NLP Project Report

Harsh (2020061) **Jay Saraf(2020438)**
Rohan Gupta (2020113) **Vatsal Chaudhary(2020549)**

1 Introduction

Text summarization aims to condense lengthy documents into shorter versions that highlight the key ideas of the original document. Three types of models are mainly used for text summarization: extractive, abstractive, and hybrid. Extractive models *([1] Dorr et al., 2003; [2] Nallapati et al., 2017)* select the most important sentences in a given document and directly copy them into the summary. Abstract summarization *([3] Rush et al., 2015; [4]See et al., 2017)*, involves generating new sentences that may not appear in the original text but convey the same meaning. Hybrid models *([5]Hsu et al., 2018; [6]Liu et al., 2018; [7]Gehrmann et al., 2018; [8]Chen and Bansal, 2018)* are the ones that combine both extractive and abstractive approaches. All the abstractive models are neural networks based, so training them from scratch requires a large amount of data and computational resources. So fine-tuning a model that's pre-trained using self-supervised objectives like masked language modeling *([9]; [10])* is the better option. But usually, in the case of text summarization, the document size that has to be summarized is quite extensive, so even fine-tuning is quite expensive computationally. This is where hybrid models come in, they first extract the most important sentences from the given document then those sentences are fed to an abstractive model to generate the summary, sort of like the best of both worlds. These days, pretty complex extractive models *([11] Xu et al., 2020)* based on neural networks are available, but we will not discuss them in this paper. Our aim in this paper is to analyze computationally inexpensive extractive methods and how they affect the performance of abstractive models that are fine-tuned using the text extracted using them. We do this analysis of text summarization in the Hindi language. The primary motivation behind considering a computationally inexpensive extractive method is to check whether a hybrid text summarization model can be trained using less computational resources without sacrificing too much performance.

2 Related Work

For good text summarization, a balance has to be maintained between staying close to the document and allowing abstractive modifications.
[14] Durrett et al. (2016) used a discriminative model that selects the textual content to include in the summary based on a rich set of sparse features whose weights are learned on a large corpus.

[15] Dorr et al. (2003) used a model that first extracts noun and verb phrases from the first sentence of an article and then generates the summary using an iterative algorithm.
[16] Gu et al. (2016) proposed a new model "CopyNet" with encoder-decoder structure which incorporated copying into neural network-based Seq2Seq learning. This model could choose and copy sub-sequences in the input sequence and put them at proper places in the output sequence and this model was also abstractive in nature. This was like an end-to-end hybrid model for text summarization.
[17] Mihalcea et al. introduced TextRank- a graph based ranking algorithm for text processing. It was an unsupervised method for keyword and sentence extraction.

3 Methodology

We first applied extractive algorithms to the data to get the most important sentences from each article in the data, and then these sentences were fed to an abstractive model to fine-tune it. We fine-tuned the abstractive models for 10 epochs. All the details about the extractive methods and the abstractive models that we used are mentioned in subsequent subsections.

3.1 Dataset

We have used the datasets provided for the Indian Language Summarization (ILSUM) conference 2023 Task-1 for the Hindi language. Their test data did not consist of corresponding summaries for articles, and hence could not be used for computation of validation and test ROUGE scores. Therefore, we split the training data file in the ratio 90:5:5 for training:validation:test purposes. The three corresponding data files have been included with the report in the submission.   While doing the preprocessing we have dropped the null values and we have added start and end tokens for every sentence.

3.2 Extractive Methods

3.2.1 TextRank

The TextRank algorithm is a graph-based approach proposed in "TextRank: Bringing Order into Texts" by *Mihalcea et al. 2004.* It takes inspiration from the PageRank algorithm proposed by Google in 1998 to rank web pages in a web search, and extends it to sentences available in a corpus of text. In this setup, each sentence is treated as a graph node, and an edge from one node to another is called a 'vote' which increases the importance or score of that sentence. The score of a sentence is determined by the number of 'votes' that it receives, along with the current score of the node casting the 'vote'. The process of computing the score of a node based on its neighbors is iteratively applied till the change in score from the previous iteration falls below a convergence threshold. The formula for computing the score is:

$S(V_i) = (1-d) + \sum_{in(Vi)} (d*S(V_j)/count(out(V_j)))$

Here, d is the damping factor (0.85), $in(V_i)$ is the set of nodes pointing towards V_i, while $out(V_i)$ is the set of nodes to which V_i points. Initial scores are chosen arbitrarily, and are determined by computing the similarity with all neighbors in subsequent iterations. The similarity between two sentences is given as follows:

$Similarity (S_i, S_j) = count(common words)/(log(wordCount(S_i)) + log(wordCount(S_j)))$

Based on final scores, sentences are sorted in reverse order to get the highest ranked sentences.

