Generating Summaries of Short Fiction using TextRank

Emily Chao, Maria Gavrilovic

***Abstract*—In this paper, we implement a distributed TextRank algorithm via MapReduce to generate text summaries for fictional works by H.P. Lovecraft. Using an undirected, weighted graph to represent our fictional text, we used ROUGE evaluation metrics to compare the automatically generated summaries with human-generated summaries. We demonstrate that due to the limited vocabulary allowed by the extractive summarization method, the TextRank implementation achieves poor performance on fiction text.**

I. Introduction

AUTOMATIC text summarization, or ATS for short, is the task of condensing a piece of text to its most important and representative semantic components. Summarization models may be classed as either extractive or abstractive on the basis of their relation to the source text: abstractive models generate new text and extractive models extract words or sentences from the source text. In this paper, we focus on extractive summarization.

While ATS strategies are often employed on nonfiction text, such as news articles[1], applications to fiction text are generally absent in the literature. Hence, we applied TextRank, an unsupervised extractive summarization strategy, on fiction text to observe its performance. Specifically, we wanted to see if a relatively simple graph-ranking algorithm could capture the more abstract semantic relationships present in fictional text.

Originally, we considered using an LSTM-RNN trained on a corpus of fiction text to generate summaries, as abstractive summarization models that make use of generative neural networks have become popular recently. [2] However, in practice we found it difficult to create a distributed neural network using ApacheSpark and the MapReduce programming model. Instead, we focused on graph-based ranking strategies, particularly the TextRank algorithm, as these can be more easily written using MapReduce.

In general, graph-based ranking strategies seek to rank vertices by their relative importance, using both local vertex information and global information from the entire graph [3]. Perhaps the most famous example of graph-based ranking is Brin and Page's PageRank algorithm.

To perform a graph-based ranking on text data, we use the following graph representation. The vertices of the graph are assigned to text units, either word-tokens or entire sentences, depending on the desired summary type. For keyword extraction, the text units are chosen to be tokens; for sentence extraction, they are chosen to be sentences. In this paper we focus on the problem of sentence extraction. Then, the edge weights of the graph are assigned to a co-occurrence relation that represents the connections between the vertices. The score-updating algorithm will then iterate until convergence, and the vertices are sorted by their final ranking score.

Graph-based ranking algorithms are versatile and can be applied on both undirected and directed graphs, as well as graphs that are weighted or unweighted.

A. Background

For our text data, TextRank was chosen as our graph ranking model because it is the foundation of many more advanced models. Therefore, we wanted to see first how TextRank would perform before moving to more sophisticated models.

TextRank is a graph ranking model introduced by Mihalcea and Tarau[1] in 2004 for the purpose of text summarization. We introduce the basic components of TextRank in the following section.

1) TextRank: Let $G = (V, E)$ denote a graph with vertices $v_i \in V$ and edges $e_i \in E$. Then the score of a vertex $S(v_i)$ is defined as

$$S(v_i) = (1-d) + d \sum_{j \in In(v_i)} \frac{1}{|Out(v_j|} S(v_j), \tag{1}$$

where $d \in [0, 1]$ is a damping factor that represents the probability of resetting to a random vertex, and $In(v_i)$ and $Out(v_i)$ refer to the incoming and outgoing vertices of v_i, respectively. In practice, the damping factor is usually set to be 0.85.

Since the true error rate for $S(v_i)$ is unknown, it must be approximated by the difference between two successive iterations of the scoring, i.e. by $S^{k+1}(v_i) - S^k(v_i)$.

This score measure is very similar to the PageRank algorithm, with the only difference being that the initial term $(1-d)$ is not divided by the number of vertices n. While this recursive formulation of TextRank is easy to understand, it does not translate well to the MapReduce programming model. In practice[4], distributed PageRank and similar algorithms are calculated as a matrix-vector multiplication

$$\boldsymbol{r} = M\boldsymbol{r} \tag{2}$$

where

$$\boldsymbol{r} = \begin{bmatrix} r_1 \\ r_2 \\ \dots \\ r_n \end{bmatrix} \tag{3}$$

is the vector of scores for each vertex and M is the transition matrix representing the edges of the graph. Here the transition

matrix refers to the matrix produced by dividing the non-zero entries of the adjacency matrix by the out-degree of the source edge, i.e. $M_{ij} = \frac{1}{|Out(v_i)|}$.

It can be shown[5] that $\boldsymbol{r}$ is the principal eigenvector of M, thus ensuring that $\boldsymbol{r} = M\boldsymbol{r}$ can be solved through power iteration. While the proof of this fact is beyond the scope of this paper, we can use the fact that successive updates $\boldsymbol{r}^k$ and $\boldsymbol{r}^{k+1}$ will eventually converge to within a certain tolerance to return an approximation for the TextRank score of each vertex at the termination of the algorithm.

