IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

A Hybrid Approach For Text Summarization

[1]Chandra Sekhar Vinukonda, [2] A. Dinesh Kumar, [3]Prasun Kumar Mandal, [4]Suja Roy, [5]M. P. Samyuktha

[1]Assistant Professor, [2]Under Graduate, [3]Under Graduate, [4]Under Graduate, [5]Under Graduate

[1]Department of Computer Science & Engineering,

[1]Dr. B. R. Ambedkar Institute of Technology, Port Blair, India

Abstract: In this new era, where tremendous information is available on the internet, it is most important to provide the improved mechanism to extract the information quickly and most efficiently . It is very difficult for human beings to manually extract the summary of a large documents of text. So there is a problem of searching for relevant documents from the number of documents available, and absorbing relevant information from it. In order to solve the above two problems, the automatic text summarization is very much necessary. Text summarization is the process of identifying the most important meaningful information in a document or set of related documents and compressing them into a shorter version preserving its overall meanings. In this paper, we present a new graph-based text summarizer method. The proposed method is fast and can be implement for text summarization.

Index Terms - Automatic Text Summarization, Extractive Summarization, Abstractive Summarization, Pegasus

I. INTRODUCTION

In the era of big data, text resources are becoming more and more accessible. Natural language processing (NLP) technology is advancing rapidly. Text summarization is a popular navigation tool in NLP. It provides summaries to help you understand the entire article at once in a minimal number of words. Generalization of text is to create a compressed version of the original document by reducing the size of the document while maintaining the main characteristics of the document. However, abstracting artificial text requires a lot of background knowledge, often requires long processing times, and the quality of the abstract is not always sufficient. For this reason, people are starting to focus on automatic text summarization. Auto text summarization was first studied by Lun almost 60 years ago and has received a lot of attention in recent years. This method is faster than the artificial method and works better than average. With the rapid growth of text resources on the Internet, various areas of text summarization are being applied. For example, question-and-answer (QA) systems generate question-based summaries to provide information. Another example is snippets of web search results that can help users to explore more. A short summary of a news article can help readers get useful information about an event or topic. In this paper we proposed a graph based hybrid text summarization technique –which mainly is based on 'extractive summarization' and 'abstractive summarization'. This proposed approach combines the concept of extraction that finds the most informative sentences within a large of text which are used to form a summary and the abstraction algorithmically generate concise phases that are semantically consistent with the large body of text.

II. RELATED WORKS

Automatic text summarization has been extensively investigated, due to its important applications in large-scale text analytics.

One more work presented a two-phase approach towards long text summarization, namely, EA-LTS [1]. In the extraction phase, it conceives a hybrid sentence similarity measure by combining sentence vector and Levenshtein distance, and integrates it into graph model to extract key sentences. In the abstraction phase, it constructs a recurrent neural network based encoder-decoder, and devises pointer and attention mechanisms to generate summaries. They tested their model on a real-life long text corpora, collected from sina.com; experimental results verify the accuracy and validity of the proposed method.

Another work presented a new multigraph-based text summarizer method [2]. This method is unique in that it produces a multi-edge-irregular-graph that represents words occurrence in the sentences of the target text. This graph is then converted into a symmetric matrix from which ranking of sentences can be produced and hence obtain the summarized text using a threshold.

A recent work [3] used a model that first trains an extractor to extract salient sentences from the original text. Next, these salient sentences are put together to get a condensed version of the original text. Then the abstractive model is used to rewrite the extracted sentences to get the final summary. In order to avoid the exposure bias, reinforcement training is used to optimize the proposed model.

The demand for query focused text summarization is growing rapidly due to the presence of irrelevant and repetitive information. A recent work [5] presents a novel hybrid methodology to extract various sentences that are relevant to a user-given question or query. The proposed model uses unsupervised algorithms to first detect the topics in the data by generating an intuitive semantic structure. It then employs a conditional semantic similarity measure to retrieve data containing keywords. An enhanced graph-based approach has been used to include additional query-relevant data as well as discard the redundant information to generate an efficient output summary using a weighted graph. The results showcase an improved performance compared to other traditional methods and encourage future work in this field.

Recent work [6] pre-training Transformers with self-supervised objectives on large text corpora has shown great success when fine-tuned on downstream NLP tasks including text summarization. In this work, they proposed pre-training large Transformer-based encoder-decoder models on massive text corpora with a new self-supervised objective. In PEGASUS, important sentences are removed/masked from an input document and are generated together as one output sequence from the remaining sentences, similar to an extractive summary. They evaluated their best PEGASUS model on 12 downstream summarization. Experiments demonstrate it achieves state-of-the-art performance on all 12 downstream datasets measured by ROUGE scores. Our model also shows surprising performance on low-resource summarization, surpassing previous state-of-the-art results on 6 datasets with only 1000 examples.

III. PROPOSED METHODLOGY

In ATS (Automatic Text Summarization) system, there are three main steps involved.

