Journal of Positive School Psychology http://journalppw.com
2024, Vol. 8, No. 4, 82-94

Detecting Islamophobic Hate Speech On Social Media Using Semi-Supervised Graph Convolutional Networks

Abdessamade El Ghandour [1], Ismail Akharraz[2]

[1,2] *Mathematical and Informatics Engineering Laboratory, Ibnou Zohr University Agadir, Morocco*
https://orcid.org/0009-0004-0875-5368

Abstract

Islamophobia hate speech is expressed with a generalized negative attitude and behavior toward Muslims and Islam. Speaking out against Muslims has a detrimental effect on how people view Islam. Hate speech towards Muslims has significantly increased on social media during the past few years. This nature of rhetoric encourages violence and prejudice against the Muslim community as well as some violent Muslim reactions. This paper introduces Text Graph Convolutional Networks (TextGCN) model for semi-supervised text classification to identify Islamophobic content on social media. Leveraging a dataset of 1,617 annotated tweets, we applied a semi-supervised approach to classify a larger corpus of 28,000 tweets. This model captures intricate relationships between words, documents, and their interconnections within the text through a heterogeneous graph structure. By using two convolutional layers on this graph, TextGCN achieves state-of-the-art performance, surpassing existing methods with a test accuracy of 93.58% using BERT embeddings and 91.38% using Word2Vec+Doc2Vec embeddings.

Keywords: Islamophobia, social media, TextGCN, Semi-Supervised

INTRODUCTION

Recent years have seen a significant increase in government attention to the frequency, consequences, and transmission of Islamophobic hate speech on social media especially in the West since September 11, 2001 (Christopher 2010). The word "Islamophobia," formed by combining 'Islam' with the suffix '-phobia,' signifies the fear of Islam. According to Conway Runnymede Trust (1997), it encapsulates the fear or hatred directed towards Islam and consequently extends to the fear or aversion towards most or all Muslims. Awan and Zempi (2020) defined Islamophobia as the fear, bias, and animosity directed at Muslims or non-Muslims, resulting in incitement, hostility, and intolerance. This encompasses threats, harassment, abuse, and intimidation in both online and offline spheres. Rooted in institutional, ideological, political, and religious hostility, it manifests as structural and cultural racism targeting symbols associated with being Muslim. Ahrari et al. (2012) elucidated Islamophobia as the term used to signify an irrational suspicion, fear, or refusal of the Islamic faith and individuals who identify as Muslim. In terms of text classification, deep learning models have shown progress and are frequently utilized, although their performance isn't always sufficient when using small labeled datasets. Due to the time-consuming nature of human labeling and the potential need for domain knowledge, the labeled dataset is extremely sparse in many real-world situations. Due to limited availability of labeled training data, semi-supervised text classification offers a

promising approach within the deep learning paradigm. Maximizing the exploitation of unlabeled data's structural and feature characteristics is fundamental to achieving optimal performance in this context.

The proposed methodology for detecting Islamophobic hate speech on social media is based on the following contributions:

1. **Data Labeling:** Labeling Islamophobic hate speech is a complex and challenging task due to the subtle and nuanced nature of this. Identifying Islamophobic hate speech often requires deep understanding of cultural, religious, and social contexts. Moreover, the evolving nature of language and the emergence of new Islamophobic hate speech expressions pose additional difficulties. To label our extensive dataset, we used a semi-supervised approach, and a subset of 1617 annotated documents as a foundation. This labeled dataset served as a starting point for training a model to classify the remaining unlabeled data.
2. **TextGCN:** TextGCN was chosen over other semi-supervised methods due to its ability to effectively capture the inherent graph structure of text data. Unlike traditional recurrent neural networks (RNNs) (Salehinejad et al., 2017) and convolutional neural networks (CNNs) (Jacovi et al., 2018), TextGCN can model long-range dependencies and complex relationships between words and documents. By representing text as a graph, where nodes represent words and documents, and edges represent their relationships, TextGCN can effectively learn from both labeled and unlabeled data. To adapt TextGCN to the specific task of Islamophobic hate speech detection, we incorporated related features into the graph construction process and fine-tuned the model's parameters using a labeled dataset.
3. **Heterogeneous Graphs:** The proposed heterogeneous graph model incorporates multiple types of nodes, including words and documents, as well as different types of edges representing various relationships between these nodes. For instance, W2W edges capture semantic and syntactic similarities between words, while W2D edges represent word occurrences within documents. D2D edges, on the other hand, indicate the similarity or dissimilarity between documents based on their content. By considering these diverse relationships, our model can capture richer semantic and syntactic information compared to traditional homogeneous graph models.
4. **Edge Features:** Edge features were incorporated to provide additional information about the relationships between nodes in the graph. By encoding distance or similarity metrics into the edge features, we can enhance the model's ability to differentiate between important and less important connections. For example, edges connecting semantically related words can be assigned higher weights, indicating a stronger relationship. By incorporating edge features, the model can learn more informative representations and improve its classification accuracy.
5. **Model Performance:** The proposed model demonstrated superior performance compared to state-of-the-art deep learning models such as CNN and LSTM on the task of Islamophobic hate speech detection. The model achieved a high accuracy rate, outperforming baseline models by a significant margin. This improvement can be attributed to the effectiveness of the heterogeneous graph representation, the incorporation of edge features, and the utilization of semi-supervised learning.

