International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 09 | Sept - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

Text Summarization System by Deep Learning & Transfer Model

Anush Sharma
Assistant Professor, Dept of CSE,
HIET, Kangra, HP, India

Nishant
Student, Dept of CSE,
HIET, Kangra, HP, India

Aryan
Student, Dept of CSE,
HIET, Kangra, HP, India

Ansh
Student, Dept of CSE,
HIET, Kangra, HP, India

Abstract:
This paper evaluates recent advances in text summarization strategies, focusing at the strategies developed and deployed in a practical assignment. The project makes use of both abstractive and extractive summarization methods, employing deep gaining knowledge of models such as the Transformer architecture. This paper will discover the comparative overall performance of various models, discuss their deployment, and highlight destiny potentialities for boosting textual content summarization technology.

Problem Statement:

- Developing an automated text summarization system that can accurately and efficiently condense large bodies of text into concise summaries is essential for enhancing business operations.
- This project aims to deploy NLP techniques to create a robust text summarization tool capable of handling various types of documents across different domains.
- The system should deliver high-quality summaries that retain the core information and contextual meaning of the original text.

Project Statement:

- Text Summarization focuses on converting large bodies of text into a few sentences summing up the gist of the larger text.
- There is a wide variety of applications for text summarization including News Summary, Customer Reviews, Research Papers, etc.
- This project aims to understand the importance of text summarization and apply different techniques to fulfill the purpose.

Approach to Solution:

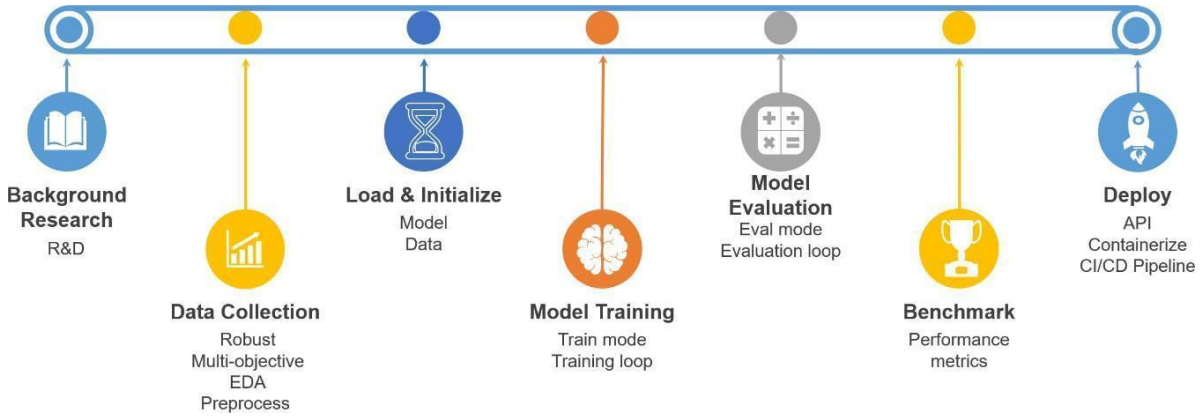


Fig.: Transformer architecture

Background Research:

Literature Review

S. No	Use-Case	Paper Title	Year	Method	Dataset	Results	Limitations
1	General text summarization	Text Summarization Using Deep Learning Techniques: A Review	2023	Deep Learning (Seq2Seq, Attention, Transformers)	CNN/Daily Mail, XSum	Improved performance in capturing semantic relationships, better coherence	Computationally expensive, requires large datasets
2	Implementation of the Transformer architecture	Attention is all you need	2023	Transformer	WMT 2014 English-German, WMT 2014 English-French	Introduced the Transformer architecture, significantly improving the performance of text summarization tasks.	Requires large datasets and computational resources for training.
3	Multi-document summarization	Surveying the Landscape of Text Summarization with Deep Learning	2023	Deep learning methods. Various techniques like RBMs and fuzzy logic employed for summarization.	CNN/DailyMail	Incorporating transfer learning enhances summary quality and reduces data demand.	Complex models, high computational resources
4	Abstractive summarization	Pegasus: Pre-training with gap-sentences for abstractive summarization	2020	Transformer (Pegasus)	XSum, CNN/DailyMail, and Reddit TIFU	Significant improvements in abstractive summarization quality	Resource-intensive
5	Extractive summarization	Text Summarization with Pre Trained Encoders	2019	Intersentence Transformer layers for summarization	CNN/Daily Mail, NYT, Xsum, DailyMail	BERT-based models outperformed other approaches in abstractive summarization.	High computational resources required

Keywords: BART Model, Abstractive Summarization, Extractive Summarization, Pretrained Models, Deep Learning for Summarization, Natural Language Processing (NLP), Transformer Models, Language Models, Neural Networks, Sequence-to-Sequence Models, Rouge Score for Evaluation, Text Compression, Model Fine-Tuning, Transfer Learning in NLP

Introduction:

The automation of textual content summarization has won growing interest, given the overwhelming volume of textual facts generated day by day. text summarization condenses big quantities of records into shorter, meaningful summaries whilst maintaining middle content. This overview focuses on a project that deployed both abstractive and extractive summarization strategies, leveraging herbal Language Processing (NLP) techniques.

The venture's intention was to increase a summarization device that efficiently procedures and summarizes numerous document types, enhancing facts accessibility. This paper evaluates the methodologies implemented, compares their effectiveness, and discusses potential enhancements and packages.

1.Motivation and Scope

Within the virtual age, the sheer volume present day textual data generated day by day affords opportunities and demanding situations. The capability to efficiently summarize big quantities of modern day textual content is crucial for expertise control, facts retrieval, and decision-making strategies. Text Summarization—automatically condenses information from a text even as preserving its critical which means—has consequently emerged as a substantial research location in herbal language processing (NLP). The inducement in the back of this overview paper lies inside the developing need to distill applicable information from significant datasets quickly and appropriately. This paper aims to provide a comprehensive assessment of brand new textual content summarization strategies, highlighting the advancements, challenges, and potential destiny directions within the area.