1. Pre-Processing

2. Processing

3. Post-Processing

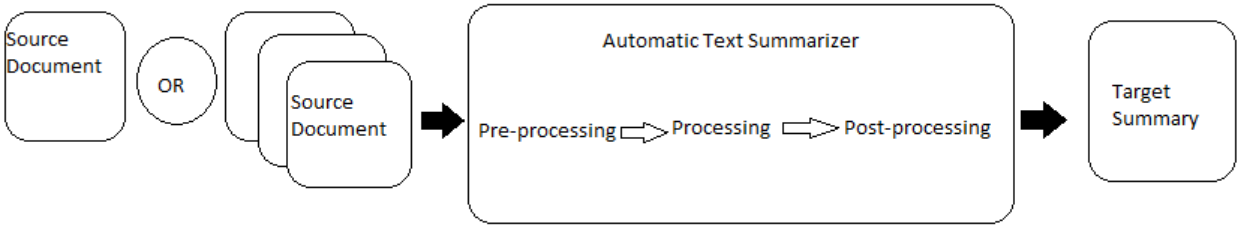

Fig 3.1 - General Architecture of ATS

1. Pre-Processing: Producing a structured representation of the original text using many linguistic techniques like sentences segmentation, word tokenization, removal of stop-words, part-of-speech tagging, stemming, etc.

2. Processing: Using one of the text summarization approaches by applying a technique to convert the input document(s) to the summary.

3. Post-Processing: solving some problems in the generated summary sentences like anaphora resolution and reordering the selected sentences before generating the final summary.

Automatic text summarization has two main approaches i.e., extractive and abstractive. We have proposed a hybrid approach for automatic text summarization, in which a new graph-based approach is introduced to produce an extractive summary, which will be fed to the abstractive summarization model that involves transformer, to produce the final summary.

The proposed approach involves three stages –

- Data pre-processing
- Extractive Text Summarization
- Abstractive Text Summarization

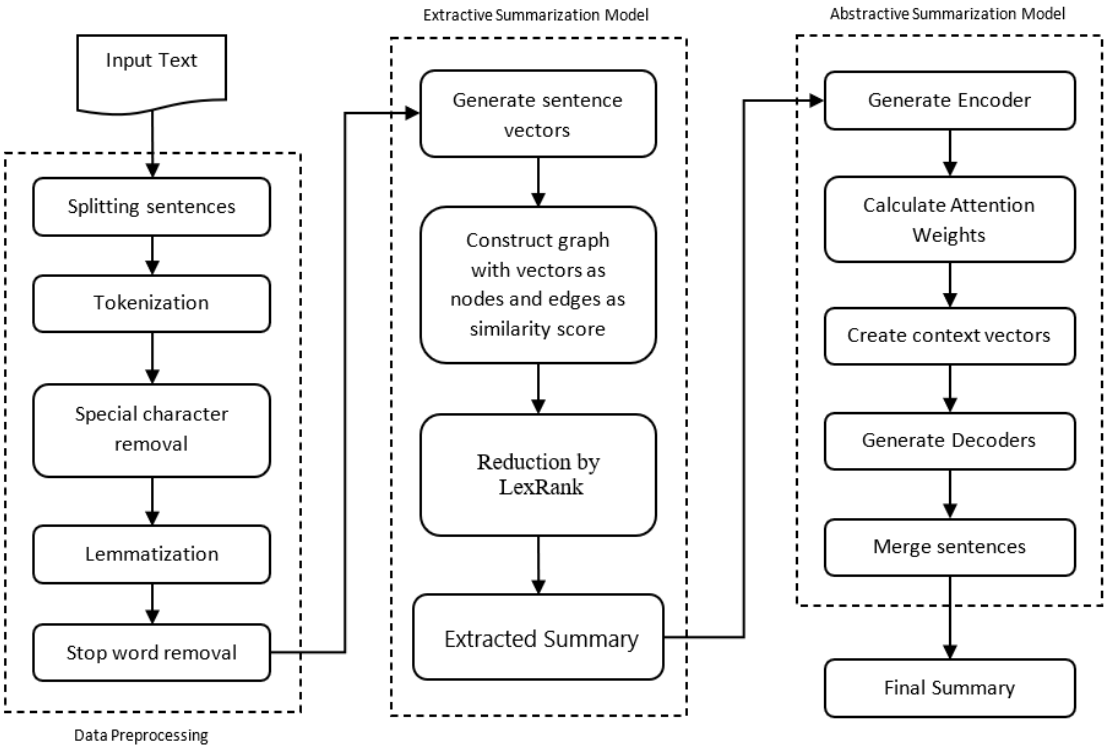

Fig. 3.2 - Architecture of proposed system

1. Data Pre-processing

Data pre-processing is a data mining technique which is used to transform the raw data in a useful and efficient format. We have done following data pre-processing steps to obtain a pre-processed text –

i. Splitting sentences

First of all, we split up the lines present in the input text, and store them in an arrayed of size n, where n is the number of lines present in it.

ii. Tokenization

Tokenization is generally a process of splitting a sentence into words. Each word present in the sentence will be referred as tokens.

iii. Special character removal

Special character involves numbers, commas, exclamation marks, punctuation marks, etc. These characters are not important for generating sentence vectors, so we need to remove them from the input text.

iv. Lemmatization

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is similar to stemming but it brings context to the words. So, it links words with similar meaning to one word.

v. Stop word removal

A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that are not very important for processing the data. These words are generally removed, as they are not important in generating sentence vectors, and doing so can reduce processing time.

