





# KGAT: An Enhanced Graph-Based Model for Text Classification

Xin Wang<sup>1</sup> , Chao Wang<sup>1</sup> , Haiyang Yang<sup>1</sup>, Xingpeng Zhang<sup>1</sup>, Qi Shen<sup>2</sup>, Kan Ji<sup>1</sup>, Yuhong Wu<sup>1</sup>, and Huayi Zhan<sup>3</sup>

<sup>1</sup> Southwest Petroleum University, Chengdu, China

{xinwang, xpzhang, jikan}@swpu.edu.cn

{202021000484, 202021000482, 202121000494}@stu.swpu.edu.cn

<sup>2</sup> Chang'an University, Xi'an, China

2018900932@chd.edu.cn

<sup>3</sup> Sichuan Changhong Electric Co. Ltd, Mianyang, China

huayi.zhan@changhong.com

**Abstract.** As a fundamental task in natural language processing, text classification, which is to predict the class label of a given text, has been intensively studied. Consequently, a host of techniques have been developed, among which techniques that are based on graph neural network and its variant *e.g.*, graph attention network (GAT) achieved impressive performances, as they show superiority in dealing with complex graph-structured data. Despite effectiveness, most of these techniques suffer from several limitations, *e.g.*, incapability in well-capturing correlation among words in a text. In light of these, we propose a comprehensive approach KGAT which incorporates multi-head GAT with enhanced attention and customized ReadOut operation for text classification. (1) Our approach constructs a *text graph*  $G_T$  with edge weights from a text such that both semantic and structural information (with correlation degree) can be well captured. (2) On text graph  $G_T$ , a novel attention mechanism is incorporated in a multi-head GAT for representation learning. (3) Our approach customizes ReadOut operation such that the representation of a text is refined by using a set of influential nodes of  $G_T$ . Intensive experimental studies on both typical benchmark datasets and a newly created one (**Sensitive**) show that our approach substantially outperforms other baseline methods and yields a promising technique for text classification.

**Keywords:** Graph attention network · Multi-head attention mechanism · Text classification

## 1 Introduction

Text classification is a classic problem in the field of natural language processing (NLP) and provides fundamental methodologies for other NLP tasks, such as topic labeling, sentiment analysis, intent detection, cyberspace security, and so

on. The problem has been investigated from the perspective of machine learning and was settled by techniques based on Naive Bayes [6], k-Nearest Neighbors [18], Support Vector Machines [3] and so on. However, these traditional techniques rely heavily on feature engineering for text representation, which leads to high labor costs and low efficiency. In recent year, with the rapid development of deep learning, neural-network-based techniques were involved to address the problem, *e.g.*, TextCNN [7], TextRNN [11], TextRCNN [8], etc. In particular, Graph Neural Network [17] (GNN), a special kind of neural network, is leveraged for the task and achieves excellent performances.

Deep learning based techniques rely on text representation heavily. In light of this, various unsupervised methods are proposed to learn word or document representations. The emergence of word vector models such as GloVe [16] and Word2Vec [13] provide solutions to transform text data from high-dimensional, high-sparse forms into continuous dense data, similar to the transformation on images and speeches. However, the transformation on a sentence is often processed sequentially with the embedding of each word, which ignores (potentially important) structural information among words/phrases within a text. To tackle the issue, investigators advocate expressing sentences with graph structures, which can well express relationship among objects. While, this task is nontrivial for classic deep learning based techniques. Fortunately, graph neural network (GNN) is proposed shortly and showed strong capability in dealing with graph data. GNN is first proposed by [17]. Then [15] proposed a graph-CNN model for text classification and achieved better performance than classical models, *e.g.*, CNN, LSTM. Essentially, GNN-based models transform a serialized text into a graph, thus node-level representation can be refined by referencing the underlying topological structure. Moreover, graph embedding, which expresses graph nodes or subgraphs in the form of vectors, provides a new type of representation for the task of classification. Following this way, [25] proposed TextGCN that builds a graph to capture the relationship of words that appeared in the entire corpus for text classification, while different meanings of the same word were not considered. Text-level-GNN [4] and Texting [26] are extensions of TextGCN, still, they did not consider the weight of each edge when constructing the graph. However, different neighbor nodes have different effects on word nodes, which should not be simply omitted.