2.Need of Examination

No matter considerable progress in text summarization, the sector keeps to face several challenges. The variety of tactics—from extractive to abstractive summarization—calls for cautious exam to recognize their effectiveness, barriers, and packages. Advances in deep today's and herbal language understanding have brought new methodologies, but they also include their personal set modern day complexities. This evaluation is essential to synthesize current studies, pick out gaps, and offer a coherent framework for evaluating exceptional summarization techniques. via inspecting the evolution of modern-day text summarization techniques and their effect on various packages, this paper pursuits to clarify the present day country today's artwork and advocate instructions for future research.

3.Literature Survey/Review

This overview paper systematically surveys the literature on text summarization, focusing on the important techniques, methodologies, and evaluation metrics. We begin with a historical attitude, tracing the improvement of brand new summarization techniques from early rule-based systems to present day gadget today's approaches. Key subjects consist of extractive summarization techniques, which includes the ones based totally on frequency and graph-based techniques, and abstractive summarization techniques leveraging deep contemporary and transformer fashions. Additionally, the evaluation covers hybrid strategies that combine elements of ultra-modern, each extractive and abstractive methods. We also talk about numerous assessment frameworks used to assess the great and effectiveness brand new summarization structures, along with each intrinsic and extrinsic metrics. By imparting a radical exam of these aspects, this paper aims to provide a holistic view in text summarization.

Research Methodology:

This section outlines the methodologies employed for both **abstractive** and **extractive** text summarization in the research. Each approach is discussed in detail, highlighting the tools, techniques, and evaluation processes used to implement and assess the models.

1. Abstractive Text Summarization Methodology

For abstractive summarization, the primary goal was to generate concise summaries by synthesizing and rephrasing content rather than extracting sentences directly from the text. This was achieved through a two-fold implementation approach using **Native PyTorch** and the **Trainer API**. Below are the steps and tools involved:

- **Native PyTorch Implementation**
  A custom implementation was done using PyTorch, one of the most widely used deep learning frameworks. This approach allowed for precise control over the model's architecture, training process, and evaluation.
- **Trainer API Implementation**
  To streamline the training process and optimize performance, the Trainer API was utilized. This API simplifies the fine-tuning of transformer models, providing an efficient interface for large-scale training and evaluation tasks.

Implemented Techniques:

- **Loading a Fine-Tuned Transformer Model**

  The fine-tuned transformer model was loaded from a saved checkpoint, leveraging pre-trained transformer models (such as BERT, GPT, or T5) for abstractive summarization. Transfer learning allowed the model to quickly adapt to the specific summarization task.

- **OOP Implementation of the Dataset**

  Object-Oriented Programming (OOP) principles were used to create a custom Dataset class, ensuring efficient data handling and pre-processing. The following steps were followed:

  - **Features and Targets:** Extracting input text and corresponding summaries (targets).
  - **Tokenization:** Tokenizing the input text using a pre-trained tokenizer corresponding to the transformer model.
  - **Padding and Truncation:** Ensuring consistent input sizes by padding and truncating sequences as per the model's requirements.
  - **Tensor Conversion:** Converting the tokenized data into tensors, the format required for PyTorch operations.
  - **DataLoader:** Passing the dataset to a PyTorch DataLoader, batching the data to enhance computational efficiency during training and evaluation.

- **Model Evaluation**
  - The model was set to **evaluation mode** to ensure no gradients were computed, and unnecessary overheads were avoided during inference.
  - **ROUGE Metrics:** The **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)** metric was used to evaluate the quality of generated summaries by comparing them to reference summaries.

- **Batch Processing:** Summaries were generated in batches to maximize efficiency. The model iterated through the batches in the DataLoader, moving the data to the available computational device (CPU/GPU).
- **Summary Generation and Decoding:** For each batch, the model generated summaries, which were then decoded from the predicted token sequences into human-readable text. The reference labels were also decoded for comparison.

2. Extractive Text Summarization Methodology

For extractive summarization, the objective was to select key sentences directly from the input text that represent the central idea, without rephrasing the content. A combination of classical and modern machine learning techniques, including **TextRank**, **TF-IDF**, and **KMeans clustering**, were used.

Implemented Techniques

- **Data Preprocessing**
  To ensure the input text was in a standardized format, a series of preprocessing steps were applied:
  - **Lowercasing:** All text was converted to lowercase to remove any case sensitivity in the analysis.
  - **Stop Words Removal:** Common words that do not contribute significant meaning (e.g., "the", "and", "is") were removed.
  - **Lemmatization:** Words were reduced to their base forms to ensure that different grammatical forms of a word were treated as a single token.
  - **Tokenization:** The text was split into individual tokens (words or phrases), which served as the fundamental units of analysis.
  - **POS Tagging:** Part-of-Speech (POS) tagging was applied to label each token with its grammatical role (e.g., noun, verb), which helped identify key information-bearing sentences.

- **TF-IDF Vectorization and Clustering**
  - **TF-IDF (Term Frequency-Inverse Document Frequency) Vectorization:** The text was transformed into a matrix where each sentence was represented as a vector of TF-IDF scores, reflecting the importance of each word relative to the document.
  - **KMeans Clustering:** The TF-IDF matrix was then fed into the KMeans clustering algorithm, which grouped sentences into clusters based on their similarity. The number of clusters was dynamically adjusted based on the number of sentences in the text, ensuring optimal grouping.
  - **Centroid Identification:** For each cluster, the sentence closest to the cluster's centroid (representative sentence) was identified as the most central sentence within that group.
- **Sentence Selection and Summary Generation**
  - **Cluster-Based Sentence Selection:** After clustering the sentences, the most representative sentence from each cluster was selected. This approach ensured that the summary covered diverse aspects of the input text.
  - **Summary Formation:** The representative sentences were joined together to form the extractive summary. Care was taken to ensure that the selected sentences were ordered in a coherent manner to preserve the logical flow of the original document.
- **Model Evaluation**
  - The extractive summaries were evaluated using **ROUGE metrics**, similar to the abstractive approach, comparing the selected sentences against reference summaries.
  - The evaluation focused on both the **precision** (how many selected sentences were relevant) and **recall** (how many relevant sentences were captured in the summary).

3. Comparison of Methods

Both the abstractive and extractive approaches were compared in terms of performance, resource usage, and suitability for specific types of text. Key areas of comparison include:

- **Abstractive Summarization:** Generates novel sentences, leading to more fluent and human-like summaries but requiring greater computational resource and training time.
- **Extractive Summarization:** Selects sentences directly from the input text, providing factually accurate summaries with less computational complexity, but the summaries may lack fluidity and coherence compared to the abstractive approach.