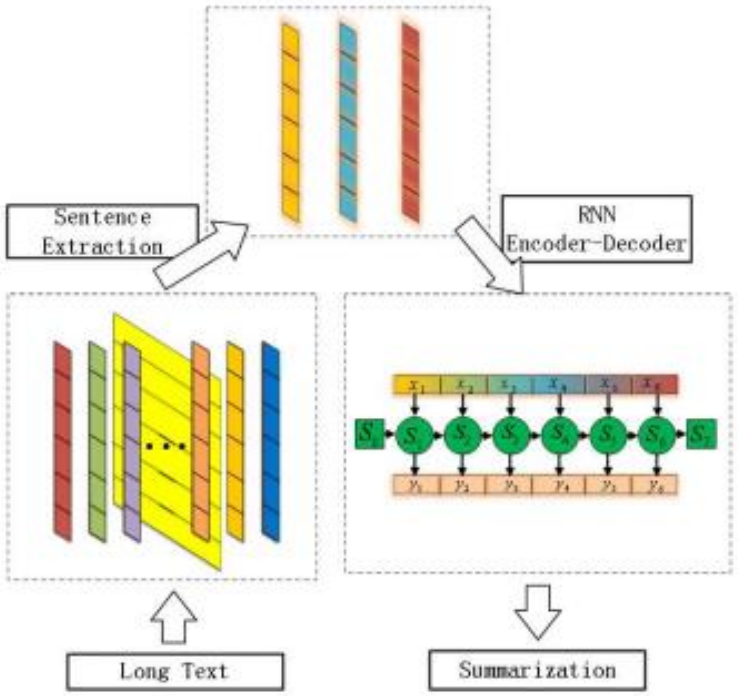

Fig 3.3 - Overall Framework of Proposed Method

2. Extractive Text Summarization

Extractive text summarization is the process of extracting most important sentences from the input text, and combine them to form the summary.

In our approach, we have proposed a graph-based extractive text summarization algorithm, which generates a graphical representation of the input text, with sentence vectors present in nodes and the mean similarity of two sentences as their edges. Then we remove some edges based on average similarity score of all sentence pairs, and select the sentences as a path with maximum profit.

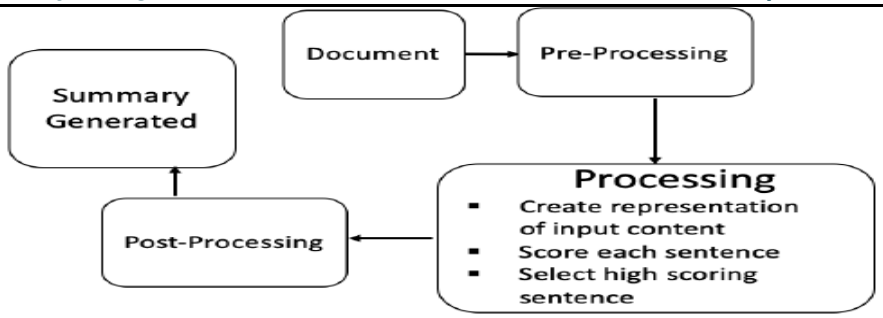

Fig 3.4 - Extractive Text Summarization Architecture

Various steps involved in extractive text summarization is explained below –

i. Sentence Embedding

Sentence embedding techniques represent entire sentences and their semantic information as vectors. This helps the machine in understanding the context, intention, and other nuances in the entire text. It allows us to understand the intention of the sentence without calculating individually the embedding of the words. It also enables the comparison of sentences to cluster them by similarity or to predict values for the sentences, such as sentiment.

We incorporate a graph model to handle the task of key sentence extraction, where a hybrid sentence similarity measure is conceived by combining sentence vector similarity and Levenshtein distance.

ii. Calculating similarity matrix

Similarity Calculation: Classic TF-IDF-based method is used. The term tf–idf stands for term frequency–inverse document frequency. It is a mathematical statistic that is planned to reflect how significant a word is to a record in a collection or corpus. The tf–idf esteem builds proportionally to the number of times a word shows up in the document. It is offset by the quantity of documents in the corpus that contain the word, which helps to adjust for the fact that a few words show up more often when all is said and done. TF–IDF is one of the most well-known term-weighting plans.

- TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization: TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

- IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing. IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

IDF can be calculated as follow:

$$idf_i = \log(\frac{n}{df_i}) \quad (1)$$

Where idf_i is the IDF score for term i, df_i is the number of documents containing term i, and n is the total number of documents.
The TF-IDF score as the name suggests is just a multiplication of the term frequency matrix with its IDF, it can be calculated as follow:

$$w_{i,j} = tf_{i,j} \times idf_i \quad (2)$$

Where w_{ij} is TF-IDF score for term i in document j, tf_{ij} is term frequency for term i in document j, and idf_i is IDF score for term i.

