Text Classification for Skewed Label Sets using Modified TextGCN and Human Pseudo-Labels

Khoa Tran
tlmkhoa@yorku.ca
York University
Toronto, Ontario, Canada

ABSTRACT

This project presents a novel approach to text classification for skewed label sets using modified TextGCN and human pseudo-labels in the context of identifying high-level feedback at TD Bank. The proposed approach generates pseudo-label data using keywords, modifies the TextGCN framework to leverage unlabeled data, adds virtual nodes to handle new samples, and evaluates the model performance using variome metrics. The experimental results show that the modified TextGCN model outperforms the baseline and existing internal models in terms of accuracy, F1-score, recall, and precision. A sensitivity analysis is performed to show the trade-off between memory and performance for different semi-samples, PMI thresholds, and preprocessing methods. In addition, a general guide for balancing the two metrics is provided for future industrial and internal development of textGCN. Finally, the implementation challenges, the limitations of the approach, and the future work directions are discussed.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; **Neural networks**; **Neural networks**; *Cross-validation.*

KEYWORDS

Text classification, skewed label set, modified TextGCN, human pseudo-labels, graph convolutional network, NLP

1 INTRODUCTION

1.1 Background

This project report is submitted by Khoa Tran to the Faculty of Graduate Studies in partial fulfillment of the requirements for the degree of Master of Science in Computer Science with a specialization in Artificial Intelligence at York University. The project is a result of a collaboration between the author, Khoa Tran, and various departments at Toronto Dominion Bank (TD). The project was conducted during an internship period under the management of Senior Data Scientist Desai Sneha and under the supervision of Professor Manos Papagelis. The main objective of this project and internship was to explore and apply state-of-the-art machine learning models to different problems and evaluate their performance and improvement over existing techniques at TD Bank. In this report, I present my implementation and modification of TextGCN, a graph convolutional network for text classification, for the task of high-level feedback detection. I will also discuss the trade-offs and challenges of the proposed model.

1.2 Project timeline

The internship session where the project is researched and implemented lasts 4 months from September 2023 to December 2023. The following is my timeline broken down by tasks achieved over the period.

- **Research the problem and TD system**: September 1, 2023 - September 15, 2023
- **Research solution and propose methods**: September 15, 2023 - October 1, 2023
- **Write and submit project proposal**: September 21, 2023 - October 07, 2023
- **Collect and submit data for labeling** : October 1, 2023 - October 14, 2023
- **Implement text GCN model**: October 1, 2023 - October 15, 2023
- **Modify text GCN model**: October 15, 2023 - October 30, 2023
- **Evaluate models and post analysis**: November 01, 2023 - November 21, 2023
- **Write and proofread the project report**: November 21, 2023 - December 10, 2023
- **Deployment of model to TD system**: December 01, 2023 - December 22, 2023
- **Submit the report**: December 22, 2023

1.3 TD Privacy

Due to the high sensitivity of the project, many technical details will be omitted, especially the data distribution or information that can imply the distribution. I will only go through an abstract description of the dataset and will mainly focus on the TextGCN model and its behavior in different configurations.

1.4 Problem Description

TD Bank, with its 1,060 branches, serves 27.5 million customers and receives a large number of feedback in my system every day. High-level feedback (HLF), is defined as feedback that needs to be escalated, reviewed, and followed up, and by nature, there is always a need for accurate detection of this type of feedback. Within the institution, an existing model is currently in use for HLF detection. This model, which is based on Convolutional Neural Networks (CNN), was initially trained for multiple-label prediction. However, its performance has been found to be subpar and there is room for improvement. This situation underscores the pressing need to develop a state-of-the-art model for HLF prediction that includes comprehensive and well-documented functions.

1.4.1 Labeling HLF: . I also faced a minor problem with labeling. HLF is inherently rare. This rarity, combined with the company's limited labeling capacity at this time frame, presents a unique problem. The challenge lies in presenting an optimal set of data to the labeling team to maximize the number of positive labels. This approach aims to prevent skewness in the label set as much as possible and reduce reliance on processing techniques.

1.4.2 The Learning Problem: The main challenge is more conventional and pertains to learning from a few sample skewed label sets. Given multiple feedbacks, the task is to distinguish between routine feedback, which is more common, and serious issues that need to be escalated

2 DATASETS

One approach to identifying HLF involves analyzing customer feedback regarding their experiences with branch services. This feedback can be collected through various channels, such as feedback forms sent to customers' emails or directly from customers at the location. Additionally, summaries of customer feedback, written and submitted to my system by branch employees, can also serve as a valuable source of information. Since some of TD's data is collected from customer direct input, text from emails or surveys is noisy due to the lack of control over people's submissions. Moreover, since French is an official Canadian language, feedback in French is significant in my dataset. Due to the limited scope, I will be focusing on English and removing other languages from my set.

3 RELATED WORK

3.1 Text classification

Text classification is a fundamental task in natural language processing that assigns predefined labels to text documents based on their content. It has various applications in sentiment analysis [Ojo et al.(2023)], topic detection [Asgari-Chenaghlu et al.(2020)], spam filtering [Tida and Hsu(2022)], etc. Traditional methods for text classification relied on hand-crafted features or word embedding. Some more complex features have been designed, such as n-grams [Wang and Manning(2012)] which may not capture the semantic and syntactic information of the text adequately. Deep learning methods for text classification use neural networks such as CNNs [Kim(2014)], RNNs [Liu et al.(2016)], and Transformers [Park et al.(2022)] to learn text representations from raw inputs. However, these methods mainly focus on local consecutive word sequences but do not explicitly use global word co-occurrence information in a corpus.

3.2 Graph neural networks

Graph neural networks (GNNs) [Kipf and Welling(2016)] is a type of neural network that can operate on graph-structured data, such as social networks, citation networks, and knowledge graphs. The topic of Graph Neural Networks has received growing attention recently [Cai et al.(2018)]. GNNs can learn node representations that capture both local and global information from the graph topology and node features. GNNs have been applied to various domains such as traffic prediction [Jiang and Luo(2021)], recommendation [Wu et al.(2020)], and NLP. In the context of NLP, GNNs have been used to model the syntactic structure of sentences, the relationships between entities, and the semantic similarity between texts.

