Archives available at journals.mriindia.com

International Journal on Advanced Computer Theory and Engineering

ISSN: 2319-2526
Volume 14 Issue 01, 2025

SmartNews: AI-Powered News Summarizer

Mrs. Abha Pathak[1], Trupti Pawar[2], Yogeshwari Pawar[3], Shreya pawar[4]
[1-3]U.G. Student, Computer Engineering. Dr. D. Y. Patil College of Engineering and Innovation, Pune
[4]Assistant Professor, Computer Engineering, Dr. D. Y. Patil College of Engineering and Innovation, Pune

Peer Review Information

Submission: 21 Feb 2025
Revision: 25 March 2025
Acceptance: 30 April 2025

Keywords

Extractive
Abstractive Summarization

Abstract

In today's digital era, the rapid influx of online news content poses a significant challenge for users seeking concise and relevant information. SmartNews: AI-Powered News Summarizer is a system designed to address this challenge by leveraging Natural Language Processing (NLP) and Machine Learning (ML) techniques to automatically generate concise summaries of lengthy news articles. The system utilizes Python- based libraries such as NLTK, spaCy, and Transformers for text preprocessing, keyword extraction, and semantic understanding. By implementing both extractive and abstractive summarization models, SmartNews delivers accurate and contextually meaningful summaries, enabling users to grasp essential news content quickly.

INTRODUCTION

The digital age has brought with it an explosion in the amount of information available to users. News articles from online media sources continue to grow at a rapid pace, leading to information overload. For users to remain informed without investing excessive time in reading full articles, automatic text summarization becomes essential. Summarization not only improves readability and user experience but also supports applications like news aggregation, personalized news feeds, and academic research.

Text summarization is a task in Natural Language Processing (NLP) aimed at generating a shorter version of a document while preserving its key information. With Python's rich ecosystem of NLP and machine learning libraries, developing summarization models has become accessible and practical.

Objective

- To investigate and compare extractive and abstractive summarization techniques.
- To implement and evaluate summarization models using Python.
- To assess the performance of various models using benchmark datasets and evaluation metrics.
- To identify the most efficient summarization approach for real-world news content.

LITERATURE REVIEW

Several studies have explored both extractive and abstractive summarization techniques, leveraging traditional statistical models and more recent deep learning methods. Early approaches, such as those using TF-IDF and sentence scoring, laid the groundwork for extractive summarization by focusing on identifying important keywords and sentence positions. Techniques like TextRank, based on the PageRank algorithm, further improved extractive summarization by modeling sentence similarity as a graph.

- **BERTSUM -** An adaptation of BERT for extractive summarization, introducing segment embeddings and classification layers to identify salient sentences.
- **BART -** Combines bidirectional encoding with autoregressive decoding, pre-trained on a denoising objective, making it highly effective for summarization.
- **T5 -** Reformulates all NLP tasks into a text-to-text format, allowing it to be flexibly used for summarization and achieving state-of-the-art results.

These models have been widely evaluated on benchmark datasets like CNN/Daily Mail and XSum, with results showing significant improvements in fluency and informativeness compared to earlier methods. Despite these advancements, challenges such as hallucination in generated summaries and high computational costs remain areas of active research

METHODOLOGY

This study follows a comprehensive methodology that includes data acquisition, preprocessing, implementation of summarization techniques, and performance evaluation. The goal is to develop and compare both extractive and abstractive models using Python-based tools and libraries.

Data Collection

We source data from three prominent datasets commonly used in summarization research:

- **CNN/Daily Mail**: A standard dataset containing news articles paired with human-written summaries. It is widely used for training and benchmarking summarization models.
- **Newsroom**: A diverse dataset collected from 38 major news publications, representing various writing styles and summarization strategies.
- **BBC News Articles**: A manually curated dataset that provides balanced coverage across multiple domains such as politics, business, sports, and entertainment.

Preprocessing

Effective summarization requires clean and well-formatted textual data. The preprocessing steps involve:

- **Text Normalization**: Lowercasing, removing non-ASCII characters, punctuation, and digits.
- **Tokenization**: Splitting text into sentences and words using libraries like NLTK or spaCy.
- **Stopword Removal**: Eliminating common words that do not contribute significant meaning.
- **Lemmatization/Stemming**: Reducing words to their base forms.
- **Length Filtering**: Removing articles that are too short or too long to maintain consistency.

Preprocessing ensures that the input fed into summarization models is noise-free and representative of the intended semantics.

