Summarizing Financial News for potential investor use with Natural Language Processing

SMITKUMAR B. SARAIYA, University of Calgary, Canada

1 Introduction

This project aims to explore the topic of financial text summarization and then developing a sample use case product with the resulting output data. The project makes use of natural language processing libraries and techniques in python and finally a website built with Next.js to display summarized text. The focus is on financial articles from websites that distribute financial information via articles.

In non-financial domains, visual aids designed with choice architecture principles help convey complex quantitative information and reduce decision biases [4]. To visualize this idea, a website was developed to show the intended use for the output from the results of this project's experimentation. Before this, we need to explore the process and methodology applied in acquiring the summarized text from articles.

2 Material and Methods

When thinking about the use cases of the output from this project, the vision was to implement it in such a way that it could be a plug and play script. The relevant libraries used for this project are *pandas*, *requests*, *BeautifulSoup*4, *sumy*, *transformers*, and *rouge_score*.

The *requests*, *pandas* and *BeautifulSoup*4 libraries were use in scraping financial news articles from moneycontrol.com [7] and marketbeat.com [8]. By making a request to the web pages we receive the initial HTML structure of the page with articles on the pages. Thereafter, each article's information, specifically the title, link and the content, is extracted and stored in a *pandas* dataframe from which we can save csv files. Since all article content was extracted without limits, some came with promotional material that could skew the content included in the final summary.

In addition to this kind of unwanted content, other unwanted content that was included was anything such as "MarketBeat keeps track of", "Our team has identified", "Enter your email address", "Get stock market alerts:Sign Up" and "Catch all the market action on our live blog".
The phrases were removed by finding the index of that phrase and getting all text before that index by slicing an array that contains the scraped article content. By doing this step we now have only the relevant article content. Now we can summarize the article content which has been cleaned of any unwanted phrases or strings.

Author's Contact Information: Smitkumar B. Saraiya, smitkumar.saraiya@ucalgary.ca, University of Calgary, Canada.
Manuscript submitted to ACM

Fig. 1. Promotional content scraped along with the main content

The main natural language processing library utilized for text summarization for this project was the *sumy* [6] library from which all the 4 models were used for text summarization from *sumy.summarizers* namely *LsaSummarizer*, *LexRankSummarizer*, *TextRankSummarizer*, and *LuhnSummarizer*.
In addition to this, the *PlaintextParser* from *sumy.parsers.plaintext* was used to parse the extracted article content to pass to each of these summarizers in order to get summarized output. Additionally, the *Tokenizer* from the *sumy.nlp* library was used to tokenize the text in order to be able to parse the text.
Each of the summarizers were configured to summarize the entire parsed article into either 1 or 2 sentences which were then evaluated using ROUGE scores discussed below in the results section.

Finally, the FinBERT [3] was used to perform sentiment analysis on the summary selected depending on ROUGE scores. The values were returned using the a piped connection to the FinBERT model that would take the text as input and return a dictionary mapped with *label* and *score* or the sentiment percentage and the sentiment it assigned.

3 Results

After conducting the above mentioned experiment, I looked to learn which summarizer performed the best under the same conditions. These conditions being, the input data parsed by the previously mentioned *PlaintextParser* and specifying the length of the output measured in number of sentences. Below you can see the experimentation setup for each of the summarizers implemented in callable methods.

```
def lexrank_summary(df=combined_df):
    for i in range(len(df)):
```

```
        c = df.iloc[i]['content']
        parser = PlaintextParser.from_string(c, Tokenizer("english"))
        summary = lex_rank(parser.document, sentences_count=1)
        summarized = ' '.join([str(sent) for sent in summary])
        df.loc[i, 'summary_lexrank'] = summarized

def luhn_summary(df=combined_df):
    for i in range(len(df)):
        c = df.iloc[i]['content']
        parser = PlaintextParser.from_string(c, Tokenizer("english"))
        summary = luhn(parser.document, sentences_count=1)
        summarized = ' '.join([str(sent) for sent in summary])
        df.loc[i, 'summary_luhn'] = summarized

def lsa_summary(df=combined_df):
    for i in range(len(df)):
        c = df.iloc[i]['content']
        parser = PlaintextParser.from_string(c, Tokenizer("english"))
        summary = lsa(parser.document, sentences_count=1)
        summarized = ' '.join([str(sent) for sent in summary])
        df.loc[i, 'summary_lsa'] = summarized

def textrank_summary(df=combined_df):
    for i in range(len(df)):
        c = df.iloc[i]['content']
        parser = PlaintextParser.from_string(c, Tokenizer("english"))
        summary = text_rank(parser.document, sentences_count=1)
        summarized = ' '.join([str(sent) for sent in summary])
        df.loc[i, 'summary_textrank'] = summarized
```

