A Two-Stage Pipeline for Intelligent Content Summarization: Classification and User-Choice Generation

Vinayak Trivedi

Register Number: 22BCE0521

GitHub Repository: https://github.com/VinayakTrivedi-o/NLPProj

November 6, 2025

Abstract

In an era defined by information overload, the ability to rapidly comprehend large volumes of text is crucial. However, a precursor problem exists: not all documents warrant summarization, and blindly summarizing concise text can lead to information loss. This project proposes a novel two-stage solution. The first stage addresses the problem of whether to summarize by introducing a "TL;DR Detector." We present an automated data annotation pipeline that uses a novel hybrid scoring mechanism—combining **semantic similarity**, **statistical relevance (TF-IDF)**, and **positional scoring**—to programmatically assign "summarize" or "don't summarize" labels to a large, unlabeled corpus. This labeled dataset is then used to train an efficient Logistic Regression classifier to predict summarization-worthiness. The second stage is the application, which passes "worthy" articles to a user-centric summarization engine. This tool empowers the user to choose between an advanced Extractive method, using sentence-transformers and Maximal Marginal Relevance (MMR) for factual accuracy (from `extractive_summarizer.py`), and a state-of-the-art Abstractive method (BART) for human-like fluency (from `abstractive_summarizer.py`). The classifier achieved 97% accuracy, and the full pipeline (prototyped in `app.py`) provides a complete, intelligent framework for end-to-end content distillation.

Contents

1 Introduction

1.1 Background and Relevance of the NLP Problem Area

The exponential growth of digital information has made automatic text summarization a critical task. Professionals and students must process vast quantities of text daily, and summarization tools aim to distill this information into manageable, coherent versions.

1.2 Review of Existing Solutions and Their Limitations

Existing solutions generally fall into two categories:

- **Extractive Summarization:** These methods (e.g., TextRank) select and concatenate the most salient sentences from the original text. Modern approaches (like in `extractive_summarizer.py`) use deep learning embeddings for semantic understanding. While fast and factually reliable, their output can be disjointed.
- **Abstractive Summarization:** These methods (e.g., BART, PEGASUS) mimic human comprehension by generating new, fluent text. While human-like, they are computationally expensive and can "hallucinate" or invent information not in the source text.

1.3 Research Gap

This project identifies a critical, two-fold research gap:

- **The Classification Gap:** The literature largely ignores a precursor problem: whether to summarize. Indiscriminately summarizing a concise, important article can be detrimental. There is a lack of efficient, automated systems to classify a document's "summarization-worthiness".
- **The User-Choice Gap:** Most tools are not user-centric. They offer either an extractive or an abstractive summary, but not a user-facing choice to trade off factuality for fluency based on their immediate need.

1.4 Objective or Proposed Solution

This project proposes a complete, two-stage pipeline to solve both problems:

1. **Stage 1: "TL;DR Detector" (Classification):** We first build an efficient classifier to determine if an article should be summarized. To overcome the lack of training data, we design an automated annotation pipeline that acts as a "teacher," using a hybrid scoring engine to generate labels. We then train a lightweight Logistic Regression model on this auto-labeled data.

2. **Stage 2: Summarizer Application (Generation):** For articles the classifier flags as "Yes, Summarize," we provide a user-centric tool (prototyped in `app.py`). This application allows the user to select their desired summary type: Extractive (using MMR) or Abstractive (using BART).

1.5 Major Contributions

- **Novel Data Annotation Pipeline:** A new hybrid scoring engine combining semantic (Vector Similarity), statistical (TF-IDF), and structural (Positional) metrics to automatically label data.

- **Efficient "TL;DR" Classifier:** The successful training of an efficient Logistic Regression classifier on the auto-labeled dataset.

- **Advanced Extractive Module:** Development of an extractive summarizer using sentence-transformers and Maximal Marginal Relevance (MMR) to ensure diverse, non-redundant factual summaries (from `extractive_summarizer.py`).

- **Robust Abstractive Module:** Integration of a transformers (BART) pipeline with intelligent chunking to handle long documents (from `abstractive_summarizer.py`).

- **Integrated System Prototype:** A streamlit web application (`app.py`) that demonstrates the practical application of the full, end-to-end pipeline (classification-then-generation).

