Information Retrievers and Extractors

ABSTRACT GENERATION FOR RESEARCH PAPERS

Ashmit Chamoli
Ashna Dua
Hardik Sharma

Introduction

The following graph shows the total number of research papers published per year 1.

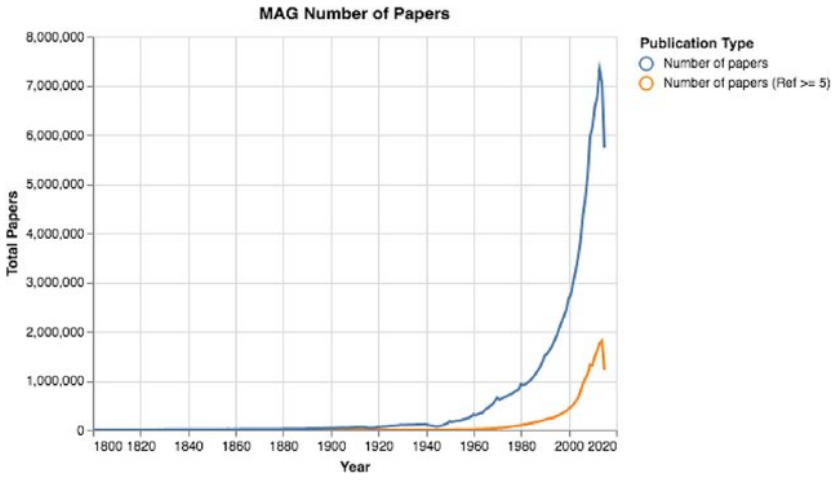

Figure 1: Number of research papers published per year.Source

In today's information-driven world, the ability to efficiently distill and summarize large volumes of text is paramount. This project aims to address the challenge of automatically generating concise and coherent abstracts for research papers.

Problem Statement

With the exponential growth of research papers, the need for automated abstract generation tools has become increasingly evident.

Abstracts serve as condensed representations of the core information contained within a document, enabling readers to quickly grasp the document's main ideas, findings, and contributions. They are essential for information retrieval, as they allow users to decide whether a document is relevant to their needs without having to read it in its entirety.

Project Objective

The primary objective of this project is to develop a robust and accurate system for automatic abstract generation. This system should take as input a full-length textual document and produce a concise and coherent abstract that captures the essential content and meaning of the document.

It will also be the focus of the project to keep the abstract size concise by removing redundant sentences. Keeping in accord with the trend of length of abstracts.

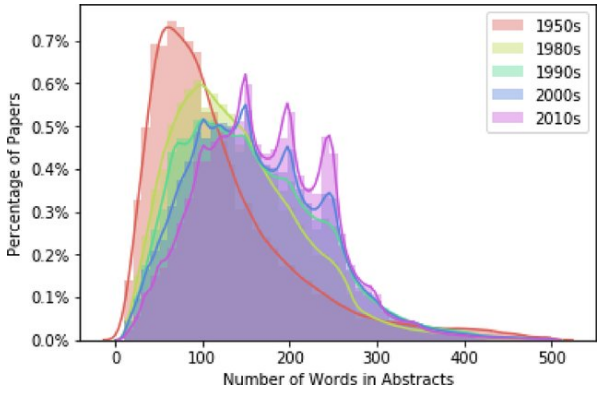

Figure 2: Length of abstracts across research papers of different eras.Source

Relevant Resources

In order to begin creating an Abstract generation model, we turn to the existing literature for inspiration/idea regarding the current state of the art systems.

There are, broadly, two different types of approaches when it comes to generating a summary(an abstract for our use case) for a general document. Namely :

- **Extractive Summary Generation**
  In ESG, sentences or tokens at similar granularity are extracted out based on scores assigned to them based on some retrieval approaches.
- **Abstractive Summary Generation**
  In ASG, some amount of mutation is added to the text at hand in order to generate summary that is more 'human-like' in the sense that phrases not present in the original text may appear in the summary.