In response to the above problems, we propose a novel text classification approach based on GAT. Instead of building a single corpus level graph, we produce a sentence level graph, referred to as *text graph*, for each input text. The *text graph* can well capture correlation relationship among words, which facilitates the calculation of attention coefficients. We improve multi-head GAT with enhanced attention mechanism for node-level feature learning. We also develop a new ReadOut function for finalizing structure-level representation, in particular, a wise strategy is incorporated for influential nodes identification. Via experimental studies, the method we proposed shows superiority in various datasets. To sum up, our contributions are as follows:

- We propose to construct an undirected weighted graph to better capture the correlation strength of the words within a text.
- We develop a multi-head GAT with enhanced attention mechanism. This new model substantially improves representation learning at node-level.
- We customize the ReadOut operation to finalize graph-level representations. In particular, an effective heuristic method for independent set searching is employed to identify influential nodes.
- We produce a labeled dataset **Sensitive**. As far as we know, the dataset is the first Chinese dataset in cyberspace security and fills a critical void in the area.
- We conduct intensive experiments on both benchmark datasets and **Sensitive**. The performance of our approach illustrates its superiority compared to other competitive baseline models.

## 2 Related Work

We now review text classification techniques that are based on deep learning and graph neural networks.

*Methods Based on Deep Learning.* For the characteristic of automatically learning high-dimensional features, deep learning models, such as CNN [7], RNN [14], are also applied to text classification tasks, which avoid tedious manual feature engineering and perform better than traditional machine learning methods. And the attention mechanism is introduced to strengthen the expressive ability of the models. Hierarchical attention networks (HAN) [24] and Attention-based LSTM [1, 20, 29] attention to networks. However, it is usually difficult for the local sliding window in the sequential learning model to capture the dependencies between words far apart in long sentences.

*Methods Based on GNNs.* Graph Neural Networks (GNNs) have attracted much attention [23, 27, 28] for their powerful representation capability in dealing with unstructured data. In GNNs, the text classification problem is abstracted as a graph node classification problem. To enable the GNN-based model to support online testing and reduce memory consumption, Text-level-GNN [4] builds graphs for each input text to obtain global information. And TextING [26] builds individual graphs for each document and learns text-level word interactions by GNN to effectively produce embeddings for obscure words in the new text. Graph theory is also combined with convolution to solve the task of text classification. TextGCN [25] builds a heterogeneous graph model and extracts co-occurrence information between overall words. And SGC [22] reduces unnecessary complexity and redundant calculations by iteratively eliminating nonlinearities and collapsing weight matrices between consecutive layers. Attention mechanism can enhance the feature learning ability of networks and is also introduced into GNNs. Graph attention network (GATs) calculates the attention weight of the neighbors of the source node. Deep attention diffusion graph neural network (DADGNN) [12] captures the connection between a word and its distant neighbors at the node-level attention layer to obtain a more accurate document-level

representation. HyperGAT proposes to learn text embeddings by applying hyper-graphs over documents. However, the aforementioned models pay less attention to polysemy and edge weight.

### 3 Method

The working flow of our approach is shown in Fig. 1a. As can be seen, the model, denoted by KGAT, consists of three parts, *i.e.*, Text Graph Construction, Message Passing, and ReadOut. We next illustrate them in details.

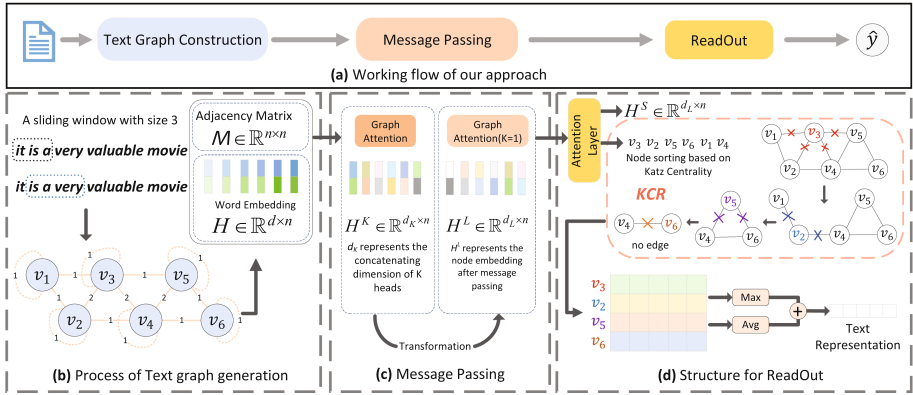


Fig. 1. The working flow and structure of KGAT

#### 3.1 Text Graph Construction

Let  $\mathbf{T} = [t_1, t_2, \dots, t_n]$  denote a text, where each  $t_i$  refers to the  $i$ -th word of  $\mathbf{T}$ . Given such a text  $\mathbf{T}$  to be classified, our approach converts it into a *text graph*, that incorporates both semantic and structural information of  $\mathbf{T}$ .