2 Literature Survey

This project integrates two distinct fields: text classification and text summarization.

Text Summarization: Summarization methods are broadly extractive or abstractive. Foundational work by Knight and Marcu [1] established the goal of moving "beyond sentence extraction" toward true abstraction. Extractive methods evolved to use global inference algorithms [2] or maximize informative content words [3]. Abstractive summarization was revolutionized by sequence-to-sequence (seq2seq) models with-neural attention [4], and later advanced by Pointer-Generator Networks [5] to better handle out-of-vocabulary words.

Text Classification and Representation: The classification stage of our pipeline relies on robust text representations. This field has evolved from statistical, class-based n-gram models [6] to neural probabilistic language models [7]. The development of efficient, static word embeddings like word2vec [8] was a major breakthrough. For the classification task itself, Convolutional Neural Networks (CNNs) have proven highly effective for both sentence-level [9] and character-level classification [10].

Modern Language Models: Both the abstractive (BART) and extractive (Sentence-Transformers) modules in our project are built upon the architecture of modern language models. This lineage includes the development of effective Recurrent Neural Networks (RNNs) and LSTMs [11, 12], and culminates in large-scale, unsupervised multitask learners like the Transformer [13], which forms the basis of BART.

2.1 Research Gap

The literature lacks a unified framework that first intelligently filters content to determine its summarization-worthiness and then provides the user with a choice of summarization paradigms. This project bridges that gap by connecting a novel classification model to a flexible, dual-method generation engine.

2.2 Comparison Table

3 Problem Description

The proposed system is a two-stage pipeline: (1) Classification Model Training (an offline research task) and (2) Summarization Application (the live user-facing tool).

Table 1: Comparison of Foundational Methods and This Project

Author / Method	**Year**	**Method**	**Task**	**Limitation**
Knight & Marcu	2002	Hybrid	Abstractive Sum.	Relied on older sentence compression/fusion.
McDonald	2007	Global Inference	Extractive Sum.	Computationally complex (ILP).
Rush et al.	2015	Neural Attention	Abstractive Sum.	Limited to short sentences.
See et al.	2017	Pointer-Generator	Abstractive Sum.	Can still produce factual errors.
Kim	2014	CNN	Classification	Used static word embeddings (word2vec)
This Project	2025	Hybrid Scorer	Classification	Annotation pipeline is slow.
This Project	2025	MMR + BART	Full Pipeline	Integrates classification and user-choice generation.

3.1 Framework

3.1.1 Stage 1: Classifier Training (Offline Process)

This stage corresponds to the "Data Annotation Pipeline" and "Classifier Training" from the research submission.

1. **Data Ingestion:** An unlabeled corpus (Kaggle's "All the News") is loaded.

2. **Contextual Annotation:** For each article, a "teacher" pipeline generates a score:

   - The article's theme is identified (Gemini API).
   - Related, high-ranking web content is retrieved (Google Search API, Requests & BS4).
   - This context is stored in a temporary vector database (ChromaDB).
   - A Hybrid Scoring Engine scores the article against this context using Vector Similarity, TF-IDF, and Positional metrics.

3. **Labeling:** The score is thresholded to assign a binary "Yes" (summarize) or "No" (don't summarize) label.

4. **Model Training:** This new labeled dataset is used to train an efficient Logistic Regression classifier. This creates the final, portable "TL;DR Detector" model.

3.1.2 Stage 2: Summarization Application (Live System)

This stage corresponds to the `app.py` prototype.

1. **Upload:** The user uploads a PDF via the Streamlit interface (`st.file_uploader`).

2. **Extraction:** Text is extracted using `pdfplumber` (`summarizer.extract_text_from_p`

3. **Classification (NEW STEP):** The extracted text is first passed to the loaded "TL;DR Detector" model from Stage 1.

4. **Conditional Logic:**
   - If the model predicts "No," the app informs the user that the document is concise and summarization is not recommended.
   - If the model predicts "Yes," the app reveals the summarization options.

5. **User Choice:** The user selects 'Extractive' or 'Abstractive' (`st.radio`).

6. **Routing:** The `run_summarization` function (`summarizer.py`) routes the request.