RELATED WORK

Islamophobia hate speech has become a popular problem in many circles, offering a challenge to people everywhere (Frieder et al. 2019). Studies on hate speech detection for various target groups regarding gender, race, and community have been more prevalent in recent years. To recognize social maltreatment, researchers have employed a variety of classification approaches.

With the use of Support Vector Machines, Davidson et al. (2017) applied a conventional feature-based classification model that integrates distributional word frequency-inverse document frequency (TF-IDF) and other linguistic variables (SVM). They applied three labels: offensive, neither offensive nor hate speech, and hate speech. The precision, recall, and F1 score they report are all impressive, at 0.91, 0.90, and 0.90 respectively. As they point out, their algorithm does poorly when it comes to hate speech, of which approximately 40% are incorrectly labeled. Their extremely unequal classifications (where 76% of the data falls under the category of "offensive speech") are a major contributor to their high F1 score. They only use one dataset to train and test their classifier, which increases the possibility of overfitting.

The paper "Detecting weak and strong Islamophobic hate speech on social media" by Vidgen and Yasseri (2020), is one of the first works using AI techniques to detect Islamophobic hate speech on social media. Their study revolved around a Twitter dataset obtained from far-right UK political parties, comprising 1,341 tweets labeled as 'Not Islamophobic,' 'Weak Islamophobic,' and 'Strong Islamophobia.' The data underwent annotation by three expert annotators in the fields of UK politics and the study of prejudice. Utilizing an SVM algorithm, they endeavored to identify instances of Islamophobic hate speech. The classifier demonstrated a performance of 74% on the trained dataset and 83% on unseen data across ten-fold.

Khan and Phillips (2021) created a Twitter dataset in English and Hindi using some hashtags about Islamophobia. The data is annotated by three annotators proficient: 'Islamophobic', 'About Islam but not Islamophobic', and, 'Neither about Islam nor Islamophobic'. They used GloVe word embedding with the LSTM model and with the CNN model to classify Islamophobic hate speech. The performance obtained of classifiers over ten-folds was 93% with the English dataset and 92% with the translated Hindi dataset for the LSTM model. 95% with the English dataset and 87% with the translated Hindi dataset for the CNN model.

Badruddin et al. (2022) generated a Twitter dataset to identify Islamophobia sentiment analysis using Twitter API and Islamophobia keywords. They collected about 4339 tweets in diverse languages (31 languages). SVM and LSTM are used to classify the issue of Islamophobia with an accuracy value of 60% for SVM with Polynomial Kernel and 73% for LSTM.

Aldreabi et al. (2023) assembled 2000 comments from Reddit to detect Islamophobia using a list of keywords that contained positive, negative, and neutral Islamic-related terms. Two annotators labeled the comments based on their propensity for Islamophobia assigning binary labels of Hateful and non-Hateful. They used LSTM and CNN as the classifiers and BERT for word embeddings. BERT+CNN achieved the highest performance, obtaining an F1-Score of 92%.

Anwar et al. (2023) Prepared a framework to detect Islamophobic content using NLP

techniques. They extracted 55000 tweets using Twitter API and Islamophobia keywords. The dataset was labeled as 'Islamophobic' and 'Non-Islamophobic' by the voting. They obtained a final dataset containing 5000 tweets. The BERT and LSTM models are used as a classifiers with a performance of 97% for BERT and 93% for LSTM.