*Text Graph.* The construction process of a text graph works as follows.

(I) A sliding window with size  $l$  ( $l < |\mathbf{T}|$ ) is initialized and then moved word by word on  $\mathbf{T}$  until reaching the rightmost side. During the period, if a pair of words are *covered* by the window, their co-occurrence frequency will be increased by one. After the above process, the co-occurrence frequency of each pair of words is obtained.

(II) The *text graph*  $G_T = (V, E, f_v, f_w)$  is generated by including a set of nodes in  $V$  such that each node  $v_i$  in  $V$  corresponds to a word  $t_i$  in  $\mathbf{T}$  and a set of edges  $(v_i, v_j)$  in  $E$  if the co-occurrence frequency  $\tau(v_i, v_j)$  (or  $\tau_{ij}$  for short when it is clear from context) of  $v_i$  and  $v_j$  is above zero. Moreover, each node  $v_i$  in  $V$  carries a tuple  $f_v(v_i)$  consisting of the node id of  $v_i$  and a  $d$ -dimensional vector  $\mathbf{h}_i \in \mathbb{R}^d$  corresponding to the embedding of  $t_i$ . Each edge  $e = (v_i, v_j)$  in  $E$  takes an integer  $f_w(e)$  as the weight of  $e$ , where  $f_w(e) = \tau_{ij}$ .

From graph  $G_T$ , one can immediately obtain two matrices  $\mathcal{H}$  and  $\mathcal{M}$ . The matrix  $\mathcal{H}$  is defined as  $[\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n] \in \mathbb{R}^{d \times n}$ , where  $\mathbf{h}_i$  ( $i \in [1, n]$ ) indicates the word embedding of  $i$ -th word in  $\mathbf{T}$ . For the (adjacency) matrix  $\mathcal{M}$ , its entry  $a_{i,j}$  indicates the edge weight  $f_w(v_i, v_j)$  of  $(v_i, v_j)$ . Taking the sentence “it is a very valuable movie” from a benchmark dataset as an example, by using a sliding window with  $l = 3$ , one can obtain a *text graph* along with its adjacency matrix as shown in Fig. 1b.

### 3.2 Message Passing with Enhanced GAT

Given a text graph  $G_T$ , a message passing layer (MPL) is developed to aggregate neighborhood information of each node in  $G_T$ . A key feature of our MPL lies in that the aggregation is performed via an enhanced multi-head GAT, which considers not only influences from neighborhood but also their strengths, *i.e.*, edge weights  $\tau_{ij}$ . Due to space constraints, we focus on key features of MPL, while omit details of the structure of a GAT, as more information can be found in [19].

*Message Passing.* Our MPL consists of an enhanced multi-head GAT followed by a single head GAT.

$$\mathcal{H}^K = \mathbf{EGAT}_K(\mathcal{H}, \mathcal{M}), \quad (1)$$

$$\mathcal{H}^L = \mathbf{EGAT}_1(\mathcal{H}^K, \mathcal{M}), \quad (2)$$

where  $\mathbf{EGAT}_K$  (resp.  $\mathbf{EGAT}_1$ ) denotes the operation of our GAT layer with with  $K$  heads (resp. a single head),  $\mathcal{H}^K \in \mathbb{R}^{d_K \times n}$  is the output of  $\mathbf{EGAT}_K$  and  $\mathcal{H}^L \in \mathbb{R}^{d_L \times n}$  as the output of  $\mathbf{EGAT}_1$  is the the final result of our MPL. In fact,  $\mathbf{EGAT}_K$  concatenates different features from multiple heads, by following Eq. 3, that is defined as follows.

