International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 09 Issue 12 ǁ 2021 ǁ PP. 71-79

Summarization of Text Based on Deep Neural Network

Saiela Bilal[#1], Pravneet Kaur[#2]
[#1]*Research scholar, M tech CSE, Department of CSE, RIMT, Punjab, India*
[#2]*Assistant Professor, Department of CSE, RIMT University, Punjab, India*

Abstract:
To automatically summarise a piece of content, the length of the original text must be reduced while keeping the crucial informative characteristics and meaning of the content. As a result, the automation of manual text summarization, which is a time-consuming and labor-intensive technique, is gaining popularity and is therefore a great stimulant for academic research. In today's world of information overload, abstracting and summarising large texts is crucial. Various techniques to text summarization have emerged throughout time. Traditional ways create a summary directly as a result of the document summary connection's duplication and omission. Deep learning systems have been shown to be effective in producing summaries. We focus on deep learning-based text summarization algorithms that have been developed throughout time.
***Keywords*:** *Summarize text, Deep Learning Techniques, Effective, Automatization*.

Date of Submission: 20-11-2021 Date of acceptance: 05-12-2021

I. INTRODUCTION

To automatically summarise a piece of content, the original text must be condensed while keeping the essential informative characteristics and meaning of the content. As a result, automation of manual text summarization, which is a time-consuming and labor-intensive operation, is gaining popularity and is therefore a significant stimulus for academic study. The volume of text data from many sources has increased in the big data era. Text books like these contain a wealth of material that must be carefully summarised in order to be useful. Humans frequently read a book in its entirety to gain a thorough comprehension before writing a summary that highlights its most important characteristics. Text summarization is challenging for computers because they lack human comprehension and linguistic competence.

Natural language processing techniques, such as page rank algorithms, can be used to summarise a piece of text. These algorithms are great at creating text summaries, but they can't generate new phrases that aren't in the content. There is also the possibility of making grammatical mistakes. As a result, we may be able to rely on Deep Learning, a text summarization model that takes new phrases into account. As a consequence, deep learning algorithms are used to create grammatically and logically correct summaries.

1.1. Deep Learning

Many nonlinear processing units are used in a cascade to conduct transformations and feature extractions, with the result of one layer being provided as an input to the next. By utilising a number of feature layers, deep learning algorithms may learn from inputs in both an unsupervised and supervised manner. People do not create and build feature layers as a result of generalised learning; rather, generalised learning teaches them to them. Text summarising techniques pull words directly from the textual material in order to generate the summary. The lemmatization procedure includes both the elimination of stop words and the discovery of noun groupings. However, because there is no record of the keywords previously selected, it's likely that certain terms will appear in the summary as they did in the main text if usual methods are followed. Furthermore, utilising established approaches, the produced summary and the document have a low connection. As a result, buyers will find it more difficult to read the material due to the condensed content. In order to overcome problems, deep learning algorithms are used for summarization.

1.2. Need for text summarization

It is critical to use terms straight from a text while summarising it in order to keep the text's meaning. During lemmatization, stop words are deleted from noun groups. Standardised procedures might have limitations, such as a lack of creativity. Because no record of the previously chosen keywords exists, it's possible that certain terms will appear in both the summary and the main text. Furthermore, there is a weak relationship between the summary and the document created using traditional procedures. As a result, customers will find it more difficult to interpret summarised material. As a result, text summarization is carried out automatically.

1.3. Approaches used for automatic text summarization:

Text can be summarized in two ways:

- Extractive Summarization.
- Abstractive Summarization.

1.3.1. Extractive Text Summarization:

Extractive text summarising is the process of obtaining relevant phrases from a source material and incorporating them into a summary (ETS). The texts are not altered in any way throughout the extraction process. This process is shown in Figure 1.

1.3.2. Abstractive Text Summarization:

As seen in Figure 2, sections of the original information are paraphrased and condensed as part of the abstraction process. Text summarization using abstraction helps overcome the grammatical faults of the extractive approach when used in deep learning. They, like people, invent new words and phrases to convey the most valuable information from the source material. As a result, abstraction performs better than extraction and is usually used for large texts.