Implementation:

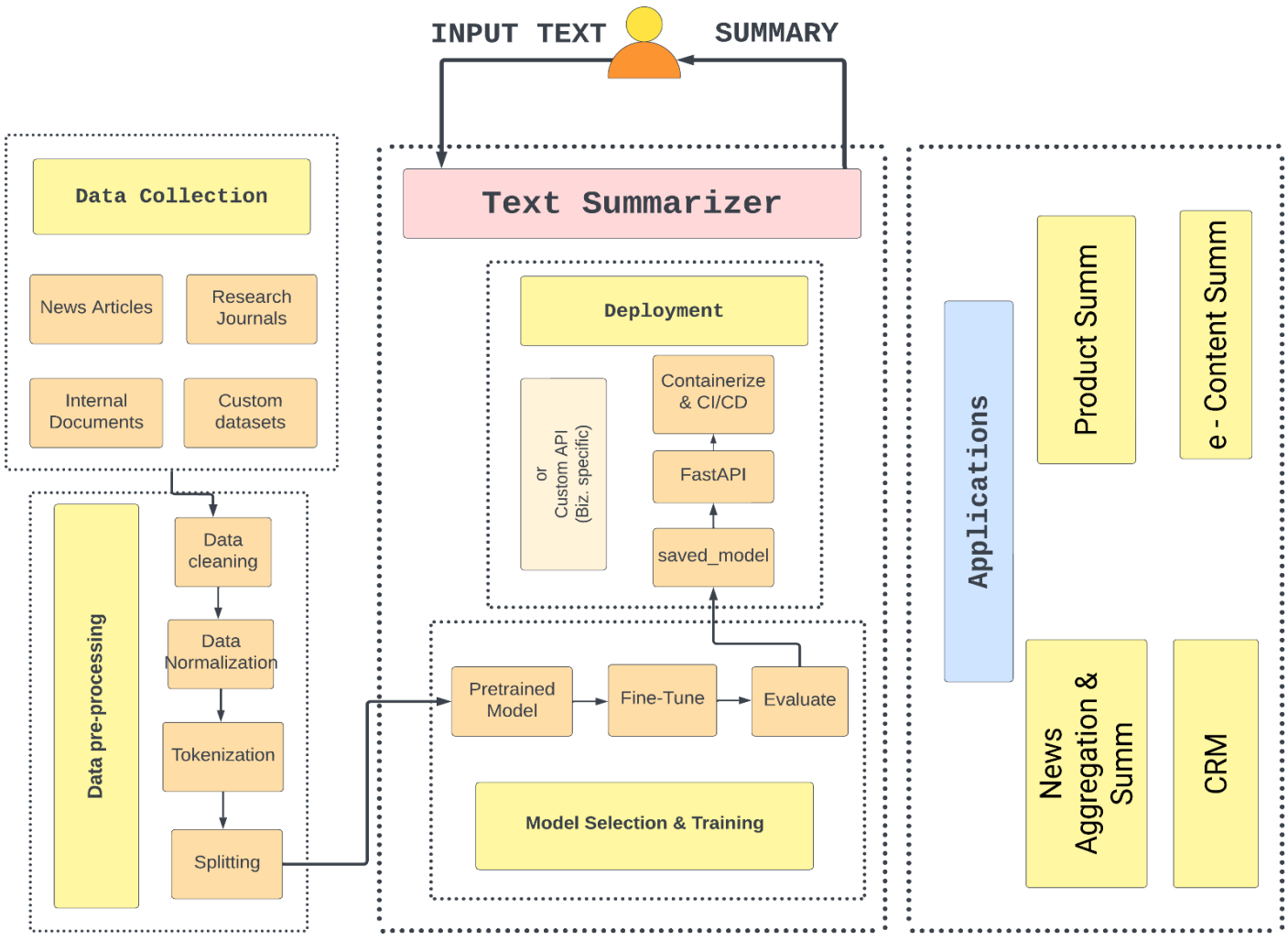


Fig.: Implementation for Text Summarizer.

Data Collection:

Data Preprocessing & Pre-processing Implemented in src/data_preprocessing.

Data collection from different source

- CNN, Daily Mail: News,
- BillSum: Legal,
- ArXiv : Scientific,
- Dialogues : Conversations.

Data was integrated from different sources, to ensure robust and multi-objective data. The objective of the data spans including News articles, Legal Documents – Acts, Judgements, Scientific papers, Conversations between persons.
The quality and consistency of the raw data is very vital for the model training, to achieve benchmark performance metrics - ROUGE.
Validated the data through Data Statistics and Exploratory Data Analysis (EDA) through Frequency Plotting, for every data source.
Saved each data in separate files (preprocessed - 1, 2, 3, 4) in csv format.

Data integration of selective data from each source.

- Selected from splits (train, test, val).
- Factors considered:Number of records
- Quality of data
- Representation of different data groups (Multi-Objective)
- To enable the model training in domestic GPU.
- Limits the number of records.
- Selected data:

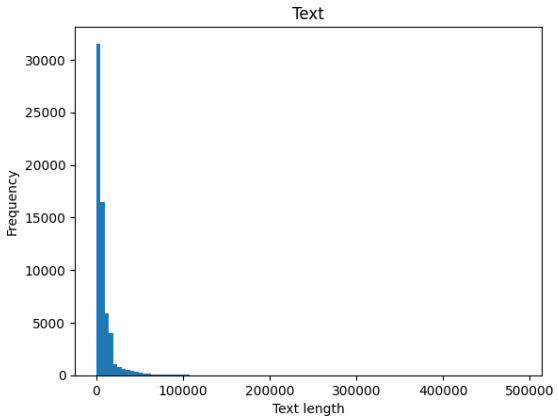
  CNN, Daily Mail - test, validation

  - BillSum - train
  - ArXiv - test
  - Dialogues - train

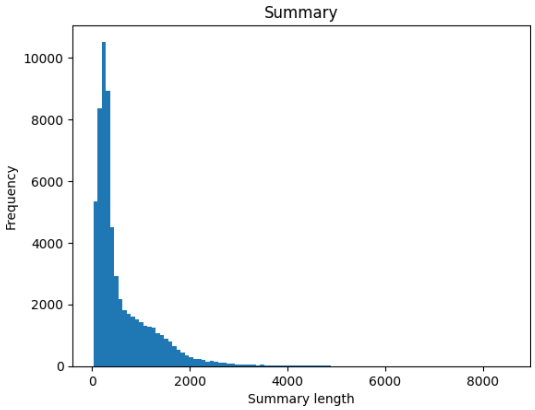
- renamed the attributes to general form (text, summary).
- Validated the data using statistics and frequency plot.