Implementation of Summarization Techniques

We explore two broad categories of summarization:

a) **Extractive Summarization**

Implemented using the TextRank algorithm via the Gensim library. This method constructs a similarity graph among sentences and uses PageRank to rank them.

```
from gensim.summarization import
summarize summary = summarize(text,
ratio=0.2)
```

It is simple to use and provides decent results for well-structured articles. However, it lacks the ability to rephrase content or combine insights across sentences.

b) **Abstractive Summarization**

Implemented using transformer-based models such as **BART** and **T5** through the Hugging Face Transformers library. These models are capable of generating fluent and coherent summaries by understanding the semantics of the input.

```
from transformers import pipeline
summarizer =
pipeline("summarization")
summary = summarizer(text, max_length=150, min_length=40, do_sample=False)
```

These models use encoder-decoder architectures where the encoder processes the input and the decoder generates a condensed
version. Fine-tuning on domain-specific datasets further enhances the accuracy.

Evaluation Metrics

To objectively assess the quality of generated summaries, we use the following metrics:

- **ROUGE-1**: Measures overlap of unigrams (words).
- **ROUGE-2**: Measures overlap of bigrams (two-word sequences).
- **ROUGE-L**: Measures the longest common subsequence.

These metrics provide a standardized way to compare extractive and abstractive summaries against human-written references.

Tools and Frameworks

Key tools used in this study include:

- Python (v3.8+)
- NLTK, spaCy, Gensim for NLP tasks
- Transformers, datasets from Hugging Face for modeling
- Scikit-learn, NumPy, Matplotlib for evaluation and visualization

This pipeline ensures a robust and reproducible experimentation framework for summarization tasks.

The methodology of this study involves implementing extractive and abstractive summarization techniques in Python and comparing their performance on benchmark datasets.

Data Collection

We utilize three widely used datasets for evaluating summarization performance:

- **CNN/Daily Mail**: A popular dataset for training and testing summarization models.
- **Newsroom**: A large collection of news articles and summaries.
- **BBC News Articles**: A curated set of articles covering various topics for experimentation.

Preprocessing

Text preprocessing includes the following steps:

- Tokenization
- Stopword removal

- Lowercasing
- Lemmatization

These tasks are performed using libraries like NLTK, spaCy, and **re** for regex-based cleaning.

Summarization Techniques

Extractive Summarization (TextRank using Gensim):

```
from gensim.summarization import
summarize summary = summarize(text,
ratio=0.2)
```

TextRank is an unsupervised graph-based ranking algorithm inspired by PageRank, where sentences are represented as nodes and their similarity as edges.

Abstractive Summarization (T5, BART using Hugging Face):

```
from transformers import pipeline
summarizer =
pipeline("summarization")
summary = summarizer(text, max_length=150, min_length=40, do_sample=False)
```

These transformer models use encoder-decoder structures to understand and generate human-like summaries. They are pretrained on large corpora and can be fine-tuned for domain-specific summarization tasks.

RESULTS AND FINDINGS

The experimental evaluation of summarization models was conducted on the three selected datasets using both extractive and abstractive methods. We analyzed the outputs in terms of accuracy, fluency, and computational efficiency using established ROUGE metrics, which compare generated summaries with human-written references.

Performance Evaluation

We computed ROUGE-1, ROUGE-2, and ROUGE-L scores for each model on a representative subset of each dataset. The results are summarized below:

Model	**ROUGE-1**	ROUGE- 2	**ROUGE- L**
TextRank	0.41	0.18	0.38
BART	0.52	0.28	0.49

T5 (fine- tuned)	0.56	0.30	0.53

These results indicate that transformer-based abstractive models outperform extractive techniques in capturing both factual content and linguistic coherence. T5, particularly when fine-tuned, produced the most contextually appropriate and fluent summaries.

Qualitative Assessment

Upon qualitative review, extractive summaries generated using TextRank often included disjointed sentences that lacked logical flow. While these models preserved key facts, they failed to paraphrase or interpret nuanced content. Conversely, abstractive models such as BART and T5 successfully generated summaries that were not only shorter but also more cohesive and natural sounding.

For instance, abstractive models were able to:

- Eliminate redundancy in the source text.
- Use synonyms and rephrasing effectively.
- Maintain grammatical correctness and context continuity.

Computational Considerations

Extractive models required significantly less computational power and training time. TextRank, being unsupervised, was fast and suitable for real-time summarization tasks. On the other hand, transformer models required access to GPU acceleration and considerable memory, especially during fine-tuning.

Approximate inference time (per article):

- TextRank: < 1 second
- BART: ~3 seconds
- T5: ~4.5 seconds

These differences are crucial when considering the deployment environment and latency requirements of a summarization application.

Summary of Findings

- **Effectiveness**: Abstractive models outperform extractive ones in ROUGE scores and readability.
- **Efficiency**: Extractive models are faster and resource-efficient.

Flexibility: Transformer models offer better domain adaptability and language generalization.

- **Deployment**: The choice of model depends on the application's constraints—TextRank for lightweight use cases, BART or T5 for quality-driven scenarios.

We conducted experiments on all three datasets and evaluated the summarization quality using ROUGE (Recall-Oriented
Understudy for Gisting Evaluation) scores.

