mittal, V. O., & Witbrock, M. J. (2000). *Headline generation based on statistical translation*. 318–325. https://doi.org/10.3115/1075218.1075259

Barzilay, R., & Mckeown, K. R. (2005). *Sentence Fusion for Multidocument News Summarization*.

Belwal, R. C., Rai, S., & Gupta, A. (2021). A new graph-based extractive text summarization using keywords or topic modeling. *Journal of Ambient Intelligence and Humanized Computing*, *12*(10), 8975–8990. https://doi.org/10.1007/s12652-020-02591-x

Bhat, I. K., Mohd, M., & Hashmy, R. (2018). SumItUp: A Hybrid Single-Document Text Summarizer. *Advances in Intelligent Systems and Computing*, *583*(April), 619–634. https://doi.org/10.1007/978-981-10-5687-1_56

Binwahlan, M. S., Salim, N., & Suanmali, L. (2009a). Swarm Based Text Summarization. *2009 International Association of Computer Science and Information Technology - Spring Conference, IACSIT-SC 2009*, 145–150. https://doi.org/10.1109/IACSIT-SC.2009.61

Binwahlan, M. S., Salim, N., & Suanmali, L. (2010). Fuzzy swarm diversity hybrid model for text summarization. *Information Processing and Management*, *46*(5), 571–588. https://doi.org/10.1016/j.ipm.2010.03.004

Binwahlan, M. S., Salim, N., & Suanmali, L. (2009b). Swarm Based Text Summarization. *2009 International Association of Computer Science and Information Technology - Spring Conference, IACSIT-SC 2009*, 145–150. https://doi.org/10.1109/IACSIT-SC.2009.61

Bloom, ; E H, Fischer, H., Charbonneau, N. K., Tonks, ; J, Mirkovitch, J. E., Sadowski, D. ; H. B., Shuai, K., Darnell, J. E., & Gilman, M. Z. (n.d.). Automatic Analysis, Theme Generation, and Summarization of Machine-Readable Texts. In *Interferon: Principles and Medical Applications* (Vol. 13).

Bouras, C., & Tsogkas, V. (2010). Noun retrieval effect on text summarization and delivery of personalized news articles to the user's desktop. *Data and Knowledge Engineering*, *69*(7), 664–677. https://doi.org/10.1016/j.datak.2010.02.005

Brin, S., & Page, L. (2012). Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, *56*(18), 3825–3833. https://doi.org/10.1016/j.comnet.2012.10.007

Carbonell, J., & Goldstein, J. (1998). The use of MMR, diversity-based reranking for reordering documents and producing summaries. In Proceedings of the 21st annual

international ACM SIGIR conference on Research and development in information retrieval, 335-336.

Chakraborty, R., Bhavsar, M., Dandapat, S. K., & Chandra, J. (2019). Tweet summarization of news articles: An objective ordering-based perspective. *IEEE Transactions on Computational Social Systems*, *6*(4), 761–777. https://doi.org/10.1109/TCSS.2019.2926144

Chen, Q. (2015). *LCSTS: A Large Scale Chinese Short Text Summarization Dataset*.
De Tre, G., Hallez, A., & Bronselaer, A. (2014). Performance optimization of object comparison. *International Journal of Intelligent Systems*, *29*(2), 495–524. https://doi.org/10.1002/int

Dohare, S., Karnick, H., & Gupta, V. (2017a). Text Summarization using Abstract Meaning Representation. *ArXiv*.

Dohare, S., Karnick, H., & Gupta, V. (2017b). *Text Summarization using Abstract Meaning Representation*. http://arxiv.org/abs/1706.01678

El-Kassas, W. S., Salama, C. R., Rafea, A. A., & Mohamed, H. K. (2020). EdgeSumm: Graph-based framework for automatic text summarization. *Information Processing and Management*, *57*(6), 102264. https://doi.org/10.1016/j.ipm.2020.102264

Farzindar, A., & Lapalme, G. (2004). Legal Text Summarization by Exploration of the Thematic Structure and Argumentative Roles. *In Text Summarization Branches Out Conference Held in Conjunction with ACL 2004*, 27–38.