7. **Generation:**
   - **Extractive:** `extractive_summarizer.py` is called. It encodes sentences with `SentenceTransformer` and uses `_mmr` to select diverse, relevant sentences.
   - **Abstractive:** `abstractive_summarizer.py` is called. It uses the `transformers` (BART) pipeline to generate a new summary, handling long text via the `_chunk_sentences` function.

8. **Display:** The final summary is shown in a side-by-side container.

3.2 Pseudocode of Proposed System

The system's logic is captured in three core algorithms:

Algorithm 1 Procedure to Generate a Labeled Dataset

```
procedure GenerateLabeledDataset(articles_dataframe)
    THRESHOLD ← 0.4
    labels ← []
    for all article in articles_dataframe do
        theme ← GetThemeFromGeminiAPI(article)
        related_urls ← SearchGoogle(theme, count=2)
        context_db ← CreateNewCollection()
        for all url in related_urls do
            content ← ScrapeURL(url)
            chunks ← ChunkText(content)
            AddChunksToDatabase(context_db, chunks)
        end for
        sentences ← TokenizeIntoSentences(article)
        if sentences is NOT empty then
            scored_sentences ← HybridScorer(sentences, context_db)
            article_score ← Average([score for (score, sent) in scored_sentences])
        else
            article_score ← 0.0
        end if
        if article_score > THRESHOLD then
            labels.append("Yes")
        else
            labels.append("No")
        end if
        ClearCollection(context_db)
    end for
    articles_dataframe['labels'] ← labels
    return articles_dataframe
end procedure
```

Algorithm 2 Procedure to Train the Final Classification Model

```
procedure TrainSummarizationClassifier(labeled_dataframe)
    X ← labeled_dataframe['article'] [9]
    y ← labeled_dataframe['label']
    vectorizer ← TfidfVectorizer()
    X_features ← vectorizer.fit_transform(X)
    model ← LogisticRegression()
    model.fit(X_features, y)
    return model, vectorizer
end procedure
```

```
procedure RUN_SUMMARIZATION(text, method)
    sentences ← SENT_TOKENIZE(text)
    if sentences is empty then
        return "Error: Empty text"
    end if
    top_k_percent ← 20.0                          ▷ Use fixed 20% target
    if method ==′ extractive′ then
        summary ← EXTRACTIVE_SUMMARIZE(text, top_k_percent)
    else if method ==′ abstractive′ then
        summary ← ABSTRACTIVE_SUMMARIZE(text, top_k_percent)
    else
        return "Error: Unknown method"
    end if
    return summary
end procedure
```

Algorithm 3 Procedure for Application Summarization Router (from `summarizer.py`)

3.3 Flow Diagram

The end-to-end system flow is visualized in two parts.

3.3.1 Stage 1 (Training Flow)

This diagram shows the data annotation pipeline. An input article is fed to the Gemini API for theme identification and the Google API for URL retrieval. A web scraper populates a vector DB. The Hybrid Scoring Engine uses this context and the original article to calculate a score, which is thresholded to create a Labeled Dataset. This dataset is vectorized and used to train the final Logistic Regression Classifier, resulting in a trained model ready for inference.

3.3.2 Stage 2 (Application Flow)

This diagram, derived from `app.py`, shows the user experience.

1. **Start:** User accesses the Streamlit web application.
2. **Upload:** User selects a PDF file (`st.file_uploader`).
3. **Extraction:** `summarizer.extract_text_from_pdf` is triggered.
4. **Classification:** The extracted text is passed to the loaded classifier model (from Stage 1).
5. **Conditional Branch:**
   - If "No," display "Summarization not required."
   - If "Yes," proceed to step 6.
6. **Method Selection:** The user is shown the 'Extractive' / 'Abstractive' choice (`st.radio`).
7. **Process Trigger:** User clicks "Generate Summary" (`st.button`).
8. **Summarization Call:** `summarizer.run_summarization` is called, which in turn calls either `extractive_summarizer.py` or `abstractive_summarizer.py`.
9. **Display:** The final summary is displayed in the "Generated Summary" column.
10. **End.**