We also measure the strength of the connection between two vertices by a weighted score

$$WS(v_i) = (1-d) + d \sum_{v_j \in In(v_i)} \frac{w_{ij}}{\sum_{v_k \in Out(v_j)} w_{jk}} WS(v_j) \tag{4}$$

,

where w_{ij} is the weight assigned to the edge connecting vertices v_i and v_j.

After assigning each sentence a vertex in the graph representation, the edge weights are then assigned to the value of the relation between the vertices they connect. For sentence extraction, the relation must be a similarity measure. Mihalcea and Tarau define the similarity between two sentences S_i and S_j in the following way:

$$Similarity(Si, Sj) = \frac{|\{w_k : w_k \in S_i \cap w_k \in S_j\}|}{\log(|S_i|) + log(|S_j|)} \tag{5}$$

While this similarity measure is intuitive and straightforward, it is not commonly used in the ATS literature. Instead, after Mao et al[6] and Yulianti et al[7], we decided to use the cosine similarity measure. This is simply the measure of the angle between two vectors, i.e.

$$CosineSimilarity(A, B) = \frac{A \cdot B}{||A|| \times |B||} \tag{6}$$

This is calculated between the vector representation of two sentences. TextRank can be used with various text embedding schemes, though we chose to use the TF-IDF representation for its simplicity and integration with the Spark MLlib API.

II. Methods

A. Data

We tested our distributed TextRank implementation on three short stories, each under 3000 words, written by H.P. Lovecraft[8][9][10]. Since the entirety of Lovecraft's writing is in the public domain and has been digitized, the data is freely accessible and does not require much pre-processing to remove boilerplate copyright text, unlike the public domain texts available through Project Gutenberg. While the data used is free of semantically-unrelated noise, it is still a challenging dataset for extractive text summarization, as each text is written in a style that favours descriptive over narrative sentences.

B. Calculation of Cosine Similarity

To obtain cosine similarity, we first created a data processing and feature extraction pipeline using Apache Spark MLlib that consisted of multiple transformers [11]. The transformers we utilized were the Tokenization, HashingTF and IDF algorithms. Tokenization was used to split each sentence into a bag of words model. HashingTF was then called to transform each sentence into a feature vector, followed by a scaling transformation using IDF. In the TF-IDF calculation, each sentence was treated as a separate document. Thus, term and inverse document frequencies were calculated at the sentence level, as we are more interested in semantic similarity between sentences within a single document rather than semantic similarity across the entire corpus. Thus, the final feature column in the transformed dataframe compared the TF-IDF vector representation of each sentence in the document.

To calculate pairwise cosine similarity between sentences, we first performed a cross-join between the sentence IDs and their TF-IDF values, then applied a user-defined cosine similarity function to each member of the join. After a few transformations to remove excess columns, the resulting dataframe was the adjacency list representation of the text graph. Even for text data with a large number of sentences, this approach was not computationally prohibitive, as the underlying dataframe operations in PySpark are relatively efficient.

C. Implementing Distributed TextRank

The input of the distributed TextRank algorithm is an RDD representing the adjacency list representation of the graph computed in the data processing and feature extraction phase. Additionally, the distributed TextRank algorithm requires three user-specified parameters: damping factor d, tolerance, and the number of maximum iterations. The tolerance value is used to determine the convergence at each iteration. We set $\delta_0 = \frac{1}{tolerance}$, so the tolerance value is usually small. The algorithm continues until δ, the L^1 norm of the difference $\mathbf{r}^{k-1} - \mathbf{r}$ is less than the value specified for the tolerance or the maximum number of iterations has been reached. According to Mihalcea and Tarau, the convergence rate of TextRank is not dependent on the initial score values. Thus, the rank score vector $\mathbf{r}$ is simply initialized as $\mathbf{r} = \begin{bmatrix} 1 \\ 1 \\ \dots \\ 1 \end{bmatrix}_{n \times 1}$.

At each iteration of the algorithm, a new mapping over the entire RDD must be performed. We first map each row of the RDD into the form (v_i, v_j, m_{ij}), where v_i is the source index, v_j is the destination index, and m_{ij} is the value of the transition matrix M corresponding to the weight of edge e_{ij}. Then for each source index, we emit the update calculation

$$\frac{\mathbf{r_j} \times m_{ij}}{|Out(v_j)|}$$

in another mapping step. The out-degree of each vertex is simply the number of non-zero edges it has. We reduce by

source index to compute the sum of each update calculation, then emit

$$(1-d) + d \times \sum_{v_j \in In(v_i)} \frac{\mathbf{r_j} \times m_{ij}}{|Out(v_j)|}$$

for each source vertex v_i.