Feblowitz, J. C., Wright, A., Singh, H., Samal, L., & Sittig, D. F. (2011). Summarization of clinical information: A conceptual model. *Journal of Biomedical Informatics*, *44*(4), 688–699. https://doi.org/10.1016/j.jbi.2011.03.008

Ferreira, R., De Souza Cabral, L., Freitas, F., Lins, R. D., De França Silva, G., Simske, S. J., & Favaro, L. (2014). A multi-document summarization system based on statistics and linguistic treatment. *Expert Systems with Applications*, *41*(13), 5780–5787. https://doi.org/10.1016/j.eswa.2014.03.023

Ferreira, R., De Souza Cabral, L., Lins, R. D., Pereira E Silva, G., Freitas, F., Cavalcanti, G. D. C., Lima, R., Simske, S. J., & Favaro, L. (2013). Assessing sentence scoring techniques for extractive text summarization. In *Expert Systems with Applications* (Vol. 40, Issue 14, pp. 5755–5764). https://doi.org/10.1016/j.eswa.2013.04.023

Gambhir, M., & Gupta, V. (2017). Recent automatic text summarization techniques: a survey. *Artificial Intelligence Review*, *47*(1). https://doi.org/10.1007/s10462-016-9475-9

Garner, R. (1982). Efficient text summarization: Costs and benefits. *Journal of Educational Research*, *75*(5), 275–279. https://doi.org/10.1080/00220671.1982.10885394

Gupta, V., & Kaur, N. (2016). A Novel Hybrid Text Summarization System for Punjabi Text. *Cognitive Computation*, *8*(2), 261–277. https://doi.org/10.1007/s12559-015-9359-3

HANSON ER. (1971). Musicassette Interchangeability. the Facts Behind the Facts. *AES: Journal of the Audio Engineering Society*, *19*(5), 417–425.

He, J., Kryscinski, W., McCann, B., Rajani, N., & Xiong, C. (2020). CTRLsum: Towards generic controllable text summarization. *ArXiv*, 1–35.

He, J., Kryściński, W., McCann, B., Rajani, N., & Xiong, C. (2020). *CTRLsum: Towards Generic Controllable Text Summarization*. http://arxiv.org/abs/2012.04281

Hovy, E., & Lin, C.-Y. (n.d.). *AUTOMATED TEXT SUMMARIZATION AND THE SUMMARIST SYSTEM*.

Hovy, E., & Lin, C.-Y. (1996). *Automated text summarization and the SUMMARIST system*. 197. https://doi.org/10.3115/1119089.1119121

Hu, M., & Liu, B. (2004). Mining opinion features in customer reviews. *Proceedings of the National Conference on Artificial Intelligence*, 755–760.

Hu, Y. H., Chen, Y. L., & Chou, H. L. (2017). Opinion mining from online hotel reviews – A text summarization approach. *Information Processing and Management*, *53*(2), 436–449. https://doi.org/10.1016/j.ipm.2016.12.002

Huang, L., He, Y., Wei, F., & Li, W. (2010). *Modeling Document Summarization as Multi-objective Optimization*. 2–6. https://doi.org/10.1109/IITSI.2010.80

Ježek, K., Katedra, J. S., & Steinberger, J. (2007). *Automatic Text Summarization (The state of the art 2007 and new challenges)*.

Jezek, K., & Steinberger, J. (2008). Automatic summarizing: (The state of the art 2007 and new challenges). *Proceedings of Znalosti*, 1–12.