Dataset before cleansing:

```
count     62707.000000
mean       8209.765879
std       11786.346696
min         190.000000
25%        1892.000000
50%        5051.000000
75%        9439.500000
max      489287.000000
Name: text, dtype: float64
```




```
count    62707.000000
mean       612.585150
std        620.618153
min         31.000000
25%        213.000000
50%        353.000000
75%        839.000000
max       8541.000000
Name: summary, dtype: float64
```

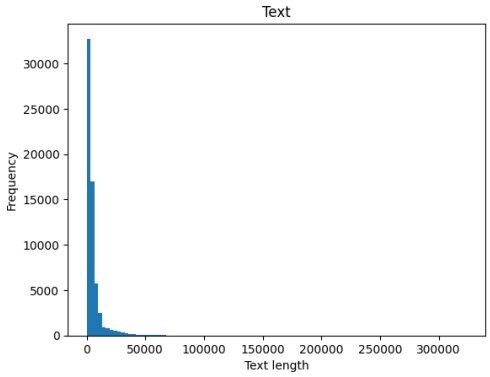


Performed data cleansing optimized for NLP tasks.

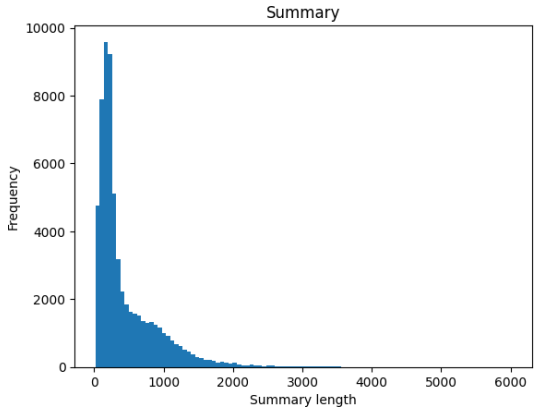
- Removed null records.
- Lowercasing.
- Punctuation removal.
- Stop words removal
- Lemmatization.

Dataset after cleansing:

```
count     62707.000000
mean       5211.270975
std        7794.860686
min          83.000000
25%        1275.000000
50%        3176.000000
75%        5684.500000
max      323742.000000
Name: text, dtype: float64
```




```
count    62707.000000
mean       448.081937
std        459.087443
min         16.000000
25%        154.000000
50%        255.000000
75%        618.000000
max       6014.000000
Name: summary, dtype: float64
```



Performed data splitting from the integrated data- train, test, validation - for training, testing and validating the model respectively, using sci-kit learn.

- Saved the data in csv format.

Abstractive Text Summarizer:

Model Training and Evaluation

Training:

The selected transformer architecture, for ABSTRACTIVE SUMMARIZATION, can be implemented in a couple of ways. Either developing the neural network from scratch and initializing it with normalized weight and biases, then training the model with massive datasets for NLP tasks, or retraining the foundational model with custom dataset i.e. fine-tuning the pre-trained model.
The former way, will pave the way to excessive unoptimized resource utilization, in terms of computation (GPU).

Also, this would not out-perform the fine-tuning of the larger foundational model.

Fig.: Fine-Tuning Overview

The choice of foundational model vests in considering lots of factors including its performance metrics and efficient trainable parameters in the model.
With due analysis, *Facebook's Bart Large* – was chosen as the foundational model for abstractive text

summarization.

- 406,291,456 total parameters.
- 406,291,456 training parameters.

Model Training can be achieved in two (2) methods:

1. Native PyTorch Implementation
2. Trainer API implementation.

Evaluation:

Performance metrics – ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

- o ROUGE – Measures the overlap between generated summary and reference summary.
- o Best suited: evaluating 'Text Summarization' tasks.
- o Other options : **BLEU**.
- o Available metrics in ROUGE: ROUGE-N = 1,2,3; ROUGE LSUM.

METHOD 1 - Native PyTorch

Native PyTorch implementation, source code: src/model.ipynb

Implemented works as follows:

Loaded pre-trained transformer

- Facebook's Bart Large

Developed OOP implementation of Dataset

- Feature, Target Loading
- Tokenized the dataset
- Padding, Truncate w.r.t maximum length
- Converted to Tensor
- Passed on to DataLoader – with batch size

Developed manual PyTorch training Loop

- Set the model in 'Train mode'.
- Utilized 'Adam' optimizer.
- Forward pass & compute loss
- Backward pass
- Updated params – compute gradient
- Updated Learning Rate
- Zeroed the gradients
- Updated total loss
- [Average Training Loss: 1.3280]

Saved the fine-tuned transformer model -> saved model

Developed manual evaluation loop:

- Set the model to 'Eval mode'.
- No gradient calculation for evaluation.
- Forward pass & compute loss.
- Accumulate batch loss.
- Printed batch information.
- Calculated average validation loss.
- Printed final evaluation results (loss and time).
- [Validation Loss: 2.4502].

METHOD 1 - MODEL

Evaluation:

Model Evaluation, source code: src/evaluation.ipynb

Implemented works:

Load fine-trained transformer

- From saved model

OOP implementation of Dataset

- Feature, Target
- Tokenize
- Padding, Truncate

Evaluate Model

- Set model to evaluation mode
- Load ROUGE metric
- Loop through batches in DataLoader

- Move data to device
- Generate summaries
- Decode predictions and labels
- Add to ROUGE metric
    - Compute ROUGE scores
- Evaluate model on validation dataset
    - Print results (ROUGE mid).
- Saved fine-tuned BART model and tokenizer. (Saved model)

Results:
Performance metrics – After fine-tune

ROUGE-1
- **Precision:** 0.000307
- **Recall:** 0.000163
- **F-Measure:** 0.000182

ROUGE-L
- **Precision:** 0.000295
- **Recall:** 0.000138
- **F-Measure:** 0.000161

ROUGE-2
- **Precision:** 0.000000
- **Recall:** 0.000000
- **F-Measure:** 0.000000

ROUGE-Lsum
- **Precision:** 0.000293
- **Recall:** 0.000141
- **F-Measure:** 0.000164

Observations:

- The trained model from method 1 was not used for deployment.