We encountered the following resources regarding the same :

1. SumItUp: A Hybrid Single-Document Text Summarizer

The key insights of SumItUp, to effectively generate abstracts are:

- **Hybrid Summarization Approach**: It utilizes a hybrid summarization approach that combines both extractive and abstractive techniques. This means it can extract key sentences from the original research paper and then transform them into a concise and coherent abstract.
- **Semantic and Statistical Features**: The system employs a range of features, including semantic and statistical features, for sentence ranking. These features include sentence length, position, TF-IDF (Term Frequency-Inverse Document Frequency), noun and verb phrases, proper nouns, aggregate cosine similarity, cue-phrases, and emotions.
- **Emotion Analysis**: One unique feature is the integration of emotion analysis. It identifies sentences with implicit emotional content, which can be crucial for conveying the writer's sentiment and the paper's emotional tone. This adds an extra layer of depth to the abstract.

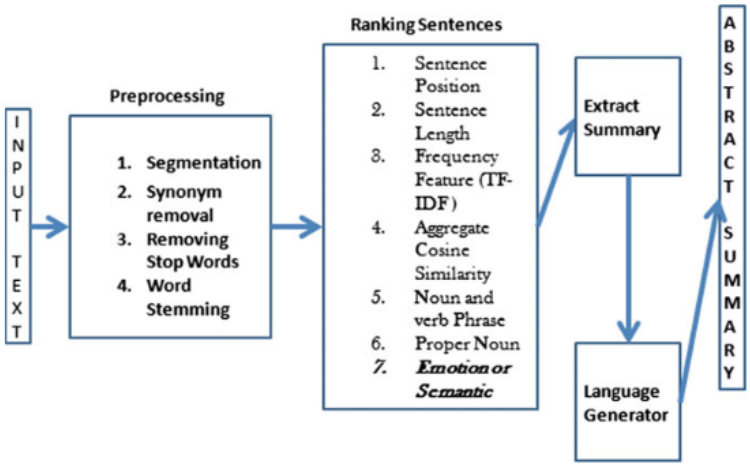

Figure 3: Summarization Pipeline.

- **Normalization**: The system normalizes the values of various features to ensure they fall within a consistent range. Normalization helps in fair ranking and ensures that no single feature dominates the summarization process.

- **Redundancy Removal**: Cosine similarity is used to remove redundancy in the generated summary which ensures that the abstract remains concise and doesn't repeat the same information multiple times.
- **Abstract Summary Generation**: After extracting salient sentences from the research paper and removing redundancy, SumItUp employs a novel language generator. This generator uses WordNet, Lesk algorithm, and POS tagging to transform the extractive summary into an abstractive one, resulting in a well-structured and coherent abstract.

2. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization

In our "Abstract Generation For Research Papers" project, understanding PEGASUS's pre-training strategies offers valuable insights into improving abstractive summarization for research papers.

- **Gap-Sentence Generation Objective**: PEGASUS employs a self-supervised "gap-sentence generation" objective to encourage more abstractive summary generation. This aligns with our goal of creating informative abstracts from research papers by extracting essential content and generating summaries that fill gaps, making our abstracts more abstractive.
- **Transformer-based Encoder-Decoder Architecture**: PEGASUS uses a Transformer-based architecture to capture context and coherence, which is vital in generating coherent abstracts from research papers. This architectural choice emphasizes maintaining logical flow in summarization, aligning with our project's objectives.