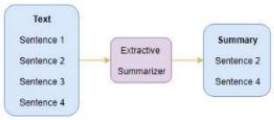

Figure 1: Extractive Summarization

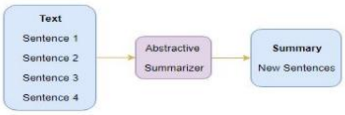

Figure 2: Abstractive Summarization

II. LITERATURE REVIEW

The automatic summarising of legal information is difficult due to the variety of writing styles and the depth of legal issues represented in the text. The authors of Legal Text Summarization examine the methodologies utilised in depth (Kanapala et al., 2019). Kim et al. (2012) summarised legal texts using an asymmetric weighted graph, in which statements are represented as nodes in a graph. For inclusion in the summary table, only sentences with high node values are chosen. A document is represented as a collection of connected graphs as a collection of sentences that are linked together. This technique promotes diversity and, as a result, ensures a continuous flow of information.

Hierarchical Latent Dirichlet Allocation (hLDA) might be used to cluster court verdicts (Venkatesh, 2013). To run hLDA and discover the summary of each document using the same topics, the similarity measure between topics and documents is employed. According to, the significance score may be calculated by summing the TF-IDF scores for each word in each phrase and then normalising by sentence length (Seth et al., 2016).

According to (Saravanan et al., 2008), the summarising work is divided into two parts: document segmentation using conditional random fields to detect rhetorical roles, and generation of a summary from the segments obtained.

A number of models for text summarization have been proposed, ranging from simple multi-layer networks to complicated neural network designs (Sinha et al., 2018). Deep learning approaches, on the other hand, have only been employed sporadically to construct legal document summaries that we are aware of. According to the authors, they use both keyword / key phrase matching and case-based techniques (Kavila et al., 2013). For multi-document summarization of Arabic text, discriminant analysis is provided in order to capture information variety (Oufaida et al., 2014).

III. OBJECTIVES

The primary purpose of this project is to:

• Use deep learning to extract the most important information from a given text and provide it to the end user.

• Put in place machine learning algorithms that can be configured to read documents and identify crucial information.

IV. METHODOLOGY

4.1. Natural Language Processing

Natural language processing (NLP) is a branch of artificial intelligence (AI) that assists computers in understanding and interpreting natural language. According to experts, natural language processing (NLP) may be used to organise and arrange knowledge in order to fulfil tasks such as translation and summarization.

4.2. Components of NLP:

of Natural Language Five main Component processing are:

- Morphological and Lexical Analysis
- Syntactic Analysis
- Semantic Analysis
- Discourse Integration
- Pragmatic Analysis

Each word is dissected into its constituent parts.

4.2.1. Semantic Analysis:

When a syntactic analyzer assigns meanings to a sentence, it is called Semantic Analysis As the name implies, this component converts a series of What it does is illustrate how the words are related to one another.

4.2.2. Syntax analysis:

Words are typically regarded as the smallest units of syntax. If you're interested in syntax, it's the principles and rules that govern every language's sentence.

4.2.3. Pragmatic Analysis:

Pragmatic Analysis focuses on the analysis of communicative and social content, as well as its effect on interpretation. It refers to the process of deriving or abstracting language's meaningful usage. As a result of this research, the focus is always on what was said and how it was communicated.

For example, "shut the window?" should be viewed as a request rather than a command.

V. SIMULATION METHODOLOGY

5.1. Extractive Approach

To summarise articles, extractive approaches find the most essential words from a set of keywords. Summary sentences are weighted based on the most significant parts of the phrases. To grade sentences based on relevance and similarity, a variety of procedures and methodologies are utilised. Because this method cannot generate text on its own, the result will always include a portion of the original content. There are several methods for extractive summarization. Simply said, we will use unsupervised learning to find and rank related keywords. We won't have to train or develop a model before using it in our project.