$$\mathbf{h}'_i = \left\|_{\kappa=1}^K \sigma \left( \sum_{j \in \mathcal{N}_i} \alpha_{ij}^\kappa \mathbf{W}^\kappa \mathbf{h}_j \right) \right\|, \quad (3)$$

where  $\|$  represents concatenation,  $K$  is the number of heads,  $\sigma$  represents the nonlinear function,  $\mathcal{N}_i$  represents all direct neighbors of  $v_i$ ,  $\mathbf{W}^\kappa$  is a learnable weight matrix, which is shared by all nodes in the  $\kappa$ -th head. Note that  $\alpha_{ij}^\kappa = \text{Softmax}(\beta_{ij}^\kappa)$  is the normalized enhanced attention coefficient of  $v_j$  to  $v_i$  computed by the  $\kappa$ -th head, and  $\beta_{ij}^\kappa$  is the enhanced attention coefficient, which indicates the importance of  $v_j$  to  $v_i$ .

$$\beta_{ij}^\kappa = \text{LeakyReLU}(\mathbf{a}^\kappa [\mathbf{W}^\kappa \mathbf{h}_i \| \mathbf{W}^\kappa \mathbf{h}_j]) \tau_{ij}, \quad (4)$$

For a pair of embedding  $\mathbf{h}_i$  and  $\mathbf{h}_j$  at  $\kappa$ -th head ( $\kappa \in [1, K]$ ), a matrix  $\mathbf{W}^\kappa \in \mathbb{R}^{d' \times d}$  is used for linear transformation. Two embedding are then concatenated through the operation  $\|$  in Eq. (3), and transformed via a learnable vector  $\mathbf{a}^\kappa \in \mathbb{R}^{1 \times 2d'}$ . Afterwards, LeakyReLU is applied as the activation function, followed by a transformation imposed by  $\tau_{ij}$ . Note that by involving  $\tau_{ij}$  in the attention

mechanism, our MPL is able to incorporate the correlation degree of words  $t_i$  and  $t_j$  in a text, and hence can capture attention coefficients more accurately.

After operation via MPL, each node  $v$  in  $G_T$  aggregates feature information of all its direct neighbors, indicating that the representation of  $v$  is refined by referencing its context information.

### 3.3 ReadOut for Prediction

After process through MPL, a customized ReadOut operation (shown in Fig. 1d) is developed for text classification.

Attention Layer. The node representation  $\mathcal{H}^L$  of a  $G_T$  is updated via an attention layer. We then obtain a new representation  $\mathcal{H}^S \in \mathbb{R}^{d_L \times n}$ , which is defined as:

$$\mathcal{H}^S = \sigma(\mathbf{W}_1 \mathcal{H}^L + \mathbf{b}_1) \odot \tanh(\mathbf{W}_2 \mathcal{H}^L + \mathbf{b}_2), \tag{5}$$

where parameters  $\mathbf{W}_1 \in \mathbb{R}^{1 \times d_L}$ ,  $\mathbf{W}_2 \in \mathbb{R}^{d_L \times d_L}$ ,  $\mathbf{b}_1, \mathbf{b}_2 \in \mathbb{R}^n$  are learned during training;  $\sigma$  and  $\tanh$  are typical non-linear functions;  $\odot$  represents the dot product of matrices. Indeed, the former part works as an attention mechanism, while the latter part is for non-linear transformation.

Identifying Influential Nodes. To predict the class label of a text, some of its words *e.g.*, stop words, are often not helpful. To downplay the influences from those useless words, it is necessary to identify *influential* nodes in  $G_T$  and obtain a representation from them for classification. To this end, we compute *Katz Centrality Ranking* (KCR) of the nodes in  $G_T$  and picks influential ones via KCR. Briefly, katz centrality [21] is a variant of eigenvector centrality that not only considers influences *e.g.*, centrality, from direct neighbors, but also leverages a coefficient to adjust centrality of the central node itself. The operation to obtain the katz centrality is defined as follows:

$$C_{Katz} = (I - \gamma \cdot \mathcal{M})^{-1} \delta, \tag{6}$$

where  $C_{Katz} \in \mathbb{R}^n$  is a  $n$  dimensional vector with each entry corresponding to the katz centrality of a node, and  $n$  is the numbers of nodes in  $G_T$ ; constant  $\gamma$  is a damping factor and usually set to be less than the largest eigenvalue  $\lambda$ , *i.e.*,  $\gamma < \frac{1}{\lambda}$ ; and constant  $\delta$  serves as a bias;  $I$  and  $\mathcal{M}$  represent the identity matrix and adjacency matrix, respectively.