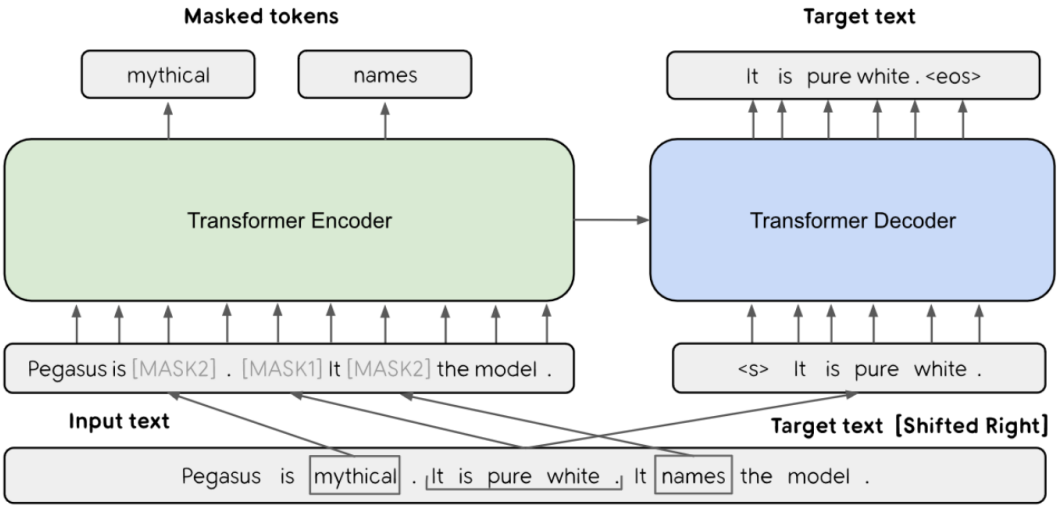

Figure 4: PEGASUS Architucture.

- **Pre-training on Diverse Text Data**: PEGASUS's pre-training on diverse text data enhances its ability to generate abstractive summaries. For our project, this means improving the model's capacity to create contextually accurate abstracts across various research domains.
- **Fine-Tuning for Specific Summarization Tasks**: PEGASUS fine-tunes on summarization tasks, adapting the model for specific contexts. In our project, applying a similar fine-tuning approach ensures that the model excels at generating abstractive summaries for research papers.
- **Performance Excellence**: PEGASUS consistently outperforms previous models in abstractive summarization, even on complex, lengthy documents. This indicates that by incorporating PEGASUS-inspired techniques, we can achieve superior results in abstract generation for research papers, regardless of document complexity.

3. Extractive Summarization as Text Matching

This paper had the following salient features :

- **Advantage of Extractive Summarization over Abstractive Summarization** Extractive summarization generates semantically and grammatically correct sentences an usually computes faster than abstractive methods. Additionally, it better preserves the factual information present in the document.
- **Traditional Approaches** This paper formulates extractive summarization task as a semantic text matching problem. In traditional extractive methods, sentences are scored and extracted one by one from the original document after which the system models the relationship between these sentences and selects several of them to form a summary.