(Trained model from method 2 was used for deployment)

Reason:

- Even though the model has very minimal training loss, but the model performed inconsistently in the validation & testing phase.
- There's a suspected tensor error while training using method 1, which could be attributed to the inconsistency of the model's output.

Dire need to implement alternative approach optimized for transformer model, to produce benchmark

METHOD 2 – Trainer Class Implementation

Trainer class implementation, source code: src/bart.ipynb.

Utilized Trainer API from Hugging face, Trainer class is optimized to train the transformer models and Pre-Trained models.
Enables to have feature-complete training and eval loop for PyTorch, optimized for transformer models.

Implemented works in this module, as follows:

Data Preparation:

- Loaded and converted datasets to Hugging Face Dataset format -PyArrow

Data Preprocessing:

- text-to-input conversion.
- mapped preprocessing to datasets with parallel processing.

Model Training:

- Load pre-trained model

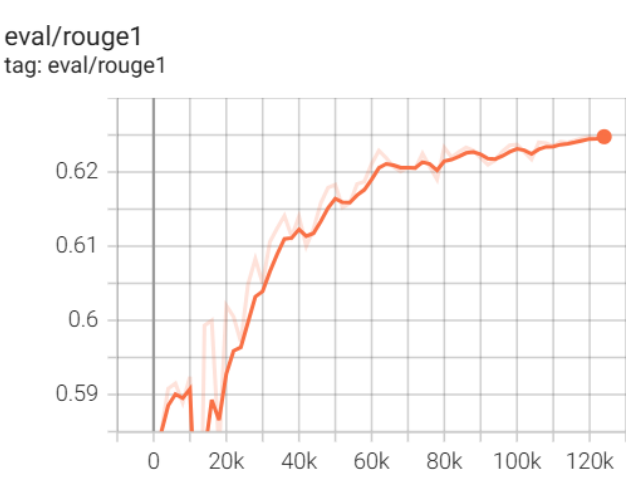
Facebook - Bart large

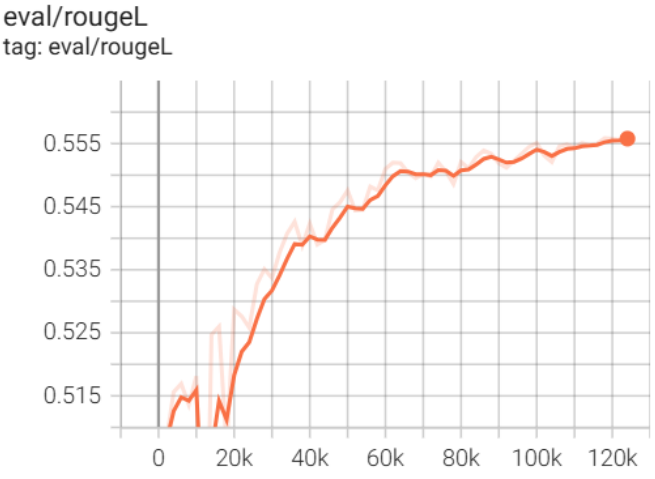
- defined evaluation metrics
- Configured training arguments
- Initialized Trainer
- Trained model and obtained training history.

Model Saving:

- Saved fine-tuned BART model and tokenizer (Saved model)

TensorBoard Training Monitor:





Results:

The model was trained with whole dataset for 10 epochs for 26:24:22 (HH:MM:SS) in 125420 steps.

- Train loss = 17.28 (final)
- ROUGE1 score = 62.52 (Last checkpoint)
- Transformer model for abstractive text summarization was successfully trained with the integrated custom data.

Observation:

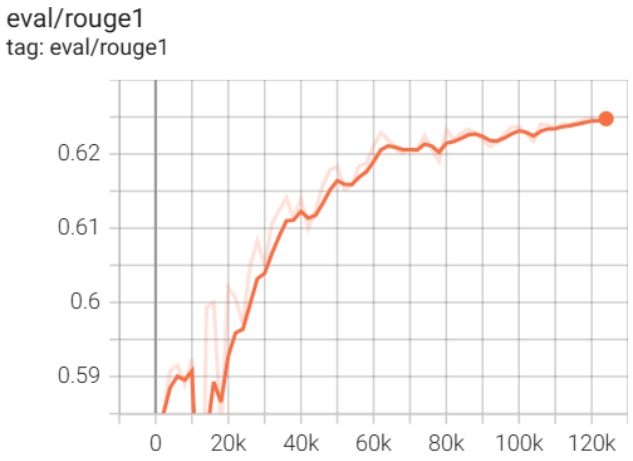
On testing the model, from method 2, consistent performance was observed.

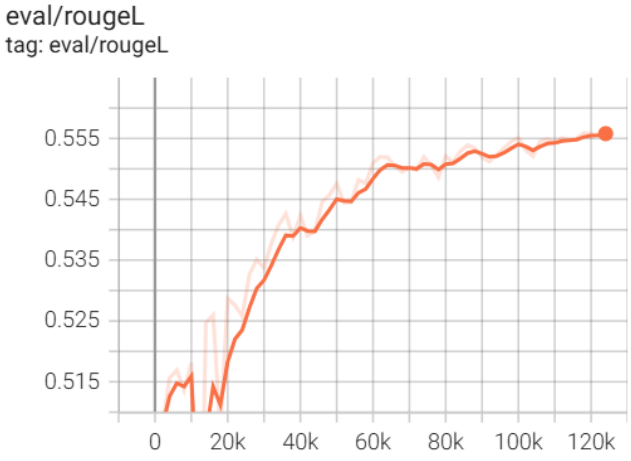
Consider the performance metrics of the models trained by the aforementioned methods.

After the due analysis, the model trained using 'Method 2' was selected for deployment.

METHOD 2 – MODEL EVALUATION

Evaluation for the model trained using Trainer class implementation, source code: src/rouge.ipynb.