3.3 GNNs for text classification

Several works have explored the use of GNNs for text classification, where the text is represented as a graph of words and documents. [Yao et al.(2019)] proposed TextGCN, which builds a single text graph for a corpus based on word co-occurrence and document-word relations, and learns word and document embeddings jointly. GCN was also explored in several NLP tasks such as semantic role labeling [Marcheggiani and Titov(2017)], relation classification [Li et al.(2018)], and machine translation [Bastings et al.(2017)], where GCN is used to encode the syntactic structure of sentences. Some recent studies explored graph neural networks for text classification [Henaff et al.(2015), Defferrard et al.(2016), Peng et al.(2018), Zhang et al.(2018)]. However, they either use an identity matrix as embeddings or only leverage fully supervised learning. They also do not pay attention to dealing with skewed data.

4 METHODOLOGY

4.1 Labeling

Currently, thanks to the auditing process, I have obtained small samples of HLFs. I proposed the usage of a pre-trained model like SBERT [Reimers and Gurevych(2019)] to encode the positive labels and no-label sets into a vector space. This process is represented in Equation 1. By identifying the positive cluster, I can measure the cosine similarity between the cluster positive centroid and other unlabeled instances, and then rank them from most to least similar.

$$\text{Rank}(\mathbf{Vector}_i) = \text{sort}\left(\frac{\mathbf{Vector}_i \cdot \mathbf{TargetVector}}{||\mathbf{Vector}_i||||\mathbf{TargetVector}||}\right) \quad \forall i \in [1, N] \tag{1}$$

Theoretically, this can provide the labeling team with an optimal set where the instances closest to the positive will be presented first and noise will be last. This approach heavily depends on the general knowledge of the model. Further improvement can be applied by using the Doc2Vec [Le and Mikolov(2014)] model to train on an internal dataset and gain knowledge directly from the dataset rather than pre-train from a public set.

4.2 Text GCN

Text Graph Convolutional Networks (Text GCN) [Yao et al.(2019)], a variant of Graph Convolutional Networks (GCN) introduced in 2019, is utilized for the automatic classification of text into corresponding labels. GCNs have proven to be effective for tasks with rich text, as they can preserve the global structural information of a dataset in graph embeddings. A sample of the textGCN graph is provided in Figure 1, where it is built from 2 examples. The selection of Text GCN for this project is motivated by its top-tier performance in various text classification and sentiment analysis tasks. For instance, Roberta [Nakajima and Sasaki(2022)] or SGNN [Karl and Scherp(2023)], variants of Text GCN, consistently rank first on the Ohsumed dataset. This high level of performance is also observed on other datasets such as 20NEWS, MR, and R8.

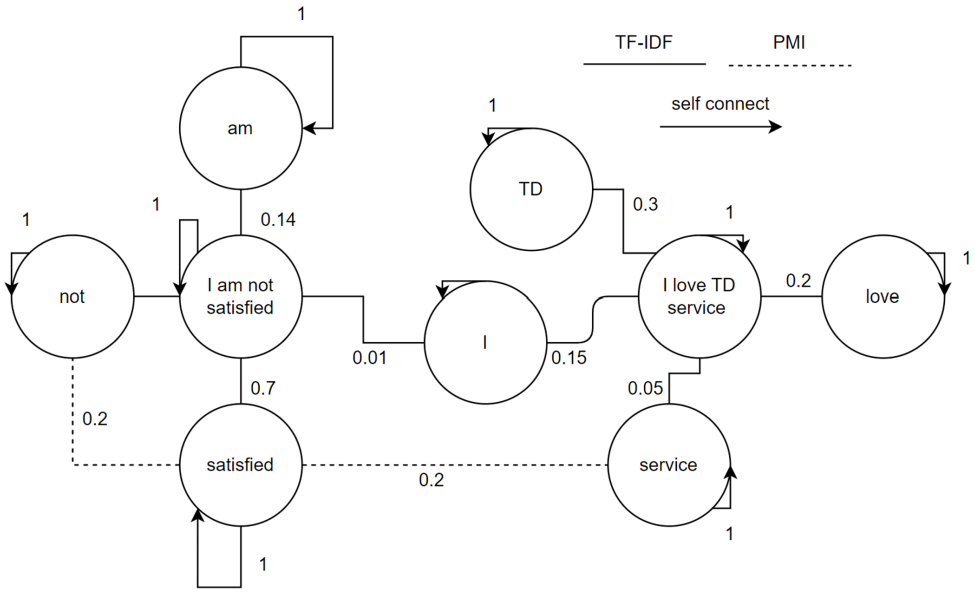

Figure 1: Sample textGCN graph. Built from 2 example sentences, "I love TD service" and "I am not satisfied". Each unique word and document is treated as a node. A solid line shows the TF-IDF relationship between a word and a node, which every word had. The dashed line portrates the PMI connection. The arrow line illustrates a self-connect at each node.

In TextGCN, the graph is represented as an adjacency matrix. This matrix is symmetric, meaning it mirrors along the main diagonal. Each row or column in this matrix corresponds to a node in the graph. In equation 2, I have shown how the edges of the graph represented in the adjacency matrix follow the rule from the original [Yao et al.(2019)] paper

$$A_{ij} = \begin{cases} \text{PMI}(i,j) & \text{if } i,j \text{ are words and PMI}(i,j) > 0 \\ \text{TF-IDF}_{ij} & \text{if } i \text{ is document and } j \text{ is word} \\ 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

For adjacency matrix ordering, referring to figure 2, let's define Ntrain as the number of samples in the training set, and Nvocab as the number of unique words in my graph vocabulary. I arranged the top Ntrain nodes for the document and the next Nvocab for the unique word. The purple area will be populated by TF-IDF values, and the blue is for PMI edges. The green area would be 0 since there is no connection between two documents. Each element on the diagonal line will also have one added representing the self-connection. After constructing the Adjacency matrix, I inputted the graph into a simple two-layer Graph Convolutional Network (GCN) as in [Kipf and Welling(2017)]. The embeddings of the second layer nodes (word/document) have the same size as the label set and are input into a softmax classifier:

$$Z = \text{softmax}(\hat{A}\text{ReLU}(\hat{A}X\hat{W}_0)\hat{W}_1) \tag{3}$$

where $\hat{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, and $\text{softmax}(x_i) = \frac{1}{Z}\exp(x_i)$ with $Z = \sum_i \exp(x_i)$