We consider a new approach, LexRank, for computing sentence importance based on the concept of eigenvector centrality in a graph representation of sentences. LexRank is an unsupervised approach to text summarization based on graph-based centrality scoring of sentences. The main idea is that sentences "recommend" other similar sentences to the reader. Thus, if one sentence is very similar to many others, it will likely be a sentence of great importance.

According to LexRank algorithm, there is a graph G that has nodes standing for sentences in X and edges placed between two sentences that are similar to each other. The weight of edges of graph G is the similar value of two sentences. After G generated, this algorithm computes the score of sentence centrality and generates the centroid-based summarization.

In LexRank, the function of sentence centrality is given by Equation

$$p(u) = \frac{d}{N} + (1-d) \sum_{v \in adj[u]} \frac{w(v,u)}{\Sigma_{z \in adj[v]} w(v,z)} p(v) \qquad (3)$$

Where N is the total number of nodes in graph G,d is a damping factor, adj[u] is the set of the nodes that are neighbors of node u in the graph G,w(v, u) is the weight between node v and node u,w(v, u) is the similarity between two sentences which the node v and u stand for.

After generating sentence vectors, we need to construct the similarity matrix, by calculating the similarity score between each sentence present in the input text. Suppose, we have two sentences Si and Sj in vector form, and we need to calculate the similarity score between them. First, we will plot those sentence vectors, and then we will calculate the cosine value of the angle they make with the x-axis.

Similarity score of Si and Sj will be calculated by –

Similarity (Si, Sj) = cos A – cos B

where, A and B are the angles made by sentence vectors Si and Sj respectively.

Similarity, we will calculate the similarity score of each sentence pairs and construct a similarity matrix.

Now, suppose we have the similarity score of Si and Sj as x and similarity score of Sj and Si as y, then we will calculate the mean value of both the scores and consider it as actual similarity score of sentence Si an Sj, which will be calculated as –

Actual Similarity (Si, Sj) = (x+y)/2

The actual similarity score will be replaced in both the places, i.e., the similarity score of Si and Sj, as well as the similarity score of Sj and Si.

The resultant matrix will be the final similarity matrix, and it will be used to construct the graph.

iii. Graph construction

Next, we will represent the entire input text as a graph, in which the sentences present will be considered as nodes and the similarity score of two sentences will be the weighted edge between the nodes of those sentences. The first sentence S0 and the last sentence Sn will be marked as start and end nodes.

iv. Graph reduction

To reduce the sentence dependencies in the graph, we will calculate the average similarity score from the similarity matrix. Then, we will remove those edges from the graph, whose weight is less than the calculated average similarity score.

The cost of an edge from the node representing sentence number i in the text, Si , to the node for Sj is calculated as:

$$\text{cost } i, j = \frac{(i-j)^2}{\text{overlap } i,j \cdot \text{weight } j} \quad (4)$$

and the weight of a sentence is calculated as:

$$\text{weight } j = (1 + \text{overlaptitle}, j) \cdot \frac{(1 + \sum_{w \in Sj} tf(w))}{(\sum_{w \in text} tf(w))} \cdot \text{early } (j). \sqrt{(1 + |\text{edgej}|)} \quad (5)$$

After that, we will obtain the reduced graph representation of input text. Since similarity is based on the number of words in common between two sentences, long sentences have a greater chance of being similar to other sentences.

v. Sentence selection

For selecting the most important sentences from the graph, we will begin from the start node and add that sentence in the resultant summary. After that, we will choose the next node, which has maximum weight and add that sentence also in the resultant summary. Similarly, we will choose next nodes also with the maximum weight, until we reach to the end node. If we are unable to reach the end node, then we will backtrack and choose any other node, which can reach to the end node. If any node is visited once, then it can't be visited again. After traversing from start to end node, we will get the initial extractive summary, which will be passed to the next stage.

3. Abstractive Text Summarization

In this type of summarization, new sentences are formed which is generally not present in the original text. We used the transformer architecture to produce the abstractive summary. It is an architecture for transforming one sequence into another. Transformers try to solve the problem by using Convolutional Neural Networks together with attention models. Attention boosts the speed of how fast the model can translate from one sequence to another.

Transformer is also an encoder-decoder mechanism. The Encoder block has 1 layer of a Multi-Head Attention followed by another layer of Feed Forward Neural Network. The decoder, on the other hand, has an extra Masked Multi-Head Attention. The encoder and decoder blocks are actually multiple identical encoders and decoders stacked on top of each other. Both the encoder stack and the decoder stack have the same number of units.