In this study, our primary focus is to detect Islamophobic content on social media, taking cues from prior research. Previous work on hate speech detection, particularly in the context of identifying Islamophobia, has demonstrated the potential for crafting a classification system using a graph-based approach. It is imperative to develop a model rooted in this methodology for the explicit purpose of detecting Islamophobia, as far as we are aware, no prior research has specifically focused on the detection of Islamophobia using the graph method.

Methods

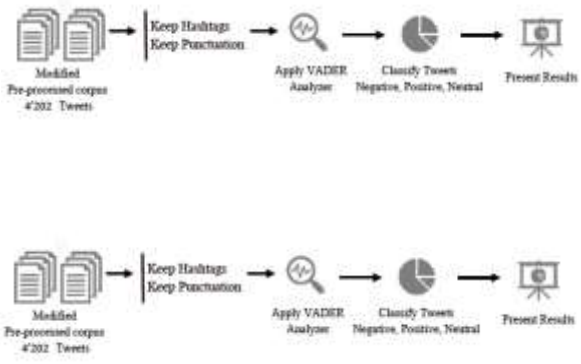
- **Data collection**

While various datasets have been generated for text sentiment analysis and hate speech detection, no complete public dataset that is, specific to Islamophobic hate speech exists to our knowledge. Consequently, we collect a new dataset comprising 28,000 tweets generated by Twitter followers using some trending Islamophobic hashtags like #MuslimsUnderAttack, #Jihadi, #rapethreatsofmuslimwomen #TablighiJamaatVirus. We gathered data over a 3 to 4-month period, roughly between March 2022 and August 2022. The dataset is diverse, encompassing a wide range of user information. Our search didn't prioritize metadata associated with user identities; instead, we focused on text, tweet likes, and timestamps during retrieval.

- **Data annotation**

The unlabeled nature of our dataset presents a significant challenge. We used a semi-supervised approach, involving the search for limited labeled data, and subsequently annotating our dataset using a TextGCN-based model. We proposed an available labeled dataset, CONAN, and filtered out their classes with purely Islamophobic hate speech content. This dataset contains 14,988 Islamophobic documents (Chung et al. 2019). After removing the duplicate documents, we obtained 856 Islamophobic documents. Afterward, to create binary labels, we require non-Islamophobic documents. Consequently, we generated a new dataset comprising 4,337 tweets by utilizing the Twitter API and keywords that contain Islamic-related terms such as 'Muslims' and 'Islam'. After cleaning and pre-processing steps, the data are fetched to the Valence Aware Dictionary for sEntiment Reasoning (VADER) (Clayton and Eric 2014), a sentiment analyzer tool used for sentiment analysis of these tweets to uncover the public's sentiment towards Islam, whether it is negative, positive, or neutral (Figure 1).

Figure 1: The process steps of the VADER sentiment analyze



In this dataset, 1871 tweets are classed as positive tweets, and others tweets are classed as negative or neutral. To establish an evenly-weighted dataset the number of positive tweets is reduced through random sampling to 869 tweets. Table 1 represents the count of documents for each label in the training dataset.

Table 1: Total count of documents for each label.

	Islamophobic	Non-Islamophobic
Dataset(**1,725**)	**856**	**869**

- **Data preprocessing**

In a typical text classification framework, preprocessing is one of the most important components (Uysal et al., 2012). We remove duplicates before proceeding because it is typical on Twitter for people to copy-paste different quotes and retweet anything. The symbol "@" on Twitter allows users to include usernames in their tweets. These are removed from the dataset since they are of no use to our analysis. Normalization is a method of converting all tweets to lowercase letters such that "Token" and "token" are not considered two separate words. Hashtags, like usernames, are not deemed to be of substantial significance for our study and, as a result, are removed. URLs and collecting terms (words used to filter tweets in the first place, such as 'islam,' and 'islams') are next in line to be removed. After that, numbers, punctuation, and special characters (@,&,#, %, etc.) must be removed. The dataset is then cleaned up by removing terms with less than three characters (short words), making feature extraction easier for the study. Tokenization is a crucial step in the pre-processing procedure. It is the process of splitting text into tokens based on whitespaces and saving each word as a separate token. We use Gensim's simple preprocess method to accomplish this step. The next phase is lemmatization and steaming, which are required for many text-mining applications. They consider the context before converting the term to its basic form. Finally, we remove stopwords with no analytic value, such as articles, prepositions, and pronouns, such as 'a,' 'and,' 'the,' etc. The default list can be tweaked and expanded to fit your needs. We've added a few new words to the Natural Language Toolkit's predefined list (NLTK). Table 2 shows the dataset after the pre-processing.