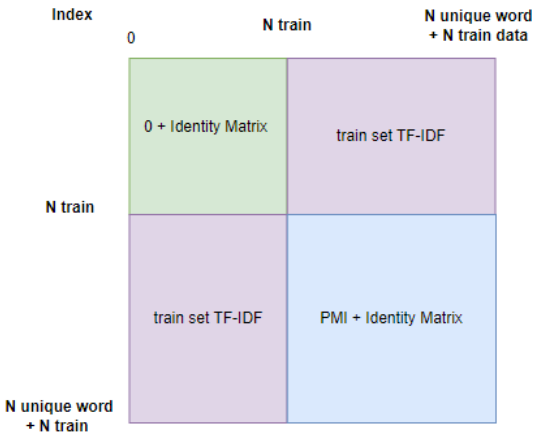

Figure 2: Arrangement of edges in train adjacency matrix for my implementation. Fully described in the text

4.3 Proposed modification of Text GCN

Another reason for choosing Text GCN is its ability to leverage unlabeled data. The original textGCN [Yao et al.(2019)] paper only uses the model in the context of full-supervised learning. However, as a GCN [Kipf and Welling(2017)] based method, Text GCN can utilize the unlabeled data for semi-supervised learning. This means that I can incorporate millions of unlabeled feedback that TD receives daily to interpret the minimal labeled set I have access to. Unlabeled

data will be represented as document nodes in the graph connected to words that are contained within themselves. These will allow labeled nodes, especially positive samples, to leverage not only other label documents but unlabeled documents to gain more context to the problem. Text GCN's ability to leverage different types of embeddings is another point of interest. While the original Text GCN [Yao et al.(2019)] uses one-hot encoding as node embedding as X = I. I believe that this can be further explored by applying different embeddings to convey more information across the network. For instance, looking at equation 4, simple TF-IDF could be used, or the power of pre-trained models can be leveraged to capture the relationship between words and documents could be harnessed. Pre-trained encoder models like SBERT [Reimers and Gurevych(2019)] are promising candidates. Alternatively, to avoid reliance on external data and maintain control over the model structure for embeddings, a Doc2Vec [Le and Mikolov(2014)] model could be trained. I believe these are better alternatives to the original one-hot encoding.

$$X = \begin{cases} \text{TF-IDF} & \text{if TF-IDF embeddings} \\ \text{Doc2Vec} & \text{if Doc2Vec embeddings} \\ \text{SBERT} & \text{if SBERT embeddings} \end{cases} \tag{4}$$

One-hot encoding also limits the model to transitive learning, as no new predictive samples can be added. By using an alternative method such as TF-IDF, I facilitated inductive learning. However, this raises the question of how to represent the new samples in the training graph. I proposed adding them as virtual nodes and removing them after prediction. The new document nodes would be connected to the word nodes contained in their sentences. In terms of the validation matrix, looking at Figure 3, a new row and column will be added at the end of the two dimensions, creating an expanded matrix that uses the training information to predict new data and support inductive learning. The newly added yellow area is represented by the TF-IDF weights of the validation set. The same process can be done for the test set. This allows me to easily extend the text graph to include new data and make predictions using the trained Text GCN model.

4.4 Loss

4.4.1 Normal Loss. The loss function is defined in equation 5 as the cross-entropy error over all labeled documents:

$$L_{normal} = -\sum_{d \in Y_D} \sum_{f=1}^{F} w_f Y_{df} \ln Z_{df} \tag{5}$$

where Y_D is the set of document indices that have labels and F is the dimension of the output features, which is equal to the number of classes. Y is the label indicator matrix. W_f is the weight for the class, to account for class imbalance, the majority class will have less weight and the minority class will have more weight

4.4.2 Keyword Loss. Inspired by the graph pseudo-labeling method of [Lee(2013)], where a model's high-confidence prediction in the previous epoch is selected to be pseudo-labeled in the next epoch, I proposed developing a potential keyword loss function. Instead of using the model's confident prediction, I can use keywords as human confident predictions. Keywords that are likely to appear in a positive label will be determined by the TD business team. Given

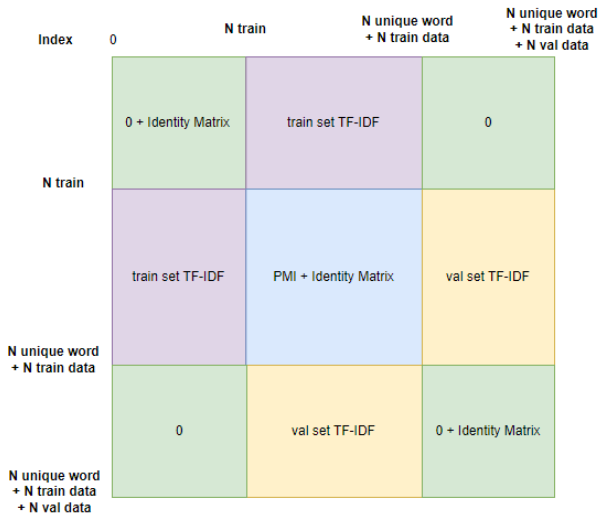

Figure 3: Arrangement of edges in validation adjacency matrix after adding virtual nodes. Fully described in the text

my limited positive labels and labeling capacity, this approach will help me detect more positive pseudo-labels and compel the model to learn sentences containing these keywords. I acknowledged that in many cases, these groups may not have negative samples, which could potentially lead to overfitting on these specific samples. Therefore, I propose adding a class weight by the factor of $1/(NP{+}1)$ to control the size of the group described in equation 6, where NP is the number of pseudo-positive samples found in keywords. I also anticipate finding labeled negative classes in these keywords. Therefore, I will apply the same weight formula, replacing NP with NN, the number of negative samples found from keywords.