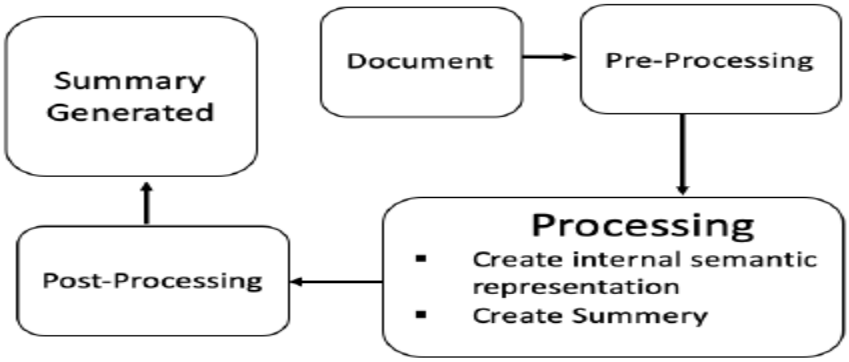

Fig 3.5 - Abstractive Text Summarization Architecture

A. Encoder-Decoder Architecture

Encoder and Decoders are the two components of a sequence-2-sequence rnn architecture. These are majorly used when inputs and outputs are of varying lengths. In this architecture, various LSTMs are placed sequentially in order to encode and then decode the input.

Encoder - It encodes the entire input into a form suitable for processing.

Decoder - It decodes the processed input into an understandable and required output.

One word at a time is provided as input to the encoder with its time stamp. This word is then processed in the LSTM by retrieving the information present in the sequenced input. It is followed by a decoder that decodes the input sequence into a more readable and desirable output format. It is also dependent upon the time stamp. Finally, we produce a summary using the model which predicts the next relevant word by considering the previous sequence of words.

- Long Short Term Memory Network
- LSTM is a special kind of recurrent neural network capable of handling long-term dependencies.
- Understand the architecture and working of an LSTM network

This model is an extension to feedforward network with at least one feedback connection. In standard LSTM sequence of fixed length is passed as an input to be encoded into a fixed dimension vector (v), which is then decoded into the output sequence of words.

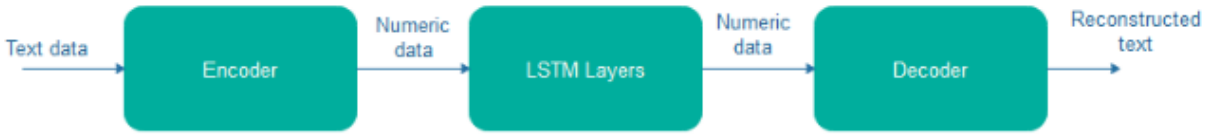

Fig 3.6 - Encoder-Decoder Architecture

The Recurrent Neural Network(RNN) is a natural generalization of feed forward neural networks to sequences. Given a sequence of inputs(x1,..., xT), a standard RNN computes a sequence of outputs (y1,..., yT) by iterating the equation 6 and 7:

$$h_t = sigmoid(W^{hx}x_t + W^{hh}h_{t-1}) \quad (6)$$

$$y_t = W^{yh}h_t \quad (7)$$

The RNN can easily map sequences to sequences whenever the alignment between the inputs and the outputs is known ahead of time. However, it is not clear how to apply an RNN to problems whose input and the output sequences have different lengths with complicated and non-monotonic relationships.

Sequence learning consists of mapping the input sequence with one RNN to a vector of fixed size and then mapping the vector with another RNN to the target sequence. Although it could work in principle, since the RNN is supplied with all relevant information, it would be difficult to train the RNNs due to the resulting long-term dependencies. However, the Long Short-Term Memory (LSTM) is known to learn problems with long-range time dependencies, so an LSTM can be successful in this setting.

The objective of the LSTM is to estimate the conditional probability p(y1,..., yM0|x1,..., xM) where (x1,..., xM) is an input sequence and (y1,..., yM0) is its corresponding output sequence whose length M0 may differ from M. The LSTM computes the conditional probability by first obtaining the fixed-dimensional representation v of the input sequence (x1,..., xM) given by the last hidden state of the LSTM, and then computing the probability of (y1,..., yM0) with a standard LSTM language model formulation whose initial hidden state is set to the representation v of (x1,..., xT):

$$p(y_1, \ldots, y_{M'} | x_1, \ldots, x_M) = \prod_{m=1}^{M'} p(y_m | v, y_1, \ldots, y_{m-1}) \quad (8)$$

In abstractive summarization, sequence-to-sequence has become a dominant framework using encoder-decoder architectures based on RNNs and more recently Transformers. Most prior work on neural abstractive summarization relied on large-scale, high-quality datasets of supervised document-summary pairs and achieved promising results. We find that masking whole sentences from a document and generating these gap-sentences from the rest of the document works well as a pre-training objective for downstream summarization tasks. In particular, choosing putatively important sentences outperforms lead or randomly selected ones. We hypothesize this objective is suitable for abstractive summarization as it closely resembles the downstream task, encouraging whole-document understanding and summary-like generation. We call this self-supervised objective Gap Sentences Generation (GSG).

Using GSG to pre-train a Transformer encoder-decoder on large corpora of documents (Web and news articles) results in our method, Pre-training with Extracted Gap-sentences for Abstractive summarization sequence-to-sequence models, or PEGASUS.