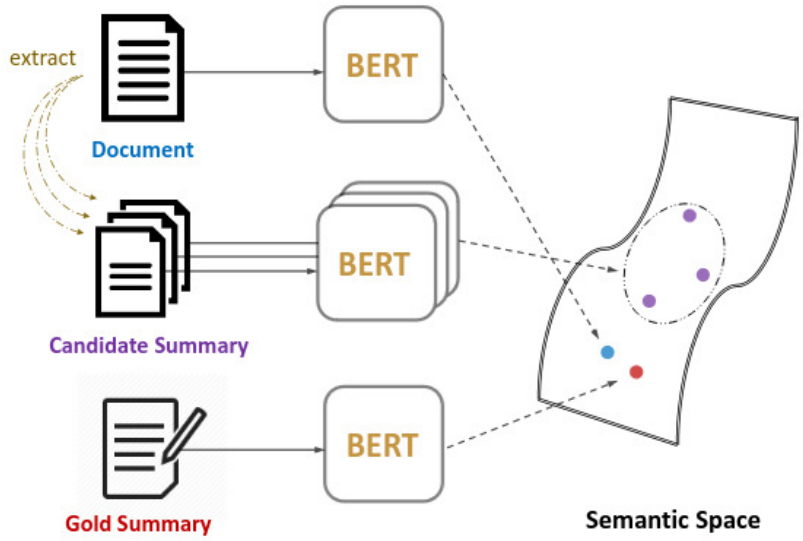

Figure 5: Overview of Extractive Summarization at Summary Level Framework

- **Novel Idea Using Sentence-level framework** This paper introduces a novel summary-level framework which is different from the traditional sentence-level frameworks described above. The main idea is that a good summary should be more semantically similar to the source document than other candidate summaries.
- **Relevance to this project** This paper provides a baseline which is superior to a number of extractive models and obtains state-of-the-art extractive result on CNN/DailyMail by only using the base BERT model.

There are some other resources that we are actively looking at for more inspiration/insights that might aid in implementaion/analysis of the project. Some prominent ones are Review of automatic text summarization and Text Summarization.

Datasets Of Relevance

To perform the training, we essentially need a data-set that contains a research paper and a sample abstract that is written by the authors/experts. We have found the following resources for such data-sets:

Arxiv and PubMed Labelled Research Papers

Both "arxiv" and "pubmed" have three features:

- article: the body of the document, paragraphs separated by "/n".
- abstract: the abstract of the document, paragraphs separated by "/n".
- section_names: titles of sections, separated by "/n".

All the features are *string features*. The data-sets are already partitioned into train, validation and test sets as shown in 6, making it easier to compare the results with the baselines.

name	train	validation	test
arxiv	203037	6436	6440
pubmed	119924	6633	6658

Figure 6: Data Split for both data-sets.

Timeline and Workflow

- **Month 1 – Understanding the Problem and Extensive Literature Review**
  - *Week 1-2: Comprehensive Literature Review*
    - Collate pertinent research papers related to text summarization and abstract generation.
    - Gain in-depth insight into established techniques and tools in the field.
  - *Week 3: Defining the Problem Statement*
    - Precisely outline the project's scope, objectives, and inherent challenges.
  - *Week 4: Data Collection and Baseline Model Establishment*
    - Develop a rudimentary baseline summarization model.
    - Identify and acquire existing datasets relevant to abstract generation for research papers.
- **Month 2 – Refining the Baseline Model and Exploring Advanced Approaches**
  - *Week 5-7: Enhancing the Baseline*
    - Enhance baseline summarization models to achieve improved performance.
    - Document encountered challenges and chart future directions for baseline improvements.
  - *Week 8-9: Advancing the Methodology*
    - Propose an advanced abstract generation method while assessing the shortcomings and metrics of prior baselines.
- **Month 3 – Implementation of Advanced Methodology and Final Reporting**
  - *Week 10-12: Implementation of the Advanced Methodology*
    - Implement the advanced abstract generation method as per the proposed framework.
    - Conduct thorough evaluations and testing, aiming to surpass the performance of the previous baselines.
  - *Week 13: Comprehensive Documentation and Reporting*
    - Prepare detailed reports and documents encapsulating the project's findings, methodologies, and outcomes.

Deliverables

Interim Deliverables

- Project Proposal: A comprehensive project proposal outlining the problem statement, objectives, methodology, and expected outcomes.
- Literature Review: A detailed literature review summarizing relevant research on abstract generation, summarization techniques, and related work in the field of information retrieval.
- Data organization and Preprocessing: An interim report on the progress of data collection, preprocessing, and any challenges encountered in preparing the research paper dataset.
- Baseline Model Development: A prototype of the abstract generation model, demonstrating progress in implementing the chosen methodology and any initial experimentation with different models or algorithms.
- Evaluation Framework: A description of the evaluation metrics and criteria to assess the quality of the generated abstracts. This may include initial baseline performance.
- Progress Report: A status update on the project's overall progress, including any setbacks or adjustments made to the project plan.

Interim Progress Report

All the codes are available in the Github Repository here.