$$w_i = \begin{cases} \frac{1}{N_P+1} & \text{if } f \text{ is a pseudo-positive class} \\ \frac{1}{N_N+1} & \text{if } f \text{ is a negative class} \end{cases} \tag{6}$$

$$L_{keyword} = -\sum_{i=1}^{N} w_{y_i} \cdot y_i \cdot \log(p(y_i)) + (1-y_i) \cdot \log(1-p(y_i)) \tag{7}$$

Keywords can be categorized into multiple groups, each associated with a different loss. For instance, one group is likely associated with keywords, which suggests that sentences containing these words highly indicate HLFs. Another group is more neutral words that are less indicative due to their usage in various contexts. Finally, I can aggregate all the losses by assigning them weights (denoted as lambda) in equation 8, allowing me to control their impact.

$$L_{total} = L_{normal} + \lambda_1 L_{likely} + \lambda_2 L_{neutral} \tag{8}$$

5 EVALUATION

For the evaluation process, I will employ the F1 score as a metric shown in equation 9. However, it will be adjusted based on the inverse proportionality of each class. This implies that the majority class will be assigned a lower weight, while the minority class will

be given a higher weight-adjusted in equation 10.

$$F1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} \tag{9}$$

$$F1_{\text{weighted}} = \sum_{i=1}^{N} w_i * F1_i \tag{10}$$

6 RESULTS

6.1 Labeling

I decided to use SBERT [Reimers and Gurevych(2019)] as my labeling model. SBERT stands for Sentence Embeddings using Siamese BERT-Networks and is implemented by SBERT.net. SBERT is a variation of the pre-trained BERT network that uses siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine similarity. This specific triplet network structure allows SBERT to excel in the task of sentence comparison in my use case. This method heavily depends on the pre-trained model knowledge of SBERT. However, thanks to this methodology, the TD business team was able to find significantly more positive HLF samples than the expected distribution. Unfortunately, due to the TD privacy policy and the highly sensitive nature of the problem, I could not disclose the data distribution I found or a sample of these data.

6.2 Preprocessing

In my problem, I defined 2 vocabularies, vocab graph and vocab feature. I found that separation helps me gain more flexibility in processing and fine-tuning the model.

6.2.1 vocab graph. for graph building, specifically for word node creation and PMI's edge calculation. Only training data will be used to create this vocab to control the number of nodes in the graph.

6.2.2 vocab feature. for feature transform, specifically for creating a feature for each node. Sentences and words are transformed from text to vector by using a set of vocab. For this, I employed both train and semi-set to build the set.

I applied standard reprocessing techniques to my data set, including the following:

- **Drop NA**: Remove rows with missing values.
- **Drop Duplicates**: Remove duplicate rows.
- **Remove Numbers**: Replace all numbers with a special token <num>.
- **Lemmatize**: Convert words to their base form using lemmatization.
- **Stopword Removal**: Remove common stop words
- **Filter English**: Remove non-English sentences by removing sentences that did not contain any English stop words.
- **Remove Corrupt**: Remove sentences that were corrupted or unreadable.
- **Minimum Character Threshold**: Remove sentences with fewer than a threshold.
- **Train-Validation-Test Split**: Split the data into training, validation, and test sets.
- **Graph Threshold**: Remove words that appear less frequently than a certain threshold in the training set.

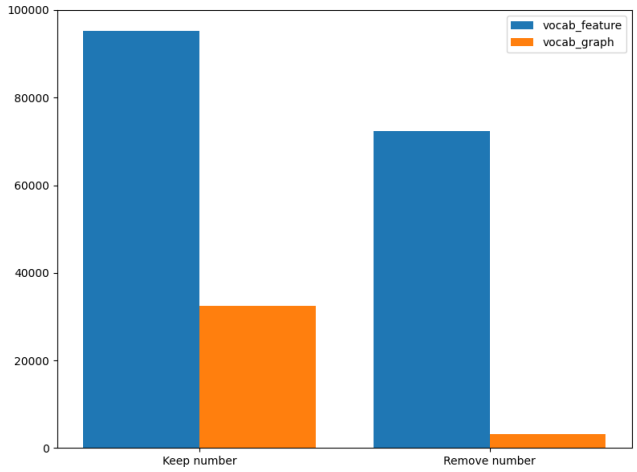

Figure 4: Unique word count before and after replace number with token <num> for both vocab graph and vocab feature

- **Feature Threshold**: Remove words that appear less frequently than a certain threshold in the training and semi-supervised sets.

Sentences with a low character count are removed due to their limited informational value. Numbers are also replaced by a placeholder <num>, as they often occur frequently but also carry minimal information related to my target of HLF detection. These steps helped me to eliminate noise and conserve computing resources. As I expected, observed in feature 4, this technique reduced my vocabulary sizes significantly, over 24% for the vocab feature and 2% for the vocab graph, in turn, this also reduced the overall graph size and complexity. I did not apply lemmatization and stopword removal in my final model. The effects of these techniques will be mentioned in the analysis section 6.5.4. I also encountered a large number of French samples in my dataset. Instead of relying on a deep learning model to detect French, which is computationally expensive, I opted for a simpler approach. I removed sentences that do not contain English stop words. The idea behind this is that stop words should appear in most English sentences. Therefore, if a sentence contains stop words, it should be English. Although there may be an overlap between French and English stop words, I found the error rate to be acceptable. Additionally, this approach has the added benefit of removing short English sentences that usually contain no information related to my problem, such as "TD good".

