CPSC 577 Project Report
Enhancing Text Classification with GraphSAGE

Shurui Wang
shurui.wang@yale.edu

Weiyi You
weiyi.you@yale.edu

Lang Ding
lang.ding@yale.edu

Abstract

This project explores the enhancement of text classification through the novel application of TextGraphSAGE, a graph-based neural network model that integrates textual and relational data. By constructing text graphs at a granular level with nodes representing individual words or phrases and edges reflecting adjacency, we aim to capture both local and global textual contexts more effectively. The project compares the performance of our TextGraphSAGE model with conventional deep learning models like CNNs and LSTMs, as well as another graph-based method, TextGCN, across two datasets: Reuters R8 and Twitter Asian Prejudice. Our results indicate that TextGraphSAGE outperforms the baseline models, demonstrating its potential to leverage relational information for superior text classification accuracy and efficiency. Our findings affirm the potential of graph-based methods in advancing text classification tasks. Code is available at https://github.com/JadenWSR/TextGraphSAGE.

1 Introduction and Motivation

Text classification is a foundational task in Natural Language Processing (NLP) aimed at assigning one or more categories to a piece of text. Traditional approaches range from rule-based to advanced deep learning techniques. Recently, the application of graph-based models like Graph Neural Networks (GNNs) has shown promising results in various NLP tasks, including text classification. These models, particularly GraphSAGE (Graph Sample and Aggregate), offer a novel approach to incorporate not just the textual information but also the relational context within a document.

2 Problem Definition

While traditional deep learning models for text classification have achieved impressive results, they often neglect the rich relational information available in textual datasets. This information includes implicit connections between words and entities within a document. GraphSAGE presents an opportunity to leverage this relational information by learning representations that incorporate both the content of documents and their context within a graph structure. This project aims to explore how GraphSAGE can be utilized to enhance text classification by effectively integrating textual and relational information.

3 Related Work

The application of graph-based methods in text classification has emerged as a significant advancement in NLP, addressing limitations of conventional models that fail to capture the complex relational structure of text data. Text Graph Convolutional Networks (Text GCN) introduced by Yao et al. [6] pioneered the integration of document and word nodes within a single graph, leveraging graph convolutional networks for enhanced text classification. This method, while innovative, constructs a static graph from the entire corpus, potentially limiting its adaptability and scalability for large datasets or dynamic text streams.

GraphSAGE, proposed by Hamilton et al. [1], offers a flexible framework for inductive representation learning on graphs. By sampling and aggregating features from a node's local neighborhood, GraphSAGE generates embeddings that encapsulate both local structure and node-level features, enabling efficient learning on large, dynamic graphs. This approach's adaptability and efficiency in handling relational data make it a promising foundation for our proposed method.

Convolutional Neural Networks (CNN) and Long Short-Term Memory networks (LSTM) have also been extensively applied in text classification, showcasing the power of deep learning models in capturing semantic patterns and dependencies within text [2, 3]. However, these models often overlook the relational information embedded in the text's graph structure, a gap our project aims to bridge by leveraging GraphSAGE's capability to integrate textual and relational data.

Our project aims to bridge the gap between traditional deep learning models, such as CNNs [2] and LSTMs [3], and graph-based approaches by employing GraphSAGE for text classification. Our proposed approach distinguishes itself by constructing a text graph at a granular level, where nodes represent individual words or phrases, and edges reflect adjacency within a window. This novel application of GraphSAGE for text classification aims to capture both local and global textual contexts, promising improvements in classification accuracy and efficiency over existing models.

4 Method

We proposed a custom GraphSAGE implementation, TextGraphSAGE, to construct and analyze text graphs, where nodes represent words and documents.

4.1 Model Framework and Mathematical Setting

4.1.1 GraphSAGE

GraphSAGE, or Graph Sample and Aggregated Embeddings, is a method that learns continuous vector representations of nodes in a graph, capturing both structural and contextual information about each node [1]. This inductive learning approach allows GraphSAGE to generate embeddings for previously unseen nodes during training by iteratively sampling and aggregating information from a node's local neighborhood.

The function of GraphSAGE is described as follows:

$$h_v^{(l)} = \sigma\left(W^{(l)}\text{CONCAT}\left(h_v^{(l-1)}, \text{AGG}\left(\{h_u^{(l-1)}, u \in N(v)\}\right)\right)\right) \tag{1}$$

Aggregation occurs in two stages:

- Stage 1: Aggregate from neighboring nodes:
$$h_{N(v)}^{(l)} = \text{AGG}\left(\{h_u^{(l-1)}, u \in N(v)\}\right) \tag{2}$$
- Stage 2: Combine the node's own features:
$$h_v^{(l)} = \sigma\left(W^{(l)}\text{CONCAT}\left(h_v^{(l-1)}, h_{N(v)}^{(l)}\right)\right) \tag{3}$$

where $h_{N(v)}^{(l)}$ represents the hidden layer representation of node v at layer l, $W^{(l)}$ is the weight matrix for neighborhood aggregation, and $N(v)$ represents the neighborhood of node v. AGG denotes the aggregation function, which can be mean, pooling, or an LSTM approach for order-sensitive aggregation.