5.2. Unsupervised approach

Without supervision, the robots are taught to use data that has not been categorised or labelled. This basically means that no training data may be supplied, and the machine must learn on its own. To categorise the data, the computer should not need any prior knowledge of it. In order for the machine to learn on its own, it must be programmed. Both organised and unstructured data must be understood and analysed by the computer. Steps involved are as follows:

5.2.1. Step 1: Importing required libraries.

There are two NLTK libraries that will be necessary for building an efficient feedback summarizer.

```
from nltk.corpus import stopwords
from nltk import word_tokenize, sent_tokenize
```

Terms used here are:

- Corpus: Corpus is a collection, as the name suggests. Texts, such as poetry by a single poet or an author's body of work, may be utilised as data sets. In this case, a set of pre-determined stop words will be employed.
- Tokenizers: It deconstructs a text into tokens. Word, phrase, and regex tokenizers are examples of tokenizers.

5.2.2. Step 2*:* Removing Stop Words and storing them in a separate array of words.

Stop Word: There are a number of words that are not necessary in a sentence, such as (is, a, an, the, for). Think about the following statement as an example.

Jammu and Kashmir is the northern most state of India and is also called as crown of India.

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

‘Jammu’, ‘and’, ‘Kashmir’, ‘northern’, ‘most’, ’state’, ‘india’,’also’, ‘called’, ’crown ’, ‘india’

.

5.2.3. Step 3: Create a frequency table of words.

When stop words are removed, a python dictionary will keep track of how many times each word appears as shown in Figure 3. In order to determine which sentences contain the most relevant material in the entire text, we may run each sentence through the dictionary.

5.2.4. Step 4: Assign score to each sentence depending on the words it contains and the frequency table

We can use the sent tokenize ( ) method to create the array of sentences as shown in Figure 4 . Secondly, we will need a dictionary to keep the score of each sentence.

Figure 3: Creating a Frequency Table

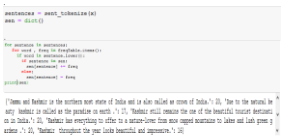

Figure 4: Assign scores to sentences

5.2.5. Step 5: Assign a certain score to compare the sentences within the feedback.

A simple approach to compare our scores would be to find the average score of a sentence as illustrated in Figure 5. The average itself can be a good threshold. Apply the threshold value and store sentences in order into the summary as shown in Figure 6.

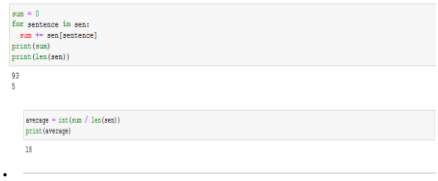

Figure 5: Assign Scores to compare the sentences

Figure 6: Apply threshold and store sentences

VI. ABSTRACTIVE SUMMARIZATION

We create new phrases depending on the original data. Certain sentences may not be present in the original text as compared to the extractive approach, in which we only employed phrases that were a product of the abstractive summarising process.

6.1. Recurrent Neural Network:

A form of neural network is referred to as "recurrent neural networks." RNNs that are particularly good in modelling sequence data, such as time series This design is built on the notion of sequential information. A huge number of the most frequently occurring words are supplied into the RNN network. In order to forecast the next word in a phrase, the computer scans the data for terms that appear frequently. As a result, do you understand how important the RNN is in our everyday lives? In actuality, it has slowed us down. Figures 7 and 8 show the basic structure of RNN as well as a diagram of the RNN's basic equations.

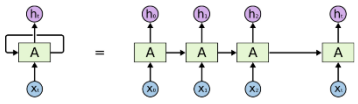

Figure 7: Basic Architecture of RNN

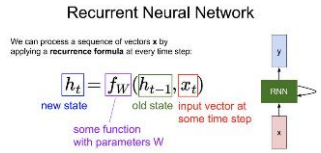

Figure 8: Basic Equations of RNN

6.2. Sequence-to-Sequence (Seq2Seq) Modeling

With the Seq2Seq paradigm, we can analyse and solve any problem with sequential information. Some of the most frequent uses of sequential information are Sentiment categorization, Neural Machine Translation, and Named Entity Recognition.

A text in one language is fed into Neural Machine Translation, which outputs a text in another language.

I love playing sports ⟶ Me encanta hacer deporte

A series of words is sent into Named Entity Recognition, which returns a list of tags for each word in the input sequence.:

Andrew ng founded coursera ⟶ B-PER, I-PER, O, O

Our goal is to create a text summarizer that takes a long series of words (in a text body) as input and outputs a brief summary of the text (which is a sequence as well). It's a Seq2Seq issue with many-to-many components, which we can model. As an example, consider the following Seq2Seq model architecture:
There are two major components of a Seq2Seq model:

- Encoder
- Decoder.