6.3 Text GCN

I found that the model structure and proposed solution of textGCN worked well with my problem. After the processing mentioned earlier, the model contained 3,133 tokens for the vocab graph and 72,350 for the vocab feature. I used these two sets of vocab to build a train graph of 450,023 nodes and 16,970,415 edges, which included all of my semi and fully-labeled samples. The graph statistics are shown in Table 1. As we can observe, the number of isolated nodes is 0 simply due to the design of textGCN, which forces every word

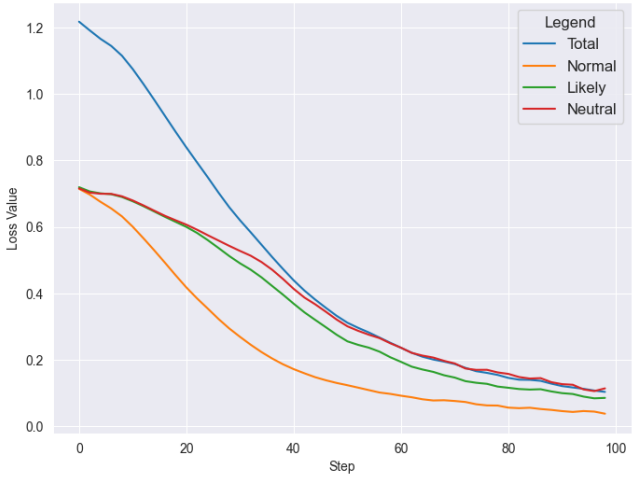

Figure 5: The decrease in different types of loss for textGCN using TF-IDF embedding over 100 epochs.

Figure 6: F1 score weighted and Inference time in seconds

node to be connected to a document node. After appending new samples to the validation graph, the number of nodes increased to 450,092 and the number of edges increased to 16,975,800. The increase in both metrics is attributed to the additional virtual node for validation later on.

Our findings suggest that there are no significant benefits to the model when leveraging advanced embeddings. Looking into the comparison in Table 2, TF-IDF is the best option for my problem. Due to the sparsity of TF-IDF, the actual values only populate around 0.05% of the feature matrix.

For the neural network layer, I employed the standard 1 hidden layer with 300 hidden nodes for each GCN layer. The feature matrix is passed through this fully connected layer and then multiplied with the adjacency matrix to apply the GCN operation. I attempted to increase the number of hidden layers and hidden nodes between GCN layers; however, no increase in performance was found. I also went with the conventional 2-layer GCN since increasing these layers will lead to information being evenly distributed across every node, hammering the model's ability to learn.

6.4 Train and Performance

I divided the loss function into three different categories: normal loss, likely keyword loss, and neutral keyword loss. These 3 losses and the combined total loss are shown in Figure 5 over 100 epochs. As mentioned, likely keywords are words that the TD business team determined to be highly likely to be positive if a sample contains them. Interestingly, the search operation also found some sentences that real negative labels containing likely keywords. This fact is good news for me since it will allow the model to learn to discriminate between real negative classes with likely keywords and pseudo-label positive classes. The same can be said for the neutral keyword class. To control the loss function, I assigned a lambda of 0.5 to the likely group and 0.2 to the neutral group with the idea that the likely group is more important than the neutral group. I do not want them to have too much effect on the real label data.

Upon analyzing the loss graph in Figure 5, I can observe that the normal loss decreased as expected over the course of 100 epochs. Both keyword losses were optimized without any trade-off between them and the normal loss. However, the neutral loss was a bit more difficult to optimize, as it contained samples that were less likely. This shows me that the keyword loss is effective in helping assist in learning the total problem without adding additional noises. Based on Table 3, my newly trained model has outperformed the existing internal model that TD is currently using. I believe a large contribution to this performance is to the specialized dataset. Since the internal model was initially trained for multi-category prediction for multiple purposes, a binary model that focuses on HLFs would naturally be better at the task. This can be observed via the baseline LSTM [Hochreiter and Schmidhuber(1997)] model, which, with no modification, would be able to perform better than the internal model. The textGCN family outperforms the LSTM baseline in many metrics. This is due to the textGCN model's ability to leverage not only the predicted sample but also the surrounding neighborhood to aggregate data, including positive, negative, and unlabelled sets. Compared to other embeddings, I can observe that TF-IDF is significantly better than other embeddings in many aspects. Firstly, pre-trained models such as Doc2Vec or BERT require separate training or fine-tuning processes. Although this helps the model to capture more information in the embedding, it also causes TextGCN to inherit errors from these pre-trained models. The fact that using simple transformations like TF-IDF on TextGCN creates slightly better or comparable performances to these models shows that the TextGCN model structure is state-of-the-art for text classification and suffices to understand the problem. Secondly, pre-trained models can be a black box due to their complexity, and relying on these could open the door to security risks for TD, especially in a high-security environment. Finally, looking at inference time, shown in Figure 6, TF-IDF is significantly faster compared to all the mentioned baselines. This is thanks to the sparsity of TF-IDF, and the model also consumes less memory and computing resources.

	Nodes	Edges	Selfloops	Isolates	Coverage
Train Graph	450023	16970415	0	0	1.0
Val Graph	450092	16975800	0	0	1.0

Table 1: Graph Statistics for the final model using TF-IDF embedding

	Train				Validate			
Model	Acc	F1 weighted	R	P	Acc	F1 weighted	R	P
Intertal	_	_	_	_	0.700	0.772	0.5	0.272
LSTM	0.88	0.908	0.833	0.384	0.826	0.854	0.667	0.285
textGCN Doc2Vec	0.985	0.985	0.909	0.909	0.956	0.954	0.667	0.800
textGCN SBERT	0.992	0.992	1.000	0.916	0.927	0.938	0.833	0.556
textGCN TF-IDF	0.992	0.992	1.000	0.916	0.971	0.9729	1.000	0.75