Table 2: Total count of documents for each label after pre-processing

	Islamophobic	Non-Islamophobic
Dataset(**1,617**)	**823**	**794**

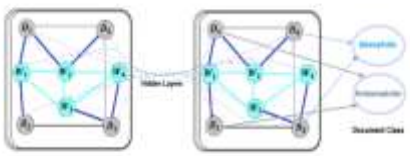
- **GCN-based Text Classification**

The Graph Convolutional Network (GCN) is a semi-supervised framework that combines convolutional techniques with graph data modeling (Kipf and Welling 2016a). It is a multi-layer neural network that works directly on a graph, inducing embedding vectors for nodes based on their neighboring nodes' attributes. With multiple GCN layers, it can capture information about larger communities within the graph. The focus of the discussion is on GCN-based text classification. Figure 2 shown the architecture of typical graph-based text classification models based on level corpus (Mao et al. 2019) and document-level GCN text classification models (He et al. 2019). These models are preferred because they capture global structural information, unlike document-

level GCN models that focus only on local information. The former models enable the analysis of rich relational structures and the preservation of the graph's global structure.

Consider the graph $G = (V; E; A)$, where $V(|V| = N)$ and E, respectively, are sets of nodes and edges while $A \in \mathbb{R}^{N\times N}$ is the graph adjacency matrix. For any v, each node is assumed to be connected to itself, i.e., $(v; v) \in E$. Let $X \in \mathbb{R}^{N\times M}$ be a matrix holding all N nodes and their features, with M being the feature vectors' dimension and each row $x_v \in \mathbb{R}^M$ being the feature vector for v. As illustrated in Figure 2 , we construct a broad and heterogeneous text graph that encompasses both word and document nodes so that global word co-occurrence may be explicitly represented and graph convolution can be easily adjusted. The number of nodes in the text graph N is equal to the sum of the number of documents (corpus size) and unique words (vocabulary size) in a corpus.

Figure 2: Framework of TextGCN. Words and documents are nodes in a single huge network, with co-occurrence between words and documents appearing as edges. Assume that the tweet classification has only two classes.



TextGCN (Mao et al. 2019) is a corpus-based graph that takes all of the words and documents in the corpus as graph nodes. Word occurrence in documents (document-word edges) and word co-occurrence across the corpus (word-word edges) are used to create edges between nodes. The point-wise mutual information (PMI) value, represents the edge between each word pair. The term frequency-inverse document frequency (TF-IDF) of the word in the document is the weight of the edge between a document node and a word node. The Jaccard similarity is used to determine the weight of the edge between each document pair. The created graph is fed into a $s-$layer GCN, $s \in \{0,1,\ldots,S-1\}$, as shown in equation 4, with the $S-$layer node embeddings for both word and document having the same size as the label set and being given to a softmax classifier for output (Equation 5). For training and optimization, the cross-entropy loss is computed over all labelled documents (Equation 6).

In this paper, we typically utilized two input embedding models to represent the node features in GCN-based text classification:

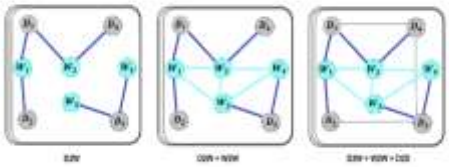
1. **Doc2Vec and Word2Vec:** TextGCN uses a Word2Vec (Corrado et al. 2013) for word nodes and a Doc2Vec (Le and Mikolov 2014) for document nodes to train its word/node feature.
2. **BERT:** TextGCN handle the BERT as one of the most often used contextual word embeddings.