Given  $C_{Katz}$ , influential nodes can be identified as follows. (a) Nodes in  $G_T$  are sorted according to their centrality specified in  $C_{Katz}$ . (b) Nodes with higher centrality are picked repeatedly, until each edge of  $G_T$  has at least one end point in a set  $\mathcal{Z}$ , that is used for maintaining influence nodes. Essentially, above process simulates the progress of identifying an independent set from a graph. As shown in Fig. 1d, a sorted list  $\{v_3, v_2, v_5, v_6, v_1, v_4\}$  is obtained according to  $C_{Katz}$  of  $G_T$ ; then  $v_3, v_2, v_5$  and  $v_6$  are selected as influential nodes as they form an independent set of  $G_T$ . Now, we are ready to generate a representation for  $G_T$ .

Graph Representations. Based on the set  $\mathcal{Z}$  of influential nodes and their representations  $\mathcal{H}^S$ , a pooling operation, specified in Eq. 7 is performed to obtain

a new representation  $\mathcal{H}_\eta \in \mathbb{R}^{d_\nu}$ , that is used for classification. Intuitively, the pooling with `avg` averages the features of all the influential words, while the other operation `max` is to highlight the role of the most influential word.

$$\mathcal{H}_\eta = \text{avg}(\mathbf{h}_1^S, \dots, \mathbf{h}_{|\mathcal{Z}|}^S) + \text{max}(\mathbf{h}_1^S, \dots, \mathbf{h}_{|\mathcal{Z}|}^S), \quad (7)$$

where  $\mathbf{h}_i^S$  ( $i \in [1, |\mathcal{Z}|]$ ) represents the feature of the  $i$ -th node in  $\mathcal{Z}$ .

*Prediction.* Given the text representation  $\mathcal{H}_\eta$ , it is fed into the multi-layer perceptron with a single layer for prediction. In particular, the `Softmax` and cross-entropy functions are used for loss evaluation:

$$\mathcal{L}_{\text{loss}} = - \sum_i y_i \log(\hat{y}_i), \quad (8)$$

where  $\hat{y} = \text{Softmax}(\mathbf{W}_c \mathcal{H}_\eta + \mathbf{b}_c)$  is the predicted label, and weight  $\mathbf{W}_c$ , bias  $\mathbf{b}_c$  are trainable parameters.

## 4 Experiments

In this section, we conduct comprehensive experimental studies to show the performance of our model.

### 4.1 Experimental Setup

*Datasets.* For fair comparison, we used a set of typical benchmark datasets for text classification. Table 1 shows the summary of the datasets we used. In a nutshell, the datasets can be categorized into two types, one for long corpus and the other one for short corpus. Specifically, *R8* and *R52* are subsets of Reuters 21578 datasets. *MR* is a movie review dataset for binary sentiment classification. *SST-1* and *SST-2* are extension of *MR*. *TREC* [9] is a question dataset. *Sensitive*<sup>1</sup> is a dataset manually labeled by us. It contains 15035 short texts in Chinese and is classified into six types: drugs, violence, accidents, gambles, covid-19, and others.

**Table 1.** Summary statistics of the datasets.

Dataset	Long Corpus			Short Corpus				
	R8	R52	Ohsumed	MR	SST-1	SST-2	TREC	Sensitive
#Docs	7,674	9,100	7,400	10,662	11,855	9,613	5,952	15,035
#Train	5,485	6,532	3,357	7,108	9,645	7,792	5,452	12,028
#Test	2,189	2,568	4,043	3,554	2,210	1,821	500	3,007
Avg.Length	41.90	44.37	79.57	18.46	16.80	16.92	10.63	10.73
Max.Length	247	248	192	46	46	46	31	52
#Class	8	52	23	2	5	2	6	6

<sup>1</sup> <https://github.com/do-Hines/textGAT-MI.git>.

**Table 2.** Accuracy (%) on benchmark datasets. We report results as mean  $\pm$  standard deviation after 10 runs. The bold font and underline are the champion and runner-up respectively.