The output of the algorithm is a list of the top five sentence IDs with the highest TextRank scores. These sentence IDs are then matched to their corresponding sentences and displayed as the generated summary. In practice, the length of human-generated summaries for fiction texts tends to vary, but a standard length was chosen to perform evaluation.

III. Results

A. Evaluation

Since graph-based summary extraction is an unsupervised learning problem, it can be difficult to choose a suitable metric to evaluate its performance. In automatic text summarization literature it is common practice to use the ROUGE scoring package, which calculates the similarity between an automatically generated summary and a human-generated reference summary provided by the user. We used the Python implementation of ROUGE developed at Google Research [12], as it is still in active development.

A five-sentence human-generated summary was produced for each text, since ROUGE requires that the reference summary is the same length as the summary to be evaluated. The Python ROUGE implementation outputs four metrics: 1-gram overlap, 2-gram overlap, longest common subsequence, and pairwise longest common subsequence. In practice, pairwise longest common subsequence is most commonly used to evaluate automatic summaries, as it calculates the pairwise longest common subsequence between the generated summary and the reference. The results of TextRank-generated summaries for each text are shown in the tables below.

Several hyperparameters were selected for tuning: the tolerance, the number of maximum iterations, and the damping factor d. Specifically, we experimented with tolerance values of 10e-6, 10e-1 and 10e-10, d values of 0.01, 0.5 and 0.8, and 10 and 100 maximum iterations. However, repeated experimentation with different values of these hyperparameters did not result in significant change to the resulting scores (though they did affect the runtime of the algorithm).

TABLE I
Results for Dagon Summary

Evaluation Metric	Score
1-gram similarity	0.0789
2-gram similarity	0.0
LCS similarity	0.0779
LCS similiarity sum	0.0779

TABLE II
Results for Memory Summary

Evaluation Metric	Score
1-gram similarity	0.287
2-gram similarity	0.122
LCS similarity	0.246
LCS similiarity sum	0.246

TABLE III
Results for The Outsider Summary

Evaluation Metric	Score
1-gram similarity	0.106
2-gram similarity	0.0
LCS similarity	0.0946
LCS similiarity sum	0.0946

B. Discussion

In general, our implementation of extractive summarization with TextRank performs poorly in terms of one-gram (word) and bi-gram (two-word) similarity, and somewhat better in terms of longest common substring (LCS) similarity. As LCS similarity takes into account sentence-level structure, this indicates that the generated summaries may still be capturing the ”gist” or general idea of the reference summary even if they do not contain many similar words.

The summaries produced are difficult to understand as accurate summarizations of the text, especially to a reader unfamiliar with the source text. The summary for Dagon is particularly nonsensical, as the ranking did not preserve the chronological order of the sentences in the narrative. However, to a reader with prior knowledge of the summarized text, the generated summaries can be understood as a collection of representative sentences that depict the main ideas presented in the source text even if they are not a narratively cohesive synopsis.

Since TextRank is an extractive summarization scheme, the summaries it produces can only use words that are present in the source text. Meanwhile, the reference summaries are not under the same restriction. Additionally, summaries of fiction text frequently do not use the exact wording of the source text, as fiction texts tend to use abstract language and implicit descriptions that must then be summarized concretely. Hence, it makes sense that there might not be much overlap in the words used by the generated and reference summaries.

The length and content of the source text also appears to have an effect on the evaluation scores. The shortest text considered, Memory, is only 355 words long and consists primarily of setting descriptions rather than narrative sentences (i.e. sentences that describe an action taken by a character). As a result, the human-generated reference summary has a

higher proportion of words present in the source text and thus more semantic similarity to the generated extractive summary. In contrast, Dagon and The Outsider are 2240 and 2581 words long, respectively. Not only do the other two source texts also contain a high proportion of setting description to narrative sentences, their greater lengths amplify the narrative ambiguity created by long passages of descriptive text. Thus, a purely extractive summarization method would have difficulty finding semantically "important" sentences in a longer text written in a descriptive style.

IV. Conclusion

We implemented a distributed TextRank algorithm for extractive summarization using MapReduce, and showed that it does not perform well on fiction text. In particular, we performed sentence extraction using TF-IDF and cosine similarity. While our distributed TextRank implementation achieved poor accuracy in 1-gram and 2-gram similarity, it performed somewhat better in pairwise longest common subsequence similarity. Hence, we can conclude that the poor performance of our TextRank implementation is a consequence of the limited vocabulary allowed by the extractive summarization method.