Model	**ROUGE**	**ROUGE- 2**	**ROUGE- L**
TextRank	0.41	0.18	0.38
BART	0.52	0.28	0.49
T5 (fine- tuned)	0.56	0.30	0.53

Extractive models were found to be quick and simple but lacked coherence and fluency. In contrast, abstractive models, especially fine-tuned T5, produced summaries that were more readable and contextually accurate..

DISCUSSION

Interpretation of Results and Findings:

The results demonstrate that transformer-based models like BART and T5 significantly outperform traditional extractive models in terms of coherence, informativeness, and fluency. This superiority is attributed to the transformer architecture's ability to capture long-term dependencies and generate paraphrased content. Fine-tuning further improves the output quality by aligning the model more closely with domain-specific language patterns.

However, extractive methods like TextRank remain valuable for their simplicity, speed, and resource efficiency, especially in environments where computational power is limited or real-time results are essential. While these methods may not generate the most natural-sounding summaries, they reliably extract key points from the text.

Implications for Students

The research offers valuable insights for students interested in natural language processing and machine learning:

- Students can use extractive models to understand the basics of text representation, graph theory, and unsupervised learning.
- Working with transformer-based models introduces students to advanced deep learning concepts, including encoder- decoder architectures and transfer learning.
- The use of open-source Python libraries like Hugging Face, Gensim, and NLTK ensures that students can reproduce and extend the experiments with minimal setup.
- Understanding trade-offs between performance and resource requirements helps build practical AI applications suited to specific use cases.

This study also encourages interdisciplinary learning, combining skills from programming, data science, linguistics, and statistics.

Limitations and Challenges

Despite the promising results, several limitations were encountered:

- **Computational Resources**: Transformer models require significant memory and GPU acceleration for training and inference, which may not be accessible to all users.
- **Dataset Bias**: The summarization quality is influenced by the biases present in the training data. Models fine-tuned on specific datasets may not generalize well to different domains.
- **Evaluation Metrics**: ROUGE scores, while standard, do not always capture summary quality from a human perspective. More nuanced metrics or human evaluations are needed.
- **Hallucinations**: Abstractive models occasionally generate content that is not factually present in the original text, posing a risk in high-stakes applications like legal or medical summaries.

Addressing these limitations requires continued research in model interpretability, dataset diversity, and hybrid summarization strategies that blend extractive and abstractive methods effectively.

While extractive summarization is computationally efficient and easier to implement, it often results in disjointed summaries that lack narrative flow. Abstractive summarization, on the other hand, provides coherent and human-like summaries but demands higher computational resources and fine-tuning efforts.

Pretrained models like BART and T5 demonstrated superior performance. The trade-off between performance and computational efficiency must be considered depending on the application. For real-time summarization, lightweight extractive methods may be preferred. For offline or batch processing, deep learning models offer higher quality.

CONCLUSION

Key Points and Contributions

This study investigated and implemented both extractive and abstractive summarization techniques for news articles using machine learning in Python. We compared traditional models like TextRank with advanced transformer-based models such as BART and T5. The results clearly showed that transformer models significantly outperform extractive methods in producing

coherent, fluent, and informative summaries.
Key contributions of the study include:

- A comprehensive implementation pipeline for news summarization in Python.
- A detailed performance evaluation across multiple datasets using ROUGE metrics.
- Comparative insights into computational efficiency and summarization quality.
- A resource for students and developers seeking practical exposure to NLP and deep learning techniques.

Future Directions and Enhancement

To build upon this research, several directions can be pursued:

- **Model Compression**: Development of more efficient transformer architectures that can operate effectively on limited hardware, enabling mobile and edge deployment.
- **Multilingual Summarization**: Extending the models to support multiple languages and code-switching texts.
- **Interactive Summarization Tools**: Integration into real-time applications such as digital assistants, chatbots, and content management systems.
- **Hybrid Models**: Combining extractive and abstractive techniques to strike a balance between speed and quality.
- **Human Evaluation**: Including qualitative feedback from human reviewers to better understand user satisfaction and summary usability.

Limitations and Challenges (Revisited)

Though abstractive models have shown superior performance, they still face several issues that need addressing:

- **Model Interpretability**: Deep models often act as black boxes, making it hard to trace decisions.
- **Factual Consistency**: Sometimes generated summaries contain hallucinated facts not present in the original content.
- **Domain Adaptation**: Models may underperform on out-of-domain data unless properly fine-tuned.

References

1. See, A., Liu, P. J., & Manning, C. D. (2017). Get to the Point: Summarization with Pointer-Generator Networks. *ACL*
2. Lewis, M. et al. (2020). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *ACL*
3. Raffel, C. et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *JMLR*
4. Mihalcea, R., & Tarau, P. (2004). TextRank: Bringing Order into Texts. *EMNLP*