Table 2: Model Performance Comparison with textGCN different embeddings

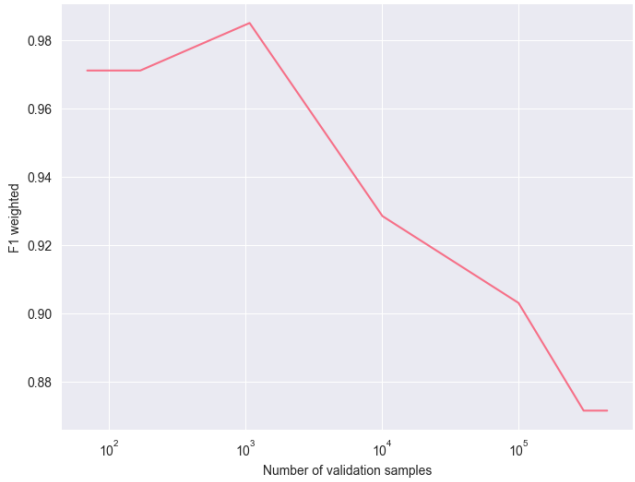

Figure 7: F1 weighted and number of validation samples in an exponential scale

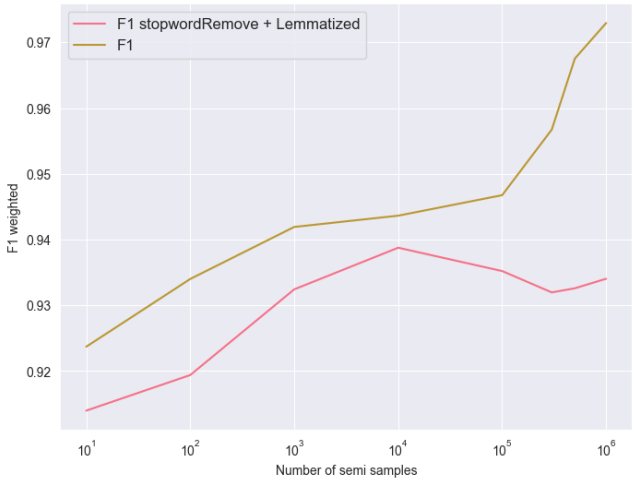

Figure 8: F1 weighted vs additional semi-samples appended to the original train graph to expand the model, break down by whether applying lemmatization and stopword removal.

6.5 Analysis

6.5.1 Inference samples vs Performance . Given the nature of the GCN model, each layer averages the surrounding neighbor nodes to generate a new embedding for the target node. As I proposed to expand the training graph each time a prediction is made, I hypothesize that if the validation set is too large, it could increase the variance in model predictions and consequently affect performance. By plotting the graph comparing the number of samples in the validation set against validation performance in Figure 7, I found this decrease in performance to be true, but only with exponential samples. Our model's performance remained stable until around 1,000 validation samples, after which it dropped dramatically. I attribute this to the introduction of a large volume of unseen data as neighbors, which is incorporated into the graph by the GCN layers and reduces the quality of the seen node embeddings in the graph. Due to this phenomenon, I would recommend splitting future inference sets into multiple small sets and predicting them in small quantities.

6.5.2 Number of semi samples vs Performance . Memory usage is a significant concern for TextGCN due to the necessity of the entire graph for each forward pass, particularly when incorporating unlabeled data. Our objective is to evaluate the trade-off between unlabeled data and model performance. In this section, I studied the effect of semi-samples on my F1-weighted performance. I trained multiple TextGCN models with an exponential increase in semi-samples and found that the increase in semi-samples does increase the model performance. As seen on Figure 8, The relationship between semi-samples and model performance is not completely linear, especially when looking at models that have stop words removed and are lemmatized. The performance peaked at around 10,000 samples but decreased again when going up to 100,000. In contrast, models that have their stop words kept and are not lemmatized have a more consistent relationship, with the performance peaking when using the full set.

I hypothesize that this could be due to the lack of information that removing stop words and lemmatizing causes. As semi-samples

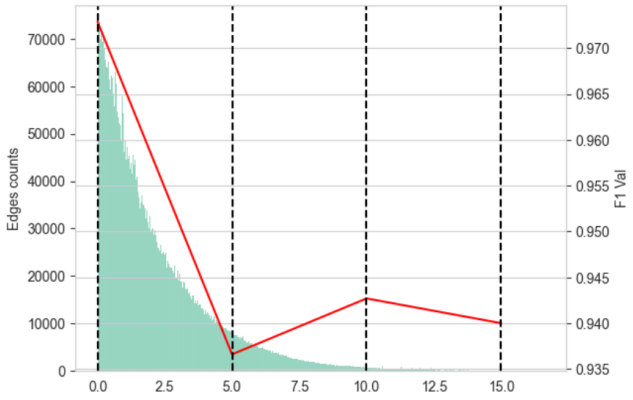

Figure 9: F1 weighted vs PMI thresholds. The figure shows the histogram distribution of the PMI edges. Each of the black horizontal lines is a cut-off threshold, where PMI edges below that number will be cut and trained to find the performance. The right axis shows the validation F1 score weighted of the model after that action is performed.

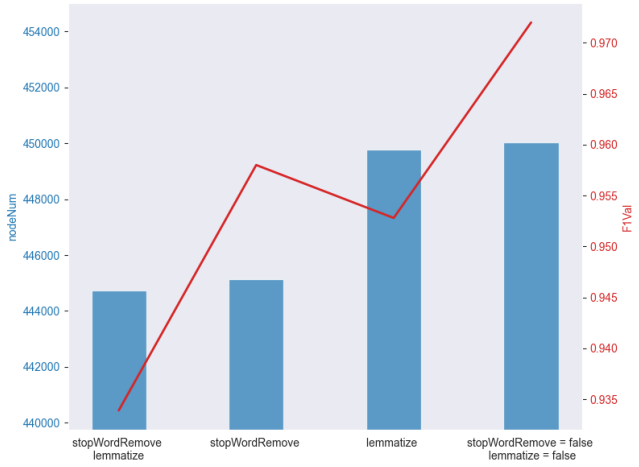

Figure 10: The effectiveness of preprocessing operation to model performance. The figure illustrates 4 combinations between lemmatization and stopword removal and how these models with different combinations performed on the validation set.

increase, more information is averaged into a node, and the node neighborhood becomes more complex to learn. Since I am removing the quality of information by performing this operation, this neighborhood is more noisy and can affect the quality of prediction. This would explain the peak in performance for the first model type as 10,000 is an appropriate amount of information for a node, and more will generate more noise. This will also explain the consistent increase when stop words are kept and not lemmatized since the quality is consistent as the semi-samples increase. However, this phenomenon could contribute to limited validation data, where a small variation in prediction can affect the F1 score, especially when the minority class has a significant weight. I would recommend more research into these effects with more labeled validation samples.

This shows that the model performance increases marginally with the exponential incline of unlabelled samples. However, this relationship is not consistent throughout different configurations, and the trade should be carefully considered in future projects.