Joshi, A., Fidalgo, E., Alegre, E., & Fernández-Robles, L. (2019). SummCoder: An unsupervised framework for extractive text summarization based on deep auto-encoders. *Expert Systems with Applications*, *129*, 200–215. https://doi.org/10.1016/j.eswa.2019.03.045

Kanapala, A., Pal, S., & Pamula, R. (2019). Text summarization from legal documents: a survey. *Artificial Intelligence Review*, *51*(3), 371–402. https://doi.org/10.1007/s10462-017-9566-2

Kavila, S. D., Puli, V., Raju, G. S. V. P., & Bandaru, R. (2020). *An Automatic Legal Document Summarization and Search Using Hybrid System An Automatic Legal Document Summarization*. *January 2013*. https://doi.org/10.1007/978-3-642-35314-7

Kiyani, F., & Tas, O. (2017). A survey automatic text summarization. *Pressacademia*, *5*(1), 205–213. https://doi.org/10.17261/pressacademia.2017.591

Ko, Y., & Seo, J. (2008). An effective sentence-extraction technique using contextual information and statistical approaches for text summarization. *Pattern Recognition Letters*, *29*(9), 1366–1371. https://doi.org/10.1016/j.patrec.2008.02.008

Koupaee, M., & Wang, W. Y. (2018). *WikiHow: A Large Scale Text Summarization Dataset*. http://arxiv.org/abs/1810.09305

Kryściński, W., McCann, B., Xiong, C., & Socher, R. (2019a). Evaluating the factual consistency of abstractive text summarization. *ArXiv*. https://doi.org/10.18653/v1/2020.emnlp-main.750

Kryściński, W., McCann, B., Xiong, C., & Socher, R. (2019b). *Evaluating the Factual Consistency of Abstractive Text Summarization*. http://arxiv.org/abs/1910.12840

Kryściński, W., Paulus, R., Xiong, C., & Socher, R. (2018). *Improving Abstraction in Text Summarization*. http://arxiv.org/abs/1808.07913

Kryściński, W., Paulus, R., Xiong, C., & Socher, R. (2020). Improving abstraction in text summarization. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, 1808–1817. https://doi.org/10.18653/v1/d18-1207

Kumar, N. V., & Reddy, M. J. (n.d.). *Factual Instance Tweet Summarization and Opinion Analysis of Sport Competition*. Springer Singapore. https://doi.org/10.1007/978-981-13-3393-4

Kutlu, M., Ciğir, C., & Cicekli, I. (2010). Generic text summarization for Turkish. *Computer Journal*, *53*(8), 1315–1323. https://doi.org/10.1093/comjnl/bxp124

Li, P., Lam, W., Bing, L., & Wang, Z. (2017). *Deep Recurrent Generative Decoder for Abstractive Text Summarization*. http://arxiv.org/abs/1708.00625

Lin, C., & Rey, M. (2001). *R OUGE : A Package for Automatic Evaluation of Summaries*.
Lin, C. Y. (1999). Training a selection function for extraction. *International Conference on Information and Knowledge Management, Proceedings*, 55–62. https://doi.org/10.1145/319950.319957

Ling, X., Jiang, J., He, X., Mei, Q., Zhai, C., & Schatz, B. (2007). Generating gene summaries from biomedical literature: A study of semi-structured summarization. *Information Processing and Management*, *43*(6), 1777–1791. https://doi.org/10.1016/j.ipm.2007.01.018

Linhares Pontes, E., Huet, S., Torres-Moreno, J. M., & Linhares, A. C. (2020). Compressive approaches for cross-language multi-document summarization. *Data and Knowledge Engineering*, *125*. https://doi.org/10.1016/j.datak.2019.101763

Liu, F., Flanigan, J., Thomson, S., Sadeh, N., & Smith, N. A. (2015). Toward abstractive summarization using semantic representations. *NAACL HLT 2015 - 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference*, 1077–1086. https://doi.org/10.3115/v1/n15-1114

Liu, S., Zhou, M. X., Pan, S., Song, Y., Qian, W., Cai, W., & Lian, X. (2012). TIARA: Interactive, topic-based visual text summarization and analysis. *ACM Transactions on Intelligent Systems and Technology*, *3*(2). https://doi.org/10.1145/2089094.2089101