Model	R8	R52	Ohsumed	MR	SST-1	SST-2	TREC
TF-IDF+LR	93.74 $\pm$ 0.00	86.95 $\pm$ 0.00	54.66 $\pm$ 0.00	74.59 $\pm$ 0.00	41.18 $\pm$ 0.00	79.63 $\pm$ 0.00	96.81 $\pm$ 0.00
CNN	95.71 $\pm$ 0.52	87.59 $\pm$ 0.48	58.44 $\pm$ 1.06	77.75 $\pm$ 0.72	42.30 $\pm$ 0.41	80.27 $\pm$ 0.42	93.62 $\pm$ 0.55
Bi-LSTM	96.31 $\pm$ 0.33	90.54 $\pm$ 0.91	49.27 $\pm$ 1.07	77.68 $\pm$ 0.86	42.63 $\pm$ 0.66	80.11 $\pm$ 0.49	93.32 $\pm$ 0.72
CNN-BiLSTM	96.66 $\pm$ 0.61	92.62 $\pm$ 0.49	52.21 $\pm$ 0.23	76.62 $\pm$ 0.39	43.12 $\pm$ 0.40	81.93 $\pm$ 0.46	94.12 $\pm$ 0.51
fastText	96.13 $\pm$ 0.21	92.81 $\pm$ 0.09	57.70 $\pm$ 0.49	75.14 $\pm$ 0.20	36.08 $\pm$ 0.81	81.45 $\pm$ 0.16	91.29 $\pm$ 0.69
Text-GCN	97.07 $\pm$ 0.10	93.56 $\pm$ 0.18	68.36 $\pm$ 0.56	76.74 $\pm$ 0.20	40.65 $\pm$ 0.06	81.25 $\pm$ 0.09	91.40 $\pm$ 0.39
SGC	97.20 $\pm$ 0.10	94.00 $\pm$ 0.20	68.50 $\pm$ 0.30	75.90 $\pm$ 0.30	41.63 $\pm$ 0.41	76.22 $\pm$ 0.13	92.29 $\pm$ 1.26
Text-level GNN	97.80 $\pm$ 0.20	94.60 $\pm$ 0.30	69.40 $\pm$ 0.60	75.47 $\pm$ 0.60	43.02 $\pm$ 0.65	81.75 $\pm$ 0.36	94.09 $\pm$ 0.36
HyperGAT	<u>97.97<math>\pm</math>0.23</u>	94.98 $\pm$ 0.27	69.90 $\pm$ 0.34	78.32 $\pm$ 0.27	41.96 $\pm$ 0.35	81.26 $\pm$ 0.72	93.55 $\pm$ 1.79
DADGNN	<b>98.15<math>\pm</math>0.16</b>	<b>95.16<math>\pm</math>0.22</b>	-	78.64 $\pm$ 0.29	45.15 $\pm$ 0.26	<b>84.32<math>\pm</math>0.15</b>	97.99 $\pm$ 0.52
w/o edge weights	97.34 $\pm$ 0.15	94.70 $\pm$ 0.33	<u>70.16<math>\pm</math>0.43</u>	<u>78.75<math>\pm</math>0.34</u>	<u>45.71<math>\pm</math>0.53</u>	83.42 $\pm$ 0.23	<u>98.16 <math>\pm</math>0.25</u>
w/o KCR	97.30 $\pm$ 0.21	94.61 $\pm$ 0.29	69.88 $\pm$ 0.52	78.64 $\pm$ 0.53	45.45 $\pm$ 0.50	83.31 $\pm$ 0.35	97.98 $\pm$ 0.38
KGAT(ours)	97.41 $\pm$ 0.16	<u>95.00<math>\pm</math>0.33</u>	<b>70.24<math>\pm</math>0.32</b>	<b>79.03<math>\pm</math>0.30</b>	<b>45.83<math>\pm</math>0.35</b>	<u>83.71<math>\pm</math>0.43</u>	<b>98.18<math>\pm</math>0.27</b>

*Baselines.* We consider three types of models as baseline methods.

- Traditional machine learning method TF-IDF+LR.
- Traditional deep learning methods, *e.g.*, CNN [7], Bi-LSTM [11], CNN-BiLSTM [10], and fastText [5].
- Graph-based methods, *e.g.*, Text-GCN [25], SGC [22], Text-level GNN [4], HyperGAT [2], and DADGNN [12].