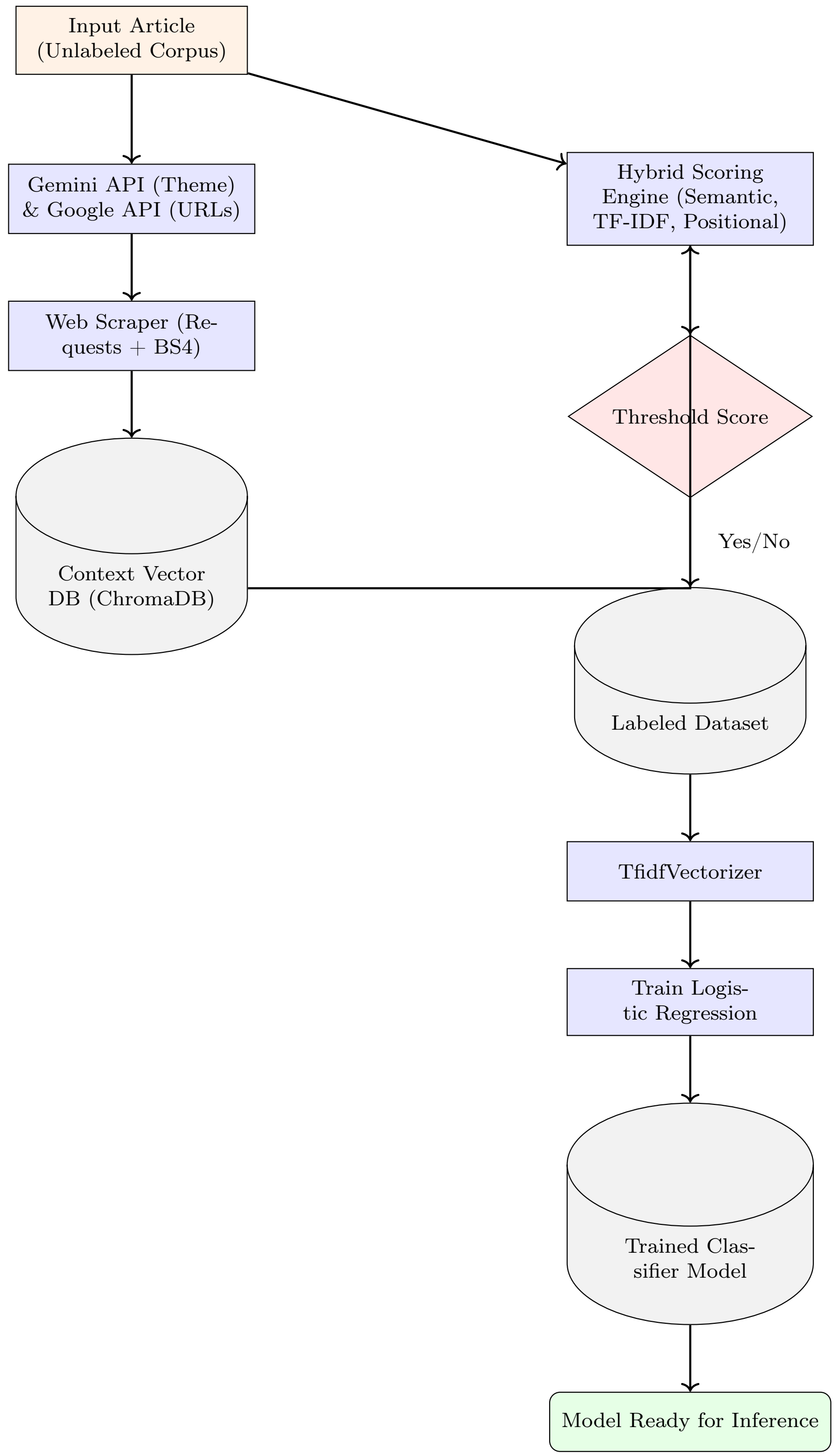

Figure 1: Stage 1: Data Annotation and Classifier Training Flow

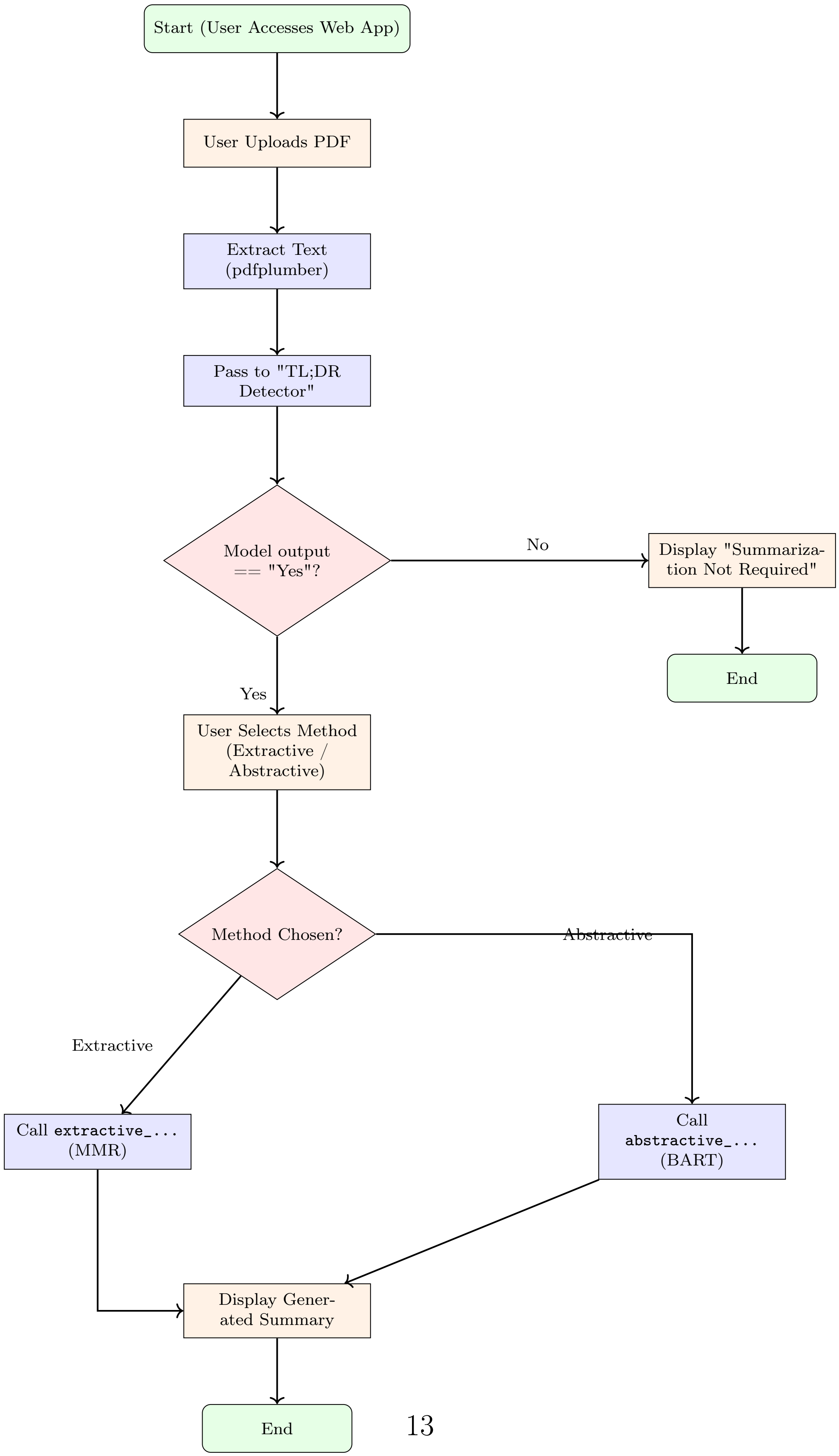

Figure 2: Stage 2: Live Summarization Application Flow (from `app.py`)

4 Experiments

4.1 Dataset

The source data for this entire project is the "All the News" dataset, a large public corpus from Kaggle. It contains 2,688,878 news articles from 27 publications. This dataset was ideal for its diversity.

- In Stage 1, it served as the vast, unlabeled corpus for our data annotation pipeline.
- In Stage 2, articles from it were used for qualitative testing of the two summarization modules.