PEGASUS proposes a new self-supervised pre-training objective for abstractive summarization, by reconstructing the target sentence with the remaining sentences in the document, it also shows strong results in low-resource settings.

In Pegasus, important sentences are removed/masked from an input document and are generated together as one output sequence from the remaining sentences, similar to an extractive summary.

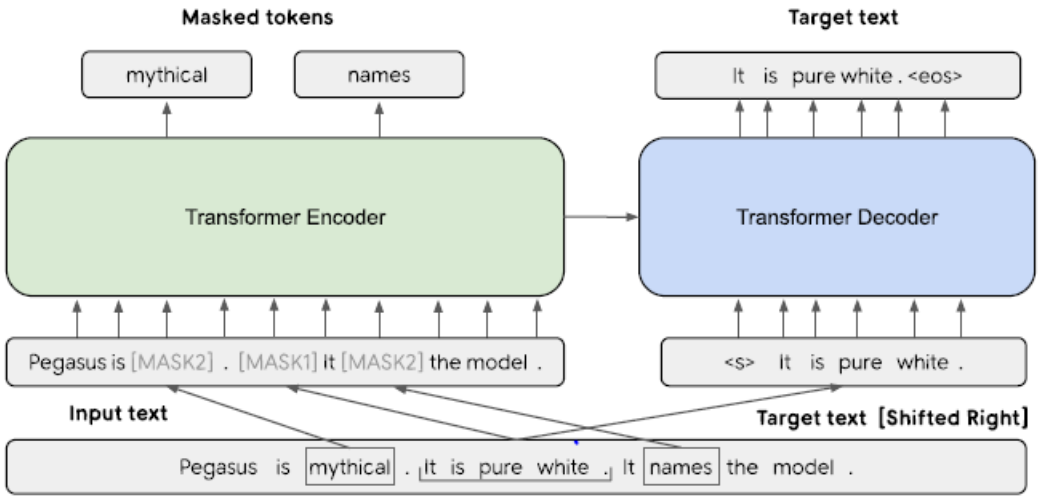

Fig 3.7 - The base architecture of PEGASUS

The base architecture of PEGASUS is a standard Transformer encoder-decoder. Both GSG and MLM are applied simultaneously to this example as pre-training objectives. Originally there are three sentences. One sentence is masked with [MASK1] and used as target generation text (GSG). The other two sentences remain in the input, but some tokens are randomly masked by [MASK2] (MLM).

B. Pre-training Objectives

1. Gap Sentences Generation (GSG)

The proposed pre-training objective involves generating summary-like text from an input document. In order to leverage massive text corpora for pre-training, we design a sequence-to-sequence self-supervised objective in the absence of abstractive summaries. A naive option would be to pre-train as an extractive summarizer; however, such a procedure would only train a model to copy sentences, thus not suitable for abstractive summarization.

Inspired by recent success in masking words and contiguous spans (Joshi et al., 2019; Raffel et al., 2019), we select and mask whole sentences from documents, and concatenate the gap-sentences into a pseudo-summary. The corresponding position of each selected gap sentence is replaced by a mask token [MASK1] to inform the model. Gap sentences ratio, or GSR, refers to the number of selected gap sentences to the total number of sentences in the document, which is similar to mask rate in other works.

To even more closely approximate a summary, we select sentences that appear to be important/principal to the document. The resulting objective has both the empirically demonstrated benefits of masking, and anticipates the form of the downstream task.

2. Masked Language Model (MLM)

Following BERT, the model selects 15% tokens in the input text, and the selected tokens are (1) 80% of time replaced by a mask token [MASK2], or (2) 10% of time replaced by a random token, or (3) 10% of time unchanged. We apply MLM to train the Transformer encoder as the sole pre-training objective or along with GSG. When MLM is the sole pre-training objective, the Transformer decoder shares all parameters with encoder when fine-tuning on downstream tasks following Rothe et al. (2019).

C. Datasets

For downstream summarization, we only used public abstractive summarization datasets, and access them through Tensor Flow Summarization Datasets 1, which provides publicly reproducible code for dataset processing and train/validation/test splits. We used train/validation/test ratio of 80/10/10 if no split was provided, and 10% train split as validation if there was no validation split.

Here we are using Google Pegasus XSum model. The model google Pegasus XSum is a Natural Language Processing (NLP) Model implemented in Transformer library, generally using the Python programming language.