In natural language processing, token representations can be created by combining tokens, segment, and position embeddings. The key distinction lies in how Word2Vec and BERT handle word order and context. Word2Vec produces context-independent embeddings, assigning a single numeric vector to each word. This method merges various meanings of a word into one vector and doesn't consider word order during training. On the

other hand, BERT generates context-sensitive embeddings, allowing for multiple numeric representations of the same word, contingent on its context. BERT's embeddings are context-dependent, capturing a word's meaning within various contexts. BERT takes word order into account by employing the Transformer model with positional encodings to represent word positions, making it a valuable tool for tasks that require contextual understanding in natural language processing.

To construct edges, we aim to comprehensively analyze all possible co-occurring relationships between every pair of node types. As shown in Figure 3, we use D2D edges in addition to W2W and D2W edges. To calculate the edge weight between two documents nodes D2D, we employ Jaccard Similarity.

Figure 3: Various edges construction in an entire corpus-based graph with four documents and four words.



The TF-IDF of the word in the document defines the weight of the edge between a document node and a word node D2W. To calculate weights between two word nodes, we use PMI, a popular measure for word associations. In our preliminary experiments, we discovered that using PMI yields superior results than using word co-occurrence count. The PMI value of a word pair (k, l) is calculated as:

$$\mathrm{PMI}(k,l) = \max\left(\log\frac{p(k,l)}{p(k)p(l)}, 0\right) \qquad (1)$$

$$p(k,l) = \frac{\#W(k,l)}{\#W} \qquad (2)$$

$$p(k) = \frac{\#W(k)}{\#W} \qquad (3)$$

where $\#W(k)$ is the number of sliding windows in a corpus that contain word k, $\#W\,(k,l)$ is the number of sliding windows that contain both word k and l, and #W is the total number of sliding windows in the corpus.

The GCN learning algorithm utilizes the input matrix $L^{(0)} = X$, which comprises the initial P_0 −dimensional features of the N nodes (the sum of the number of documents and the number of unique words in the corpus) in V, and executes layer propagation using the approach in equation (4), which formulates the propagation operation from layer s to the next layer $(s+1)$ Where, $s \in \{0,1,\ldots,S-1\}$, $\widetilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is the normalized symmetric adjacency matrix and D is degree matrix of A where $D_{ii} = \sum_j A_{ij}$. The diagonal elements of A are set to 1 because of self-loops. $W_s \in \mathbb{R}^{P_s \times P_{s+1}}$ is a weight matrix for the s layer and $\mathrm{ReLU}(\mathbf{x}) = \max(0;\mathbf{x})$ is an activation function. Using Word2Vec and Doc2Vec or BERT, the initial feature matrix of nodes is $L^{(1)}$. For both word and document node embeddings, we set the last layer output to be the same size as the label set and apply a softmax classifier to the output.

$$L^{(s+1)} = \mathrm{ReLU}\left(\widetilde{A}L^{(s)}W_s\right) \qquad (4)$$

$$Z = \mathrm{softmax}\left(\widetilde{A}L^{(S)}W_S\right) \qquad (5)$$

The loss function is defined in equation (6) as the cross-entropy error over all labeled

documents

$$\text{Loss} = -\sum_{d \in \mathcal{Y}_D} \sum_{j=1}^{F} Y_{dj} \ln Z_{dj} \qquad (6)$$

Where $\mathcal{Y}_D$ is the collection of document indices with labels, and F denotes the output feature dimension, which is equal to the number of classes ($F = 2$). The label indicator matrix is denoted by the letter Y. Gradient descent can be used to train the weight parameters W_s.