Listing 1. Extractive summarizers using Sumy

Figure 2 shows the calculated average ROUGE scores when the summarizers were configured to generate a summary of sentence length 1. From the diagrams, we can see that the best performing summarizers are the *TextRankSummarizer* and the *LuhnSummarizer* were the best performing models across the board when producing summaries with a sentence length of only 1 sentence. This means they were able to capture better information within one sentence from a whole article of content which could be many sentences long.

Figure 3 shows the average ROUGE scores calculated when the summarizers were configured to generate a summary of sentence length 2. This visualization also shows the same result of the

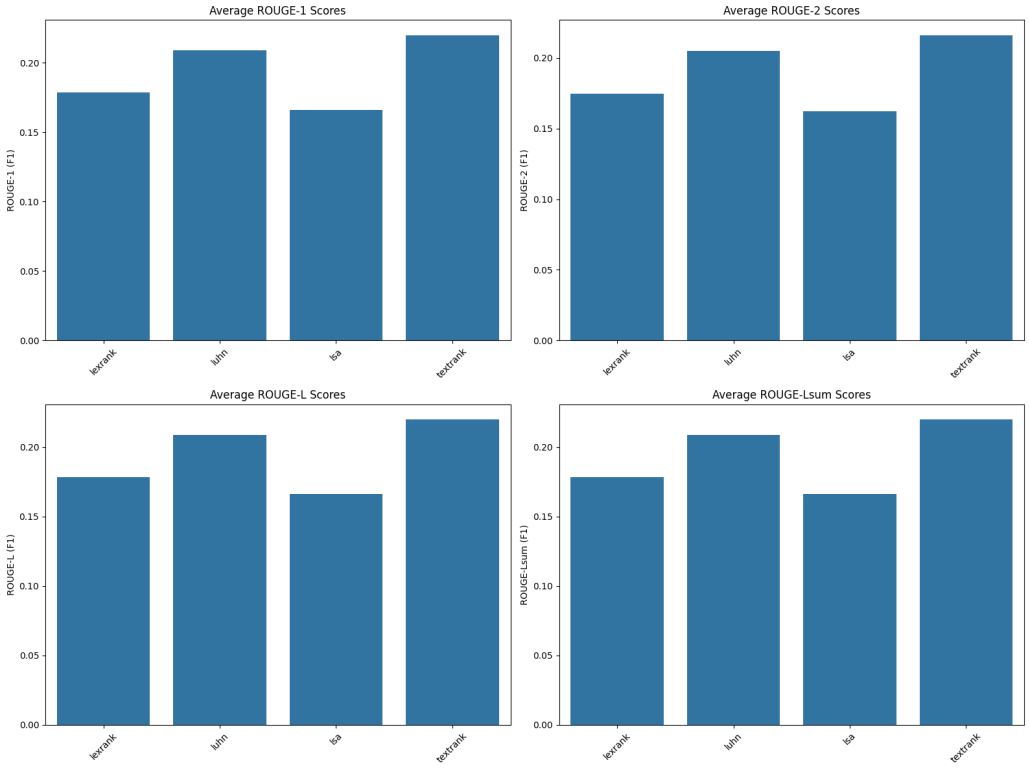


Fig. 2. ROUGE scores with summarized sentence length (sentences_count) set to 1

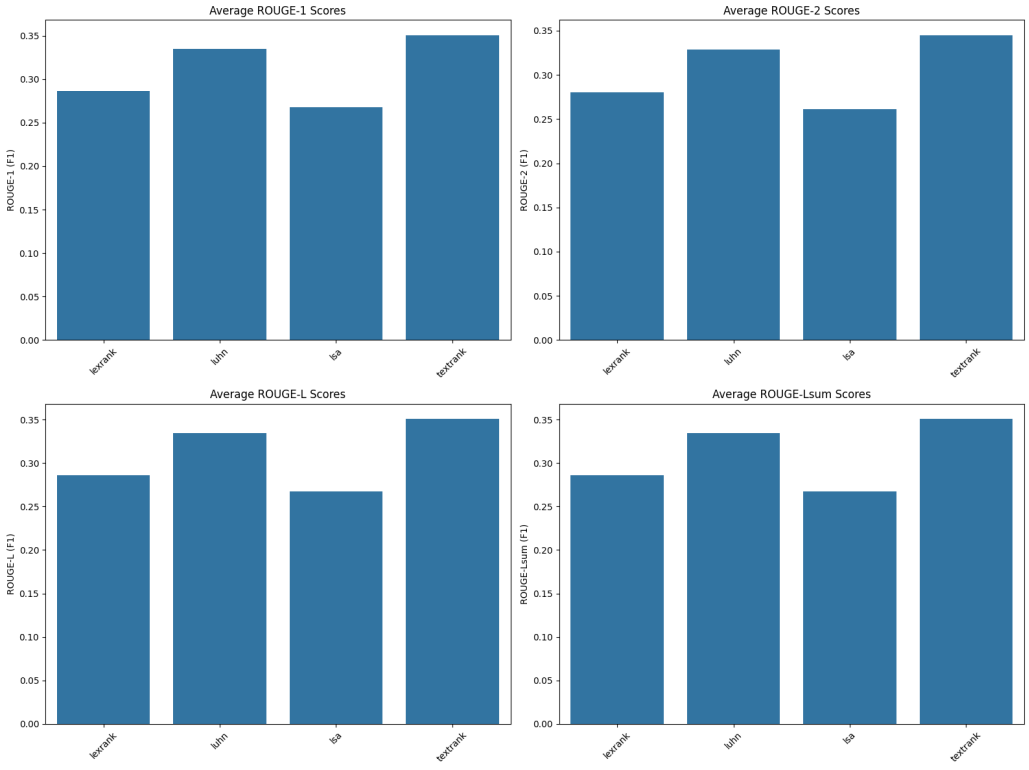


Fig. 3. ROUGE scores with summarized sentence length (sentences_count) set to 2

TextRankSummarizer and the *LuhnSummarizer* being the best summarizing models across the board. However, one factor to note is that the overall order of best summarizer stayed the same.

Therefore, the assumption can be made that as you increase the sentence_count parameter for the summarizers their respective ROUGE scores will increase as well. With the current setup, we could set the sentences_count to be higher and get higher ROUGE scores dependent on the final application use of the summaries. However, one could argue that setting the sentences_count higher could lead to a sentences_count=N where N is the actual length on the article in terms of number of sentence length of the article. Figures 4 and 5 below show use an example of this scenario.

The highest plotted point with ROUGE score 1.0 across all calculated ROUGE scores tells us that for that specific article, it had a sentence length of exactly $sentences\_count = 1$ or $sentences\_count = 2$. Therefore, resulting in a total capture of information of the article content by each of the summarizers.

Therefore, if the longest article had some sentence length θ and sentences_count = θ, then we would see all the ROUGE scores equal 1 meaning that all information from the source text was captured in the generated summary by the summarizers.

Finally, the ROUGE score distributions in figures 4 and 5 show us what the whiskers and inter quartile rangers widen a bit for 2-sentence summaries, especially for the *TextRankSummarizer*, implying greater dispersion in performance across documents when you allow longer summaries.