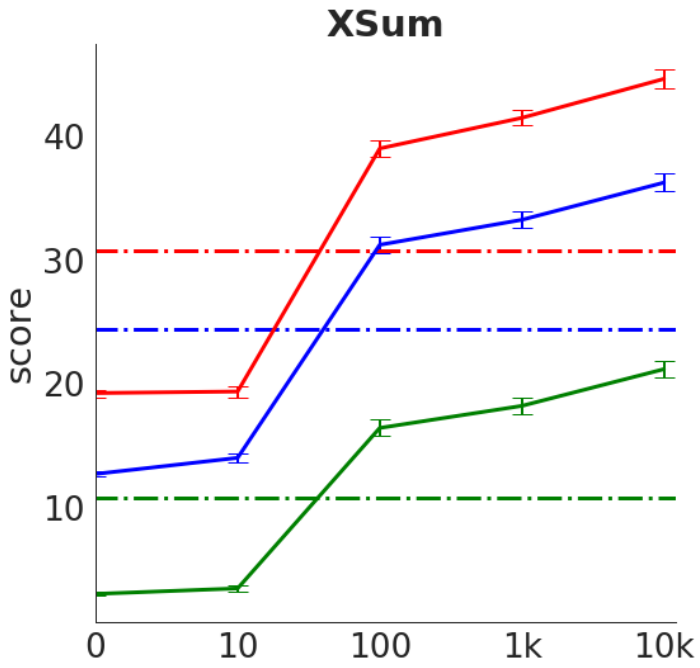

Fig 3.8 - ROUGE scores (three variants, higher is better) vs. the number of supervised examples across four selected summarization datasets. The dotted-line shows the Transformer encoder-decoder performance with full-supervision, but without pre-training.

The Extreme Summarization (**XSum**) dataset is a dataset for evaluation of abstractive single-document summarization systems. The dataset consists of 226,711 news articles accompanied with a one-sentence summary. The articles are collected from BBC articles (2010 to 2017) and cover a wide variety of domains (e.g., News, Politics, Sports, Weather, Business, Technology, Science, Health, Family, Education, Entertainment and Arts). The official random split contains 204,045 (90%), 11,332 (5%) and 11,334 (5) documents in training, validation and test sets, respectively.

V. RESULTS

In our work, we have reviewed various studies on research papers based on text summarization from 2015 to 2020. published in conference proceedings and journals of high repute nationally and internationally. In the proposed model, we have used both extractive and abstractive method, so that we can get a more precise summary in abstractive form. We have observed that our method is extracting important sentences from the original text, and producing a topic relevant summary as output.The model that is proposed in this paper are applied on random text which we get from internet, like we use news article, scientific article or paper, research paper and short story. Some of the screens shown here, on which the method is applied. The corresponding outputs are also shown below.