Performance metrics – ROUGE (Recall-Oriented Understudy for Gisting Evaluation).

Implemented works:

- Load the validation data.
- feature - load & tokenize & convert to tensor
- Generated summary IDs with specified parameters
- Decoded summary IDs to text and skip special tokens
- Generated summaries for the validation set
- computed rouge metrics based on generated summary and original summary (target)

Results:

Performance Metrics - Before fine-tuning:

ROUGE-1

- **Precision:** 0.3097
- **Recall:** 0.4397
- **F-Measure:** 0.2905

ROUGE-2

- **Precision:** 0.1254
- **Recall:** 0.1778
- **F-Measure:** 0.1161

ROUGE-L

- **Precision:** 0.2009
- **Recall:** 0.2908
- **F-Measure:** 0.1879

ROUGE-Lsum

- **Precision:** 0.2409
- **Recall:** 0.3450
- **F-Measure:** 0.2271

Performance Metrics - After fine-tuning:

ROUGE-1

- **Precision:** 0.6592
- **Recall:** 0.6325
- **F-Measure:** 0.6132

ROUGE-2

- **Precision:** 0.5080
- **Recall:** 0.4969
- **F-Measure:** 0.4787

ROUGE-L

- **Precision:** 0.5735
- **Recall:** 0.5600
- **F-Measure:** 0.5397

ROUGE-Lsum

- **Precision:** 0.6167
- **Recall:** 0.5946
- **F-Measure:** 0.5762

Observations:

The trained model from method 1 was not used for deployment:

Trained model from method 2 was used for deployment

Reason:

- Even though the model has very minimal training loss but, the model performed inconsistently in validation & testing phase.
- There's a suspected tensor error while training using method 1, which could be attributed to the inconsistency of the model's output.
- Model 2 - results outperformed that of method 1.

ROUGE1 (F-Measure) = 61.32 -> Benchmark grade
GPT4 performance for text summarization - ROUGE1 (F-Measure) is 63.22

Comparative Analysis:

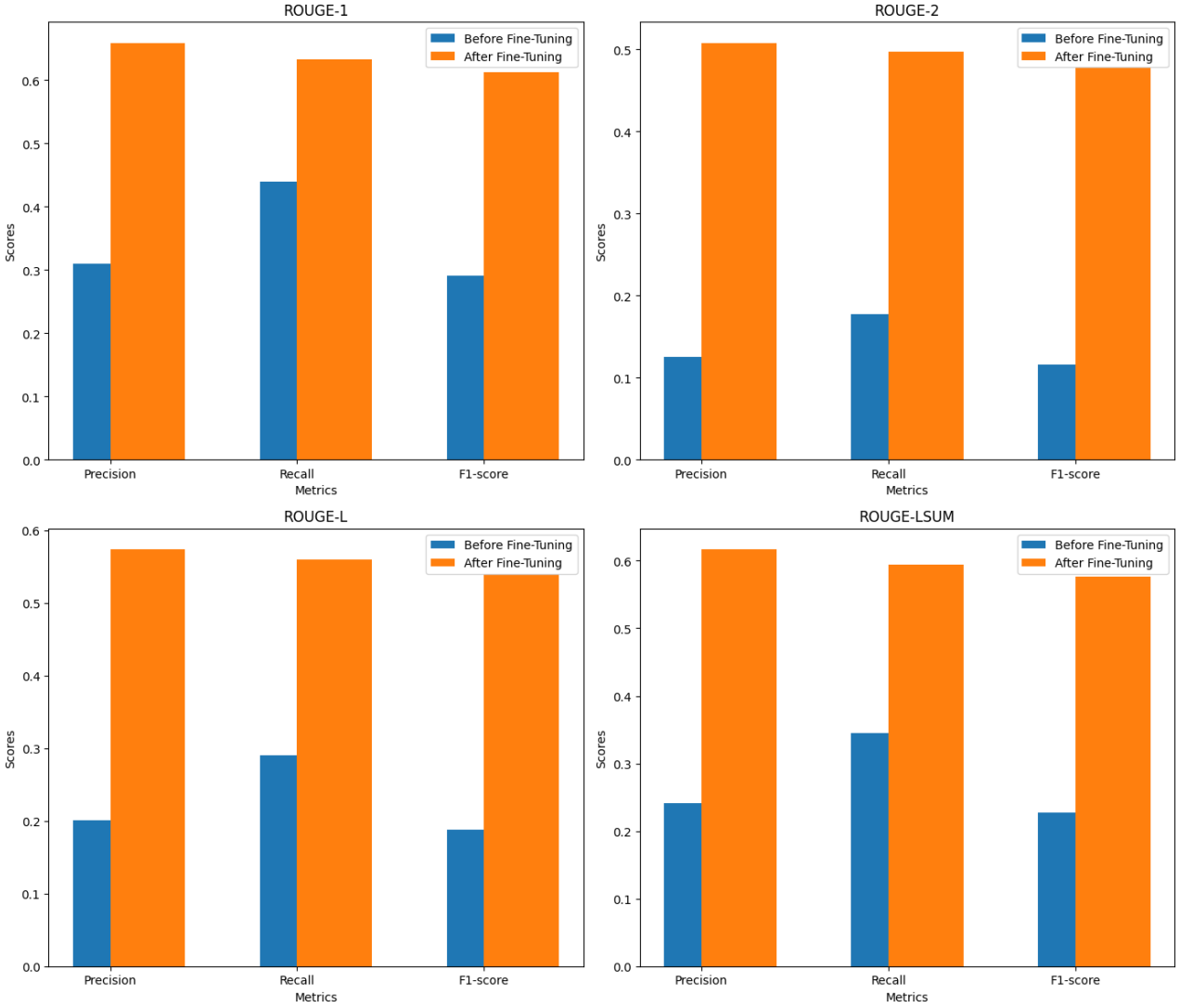


Fig.: Comparison of model's performance metrics.

Performance comparison of the fine-tuned transformer model, source code: src/compare.ipynb.

Implemented works:

- Before fine tuning vs After fine tuning

Basis: ROUGE scores - mid values

Observations:

The fine-tuning process significantly improved the transformer's performance across all ROUGE metrics:

- ROUGE-1 and ROUGE-LSUM show over 100% improvement in F1-score.
- ROUGE-2 demonstrates the highest improvement with over 300% increase in F1-score

- ROUGE-L scores also saw substantial improvements, particularly in Precision and F1-score.