- **Parser**
  We have built a parser, which can take a pdf file as input and convert the contents to a simple text file as the first step of the summarization pipeline.
- **PEGASUS**
  We have a working PEGASUS baseline implemented using transformers that can take a text file and then give back a abstract for the same.
  There are multiple variations of the same, one using the PEGASUS$_{BASE}$ model, and one using the PEGASUS$_{LARGE}$.
- **Scoring**
  We have also built a module to take in two text documents, and compute a variety of scores for them , including *ROUGE-n*, etc.

  For a random research paper from arxiv, we get the following scores.

In [7]:
```
paperName = "2301.03669"

content = Parser(pdfFile = "../papers/" + paperName +".pdf")

pegasusSummary = get_response(content)[0]

print(pegasusSummary)
```

```
We performed small-domain convection-resolving simulations of an Earth-like atmosphere over a wide range of surface
temperatures and found that there's indeed a peak in convective vigor. We also show that a similar peak in convectiv
e vigor exists when relative abundance of water vapor is changed by varying amount of background (non-dilute) gas. T
hese may have implications for Earth's climate and atmospheric chemistry during the Hadean and Archean.
```

In [8]:
```
goldenSummary = ""

with open ( "../papers/" + paperName + "_ABSTRACT.txt") as f :
    goldenSummary = f.read()

score = Score(trueSummary = goldenSummary, predSummary = pegasusSummary)
```

In [9]:
```
for key, value in score.rougeScore().items():
    print(f"Criteria:{key}, Score:{value}" )
```

```
Criteria:rouge1, Score:Score(precision=0.9726027397260274, recall=0.27307692307692305, fmeasure=0.4264264264264264)
Criteria:rouge2, Score:Score(precision=0.8611111111111112, recall=0.23938223938223938, fmeasure=0.37462235649546827)
Criteria:rougeL, Score:Score(precision=0.9315068493150684, recall=0.26153846153846155, fmeasure=0.4084084084084085)
```

Figure 7: Summarization for A random research paper

For the PEGASUS paper itself, we obtain the following scores.

```
In [4]: paperName = "pegasus"

        content = Parser(pdfFile = "../papers/" + paperName +".pdf")

        pegasusSummary = get_response(content)[0]

        print(pegasusSummary)

        A Transformer-based pre-training model for abstractive text summarization has been developed. In PEGASUS, important
        sentences are removed/masked from an input doc-ument and are generated together as one output sequence from the rema
        ining sentences, similar to an abstract summary. We evaluated our model on 12 downstream summariza- tion tasks spann
        ing news, science, stories, emails, patents, and legislative bills.

In [5]: goldenSummary = ""

        with open ( "../papers/" + paperName + "_ABSTRACT.txt") as f :
            goldenSummary = f.read()

        score = Score(trueSummary = goldenSummary, predSummary = pegasusSummary)

In [6]: for key, value in score.rougeScore().items():
            print(f"Criteria:{key}, Score:{value}" )

        Criteria:rouge1, Score:Score(precision=0.9672131147540983, recall=0.30256410256410254, fmeasure=0.4609375)
        Criteria:rouge2, Score:Score(precision=0.8166666666666667, recall=0.25257731958762886, fmeasure=0.3858267716535433)
        Criteria:rougeL, Score:Score(precision=0.9016393442622951, recall=0.28205128205128205, fmeasure=0.4296875)
```