6.5.3 PMI vs Performance. Since I am only considering the positive PMI (PPMI), the following is my final model PPMI distribution in Figure 9. On top of the histogram, I overlay different PMI thresholds that I cut off the final model. By measuring the performance of these models, I can understand the effect of these thresholds.

$$ppmi(word1, word2) = \max\left(\log_2 \frac{p(word1, word2)}{p(word1)p(word2)}, 0\right) \quad (11)$$

I found that raising the PMI threshold does affect the model performance. The decrease in performance level was down to around 0.94 after the threshold was raised to 5, however, it stayed at that point even after the complete removal of PMI edges. This shows the case that the performance only decreases down to a point and remain stable afterward. This also indicates that the PMI edges are essential for model learning, but only to a point. Without these edges, TextGCN can still reliably learn the problem, just with weaker performances.

6.5.4 Stop word removal and lemmatization vs Performance. Finally, recent research from [Pradana and Hayaty(2019)] in addition to [DiPietro(2022)] suggest that refraining from removing stop words and not applying lemmatization could enhance the performance of my model. I believe this will be particularly true for TextGCN, as the model relies on words for both information transfer and prediction. I benchmarked the performance of my model with and without the application of stop word removal and lemmatization. The results in Figure 10 show that my model with stop word removal and lemmatization significantly underperforms compared to not applying both processes. This is due to many reasons. First of all, many available stop word libraries not only contain common English stop words like 'a' and 'the', but also many sentiment-related words like 'not' and 'no'. Removing these can dramatically alter the context of a sentence, especially shorter sentences without any additional information. Moreover, lemmatization could reduce the model's confidence by converting a highly sentimental token like 'the best' to the root 'good'. There is also a trade-off between the number of nodes when applying and not applying both operations. With lemmatization and stop word removal, the number of nodes decreases by about 6,000. However, this also reduces the model performance from 0.97 to 0.93 in F1 validation.

6.5.5 Overall trade-off between memory and performance. In this section, I will take a look into how to balance the memory consumption of the modified version of textGCN for industrial usage. Noticed that, so far, I have not mentioned memory consumption in MBs this decision is due to my belief that Mbs usage is affected by many factors, from dataset, length of each sample, number of unique tokens, and how noisy is the data. Instead, I want to base my analysis on graph information like nodes and edges to give an

overview of the relationship. The trade-off between memory and performance is complex, and many different variables interact with each other and affect this trade-off. For my use case, to maximize the model performance, I leverage the maximum semi-sample, keeping the PMI threshold to 0, and do not perform stop word removal and lemmatization. These configs will use memory to a maximum level but also optimize my model performance. Based on my analysis of semi-sample, PMI, and preprocess pipeline, I suggest the following general recommendations for balancing memory and performance for further development and other use cases in an industrial setting. I found an exponential relationship between semi-sample and performance. Therefore, I recommend reducing the number of semi-samples if a reduction in memory is needed since you would have to reduce a significant amount to observe a large drop in performance. This relationship only holds true if I keep stop words and do not apply lemmatization, therefore I would recommend keeping these configs. For the PMI threshold, the model's performance decreases significantly after raising, but only up to a point, the metric will level down and will not decline even after complete PMI removal. If a significant decrease in memory is needed, I recommend the total removal of PMI edges rather than raising the threshold. I acknowledge that the problem is empirical and depends on the problem. However, I found these relationships to be generally true for most of my textGCN applications internally. I also provided a grid search in Table 3 for parameter interaction based on different configurations for anyone interested in studying the interactions or needing a reference for model development.

6.6 Discussion

In this project, I have implemented and modified TextGCN, a graph-based neural network model for text classification, to address the problem of identifying HLFs at TD Bank. I have experimented with different configurations of the model, such as using TF-IDF or Doc2Vec features, removing PMI edges, and leveraging both labeled and unlabeled data via semi-supervised learning. I also incorporate keyword loss by humans to insert human-confident labels into the data, which assists the model in learning and converting. Adding virtual nodes is also proposed to bypass the transitive learning problem. I have evaluated the model's performance using a weighted F1 score and compared it with several baseline methods. Our results show that TextGCN can achieve state-of-the-art results on the HLF classification task, especially when using TF-IDF embedding and keyword loss. I have also observed that adding too large a number of virtual nodes in one inference batch can negatively impact the model's ability to predict and should be carefully regulated. Removing PMI edges can degrade the model's performance, as it may lose some semantic information between words. However, the decline is only to a point and provides an acceptable model if needed. I found an exponential positive relationship between unlabeled data and performance and confirmed the research indicating the positive effect of not applying stop word removal and lemmatization. Based on these findings, I also provided a recommendation guideline on how to balance memory and performance. Some of the limitations and future directions of my project are the following:

- One of the important factors of textGCN is the adjacency matrix; although powerful, this also limited the model to the train data distribution. For some other text classification tasks, like social media, the distribution would shift frequently and require frequent retraining. One could replace the adjacency matrix with another build based on the new distribution and study the model's performance. Theoretically, assuming that the model learned the interaction between words, the neural network layer would be able to generalize with limited loss in performance.
- Due to my limited data capacity, I have used a relatively small and imbalanced dataset, which may expose my model to variation in learning and evaluation. A larger and more balanced dataset may improve the model's performance and provide a more accurate analysis.
- The same keyword loss and analysis method could be applied to other graph-based models or architectures, such as Graph Attention Networks (GAT) [Veličkovi'c et al.(2018)] or Graph Convolutional Networks with Edge Features (GCN-EF) [Liu et al.(2020)], which may offer better performance or interpretability for the HLF classification task.

6.7 Implementation

The project leverages multiple Python libraries for development. For preprocessing, nltk and sklearn's TfidfVectorizer are used to process data and create TF-IDF objects. For graph creation, networkx is mainly employed. And for the modeling section, I used PyTorch to train on the problem.

7 ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my industrial supervisor, Sneha, for her guidance, encouragement, and support throughout this project. She helped me to define the problem, design the methodology, and evaluate the results. I also thank my academic supervisor, Prof. Manos for his valuable insights and suggestions on the scope and direction of my research. He provided me with constructive feedback and useful resources to improve my work. Finally, I am grateful to Prof. Ruth for serving as a committee member and reviewing my paper. Her comments and questions helped me to refine my arguments and presentation.