These results indicate that fine-tuning has greatly enhanced the model's ability to generate accurate and relevant summaries.
Hence, the *fine-tuning of the transformer model is successful.*

Extractive Text Summarization:

MODEL

Utilized a rule based approach, source code: [src/Extractive_Summarization.ipynb.]

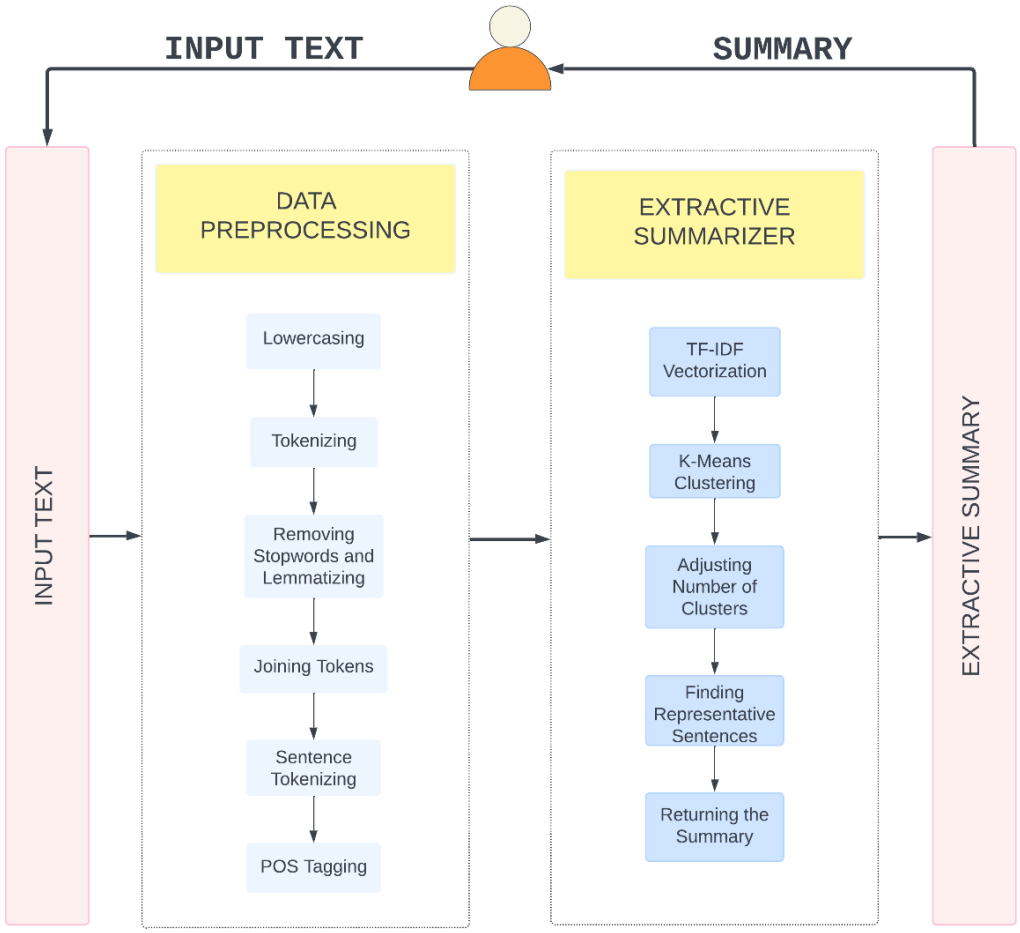


Fig.: Workflow for Extractive Text Summarizer.

The existing ways for extractive summarization: Text Rank, TF-IDF, ML, DL – models.

Rather than choosing computationally intensive deep-learning models, utilizing a rule based approach will result in optimal solutions. Utilized a new-and-novel approach of combining the matrix obtained from TF-IDF and KMeans Clustering methodology.
It is the expanded topic modeling specifically to be applied to multiple lower-level specialized entities (i.e., groups)

embedded in a single document. It operates at the individual document and cluster level.
The sentence closest to the centroid (based on Euclidean distance) is selected as the representative sentence for that cluster.

Implemented works:

- Preprocesses the input text to get POS-tagged sentences.
- Data Preprocessing:

  Lowercasing

  Stop Words Removal.

  - Lemmatization.
  - Tokenization.
  - POS Tagging.

Adjusts the number of clusters if there are fewer sentences than clusters.

Initializes a TF-IDF vectorizer with stop words removed.

Transforms the sentences into a TF-IDF matrix.

- Matrix where each sentence is represented as a vector of TF-IDF scores

TF-IDF matrix is fed into the KMeans clustering algorithm.

- Each sentence is assigned a cluster label.

Identifies the cluster labels for each sentence.

For each cluster:

- Finds the indices of sentences in the cluster.
- Calculates the centroid of the cluster
  .
- Identifies the sentence closest to the centroid (representative sentence).
- Collects the representative sentences from each cluster.
- Joins and returns the representative sentences as a extractive summary.
- Evaluating Summary.

ROUGE scores as performance metrics

Results:

Rule-based approach for extractive summarization was implemented and evaluated successfully.

ROUGE1 (F-Measure) = 24.72.