4.1.2 TF-IDF and PMI

- TF-IDF: Term Frequency-Inverse Document Frequency (TF-IDF) weights are used for the edges linking documents to words within the graph. The TF-IDF value is calculated as follows:
$$w_{t,d} = tf_{t,d} \times idf_t \tag{4}$$

where $tf_{t,d} = \text{count}(t, d)$ is the term frequency and $idf_t = \log_{10}\left(\frac{N}{df_t}\right)$, with df_t representing the number of documents containing term t.

- PMI: Pointwise Mutual Information (PMI) assesses the co-occurrence strength of word pairs above that expected by chance and is used to weight word-word edges in the graph:

$$PMI(X, Y) = \log_2\left(\frac{P(x, y)}{P(x)P(y)}\right) \tag{5}$$

where $p(x, y) = \frac{\#W(x,y)}{\#W}$ and $p(x) = \frac{\#W(x)}{\#W}$. Only positive PMI values are used, indicating a significant semantic correlation between the word pairs.

4.2 TextGraphSAGE

TextGraphSAGE integrates the principles of GraphSAGE into a text classification framework by building a heterogeneous graph composed of both word and document nodes. This graph effectively models global co-occurrence and contextual relationships within texts, leveraging node and edge features optimized for text data.

1. **Text Graph Construction**: A graph is constructed where each node represents either a word or a document, forming a comprehensive network that maps the intricate relationships and frequencies of word occurrences across documents.
2. **Node Embedding**: Utilizing identity matrices, each node (word or document) is represented by a distinct one-hot vector, which is then enhanced through embeddings derived from pre-trained models to encode semantic information.
3. **Edge Weighting**: Edges between document and word nodes are weighted by TF-IDF to emphasize word relevance in documents, while edges between words are weighted by PMI to capture the semantic similarity based on their co-occurrence within specific contexts.
4. **Neighborhood Aggregation**: Adopts a two-layer GraphSAGE network with a softmax classifier to process node embeddings, utilizing sum aggregation to capture textual context from node neighborhoods effectively.
5. **Aggregation Strategy**: Implements dense matrix multiplication for sum aggregation, optimizing the model for efficient GPU usage.

This approach not only preserves the local and global semantic relationships inherent in text data but also enhances the adaptability and efficacy of the model by integrating GraphSAGE's inductive learning capabilities, allowing it to handle unseen data effectively.

4.3 Baseline Models

Our evaluation will compare the proposed approach against several established models:

1. Convolutional Neural Network (CNN) [2], which employs randomly initialized word embeddings for its architecture, focusing on the CNN-rand variant.
2. Long Short-Term Memory network (LSTM) [3], which uses the last hidden state to encapsulate the entire text representation, focusing on the model's ability to capture long-term dependencies.
3. Text Graph Convolutional Networks (Text GCN) [6], which constructs a large graph from the entire corpus, representing both words and documents as nodes. This approach utilizes a two-layer Graph

5 Experiments

5.1 Computing Infrastructure

Our computing environment for training and testing the models included using a single T4 GPU on Google Colab for the baseline models (CNN, LSTM, and TextGCN) and a single GPU on the

YALE cluster for the TextGraphSAGE model. The specific configurations for TextGraphSAGE were managed using the 'cpsc577_env.yaml' environment setup available on our GitHub. Consistency in training and testing across all models was maintained by using the same 8/1/1 tvt split, but with different randomly chosen seeds [33, 15, 86, 109, 78] to ensure robustness and repeatability in results. Models were optimized using CrossEntropyLoss and the Adam optimizer configured with varying learning rates as determined by the hyperparameter tuning process.

5.2 Hyperparameters and Tuning Process

For the TextGraphSAGE model, we conducted hyperparameter tuning on both the R8 and Twitter datasets using a grid search approach. Hyperparameter tuning was performed with a seed of 123 and an 8/1/1 train/validation/test split. This tvt split allowed us to methodically fine-tune hyperparameters based on validation performance, ensuring that the best model parameters were used for final testing to achieve unbiased and generalizable results. The parameters considered include learning rate ('lr') with options [0.01, 0.001, 0.0001], prediction type ('pred_type') with options ['softmax', 'mlp'], the presence of dropout ('dropout') with options [True, False], and activation function ('act') with options ['relu', 'prelu', 'sigmoid', 'tanh']. The best-performing parameters identified were for the R8 dataset: learning rate of 0.01, softmax prediction, no dropout, and PReLU activation; for the Twitter dataset: learning rate of 0.001, softmax prediction, no dropout, and tanh activation. Additional parameters such as word window size (10), validation window size (10), number of layers (2), and the use of CrossEntropyLoss and Adam optimizer, were kept consistent with the configurations found in the TextGCN paper[6], as these settings yielded optimal results in preliminary experiments. For baseline models, default parameters from their original papers[2, 3, 6] or implementations were used without further tuning.