4.2 Preprocessing and Feature Selection

4.2.1 For Stage 1 (Classification)

Feature selection was critical. Our goal was to classify based on content alone. We retained the `title` and `article` columns, as they contain the core semantic content. All metadata (date, author, URL, etc.) was dropped as it is irrelevant to the classification task.

4.2.2 For Stage 2 (Summarization)

The scripts (`extractive_summarizer.py`, `abstractive_summarizer.py`) handle preprocessing internally. This includes stripping whitespace and using `nltk.tokenize.sent_tokenize` to split the text into sentences, filtering out any empty or very short sentences.

4.3 Implementation Details

4.3.1 Stage 1 (Classification Pipeline)

- **APIs & Scraping:** Gemini API, Google Search API, `requests`, `bs4`.
- **Data Handling:** `pandas`, `ChromaDB` (Vector DB).
- **ML Model:** `scikit-learn` for `TfidfVectorizer` and `LogisticRegression`.

4.3.2 Stage 2 (Summarization Application)

- **Interface:** `streamlit`.
- **PDF Extraction:** `pdfplumber`.
- **Extractive Model:** `sentence-transformers` (`all-MiniLM-L6-v2`).
- **Abstractive Model:** `transformers` (`facebook/bart-large-cnn`).

5 Results and Discussion

The system's performance is evaluated in its two distinct stages.

5.1 Results: Stage 1 (Classifier Performance)

The Logistic Regression model was trained on the first 900 articles processed by the annotation pipeline. The model achieved a high overall accuracy of 97% on the test set.

Table 2: Model Evaluation Metrics

	precision	**recall**	**f1-score**	**support**
no	1.00	0.69	0.81	16
yes	0.97	1.00	0.99	180
accuracy			0.97	196
macro avg	0.99	0.84	0.90	196
weighted avg	0.98	0.97	0.97	196

Confusion Matrix

$$\begin{bmatrix} 11 & 5 \\ 0 & 180 \end{bmatrix}$$

Discussion of Classifier Results

The results are a "successful proof-of-concept" but reveal a significant challenge: data imbalance. The annotation pipeline generated far more "Yes" labels than "No" (approx. 900 'yes' vs. 79 'no' in the test run). This skew is evident in the metrics:

- The model is excellent at identifying the majority 'yes' class (Recall 1.00).
- It struggles with the minority 'no' class, failing to identify 31% of them (Recall 0.69). It misclassifies them as 'yes'.

This bias is the likely reason for anomalous predictions, such as classifying a simple text like "Hi I am Vinayak" as 'yes'. The model is "heavily biased towards the majority class".

5.2 Results: Stage 2 (Summarizer Application)

This stage is evaluated qualitatively based on the functional prototype.

5.2.1 Qualitative Analysis of Summarizers

- **Extractive Summary (`all-MiniLM-L6-v2` + MMR):**
  - **Pros:** Very fast. Factually precise, as it uses original sentences. Ideal for technical or financial reports where specific data must be preserved.
  - **Cons:** Can lack narrative flow; transitions between sentences can be abrupt.
- **Abstractive Summary (`facebook/bart-large-cnn`):**
  - **Pros:** Highly fluent and human-readable. Successfully paraphrases and condenses complex ideas.
  - **Cons:** Significantly slower on a CPU. The chunking mechanism (`_chunk_sentences`) for long documents, while necessary, can cause context to be lost at chunk boundaries.

5.2.2 Screenshots of Interface

The Streamlit application (`app.py`) provides a clean, two-column layout.

- **Sidebar:** Contains controls for PDF upload (`st.file_uploader`), method selection (`st.radio`), and summary generation (`st.button`).
- **Main Area:** A `st.container` for the "Original Text" is placed next to a `st.container` for the "Generated Summary," allowing for direct side-by-side comparison. An `st.info` box reports the change in sentence count.