ROUGE-1

- **Precision:** 0.3356
- **Recall:** 0.2516
- **F-Measure:** 0.2472

ROUGE-2

- **Precision:** 0.1298
- **Recall:** 0.0929
- **F-Measure:** 0.0915

ROUGE-L

- **Precision:** 0.2489
- **Recall:** 0.1786
- **F-Measure:** 0.1769

ROUGE-Lsum

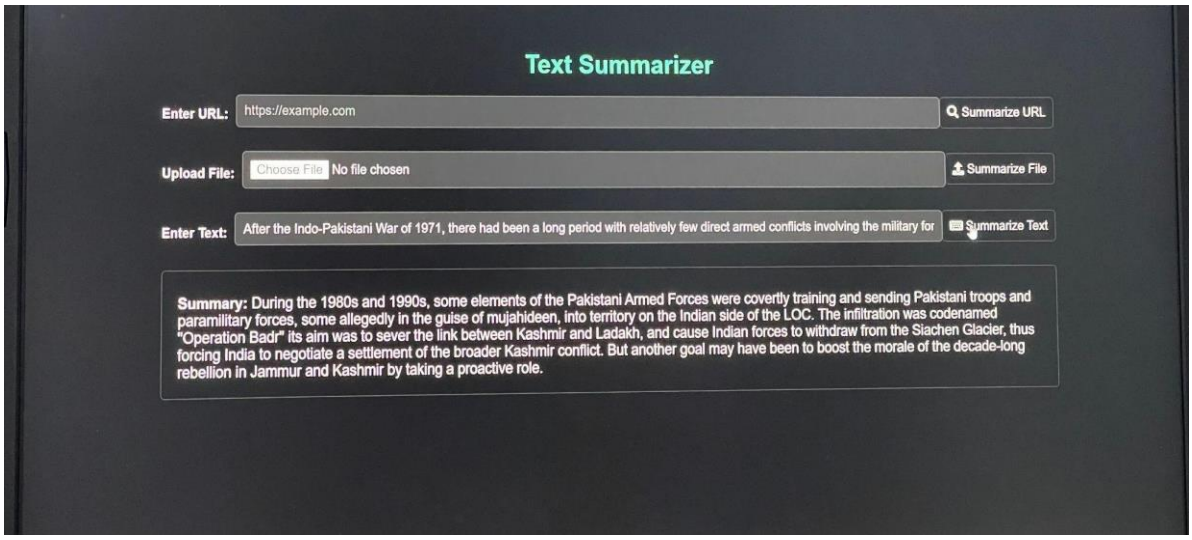
- **Precision:** 0.2489
- **Recall:** 0.1784
- **F-Measure:** 0.1768

Testing:

Implemented works, as follows, source code: src/interface.ipynb.

- text summarization application (Abstractive text summarization).
- using a fine-tuned transformer model (from method 2)Gradio library - for web-based interface
- Utilized for testing abstractive model's inference capabilities in real-time .

:



Deployment:

The summarization device changed into deployed the usage of a FastAPI framework and containerized the use of Docker. The version was hosted on Azure, reaching excessive availability and fast inferencing speeds. The mission also incorporated a CI/CD pipeline and the use of GitHub movements, ensuring seamless updates and scalability.

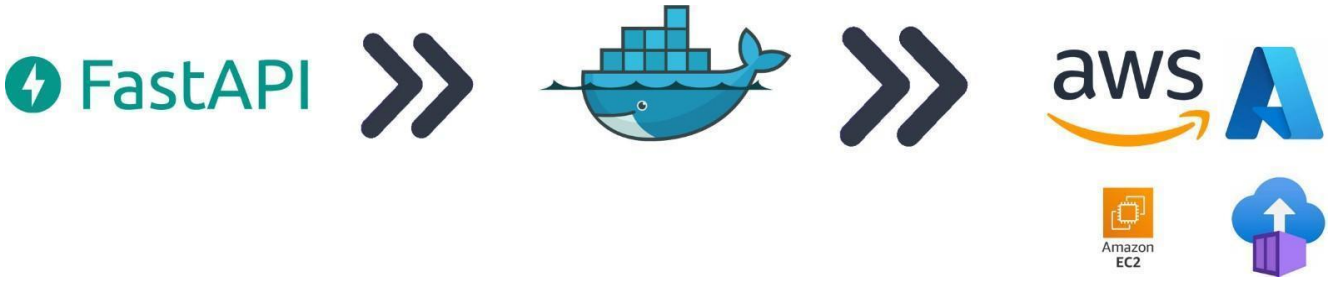


Fig.: Roadmap for deployment.

CONCLUSION:

- 'Text Summarization' Project achieved its goal by developing & deploying a robust 'Text Summarizer'
- Processed and analyzed a combined dataset of over 5,64,522 text samples from various sources.
- Implemented both abstractive and extractive methods, & took deep research on the different ways to optimally solve the sub-tasks.
- Trained the model on the most robust and comprehensive – custom dataset.
- Achieved Benchmark result in abstractive summarization model. (ROUGE 1 = 61.32)
- Showed a new paradigm in extractive summarization. (using TF-IDF and KMeans Clustering)
- Developed the solution with regress check on performance metrics, to ensure quality & standard.
- Created diversified API endpoints, to enable summarization for different text sources.
- By deploying on Azure using Docker and GitHub Actions with a CI/CD pipeline.
- Got 99.9% uptime, ensuring high availability and reliability.
- Accessible via user-friendly interfaces and API endpoints.

FUTURE SCOPE:

- SaaS Platform Development
    - API for Screen and Browser Summarization.
    - Platform Integration
        - Plugins - web browsers and document editors.
    - Domain-Specific Models
        - Training models using specialized datasets for industry-specific summaries.
    - User Customization

Customizable Summaries.

- Adjustable summary length & detail.
    - UI to change 'max_length' in model inferencing (model.generate()).
    - UI to adjust the required number of clusters, in an extractive model.
- Multi-Language Support. (too out of scope)
    - Continuous improvement steps will transform this project into a scalable, user-friendly summarization service. Will play as a value booster to both individual users and businesses.

References:

- N. D. (2016). Automatic Text Summarization Using Supervised Machine Learning Technique for Hindi Language. 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
- 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
- Binwahlan, M. S., Salim, N., and Suanmali, L. (2010). Fuzzy swarm diversity hybrid model for text summarization. Information processing & management, 46(5):571–588.
- Chang, T.-M., Hsiao, W.-F., et al. (2008). A hybrid approach to automatic text summarization. In 2008 8th IEEE International Conference on Computer and Information Technology, pages 65–70. IEEE.
- Chaudhari, M. and Mattukoyya, A. N. (2018). Tone biased mmr text summarization. arXiv preprint arXiv:1802.09426.
- Das, D. and Martins, A. F. (2007). A survey on automatic text summarization. Literature Survey for the Language and Statistics II course at CMU, 4(192-195):57.
- Goldstein, J. and Carbonell, J. (1998). Summarization:(1) using mmr for diversity-based reranking and (2) evaluating summaries. In Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998, pages 181–195.
- He, R., Tang, J., Gong, P., Hu, Q., and Wang, B. (2016). Multi-document summarization via group sparse learning. Information Sciences, 349:12–24.