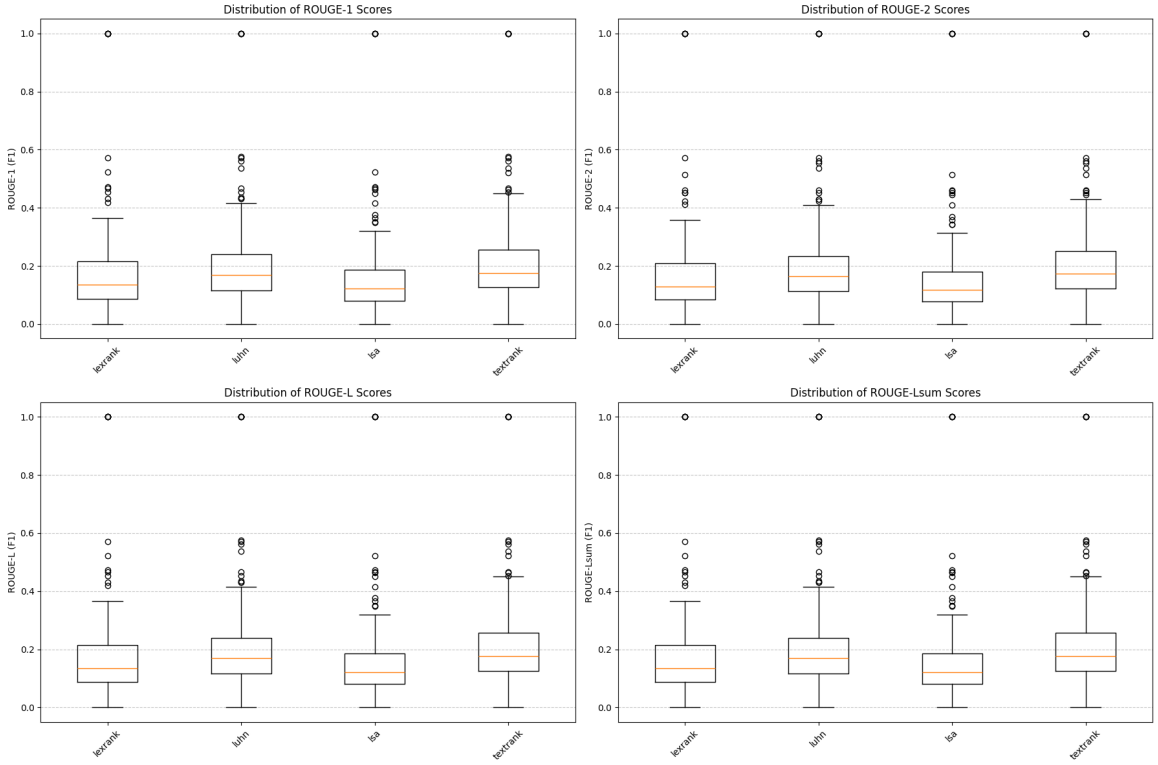


Fig. 4. ROUGE score distribution with sentence length (sentences_count) set to 1

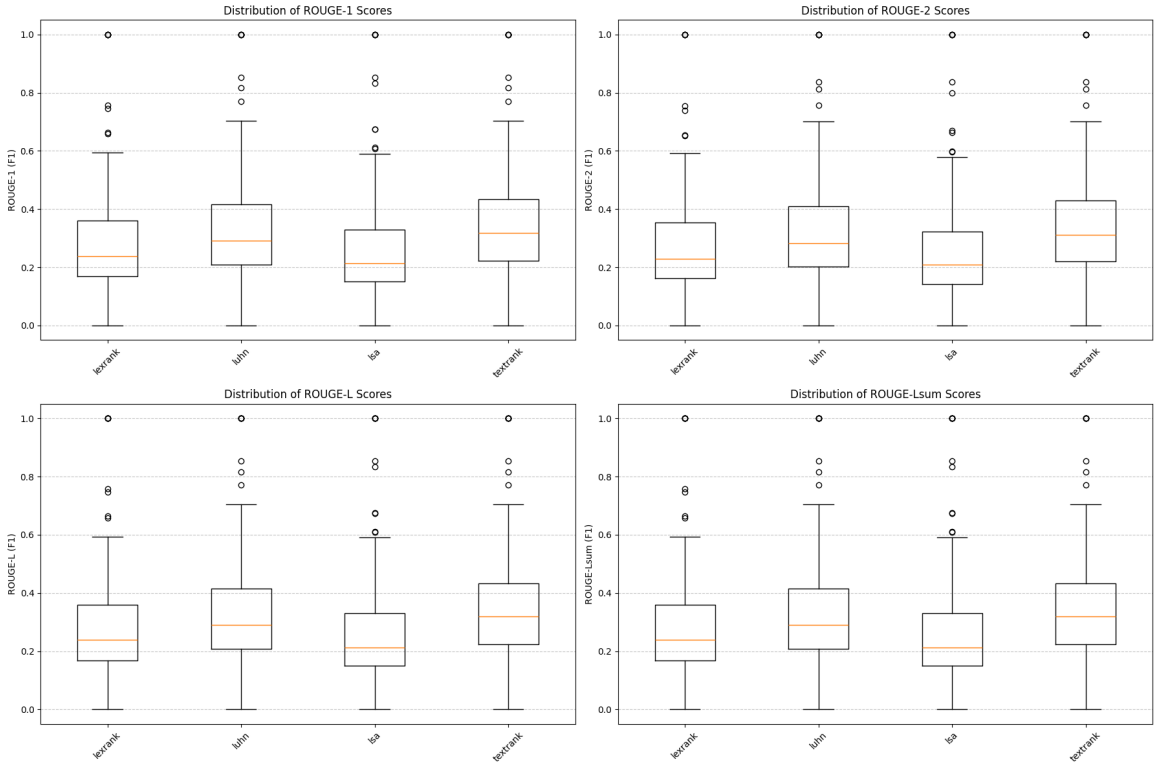


Fig. 5. ROUGE score distribution with sentence length (sentences_count) set to 2

4 Discussions and Conclusions

4.1 Summarization Performance

- The `TextRankSummarizer` consistently outperformed other methods across all ROUGE metrics (ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-Lsum)
- Performance ranking from best to worst was: TextRank > Luhn > LexRank > LSA
- All methods were able to generate coherent single-sentence summaries while maintaining the key information from financial news articles

4.2 ROUGE Score Analysis

The ROUGE scores revealed that:

- TextRank achieved higher precision and recall in capturing both unigram (ROUGE-1) and bigram (ROUGE-2) overlaps
- The higher ROUGE-L scores for TextRank indicate better preservation of the longest common subsequence
- Distribution of scores showed consistent performance across different articles, suggesting robustness of the method

4.3 Financial Domain Specific Insights

When combined with FinBERT sentiment analysis:

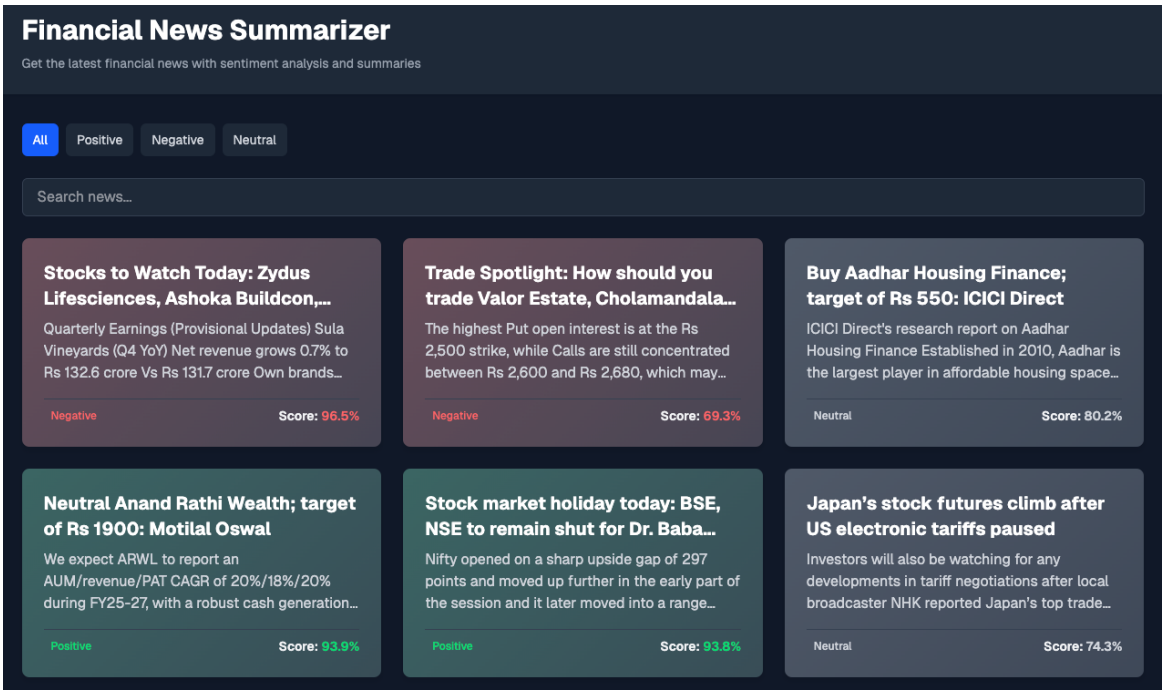
- The summaries retained enough financial context for meaningful sentiment classification
- TextRank summaries provided a balanced representation of market sentiment
- The summarization-sentiment pipeline proved effective for rapid market intelligence

4.4 Limitations and Future Work

While the current implementation shows promise, several areas deserve further investigation:

- Fine-tuning of parameters beyond sentence count could potentially improve results
- Domain-specific adaptations for financial text could enhance summarization quality
- Exploration of hybrid approaches combining multiple summarization techniques
- Integration of financial entity recognition for more targeted summaries

The experimental results demonstrate that TextRank is the most suitable summarizer for financial news articles among the tested methods. However, this represents just a starting point for financial text summarization. Future work should focus on domain adaptation and parameter optimization to further improve performance for specific financial use cases such as the one shown below in figure 6.



[9]

Fig. 6. Sample summarized news application on a article summary website

References

[1] Dogu Tan Araci. 2019. *FinBERT: Financial sentiment analysis with pre-trained language models*. Master's thesis. University of Amsterdam. arXiv:1908.10063 https://doi.org/10.48550/arXiv.1908.10063

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, 2019. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv:1810.04805. https://doi.org/10.48550/arXiv.1810.04805

[3] Yi Yang, Mark Christopher, Siy UY Allen Huang, 2020. *FinBERT: A Pretrained Language Model for Financial Communications*. School of Business and Management, Hong Kong University of Science and Technology. arXiv:2006.08097. https://doi.org/10.48550/arXiv.2006.08097

[4] Brian Scholl, et al. 2023. *A Picture Is Worth a Thousand Dollars: Visual Aids Promote Investor Decisions. Journal of the Association for Consumer Research*, vol. 8, no. 4, pp. 416–428. https://doi.org/10.1086/726428

[5] Richard P. Larrick and Jack B. Soll. 2008. *The MPG Illusion. Science*, vol. 320, pp. 1593–1594. https://doi.org/10.1126/science.1154983

[6] Belica, M. 2022. sumy 0.11.0: Module for automatic summarization of text documents and HTML pages. Python Package Index. Released Oct. 23, 2022. https://pypi.org/project/sumy/. (Accessed April 16, 2025).

[7] https://www.moneycontrol.com/news/business/stocks

[8] https://www.marketbeat.com/headlines/

[9] https://finbrief-orpin.vercel.app/