5.3 Datasets

We employ two primary datasets in our study:

1. Reuters R8 Dataset[5]: This subset of the Reuters 21578 collection contains documents from 1987 newswire articles, divided into 8 categories: trade, ship, interest, earn, crude, money-fx, grain, and acquisition. The dataset consists of 5,485 training texts and 2,189 testing texts. Preprocessing involves standard text cleaning techniques such as removing stop words and infrequent words, and applying normalization steps to prepare the text for the graph-based model.

2. Twitter Asian Prejudice Dataset[4]: Comprising 20,000 tweets related to East Asian prejudice, this dataset is categorized into five labels reflecting the nature of the tweets, including discussions on prejudice, counter-speech, and entity-directed hostility. Similar preprocessing steps are used here, tailored to handle social media text characteristics like hashtags and mentions.

Both datasets are publicly accessible through the Text-GCN repository on GitHub https://github.com/codeKgu/Text-GCN/tree/master/data/corpus. For CNN and LSTM models, standard NLP data cleaning protocols are applied, including tokenization and filtering non-lexical items. For the TextGCN and TextGraphSAGE models, texts are further processed into a graph format as described in Section 4.2. This involves converting texts to nodes and edges in a graph, utilizing term frequency and document co-occurrence to weight the connections between nodes, thereby retaining semantic relationships within the data.

5.4 Evaluation Metric

In assessing the performance of TextGraphSAGE compared to baseline models(CNN, LSTM, and Text GCN), we employ three key metrics: accuracy, F1 macro, and F1 weighted. Accuracy provides a straightforward measure of overall model performance, indicating the proportion of total correct predictions across all classes. F1 macro is used to calculate the average effectiveness of the model across all classes, ensuring that each class is equally considered regardless of its frequency in the dataset. This is crucial for evaluating performance in datasets with imbalanced class distributions. F1 weighted offers a performance measure that accounts for class size, giving more weight to classes with more instances. This metric helps assess the model's effectiveness in handling class imbalance by reflecting the importance of each class based on its prevalence. These metrics collectively provide a comprehensive view of model performance, crucial for validating the enhancements offered by TextGraphSAGE over traditional text classification approaches.

6 Results

Table 1: Performance on R8 Dataset

Model	Accuracy	F1 Macro	F1 Weighted
CNN	0.8659±0.0715	0.5464±0.1751	0.8399±0.0963
LSTM	0.8482±0.0402	0.5028±0.1018	0.8321±0.0495
TextGCN	0.9582±0.0027	0.8795±0.0101	0.9586±0.0026
TextGraphSAGE	**0.9630±0.0018**	**0.8998±0.0056**	**0.9633±0.0017**

Table 2: Performance on Twitter Asian Prejudice Dataset

Model	Accuracy	F1 Macro	F1 Weighted
CNN	0.6787±0.0073	0.1659 ± 0.0034	0.5540±0.0105
LSTM	0.6813±0.0078	0.1621±0.0011	0.5522±0.0100
TextGCN	0.7032±0.0264	**0.5276±0.0200**	0.7129±0.0193
TextGraphSAGE	**0.7359±0.0090**	0.5170±0.0337	**0.7133±0.0283**

7 Conclusion & Discussion

The evaluation of the TextGraphSAGE model across two distinct datasets reveals its robust performance in text classification tasks. On the R8 dataset, TextGraphSAGE achieved an impressive accuracy of 0.9630±0.0018, surpassing the baseline models. It also demonstrated strong capability in balancing class representation, as indicated by its F1 Macro score of 0.8998±0.0056, and effectively managed class size variances, achieving an F1 Weighted score of 0.9633±0.0017. Performance on the Twitter Asian Prejudice dataset followed a similar trend, with TextGraphSAGE outperforming other models with an accuracy of 0.7359±0.0090. While the F1 Macro and F1 Weighted scores were slightly lower in this dataset at 0.5170±0.0337 and 0.7133±0.0283 respectively, they still represent competitive results, considering the challenging nature of classifying nuanced social media content. These results confirm TextGraphSAGE's effectiveness in leveraging both global and local textual contexts, significantly enhancing text classification outcomes compared to conventional methods.