To address the Seq2Seq challenge, encoder-decoder architecture is employed. To further comprehend this, let's look at it from the standpoint of text. As an input, you provide a long string of words, and as an output, you provide a condensed version of input sequence.
We can set up the Encoder-Decoder in 2 phases:

- Training phase.
- Inference phase.
-

6.2.1. Training phase:
After setting up encoder and decoder, we will move on to the training phase. In the next phase, we will train our model to predict the target sequence with a Encoder.

6.2.2. Encoder*:*
One word is supplied into the encoder at each timestep by an Encoder Long Short Term Memory Model (LSTM). Each timestep is processed and contextual information from the input sequence is captured. This can be seen in Figure 9. To initialize the decoder, the hidden state (hi) and cell state (ci) of the last time step are utilized. Keep in mind that this is due to the fact that the encoder and decoder are two separate sets of the LSM architecture.

6.2.3. Decoder:
As with the encoder, the decoder is an LSTM network that reads the whole target sequence word-by-word and predicts the same sequence, but using the preceding word as a cue, the decoder is taught to anticipate the following word. Basic structure of Decoder can be shown in Figure 10.

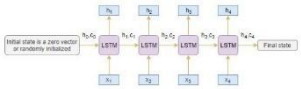

Figure 9: Basic Structure of Encoder

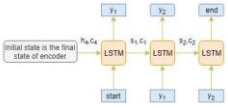

Figure 10: Basic structure of Decoder

In order to decode the target sequence, special tokens called <start> and <end> must be appended to it. While decoding the test sequence, the target sequence is unknown. In order to anticipate the target sequence, the initial word, which is always <start>, is passed to the decoder. It's also worth noting that <end> signifies the end of the sentence.

6.2.4. Inference Phase:
After training, the model is tested on new source sequences for which the target sequence is unknown. So, we need to set up the inference architecture to decode a test sequence.

6.2.4.1. Working of Inference:
Here are the steps to decode the test sequence:

1. To do this, encrypt the whole input stream, then use the encoded data to initialize the decoder.
2. Decoder should be given the <start> token as an input.
3. Then, run the decoder for a single timestep.
4. The likelihood of the following word will be returned. The most likely term will be chosen.
5. Pass the sampled word to the decoder in the next timestep and update the internal states with the current timestep.
6. Tokenize the target sequence by repeating steps 3 through 5 until the <end> token is generated or we reach the target sequence.
7. Encode the test sequence into internal state vectors.
8. Observe how the decoder predicts the target sequence at each timestep.

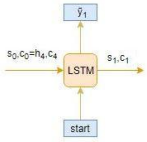

Figure 11: Timestep t=1

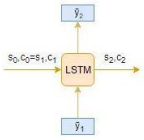

Figure 12: Timestep t=2

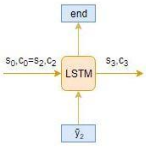

Figure 13: timestep t=3

VII.RESULTS

In order to create the model, following terms are important.

- Return Sequences = True: This is because LSTM creates the hidden and cell states for every timestep when return sequences is set to True.
- Return State = True: When return state = True, as its name suggests, LSTM generates just the hidden state and cell state of last timestop.
- Initial State: Initializes the LSTM's internal states for the first timestep.
- Stacked LSTM: It is composed of several layers of the LSTM that are placed on top of each other. A better representation of the sequence results from this. Stacking many LSTMs on top of each other is an excellent method to learn.