3.2.2 Tf-Idf Vectorizer

The Term Frequency - Inverse Document Frequency vectorizer algorithm is a ranking system that was initially proposed in “A Statistical Interpretation of Term Specificity and its Applications in Retrieval” by *Jones (1972)*. While this approach is more commonly applied in Information Retrieval for determining the important words, we apply a modified approach to compute scores for sentences. The Term Frequency (TF) is the ratio of the count of the particular word to the number of words in the sentence.

$TF (w_i, s) = count(w_i \text{ in } s)/wordCount(s).$

The calculation of the Inverse Document Frequency (IDF) remains the same.

$IDF (w_i) = log(\text{Number of Documents in corpus}/\text{Number of Documents containing } w_i)$

Upon calculation of these two values, the score of the sentence is given by the following formula:

$Score (S_i) = \sum_{wi} TF(w_i, s_i)*IDF(w_i)$, which is basically the sum of the product of TF and IDF values of all words in the sentence. These scores can again be sorted in reverse order, and the higher scores denote more relevant sentences for summarization.

3.2.3 BM25

Best Match 25 (BM25) is a ranking algorithm, which is used in information retrieval and data mining tasks. It is used for ranking documents based on the specific topic.

The formula for computing BM25 is

$$MB25(Q,D) = \Sigma^{n}_{i=1} IDF(q_i) \times \frac{f(qi,D) \times (k_1+1)}{f(qi,D) + k_1 (1-b+b \times \frac{|D|}{avgdl})}$$

n is the number of terms in the query.

$q_{\_i}$ is the i^{th} term in the query.

$|D|$ is the length of document D (in terms, usually words).

'avgdl' is the average document length in the entire collection.

k_1 and b are free parameters that adjust the impact of term frequency normalization and document length normalization, respectively.

f(q_i , D) is the frequency of term q_i , which is calculated as
IDF(q_i) is the Inverse Document Frequency of term q_i , which is calculated as
IDF(q_i) = $\log(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5})$
In the above equation, N is the total number of Articles in the input file.
n(q_i) is the number of documents containing term q_i .

3.3 Abstractive Models

3.3.1 IndicBART

IndicBART is a pre-trained sequence-to-sequence model for a group of 11 Indian languages. Most of the Indian languages have linguistic similarities between them due to them being derived from common ancestor languages. Also there is evidence that this linguistic similarity can result in positive-lingual transfer for NLP applications like NMT *([20] Goyal et al., 2020)*. Hence IndicBART is a compact model with 244 M parameters only because of linguistic similarity of Indian languages. IndicBART is trained on the publicly available Indic language corpora IndicCorp *([21] Kakwani et al. 2020)*
IndicBART is conceptually based on the mBART25/50 model family *([10])*. English is also included in the pretraining data as English words are widely used with other Indian languages.
Model architecture and Pre-training details: IndicBART uses 6 encoder and decoder layers with hidden and filter sizes of 1024 and 4096, respectively, and 16 attention heads which equate to 244M parameters. As in mBART, 35% of the words in each sentence was masked by randomly sampling a span of length according to a poisson distribution(lambda = 3.5).
Why we used this model: We needed a model with an encoder + decoder, an encoder could encode the meaning of sentences in a latent space and a decoder for autoregressive generating the summaries from this latent representation. Hence model similar to BART was the best choice.

We fine-tuned this IndicBART model using data extracted from the hindi dataset of articles and headline pairs from several leading newspapers of the country.

3.3.2 mT5

MT5 is a Multilingual Translation Transformer model and is a powerful pre-trained sequence-to-sequence model designed for multilingual natural language processing tasks, including summarization. MT5 is particularly well-suited for handling summarization tasks in Hindi and other Indic languages due to its extensive coverage and robust architecture.

Model Architecture and Pre-training Details: MT5 consists of 12 encoder and decoder layers, with hidden and filter sizes of 512 and 2048, respectively , MT5 incorporates 8 attention heads, resulting in a total of 245M parameters. During pre-training, a masked language modeling objective is employed, in which a percentage of tokens in each sentence are masked, and the model is trained to predict the masked tokens based on the surrounding context

Why We Chose This Model: We selected MT5 for Hindi summarization due to its superior performance and versatility in handling multilingual text. By fine-tuning MT5 with a dataset of articles and Summary pairs from Hindi newspapers, we aimed to leverage the model's advanced architecture and pre-trained representations to generate high-quality summaries in Hindi.