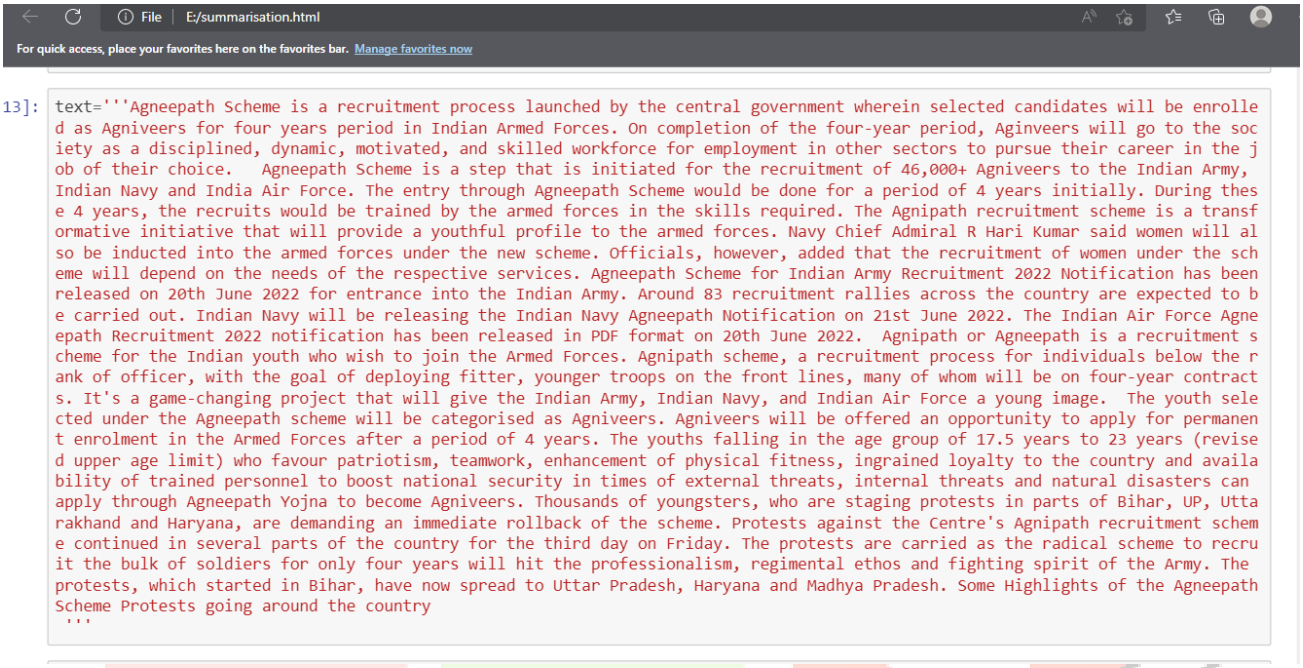

Fig 5.1 – Our Test Data

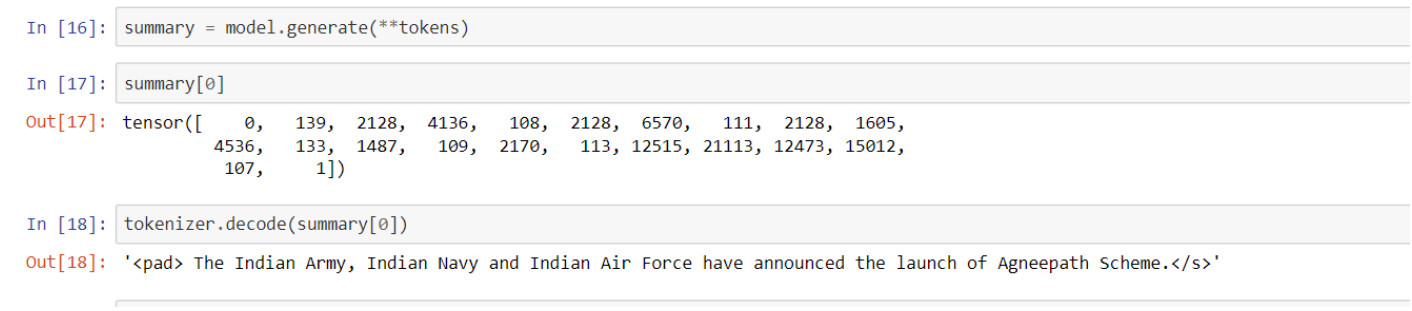

Fig 5.2 – Result

IV. CONCLUSION

In this paper, we have introduced a new graph-based text summarizer that can efficiently reduce the size of a text while maintaining the integrity. The proposed method is simple to implement. It is expected that our model will improve if an efficient preprocessing routine is used prior to the implementation of the proposed method. Such practice is common in Text summarization. Finally, the proposed method can be used for real time applications.

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

Abstractive summarization **uses the Pegasus model**. The model uses Transformers Encoder-Decoder architecture. The encoder outputs masked tokens while the decoder generates Gap sentences. Abstractive summarization aims to take a body of text and turn it

into a shorter version.In this work, we used PEGASUS, a sequence-to sequence model with gap-sentences generation as a pre-training objective tailored for abstractive text summarization. We studied several gap-sentence selection methods and identified principle sentence selection as the optimal strategy. We finally showed our model summaries achieved human performance on multiple datasets using human evaluation.

VI. REFERENCES

[1] Saif alZahir, Qandeel Fatima and Martin Cenek," New Graph-Based Text Summarization Method", IEEE, 2015.

[2] Shuai Wang,Xiang Zhao, Bo Li,Bin Ge and Daquan Tang," Integrating Extractive and Abstractive Models for Long Text Summarization", IEEE, 2017.

[3] Si Huang ,Rui Wang, Qing Xie, Lin Li and Yongjian Liu," An Extraction-Abstraction Hybrid Approach for Long Document Summarization",IEEE,2019.

[4] Mahmoud R. Alfarra, Abdalfattah M. Alfarra AND Jamal M. Alattar ,"Graph-based Fuzzy Logic for Extractive Text Summarization (GFLES)",ICPET,2019.

[5] Samridhi Murarka and Akshat Singhal," Query-based Single Document Summarization using Hybrid Semantic and Graph-based Approach", IEEE, 2020.

[6] Jingqing Zhang,Yao Zhao,Mohammad Saleh and Peter J. Liu, "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization", arXiv:1912.08777v3 [cs.CL], 2020.

[7] Chandra Khatri et. al., "Abstractive and Extractive Text Summarization using Document Context Vector and Recurrent Neural Networks", arXiv: 1807.08000v2 [cs.CL] 29 Jul, 2018.

[8] Ramesh Nallapati et. al., "Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond", arXiv: 1602.06023v5 [cs.CL] 26 Aug, 2016.

[9] Chrysoula Zerva1, Minh-Quoc Nghiem, Nhung T. H. Nguyen, Sophia Ananiadou, "Cited text span identification for scientific summarization using pre-trained encoders", Springer, 7 May 2020.

[10] N. Moratanch and Dr. S. Chitrakala, "A survey on abstractive text summarization", ICCPCT, 2016.

[11] Dima Suleiman and Arafat Awajan, "Deep Learning Based Abstractive Text Summarization: Approaches, Datasets, Evaluation Measures, and Challenges", Hindawi, 24 August 2020.

[12] Yang Liu and Mirella Lapata, "Text Summarization with Pretrained Encoders", arXiv: 1908.08345v2 [cs.CL], Sep 2019.

[13] Gunes Erkan and Dragomir R. Radev**, "**LexRank: Graph-based Lexical Centrality as Salience in Text Summarization", arXiv: 1109.2128v2 [cs.CL], 2004.

[14] Ekaterina Zolotareva, Tsegaye Misikir Tashu and Tomáš Horváth, "Abstractive Text Summarization using Transfer Learning", CEUR-WS.org, 2020.

[15] Deepali K. Gaikwad and C. Namrata Mahender, "A Review Paper on Text Summarization" , IJARCCE, March 2016.