Here, we are building a 3 stacked LSTM for the encoder which is seen in Figure 14 and the output can be seen in Figure 15:

Figure 14: Three stack LSTM for encoder

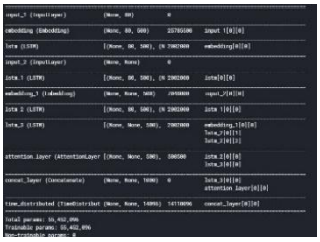

Figure 15: Output for 2 stack LSM for encoder

We are using sparse categorical cross-entropy as the loss function since it converts the integer sequence to a one-hot vector on the fly. This overcomes any memory issues.
i.e.,

model.compile(optimizer='rmsprop', loss='sparse_ categorical_ crossentropy')

Using a user-specified measure, it determines when to cease training the neural network. As we are keeping an eye on the validity loss (Once the validation loss rises, our model will stop training
i.e.,

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1

We'll train the model on a batch size of 512 and validate it on the holdout set (which is 10% of our dataset):
i.e.,

history=model.fit([x_tr,y_tr[:,:-1]], y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)[:,1:] ,epochs=50,callbacks=[es],batch_size=512, validation_data=([x_val,y_val[:,:-1]], y_val.reshape(y_val.shape[0],y_val.shape[1], 1)[:,1:]))

7.1. Understanding the Diagnostic plot

Now, we will plot a few diagnostic plots to understand the behavior of the model over time:
i.e.,

```
from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend() pyplot.show()
```

7.2. Output:

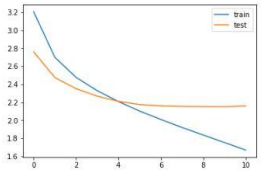

Figure 16: Output of Cross Entropy

After epoch 10, we may deduce that the validation loss has increased slightly. Because of this, we will no longer train the model after this epoch.
Next, let's build the dictionary to convert the index to word for target and source vocabulary:

i.e.,

```
reverse_target_word_index=y_tokenizer.index_word
reverse_source_word_index=x_tokenizer.index_word
target_word_index=y_tokenizer.word_index
```

7.3. Inference

The various steps involved are

Step 1*:* Set up the inference for the encoder and decoder as shown in Figure 17.

```
# encoder inference
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h, state_c])

# decoder inference
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_len_text,latent_dim))

# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)

# To predict the next word in the sequence, set the initial states to the states from the previous time
step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2,
initial_state=[decoder_state_input_h, decoder_state_input_c])

#attention inference
attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input, decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2, attn_out_inf])

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_inf_concat)

# Final decoder model
decoder_model = Model(
[decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_state_input_c],
[decoder_outputs2] + [state_h2, state_c2])
```

Figure 17: Inference setup for Encoder and Decoder

Step 2: This function implements the inference procedure can be illustrated from Figure 18

```
def decode_sequence(input_seq):
  # Encode the input as state vectors.
  e_out, e_h, e_c =encoder_model.predict(input_seq)

  # Generate empty target sequence of length 1.
  target_seq =np.zeros((1,1))

  # Chose the 'start' word as the first word of the target sequence
  target_seq[0, 0] = target_word_index['start']

  stop_condition = False
  decoded_sentence = ''
  while not stop_condition:
    output_tokens, h, c =decoder_model.predict([target_seq] + [e_out, e_h, e_c])

    #Sample a token
    sampled_token_index = np.argmax(output_tokens[0, -1, :])
    sampled_token =reverse_target_word_index[sampled_token_index]

    if(sampled_token!='end'):
      decoded_sentence = decoded_sentence + ' '+sampled_token

      #Exit condition: either hit max length or find stop word.
      if (sampled_token == 'end' or len(decoded_sentence.split()) >=(max_len_summary-1)):
        stop_condition =True

    #Update the target sequence (of length 1).
    target_seq = np.zeros((1,1))
    target_seq[0, 0] = sampled_token_index

    #Update internal states
    e_h, e_c =h, c

  return decoded_sentence
```

Figure 18: The interference procedure implementation

Step 3: We now, define the functions to convert an integer sequence to a word sequence for summary as well as the reviews as shown in Figure 19.

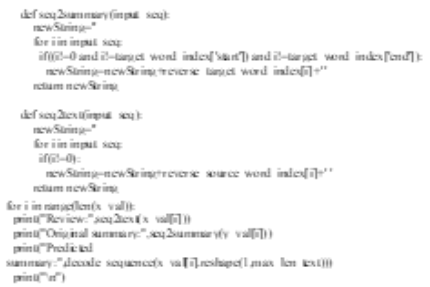

Figure 19: Defining functions and converting into a word sequence

Step 4*:* Here, in Figure 20 are a few summaries generated by the model:

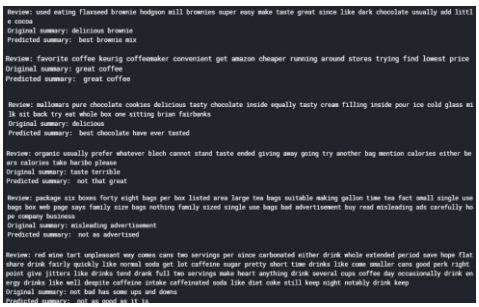

Figure 20: Examples of the Output Summaries

VIII. CONCLUSION

While our model's output summary and the actual one do not have the same word count, but the message conveyed is the same. Using context from the text, our model is able to create a readable summary of the content. Our technique was able to give 90% accuracy. We used the dataset from Amazon that contained reviews from customers related to products.

REFERENCES

[1]. Amjad Abu-Jbara and Dragomir Radev. 2019. Coherent citation-based summarization of scientific papers. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1. Association for Computational Linguistics, 500–509.

[2]. Rasim M Alguliev, Ramiz M Aliguliyev, Makrufa S Hajirahimova, and Chingiz A Mehdiyev. 2011. MCMR: Maximum coverage and minimum redundant text summarization model. Expert Systems with Applications 38, 12 (2011), 14514–14522.

[3]. Rasim M Alguliev, Ramiz M Aliguliyev, and Nijat R Isazade. 2013. Multiple documents summarization based on evolutionary optimization algorithm. Expert Systems with Applications 40, 5 (2019), 1675–1689.

[4]. Mehdi Allahyari and Krys Kochut. 2020. Automatic topic labeling using ontology-based topic models. In Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on. IEEE, 259–264.

[5]. Mehdi Allahyari and Krys Kochut. 2016. Discovering Coherent Topics with Entity Topic Models. In Web Intelligence (WI), 2016 IEEE/WIC/ACM International Conference on. IEEE, 26–33.

[6]. Mehdi Allahyari and Krys Kochut. 2019. Semantic Context-Aware Recommendation via Topic Models Leveraging Linked Open Data. In International Conference on Web Information Systems Engineering. Springer, 263–277.

[7]. Mehdi Allahyari and Krys Kochut. 2020. Semantic Tagging Using Topic Models Exploiting Wikipedia Category Network. In Semantic Computing (ICSC), 2016 IEEE Tenth International Conference on. IEEE, 63–70.

[8]. M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut. 2017. A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques. ArXiv e-prints (2017). arXiv:1707.02919

[9]. Einat Amitay and Cécile Paris. 2000. Automatically summarizing web sites: is there a way around it?. In Proceedings of the ninth international conference on Information and knowledge management. ACM, 173–179.

[10]. Elena Baralis, Luca Cagliero, Saima Jabeen, Alessandro Fiori, and Sajid Shah. 2013. Multi-document summarization based on the Yago ontology. Expert Systems with Applications 40, 17 (2013), 6976–6984.

[11]. Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. 2011. Jointly learning to extract and compress. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1. Association for Computational Linguistics, 481–490.

[12]. David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. the Journal of machine Learning research 3 (2003), 993–1022.

[13]. Asli Celikyilmaz and Dilek Hakkani-Tur. 2019. A hybrid hierarchical model for multi-document summarization. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 815–824.

[14]. Yllias Chali and Shafiq R Joty. 2018. Improving the performance of the random walk model for answering complex questions. In Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers. Association for Computational Linguistics, 9–12.

[15]. Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, and others. 2006. Semisupervised learning. Vol. 2. MIT press Cambridge.
[16]. Ping Chen and Rakesh Verma. 2018. A query-based medical information summarization system using ontology knowledge. In Computer-Based Medical Systems, 2006. CBMS 2019. 19th IEEE International Symposium on. IEEE, 37–42.
[17]. Freddy Chong Tat Chua and Sitaram Asur. 2019. Automatic Summarization of Events from Social Media.. In ICWSM.
[18]. John M Conroy and Dianne P O'leary. 2019. Text summarization via hidden markov models. In Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 406–407.
[19]. Hal Daumé III and Daniel Marcu. 2016. Bayesian query-focused summarization. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 305–312.
[20]. Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. 1990. Indexing by latent semantic analysis. JASIS 41, 6 (1990), 391–407.