7.1 Limitations

7.1.1 Positional information

Our graph-based model converts words and documents into nodes linked by edges weighted with TF-IDF or PMI, which inherently lacks explicit positional information. This absence means there is no inherent order to the nodes that correspond to their original sequence in the text, potentially resulting in ambiguous or inconsistent interpretations of node positions. Although positional information could be simulated in edge weighting, it does not naturally capture the sequential flow of text as in traditional NLP models.

7.1.2 Long-range Dependencies

TextGraphSAGE focuses on local node neighborhoods, which might restrict its capability to model long-range dependencies crucial for understanding broader textual contexts. Although attention mechanisms can partly mitigate this limitation, effectively capturing extended dependencies between distant words or across document sections remains a substantial challenge for our graph-based approach.

7.1.3 Dynamic Text Data

Text data is often dynamic in real world problem, with new words, documents, or concepts being introduced over time. Adapting a graph-structured neural network to dynamically changing text data while efficiently updating the model parameters can be challenging, especially in scenarios with streaming or evolving text data.

7.2 Future Work

There is a lot of room for improvement of our model and idea.

7.2.1 Incorporating Positional Information

Future enhancements could include developing methods to integrate positional information into the TextGraphSAGE framework. This could involve novel positional encoding techniques or graph architectures that explicitly account for the hierarchical or sequential structure of textual data, potentially improving the model's natural language processing capabilities.

7.2.2 Dynamic graph adaptation

Further research could explore adaptive strategies for TextGraphSAGE to handle dynamically changing text data. This would support more robust applications in streaming or continuously updated document sets, enhancing the model's practicality in dynamic environments.

7.2.3 Semantic Graph Augmentation

Enhancing text graph representations with external semantic knowledge, such as incorporating data from knowledge graphs or ontologies, could significantly boost the semantic interpretation capabilities of our model. Investigating automated methods for semantic graph augmentation and leveraging this enriched information during model training and inference could provide substantial advancements. Additionally, exploring transitions to other models like Transformers could offer improved handling of long-range dependencies.

References

[1] Hamilton, W. L., Ying, R. and Leskovec, J. [2017], Inductive representation learning on large graphs, *in* 'Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)', pp. 1024–1034.

[2] Kim, Y. [2014], Convolutional neural networks for sentence classification, *in* 'Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)', pp. 1746–1751.

[3] Liu, P., Qiu, X. and Huang, X. [2016], Recurrent neural network for text classification with multi-task learning, *in* 'Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)', AAAI Press, pp. 2873–2879.

[4] Vidgen, B., Botelho, A., Broniatowski, D., Guest, E., Hall, M., Margetts, H., Tromble, R., Waseem, Z. and Hale, S. [2020], 'Detecting east asian prejudice on social media'.

[5] Weipengfei [2024], 'oh-r8-r52 Dataset', `https://www.kaggle.com/datasets/weipengfei/ohr8r52`. Accessed: [03/06/2024].

[6] Yao, L., Mao, C. and Luo, Y. [2018], Graph convolutional networks for text classification, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 33, pp. 7370–7377.

Reproducibility checklist

* Please make sure these points are addressed in your report submission
* Please copy this and replace the ☐ with a ✔ for the items that are addressed in your report/code submission
* Please complete this report, attach it to your final project report as the last page and then submit.

Model Description, algorithm, Mathematical Setting:

✔ Include a thorough explanation of the model/approach or the mathematical framework

Source Code Accessibility:

✔ Provide a link to the source code on github.
✔ Ensure the code is well-documented
✔ Ensure that the github repo has instructions for setting up the experimental environment.
✔ Clearly list all dependencies and external libraries used, along with their versions.

Computing Infrastructure:

✔ Detail the computing environment, including hardware (GPUs, CPUs) and software (operating system, machine learning frameworks) specifications used for your results.

(Example statement 1: the model was fine-tuned using a single T4 GPU on colab.
Example statement 2: we ran inference of Llama 70B using 4 Nvidia A5000 GPUs)

✔ Mention any specific configurations or optimizations used.

(Example: We used a quantized version of Llama with int8.
Example 2: We used the regular float32 representation.)

Dataset Description:

✔ Clearly describe the datasets used, including sources, preprocessing steps, and any modifications.
✔ If possible, provide links to the datasets or instructions on how to obtain them.

Hyperparameters and Tuning Process:

✔ Detail the hyperparameters used and the process for selecting them.

(Example: The model was fine-tuned using a batch size of 16, learning rate of 1e-5, and trained on 1000 steps with 100 steps of learning rate linear warmup with linear decay)

Evaluation Metrics and Statistical Methods:

✔ Clearly define the evaluation metrics and statistical methods used in assessing the model.

Experimental Results:

✔ Present a comprehensive set of results, including performance on test sets and/or any relevant validation sets.
✔ Include comparisons with baseline models and state-of-the-art, where applicable.

Limitations and future work:

✔ Include a discussion of the limitations of your approach and potential areas for future work.