*Parameter Settings.* In our test, we used the following settings: batch size of 512, initial learning rate of 0.001, sliding window of size 5. To avoid over-fitting, we also adopt the dropout operation with a rate of 0.5. To calculate KCR,  $\gamma$ ,  $\delta$  are fixed as 0.01 and 1, respectively. We implemented an 8-head KGAT (by default) and used the Adam optimizer to train KGAT for 200 epochs with early-stopping strategy. The length of the text we intercept varies according to different datasets. For typical benchmark datasets, their original split for training and testing is followed; for Sensitive, we randomly pick 80% as training and use the remaining for testing (see Table 1 for details). We used pre-trained GloVe word vectors [16] with  $d = 300$  as the default input features while out-of-vocabulary words are randomly sampled from a uniform distribution [-0.01, 0.01].

*Evaluation Metrics.* On benchmark datasets, Accuracy is used as the evaluation metric. While on Sensitive, Precision, Recall, F1-Score, and Accuracy are used.

## 4.2 Prediction Accuracy

We show the prediction accuracy of our approach vs. baseline models on both benchmark datasets and Sensitive.



*Comparison on Benchmark Datasets.* Table 2 shows the accuracy of various models on benchmark datasets. We find the following. (1) KGAT performs better than baseline models, as it ranks top 1 *w.r.t.* accuracy on 4 datasets and top 2 on *R52* and *SST2*. (2) On 4 datasets with short text, our KGAT achieve the best performance on three and reached second place on the *SST2*. This shows that our approach works well on short texts. (3) On *R8* with long text, all models achieve high accuracy. Though KGAT performs slightly worse than some GNN models, it still beats other counterparts, showing that it can effectively capture long-distance semantic relations.

**Table 3.** The performance of different models On Sensitive.

Model	Precision	Recall	F1-Score	Accuracy
CNN	92.58±0.60	91.83±0.40	92.17±0.43	94.01±0.27
Bi-LSTM	91.20±0.75	91.02±0.67	91.02±0.69	93.25±0.52
CNN-BiLSTM	92.63±0.53	92.16±0.74	92.36±0.53	94.25±0.39
fastText	89.15±0.08	87.17±0.09	88.12±0.08	91.15±0.09
Text-GCN	89.59±0.23	90.36±0.30	89.93±0.25	92.32±0.17
SGC	86.18±0.21	89.49±0.13	87.71±0.17	89.69±0.18
Text-level GNN	93.64±0.21	93.80±0.11	93.70±0.18	94.21±0.17
w/o edge weights	95.57±0.23	95.53±0.21	95.54±0.18	96.48±0.15
w/o KCR	96.12±0.30	95.56±0.30	95.82±0.19	96.68±0.15
KGAT(ours)	<b>96.23±0.30</b>	<b>95.60±0.32</b>	<b>95.90±0.27</b>	<b>96.78±0.22</b>

*Comparison on Sensitive.* Results on Sensitive are shown in Table 3. Our KGAT exhibits the best performance on all metrics, increasing more than 2% points. It demonstrates that KGAT works quite well on Chinese dataset. Since Sensitive is a dataset regarding cyberspace security, the excellent performance of KGAT on Sensitive also shows that our method is of great practical significance in the field of cybersecurity.

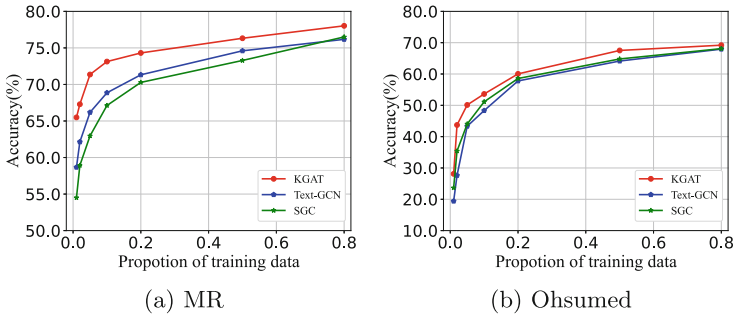
*Ablation Study.* To investigate the contribution of each module in KGAT, we conduct a series of ablation studies on all evaluation datasets. Concretely, *w/o edge weight* is a variant that calculates attention coefficient without edge weight, and *w/o KCR* is a variant that Readout without Katz Centrality Ranking. The results are shown in the last three rows of Tables 2 and 3, respectively. We find that including edge weights when calculating attention improves accuracy. This observation verifies that a text graph with edge weight can better capture the contextual relationship between words, which is beneficial for calculating more accurate attention coefficients. Moreover, the performance gap between *w/o KCR* and KGAT shows the effectiveness of choosing key nodes for graph-level representation.