REFERENCES

[Asgari-Chenaghlu et al.(2020)] Meysam Asgari-Chenaghlu, Mohammad-Reza Feizi-Derakhshi, Leili farzinvash, Mohammad-Ali Balafar, and Cina Motamed. 2020. TopicBERT: A Transformer transfer learning based memory-graph approach for multimodal streaming social media topic detection. *arXiv preprint arXiv:2008.06877* (2020).

[Bastings et al.(2017)] Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Simaan. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 1957–1967.

[Cai et al.(2018)] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[Defferrard et al.(2016)] Micha"el Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.

[DiPietro(2022)] Daniel M DiPietro. 2022. Quantitative Stopword Generation for Sentiment Analysis via Recursive and Iterative Deletion. *arXiv preprint arXiv:2209.01519* (2022).

[Henaff et al.(2015)] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).

[Hochreiter and Schmidhuber(1997)] Sepp Hochreiter and J"urgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[Jiang and Luo(2021)] Weiwei Jiang and Jiayun Luo. 2021. Graph Neural Network for Traffic Forecasting: A Survey. *Expert Systems with Applications* 207 (2021), 117921. arXiv:2101.11174 [cs.LG]

[Karl and Scherp(2023)] Fabian Karl and Ansgar Scherp. 2023. Transformers are Short Text Classifiers: A Study of Inductive Short Text Classifiers on Benchmarks and Real-world Datasets. *arXiv preprint arXiv:2211.16878* (2023).

[Kim(2014)] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. *arXiv preprint arXiv:1408.5882* (2014).

[Kipf and Welling(2016)] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[Kipf and Welling(2017)] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[Le and Mikolov(2014)] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053* (2014).

[Lee(2013)] Dong-Hyun Lee. 2013. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, Vol. 3.

[Li et al.(2018)] Yan Li, Zhi Jin, and Yongkun Luo. 2018. Classifying relations via long short term memory networks along shortest dependency paths. *IEEE Transactions on Knowledge and Data Engineering* 30, 10 (2018), 1790–1803.

[Liu et al.(2016)] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent Neural Network for Text Classification with Multi-Task Learning. *arXiv preprint arXiv:1605.05101* (2016).

[Liu et al.(2020)] Zhenfeng Liu, Jia Li, Yu Wang, Zhiyuan Li, Guoping Zhou, and Maosong Sun. 2020. Graph Convolutional Networks with Edge Features. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 2249–2255.

[Marcheggiani and Titov(2017)] Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 1506–1515.

[Nakajima and Sasaki(2022)] Hiromu Nakajima and Minoru Sasaki. 2022. Text Classification Using a Graph Based on Relationships Between Documents. In *Proceedings of the 36th Pacific Asia Conference on Language, Information and Computation*. Association for Computational Linguistics, Manila, Philippines, 119–125. https://doi.org/10.48550/2022.paclic-1.14

[Ojo et al.(2023)] Olumide Ebenezer Ojo, Hoang Thang Ta, Alexander Gelbukh, Hiram Calvo, Olaronke Oluwayemisi Adebanji, and Grigori Sidorov. 2023. Transformer-based approaches to Sentiment Detection. *arXiv preprint arXiv:2303.07292* (2023).

[Park et al.(2022)] Hyunji Hayley Park, Yogarshi Vyas, and Kashif Shah. 2022. Efficient Classification of Long Documents Using Transformers. *arXiv preprint arXiv:2203.11258* (2022).

[Peng et al.(2018)] Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. 2018. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the 2018 World Wide Web Conference*. 1063–1072.

[Pradana and Hayaty(2019)] Aditya Wiha Pradana and Mardhiya Hayaty. 2019. The effect of stemming and removal of stopwords on the accuracy of sentiment analysis on indonesian-language texts. *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control* 4, 4 (2019), 375–380.

[Reimers and Gurevych(2019)] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 3982–3992.

[Tida and Hsu(2022)] Vijay Srinivas Tida and Sonya Hsu. 2022. Universal Spam Detection using Transfer Learning of BERT Model. *arXiv preprint arXiv:2202.03480* (2022).

[Veličkovi'c et al.(2018)] Petar Veličkovi'c, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*.

[Wang and Manning(2012)] Sida Wang and Christopher D. Manning. 2012. Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, 90–94.

[Wu et al.(2020)] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2020. Graph Neural Networks in Recommender Systems: A Survey. *ACM Computing Surveys (CSUR)* 54, 6 (2020), 1–37. arXiv:2011.02260 [cs.IR]

[Yao et al.(2019)] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 7370–7377.

[Zhang et al.(2018)] Meishan Zhang, Pengfei Liu, and Yubo Song. 2018. End-to-end neural relation extraction with global optimization. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. 88–94.

Number of Semi Labels	PMI Threshold	if removeStopword+lemmatized	F1 Score
10	0	Yes	0.936
10	0	No	0.914
10	5	Yes	0.912
10	5	No	0.891
10	10	Yes	0.889
10	10	No	0.869
10	15	Yes	0.867
10	15	No	0.848
100	0	Yes	0.903
100	0	No	0.9399
100	5	Yes	0.881
100	5	No	0.919
100	10	Yes	0.86
100	10	No	0.899
100	15	Yes	0.84
100	15	No	0.88
1000	0	Yes	0.9447
1000	0	No	0.9563
1000	5	Yes	0.923
1000	5	No	0.936
1000	10	Yes	0.902
1000	10	No	0.916
1000	15	Yes	0.882
1000	15	No	0.897
10000	0	Yes	0.9457
10000	0	No	0.944
10000	5	Yes	0.924
10000	5	No	0.923
10000	10	Yes	0.904
10000	10	No	0.903
10000	15	Yes	0.885
10000	15	No	0.884
100000	0	Yes	0.902
100000	0	No	0.958
100000	5	Yes	0.882
100000	5	No	0.938
100000	10	Yes	0.863
100000	10	No	0.942
100000	15	Yes	0.845
100000	15	No	0.9

Table 3: Grid search for parameter interaction of textGCN