Figure 8: Summarization for the PEGASUS Paper

For another paper that we used as input, the scores were low, highlighting scope for improvement.

```
In [10]: paperName = "2310.10737"

         content = Parser(pdfFile = "../papers/" + paperName +".pdf")

         pegasusSummary = get_response(content)[0]

         print(pegasusSummary)

         A large, flexible class of long, high-redundancy error correcting codes, GRAND, can be efficiently decoded with gues
         sing random additive noise dec oding (GRAND). Grand is a recently developed family of code-agnostic decoding algorit
         hms that can accurately decode long, high- redundancy codes. It can be used to decode long, high-redundancy codes wh
         ile LDPC codes are used to decode long, high-redundancy codes.

In [11]: goldenSummary = ""

         with open ( "../papers/" + paperName + "_ABSTRACT.txt") as f :
             goldenSummary = f.read()

         score = Score(trueSummary = goldenSummary, predSummary = pegasusSummary)

In [12]: for key, value in score.rougeScore().items():
             print(f"Criteria:{key}, Score:{value}" )

         Criteria:rouge1, Score:Score(precision=0.703125, recall=0.17928286852589642, fmeasure=0.2857142857142857)
         Criteria:rouge2, Score:Score(precision=0.2857142857142857, recall=0.072, fmeasure=0.11501597444089456)
         Criteria:rougeL, Score:Score(precision=0.515625, recall=0.13147410358565736, fmeasure=0.2095238095238095)
```

Figure 9: Summarization for A random research paper

- **MatchSum**

  Code Base

  The codes for this are available in the *MatchSum* branch of the Github repository. The folder `MatchSum/data/` contains the papers and their abstracts. The main notebook is present inside the folder `MatchSum/SentenceTransformers/`. The code is well documented and can be understood by going through the notebook.

  Explanation and Approach

  For the first part of this pipeline, we have implemented a baseline to measure the semantic similarity between 2 documents.

  – We have used an mpnet-base model by microsoft which is fine-tuned on 1B sentence pairs.

  – The dataset is formed by concatenating sentences from several datasets. For the training, a contrastive learning objective was used: given a sentence from the pair, the model should

predict which out of a set of randomly sampled other sentences, was actually paired with it in the dataset.

- It works very well on the arxiv dataset and is able to give high similarity between the research paper content and the abstract while giving low similarity with other texts.

For the second part of this pipeline, we need to generate candidate summaries and select the summary with the highest similarity to the document. Before this, we need to have a sentence level summarizer to select the most relevant sentences from the research paper content and this model will also serve as a separate baseline as well.
We also propose a new variation to this MatchSum pipeline which uses n-grams to select sentences. The idea is to generate 1-grams, 2-grams, ... upto k-grams from the sentences of the original document. After this, we will rank all the n-grams according to the semantic similarity with the source document. Then we will select the top scoring n-grams until we have the desired summary length. Then we can use this generated extractive summary to further improve upon or treat is as an individual summary. At this point, we have the generated n-grams, ranked according to their semantic similarity score. Once we have an extractive sentence-level summarizer, we will be in a position to compare the results of our 3 pipelines.

- **SumItUp** This baseline, was implemented, which makes use of both Extractive and Abstractive Summarization to provide a relevant abstract for the research paper.
  - **Stage 1: Preprocessing** In this stage, 4 major steps were performed to preprocess and clean the data. They are: Segmentation (Tokenizing sentences), removing synonyms, removing stop words and word stemming using the Potrter Stemmer algorithm.
  - **Stage 2: Ranking** In this stage, each sentences is scored using 7 different measures (instead of 8 as suggested by the research paper).
    - **Sentence Length:** Number of words in every sentence.
    - **Sentence Position:** The position of the sentence in the document is calculated using the equation:

$$\text{Sentence Position} = 1 - \frac{S_i - 1}{N} \quad (1 < S_i < N) \tag{1}$$

where:

N is the total number of sentences in the document,

S_i is the sentence number of the i^{th} sentence.

    - **Frequency (TF-IDF):** The TF-IDF score is calculated using the equation:

$$TF_i \times IDF_i = f(w) \times \log\left(\frac{bg}{bg(w)}\right) \tag{2}$$

where:

TF_i is the term frequency of the i^{th} word in the document,

IDF_i is the inverse document frequency of the i^{th} word in the document,

$f(w)$ is the frequency count of the i^{th} word in the document,

bg is the total number of background documents taken,

$bg(w)$ is the number of background documents that contain the i^{th} word.