- **Experiments**

In this subsection, we provide an in-depth discussion of the experimental setup and parameter selection. The model is implemented using Google Colab, which provides up to 12.68 GB of free RAM. The scikit-learn and Pytorch deep learning frameworks serve as the foundation for the proposed framework, which is created and implemented in Python 3. We divided the TextGCN experiments into 6 tasks based on the models of node embeddings and edges. For The training hyperparameters, we used two layers in the textGCN training and $2e-5$ as the learning rate; 0.5 as the dropout rate; 0 as the L2 loss weight; 100 as the maximum number of epochs with the early stopping of 10 epochs. We chose 200 for the first convolutional layers embedding size and 20 for the window size. We also experimented with various settings and found that making minor adjustments did not significantly alter the outcomes. As a validation set, we chose 10% of the training set at random. We used an Adam optimizer in the training of our model (Kingma and Ba 2014). In Task-1 we are using only D2W as edges and we calculated the TF-IDF for determining the weight edges. Each word is represented as a node with Word2Vec embeddings, and each document is a node with Doc2Vec embeddings. In Task-2, we maintain D2W as edges, but we employ BERT embeddings for words and documents, using a dimension of 768 based on the "bert-base-uncased" (for English-based) model (Cistac et al. 2019). Task-3 incorporates D2W and W2W as the edges, with Word2Vec embeddings for words and Doc2Vec embeddings for documents. We establish word relationships, utilizing a window size of 20 for PMI calculation in W2W edges. In Task-4, D2W and W2W edges are employed, and BERT embeddings are applied to words and documents. Task-5 introduces D2W, W2W, and D2D edges, with Word2Vec embeddings for words and Doc2Vec embeddings for documents. Weighted edges are constructed using Jaccard similarity measures for D2D edges, with a threshold of 0.2. Finally, Task-6 features D2W, W2W, and D2D edges (full edges), with BERT embeddings used for both words and documents. For each of the tasks mentioned above, We feed the text graph into a simple two-layer GCN. This choice is based on the observation that deep GCNs tend to exhibit suboptimal performance in multi-layer setups. Graph convolution can be viewed as a process of sharing information among neighboring nodes, and if this process is iterated, it leads to a convergence where the features of all nodes become progressively more similar (Kipf and Welling 2016b).

Results and Discussion

- **Results:**

The study's results show that the model's performance is influenced by the choice of edge construction and node embeddings. Table 3 displays the average test accuracy in a 10-fold cross-validation setup for various edge constructions and node embeddings.

Table 3: Test accuracy across different node embeddings and edge constructions.

	Word2Vec +Doc2Vec	BERT
D2W	0.8885	0.8805
D2W+W2W	0.8253	0.8777
D2W+W2W + D2D	0.9138	0.9358

Overall, a graph constructed with only D2W edges consistently performs worse than the full edges. This suggests that D2W co-occurrence alone does not convey enough global structural information. When both W2W and D2D edges are added (full edges), the model's test accuracy improves, indicating more structural information is captured with the model. In this case, it exhibits with the same performance as Word2Vec+Doc2Vec and BERT node embeddings. Where the test accuracy increases with both node embeddings in the full edges. This explains that add other edges construction, more structural information has been captured with the model. However, using D2W with W2W decreases test accuracy to 82.53% with Word2Vec+Doc2Vec embeddings and to 87.77% with BERT embeddings. Moreover, using W2W edges alone is insufficient for performance enhancement, indicating that a heterogeneous graph involving words alone cannot capture all information. Despite both models being pre-trained, there's about a 5% difference in performance between Word2Vec+Doc2Vec embeddings and BERT embeddings. BERT embeddings are trained on a larger dataset, but it is suggested that a smaller dataset might perform better on pre-trained embeddings due to the need for global information.

Overall, on our dataset, the proposed model performs noticeably better than the baseline models, proving the viability of the TextGCN for semi-supervised text classification. Table 4 shows the test accuracy of this baseline compared to the TextGCN model.

Table 4: Test accuracy of various models.

Algorithm	Accuracy
TF-IDF + SVM	0.8209
TF-IDF + LR	0.8456
CNN (Word2Vect)	0.9012
LSTM (Word2Vect)	0.8148
BILSTM (Word2Vec)	0.8827
TextGCN(Word2Vec +Doc2Vec)	**0.9138**
TextGCN(BERT)	**0.9358**

With BERT embeddings or Word2Vec+Doc2Vec embeddings, TextGCN outperforms all baseline models on our dataset and performs well, proving the usefulness of the suggested method on short datasets. For a more thorough performance analysis, we point out that CNN with randomly initialized word embeddings can outperform LSTM with randomly initialized word embeddings on small datasets. BILSTM performs significantly better when pre-trained Word2Vec embeddings are made available. CNN produces high outcomes using trained Word2Vec embeddings on a small text dataset. Similar to the TF-IDF + LR model, shorter datasets improve performance. Worst baselines are achieved with TF-IDF + SVM, although the results on shorter datasets are inferior to others. Despite having higher accuracy than the previous baselines on our dataset, TextGCN consistently outperforms CNN. This illustrates the effectiveness of using full-edge features to preserve rich information.