4 Observation Tables

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is the widely used performance metric in text summarization. To calculate ROUGE, we compare our generated summary with a reference summary and find how much they overlap. ROUGE1 measures the amount of overlap of unigrams, and ROUGE2 measures the overlap between bigrams. RougeL measures the most extended matching sequence of words by looking for the longest common substrings in generated and reference summaries and averaging over individual sentences. The rouge scores are scaled to 100 in our observations. In the result table, the base case is the one in which no extractive method was used, and the whole article was fed to the abstractive models during finetuning and summary generation.

4.1 IndicBART

	Rouge 1	Rouge 2	Rouge L
Base	18.76	7.66	18.31
TextRank	16.51	6.27	15.92
TF-IDF	16.39	6.63	15.79
BM25	18.27	7.60	17.69

4.2 MT5

	Rouge 1	Rouge 2	Rouge L
Base	13.8	5.18	13.41
TextRank	12.27	4.30	11.66
TF-IDF	12.72	4.47	12.1
BM25	15.00	6.05	14.46

5 Results and Analysis

As you can see from the observation tables, performance in TextRank and Tf-Idf is not much less than that of the base case, and in the case of BM25 it is almost comparable to the base case. From these observations, we can use these extraction techniques before fine-tuning an abstraction model to decrease the computation required.

6 Future work

We plan on developing a neural abstractive model that combines both extractive and abstractive techniques in a single end-to-end model. Instead of doing the extractive and abstractive task separately, both would be handled by a single model similar to *[22]Gu et al. 2016*.

References

1) Bonnie Dorr, David Zajic, and Richard Schwartz. 2003. Hedge trimmer: A parse-and-trim approach to headline generation. In HLT-NAACL.
2) Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In AAAI.
3) Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. Proceedings of EMNLP.
4) Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer generator networks. In ACL.
5) Wan Ting Hsu, Chieh-Kai Lin, Ming-Ying Lee, KeruiMin, Jing Tang, and Min Sun. 2018. A unified model for extractive and abstractive summarization using inconsistency loss. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers.
6) Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.
7) Sebastian Gehrmann, Yuntian Deng, and Alexander M.Rush. 2018. Bottom-up abstractive summarization. In EMNLP, pages 4098–4109. Association for Computational Linguistics.
8) Yen-Chun Chen and Mohit Bansal. 2018. Fast abstractive summarization with reinforce-selected sentence rewriting. In ACL (1), pages 675–686. Association for Computational Linguistics.
9) [1810.04805] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
10) BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension
11) Jiacheng Xu, Zhe Gan, Yu Cheng, and Jingjing Liu. 2020. Discourse-Aware Neural Extractive Text Summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5021–5031, Online. Association for Computational Linguistics.

12)Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. 2018. Bottom-Up Abstractive Summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109, Brussels, Belgium. Association for Computational Linguistics.

13)Wojciech Kryscinski, Nitish Shirish Keskar, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Neural Text Summarization: A Critical Evaluation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 540–551, Hong Kong, China. Association for Computational Linguistics.

14) [1603.08887] Learning-Based Single-Document Summarization with Compression and Anaphoricity Constraints

15) Bonnie Dorr, David Zajic, and Richard Schwartz. 2003. Hedge Trimmer: A Parse-and-Trim Approach to Headline Generation. In *Proceedings of the HLT-NAACL 03 Text Summarization Workshop*, pages 1–8.

16)Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. arXiv preprint arXiv:1603.06393.

17)Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing Order into Text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain. Association for Computational Linguistics.

18)[2109.02903] IndicBART: A Pre-trained Model for Indic Natural Language Generation

19) Vikrant Goyal, Anoop Kunchukuttan, Rahul Kejriwal,

20) Siddharth Jain, and Amit Bhagwat. 2020a. Contact relatedness can help improve multilingual NMT: Microsoft STCI-MT @ WMT20. In Proceedings of the Fifth Conference on Machine Translation, pages 202–206, Online. Association for Computational Linguistics.

21) Divyanshu Kakwani, Anoop Kunchukuttan, Satish Golla, Gokul N.C., Avik Bhattacharyya, Mitesh M.Khapra, and Pratyush Kumar. 2020. IndicNLPSuite: Monolingual corpora, evaluation benchmarks and pre-trained multilingual language models for Indian languages. In Findings of the Association for Computational Linguistics: EMNLP 2020, pages 4948–4961, Online. Association for Computational Linguistics.

22) Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. arXiv preprint arXiv:1603.06393.