In addition, fiction text often contains more implicit, rather than explicit, descriptions while human-generated summaries are often more explicit, which explains their dissimilarity.

A. Future Work

To further expand on the performance of TextRank with fiction text, we could explore the usage of weighted, directed graphs to represent our text data. The orientation of the edges would be directed either following the order of the sentences in the text or in the reverse direction.[3] Because the ranking score of a vertex is affected by the number of outdegrees and the edge weights, this could potentially improve the results of the text summarization as the order of the sentences would be taken into account. Especially in a longer fiction text, sentences that are next to each other tend to share context, which therefore could result in a more cohesive summary.

We could also assess TextRank on other fiction text from authors with different writing styles. Because H.P. Lovecraft's writings are of the fantasy genre and contain more abstract language, it would be interesting to see if TextRank performs better on different writing styles and genres.

Appendix A
Generated Summary for Dagon

The end is near. I think I went mad then. The change happened whilst I slept. The great war was then at its very beginning, and the ocean forces of the Hun had not completely sunk to their later degradation; so that our vessel was made a legitimate prize, whilst we of her crew were treated with all the fairness and consideration due us as naval prisoners. It is at night, especially when the moon is gibbous and waning, that I see the thing.

Appendix B
Generated Summary for The Outsider

Unhappy is he to whom the memories of childhood bring only fear and sadness. Wretched is he who looks back upon lone hours in vast and dismal chambers with brown hangings and maddening rows of antique books, or upon awed watches in twilight groves of grotesque, gigantic, and vine-encumbered trees that silently wave twisted branches far aloft. Such a lot the gods gave to me—to me, the dazed, the disappointed; the barren, the broken. And yet I am strangely content, and cling desperately to those sere memories, when my mind momentarily threatens to reach beyond to the other. I know not where I was born, save that the castle was infinitely old and infinitely horrible; full of dark passages and having high ceilings where the eye could find only cobwebs and shadows.

Appendix C
Generated Summary for Memory

These beings were like the waters of the river Than, not to be understood. And in trees that grow gigantic in crumbling courtyards leap little apes, while in and out of deep treasure-vaults writhe poison serpents and scaly things without a name. At the very bottom of the valley lies the river Than, whose waters are slimy and filled with weeds. And within the depths of the valley, where the light reaches not, move forms not meet to be beheld. Their deeds I recall not, for they were but of the moment.

References

[1] R. Mihalcea and P. Tarau, "TextRank: Bringing order into text," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, D. Lin and D. Wu, Eds., Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 404–411. [Online]. Available: https://aclanthology.org/W04-3252.

[2] R. Jie, X. Meng, X. Jiang, and Q. Liu, *Unsupervised extractive summarization with learnable length control strategies*, 2023. arXiv: 2312.06901 [cs.AI].

[3] R. Mihalcea, "Graph-based ranking algorithms for sentence extraction, applied to text summarization," in *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 170–173. [Online]. Available: https://aclanthology.org/P04-3020.

[4] A. Das Sarma, A. R. Molla, G. Pandurangan, and E. Upfal, "Fast distributed pagerank computation," *Theoretical Computer Science*, vol. 561, pp. 113–121, Jan. 2015, ISSN: 0304-3975. DOI: 10.1016/j.tcs.2014.04.003. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2014.04.003.

[5] J. Su. "Cs 224w – pagerank." (Jan. 2020), [Online]. Available: https://snap.stanford.edu/class/cs224w-2017/slides/handout-page_rank.pdf.

[6] Q. Mao, S. Zhao, J. Li, *et al.*, "Bipartite graph pre-training for unsupervised extractive summarization with graph convolutional auto-encoders," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 4929–4941. DOI: 10.18653/v1/2023.findings-emnlp.328. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.328.

[7] E. Yulianti, N. Pangestu, and M. Jiwanggi, "Enhanced textrank using weighted word embedding for text summarization," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, p. 5472, Oct. 2023. DOI: 10.11591/ijece.v13i5.pp5472-5482.

[8] H. Lovecraft. "Dagon." (), [Online]. Available: https://www.hplovecraft.com/writings/fiction/d.aspx.

[9] H. Lovecraft. "The outsider." (), [Online]. Available: https://www.hplovecraft.com/writings/fiction/o.aspx.

[10] H. Lovecraft. "Memory." (), [Online]. Available: https://www.hplovecraft.com/writings/fiction/m.aspx.

[11] A. Spark. "Mllib: Main guide." (2024), [Online]. Available: https://spark.apache.org/docs/latest/ml-guide.html.

[12] G. Research. "Python rouge implementation." (2024), [Online]. Available: https://github.com/google-research/google-research/tree/master/rouge.