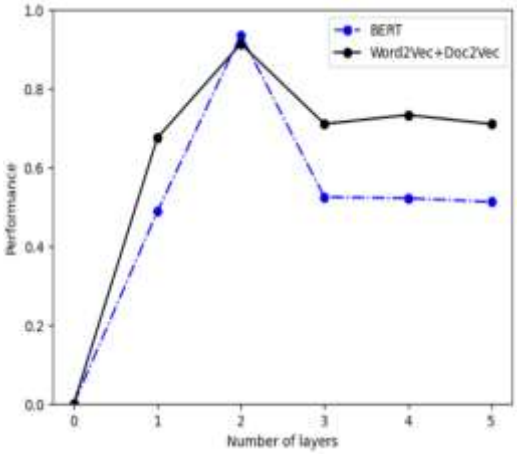
TextGCN with full-edge construction consistently surpasses the baselines in terms of performance. Notably, BERT excels when

employed in full-edge construction, potentially due to the significance of word order in sentiment analysis and the advantage of leveraging pre-trained general semantic knowledge derived from extensive external text data. However, TextGCN still outperforms these pre-trained models across all six tasks, despite not relying on external resources. This underscores the potential benefits of self-exploration within the corpus through the utilization of diverse edge networks, even in the absence of substantial external resources.

- **Discussion:**

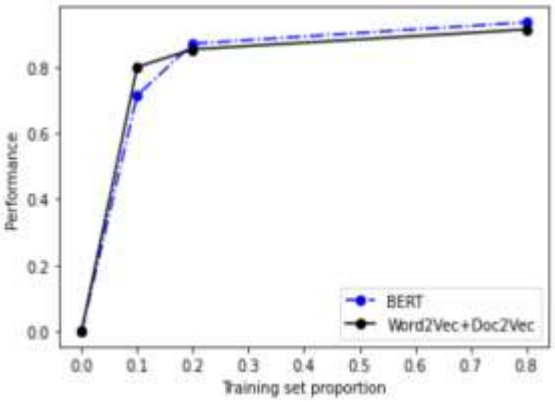
The study delves into graph convolutional network learning, aiming to leverage convolution layers for gathering information from nearby nodes and larger neighborhoods. Through text classification evaluation, we assessed the optimal range of neighbor information for each node, discovering that using two GCN layers for each node yields the best performance. Figure 4 shows the test accuracy of the two embedding models in the full edges. This suggests that capturing two levels of neighborhood nodes is ideal; beyond this, the representation of nodes becomes less effective. Comparing methods, BERT and Word2Vec+Doc2Vec demonstrate more consistent performance across evaluation parameters. Despite limited labeled data in a low-resource language dataset, the study finds that utilizing two layers consistently performs the best overall. Increasing the number of layers shows steady performance between 2 and 5, without a sudden drop in the two-node embeddings.

Figure 4: The performance of the TextGCN model with varying GCN hidden layer stacks applied to the full edges construction and two embedding models.



We evaluated multiple top-performing models with various percentages of the training data to assess the impact of the quantity of the labeled data. Figure 5 shows test accuracies using our training data at 10%, 20%, and 80% applied to a full-edge model with BERT embeddings and Word2Vec+Doc2Vec embeddings. We point out that TextGCN can increase test accuracy even with a small number of labeled documents. For example, TextGCN achieves test accuracy of 0.85 on BERT embeddings and 0.87 on Word2Vec+Doc2Vec embeddings with only 20% training documents, as well as 0.80 with BERT embeddings and 0.71 with Word2Vec+Doc2Vec embeddings using only 10% training documents.

Figure 5: Comparison of test accuracy using various numbers of labeled documents in the training set.



These accuracies surpass certain baseline models even when utilizing the complete set of training documents. This implies that TextGCN effectively spreads document label details throughout the entire graph, indicating its proficiency in this task. Moreover, it indicates that our word document graph maintains comprehensive global word co-occurrence data, contributing to the model's success.