    - **Noun Phrase and Verb Phrase:** NLTK tagger was to tag the input text. The tagging process is used assign the various parts-of-speech like de- terminers (DT), verbs (VBZ), adverbs (ADVB), nouns (NN), adjectives (JJ), coordinating conjunction (CC), etc. to each word in the input text. Tagger also assigns the respective labels to the syntactically correlated expressions as adjective phrase (AP), verb phrase (VB), Noun phrase (NP), etc. After tagging each sentence is allotted a numeric variable "nvp" whose value is the count of noun and verb phrases contained by it.

* **Proper Noun:** The number of proper nouns in a sentence is maintained.
* **Aggregate Cosine Similarity:** Aggregate cosine similarity for a given sentence is the addition of its cosine similarities with other all sentences in input text.

$$\text{sim}(S_i, S_j) = \sum_{k=1}^{n} W_{ik} \cdot W_{jk} \tag{3}$$

$$\text{Aggregate Cosine Similarity}(s_i) = \sum_{\substack{j=1 \\ j \neq i}}^{n} \text{sim}(S_i, S_j) \tag{4}$$

* **Cue Phrases:** The sentences that begin with phrases like "the paper describes", "in conclusion" "in summary", "our investigation", and emphasizes such as "the best", "the most important", "in" "particular", "according to the study", "significantly", "important", "hardly", "impossible" were used as a cue-list to identify the cue-phrases in a given sentence. This score was assigned as the number of cue phrases contained by a sentence.
* **Emotion:** As suggested by the research paper, seven emotion class labels (positive, negative, fear, joy, surprise, hate and disgust) were advised. But in the context of research papers, emotions hold no significance. Hence, this feature was not used while calculating the scores.

– **Step 3: Normalizing** In this stage, the scores were normalized, according to individual normalization rules. These normalized scores were added for every sentence using the rule:

$$\text{Total Score} = \sum_{n=1}^{7} F_n \tag{5}$$

The sentences were sorted in the basis of the *Total Score*, and the sentence having the highest total score, is the most important sentence.

– **Step 4: Cosine Similarity to remove Redundancy** A threshold value of 0.15 was used to calculate the cosine similarity of the highest ranked sentence and rest of the sentences. All the sentences with their cosine similarity greater that the threshold are used for Abstractive Summary in the next step.

– **Step 5: Abstract Summary** The words are lemmatized, to create an easily readable summary. A similarity matrix is built, using the TF-IDF values of the sentences and the sumamry is created using PageRank algorithm.

This algorithm generates a preliminary summary that may not meet the gold standard. To improve its performance, hyper-parameter tuning and feature adjustments are necessary. As a promising enhancement, integrating BERT or other pre-trained models for summary generation on top of this baseline can significantly elevate the quality of generated summaries.

Final Deliverables

- Final Project Report: A comprehensive report summarizing the entire project, including problem statement, methodology, data collection and preprocessing, model development, evaluation results, and conclusions.
- Abstract Generation System: The fully developed abstract generation system, which should include the model, codebase, and any necessary documentation for deployment.
- Research Paper Dataset: The curated and preprocessed research paper dataset used in the project, ready for future use or sharing with the research community.
- Evaluation Results: Detailed results of the model's performance on the research paper dataset, including metrics like ROUGE, BLEU, and any custom evaluation criteria developed during the project.

- User Interface (Possibly): A user-friendly interface or tool for users to input research papers and receive generated abstracts.
- Documentation: Comprehensive documentation on how to use the abstract generation system, including setup instructions, dependencies, and best practices.
- Presentation: A presentation summarizing the project's objectives, methodology, key findings, and implications. This can be used for project defense or presentation to stakeholders.
- Future Recommendations: Suggestions for future work and improvements to the abstract generation system, including potential areas for further research.
- Code Repository: A well-organized code repository containing the project's source code, version control history, and any additional resources or scripts used in the project.