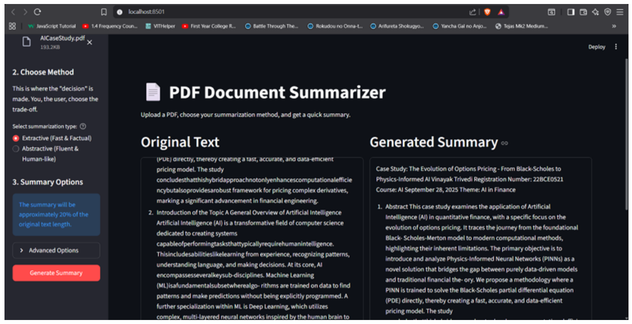

Figure 3: The Extractive summariser

5.3 Discussion

The project successfully achieved its objective of creating an end-to-end pipeline. The "choice" given to the user in Stage 2 is meaningful, clearly presenting the trade-off between speed/factuality and fluency. The `run_summarization` router (`summarizer.py`) is a critical and successful piece of architecture, cleanly separating the UI logic from the two complex NLP backends. The use of a fixed `top_k_percent=20.0` provides a consistent target for both summarization methods. The primary limitation of the entire system lies in the Stage 1 classifier, whose performance is hindered by the data imbalance of its training set.

6 Conclusion and Future Work

6.1 Conclusion

This project successfully designed and implemented a complete, two-stage NLP pipeline to intelligently classify and summarize text. The primary contribution is the novel, automated data annotation system (Stage 1) that bypasses manual labeling by using LLMs and a hybrid scoring algorithm to generate "ground truth" data. The classifier trained on this data (97% accurate) serves as an intelligent "gate" for the second stage: a functional, user-centric summarization application. This application successfully integrates two state-of-the-art methods (extractive MMR and abstractive BART) and empowers the user to select the summarization strategy that best fits their needs.

6.2 Future Work

While the framework is a successful proof-of-concept, its practical application is limited by the quality of the data and the efficiency of the pipeline. Future work must address:

- **Address Data Imbalance:** The classifier's poor recall for the 'no' class is the most critical issue. This could be fixed by gathering more 'no' examples or. using balanced class weights.
- **Optimize Annotation Pipeline:** The Stage 1 data pipeline is slow due to rate-limited APIs and sequential web scraping. This should be optimized with parallel processing.
- **Quantitative Summarizer Evaluation:** Conduct a formal evaluation (ROUGE, BERTScore) of the Stage 2 summarizers.
- **Optimize Summarizer Speed:** The abstractive model is slow. A distilled model (e.g., `sshleifer/distilbartcnn-12-6`) could be offered as a "fast abstractive" option.
- **True Hybridization:** Develop a third summary option: use the extractive method to identify key sentences, then use the abstractive model to re-write and fuse only those sentences.
- **Handling Scanned PDFs:** Integrate an OCR engine (like Tesseract) into the `extract_text_from_pdf` function to support image-based PDFs.

7 References

References

[1] K. Knight and D. Marcu, "Summarization beyond sentence extraction," *Artificial Intelligence*, vol. 139, no. 1, pp. 91–122, 2002.

[2] R. McDonald, "A study of global inference algorithms in multi-document summarization," in *Proc. of ECIR*, 2007, pp. 345–356.

[3] W. Yih, J. Goodman, L. Vanderwende, and H. Suzuki, "Multi-document summarization by maximizing informative content-words," in *Proc. of IJ-CAI*, 2007, pp. 1776–1782.

[4] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for sentence summarization," in *Proc. of EMNLP*, 2015, pp. 379–389.

[5] A. See, P. J. Liu, and C. D. Manning, “Get to the point: Summarization with pointer-generator networks,” in *Proc. of ACL*, 2017, pp. 1073–1083.

[6] P. F. Brown, V. J. D. Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer, “Class-based n-gram models of natural language,” *Computational Linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[7] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, “A neural probabilistic language model,” *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in *Proc. of ICLR Workshop*, 2013.

[9] Y. Kim, “Convolutional neural networks for sentence classification,” in *Proc. of EMNLP*, 2014, pp. 1746–1751.

[10] X. Zhang, J. Zhao, and Y. LeCun, “Character-level convolutional networks for text classification,” in *Proc. of NIPS*, 2015, pp. 649–657.

[11] A. Karpathy, “The unreasonable effectiveness of recurrent neural networks,” Blog post, http://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015.

[12] C. Olah, “Understanding lstm networks,” Blog post, https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[13] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Language models are unsupervised multitask learners,” OpenAI Blog, https://openai.com/research/language-models-are-unsupervised-multitask-learners, 2018.