Conclusion

The purpose of our work was to use the graph method for heterogeneous graphs to develop a classifier for detecting Islamophobic content on social media. In this study, we collected about 28,000 tweets with Islamophobia hashtags and introduced a dataset consisting of 1,725 labeled documents. This dataset was employed to label the remaining documents using TextGCN for semi-supervised text classification. It fully exploits both small amounts of labeled data and huge amounts of unlabelled data through rich node and edge information propagation. All graph components are based on the provided text corpus, and we provide several edge features to effectively handle the distance/closeness between words and documents. By surpassing various state-of-the-art on our semi-supervised text classification dataset, TextGCN shows promising results in full edges with the different node embeddings. For semi-supervised text classification in the future, we propose leveraging few-shot learning with small data. It requires a small amount of labeled data for each label the model is expected to recognize. The objective is for the model to generalize quickly and effectively to new unseen data in the same categories. The distant supervision can be used in this task as another approach. It is a successful strategy to generate weakly labeled training data for neural language models.

References

1. Christopher Allen. Islamophobia and Its Discontents. Verso Books, Brussels, 2007.

2. Awan and I. Zempi. A working definition of islamophobia: written evidence submission prepared for the united nations special rapporteur on contemporary forms of racism, racial discrimination, xenophobia and related intolerance. 2018.

3. Iqbal Z. Gazzaz O. B. Khan, F. R. and S. Ahrari. Global media image of islam and muslims and the problematics of a response strategy. Islamic studies, pages 5–25, 2012.

4. Yao H.-R. Yang E. Russell K. Goharian N. MacAvaney, S. and O. Frieder. Hate speech detection: Challenges and solutions. PloS one, 14(8), 2019.

5. Warmsley D.-Macy M. Davidson, T. and I. Weber. Automated hate speech detection and the problem of offensive language, 2017. In Eleventh international aaai conference on web and social media.

6. Z. Waseem and D. Hovy. Hateful symbols or hateful people? predictive features for hate speech detection on twitter, 2016. In Proceedings of the NAACL Student

Research Workshop. San Diego, California, 88–93.

7. C. Ali H. Mulki, H. Haddad and H. Alshabani. L-hsab: A levantine twitter dataset for hate speech and abusive language, 2019. In Proceedings of the Third Workshop on Abusive Language Online. Florence, Italy, 111–118.

8. B. Vidgen and T. Yasseri. Detecting weak and strong islamophobic hate speech on social media. Journal of Information Technology and Politics, 17(1):66–78, 2020.

9. Heena Khan and Joshua L. Phillips. Language agnostic model: Detecting islamophobic content on social media, 2021. Paper presented at 2021 ACM Southeast Conference (ACMSE 2021), April 15–17, Virtual Event, USA. ACM, New York, 5 pages.

10. Badruddin B. Kurniawan, F. and P. A. Wibawa. Identification of islamophobia sentiment analysis on twitter using text mining language detection. Journal of Positive School Psychology, 6(5):8286–8294, 2022.

11. Lee J. M. Aldreabi, E. and J. Blackburn. Using deep learning to detect islamophobia on reddit, 2023. In The International FLAIRS Conference Proceedings, vol. 36.

12. Anwar M. Ali F. Mukhtar R. Jaleel, A. and M. Farooq. Islamophobia content detection using natural language processing. Journal of Computing and Biomedical Informatics, 4(02):88–97, 2023.

13. S.S. Tekiroglu M. Guerini Y.L. Chung, E. Kuzmenko. Conan – counter narratives through nichesourcing: a multilingual dataset of responses to fight online hate speech. ACL, 78:2819–2829, 2019.

14. C. J. Hutto and E. Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text, 2014. Eighth international AAAI conference on weblogs and social media. Michigan: AAAI Press.

15. Gunal S. Ergin S. Uysal, A. K. and E. S. Gunal. A novel framework for sms spam filtering, 2012. In Proceedings of the IEEE international symposium on innovations in intelligent systems and applications. Trabzon, Turkiye.

16. T. N. Kipf and M. Welling. Semisupervised classification with graph convolutional networks, 2017. Paper presented in ICLR.

17. C. Mao L. Yao and Y. Luo. Graph convolutional networks for text classification. Artificial Intelligence, 33(1):7370–7377, 2019.

18. J. Huang Y. Tang X. He M. Tu, G. Wang and B. Zhou. Multi-hop reading comprehension across multiple documents by reasoning over heterogeneous graphs, 2019. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 2704–2713.

19. G. S. Corrado Tomas Mikolov, Kai Chen and J. Dean. Efficient estimation of word representations in vector space, 2013. In International Conference on Learning Representations.