*Inductive Capability.* To examine the inductive capability of KGAT, we vary the proportion of training data from 1% to 80% on *MR* and *Ohsumed*. Two baseline models Text-GCN and SGC are used for comparison. Figure 2 shows that (1) KGAT achieves the best accuracy, showing a better capability to summarize new words; and (2) all models perform better with larger training data, as expected.

### 4.3 Supplementary Studies

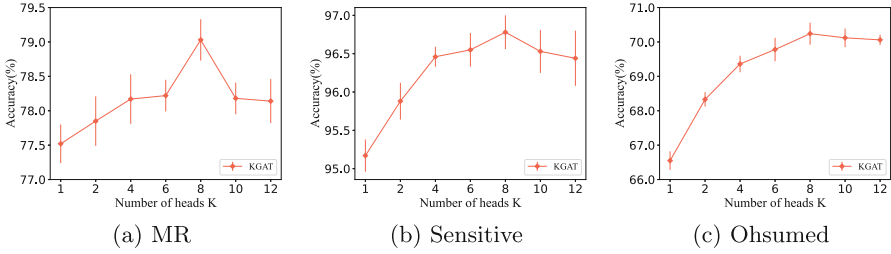
We conduct three supplementary experiments to reveal influences caused by hyper-parameters.

*Number of Heads.* To see how model performance is influenced by the change of head numbers, we conduct a supplementary study *w.r.t.* varied heads. Figure 3 shows the accuracy changes under varied head numbers on *MR*, *Sensitive* and *Ohsumed*, respectively. As can be seen, starting from  $K = 1$ , the accuracy increases when the number of attention heads increases. While the accuracy decreases in both datasets when  $K > 8$ . It shows that multi-head attention with appropriate head numbers can improve model performance.

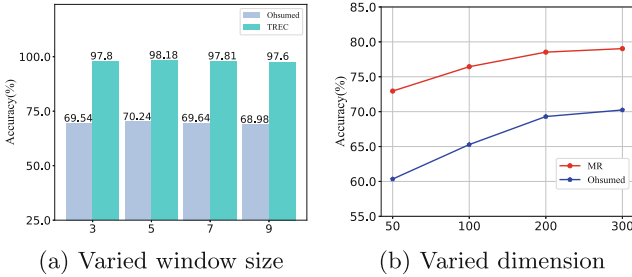


**Fig. 2.** Test with varied training data (1%, 2%, 5%, 10%, 20%, 50%, 80%) on MR and Ohsumed. The less training data is, the more new words are in the test.

*Size of Sliding Window.* The construction of a text graph is influenced by the size  $l$  of the sliding window. Therefore, the parameter  $l$  will inevitably affect the performance of KGAT. Figure 4a shows the accuracy of KGAT under different window sizes on *Ohsumed* and *TREC* respectively. The x-axis represents the window size, and the y-axis represents accuracy. It can be seen that the accuracy reaches top when  $l = 5$  (resp.  $l = 5$ ) on *Ohsumed* (resp. *TREC*), hence the optimal window size of texts is  $l = 5$  in our method.



**Fig. 3.** Tests on three datasets with different number of heads. Other datasets show the same trend, omitted for space.



**Fig. 4.** Accuracy changes under varied window size and embedding dimension.

*Dimensions.* Figure 4b depicts the accuracy on *MR* and *Ohsumed* with different embedding dimensions. As is shown, the model accuracy improves with the increase of dimension  $d$ , until reaching  $d = 300$ . In particular, the increase of accuracy slows down after  $d > 200$ . For a large dimension ( $d > 300$ ), the model accuracy begins to decline. This shows that an embedding with too low dimension cannot propagate label information to neighbor nodes well, while a too high dimensional embedding still can not improve model performance, and may cost extra training time.

## 5 Conclusion and Future Work

In the paper, we propose a comprehensive approach for text classification. We have introduced techniques to construct *text graphs*, that captures correlation degrees among words. We have also developed a GAT-based model with multi-head and enhanced attention mechanism for representation learning. We have proposed a customized ReadOut operation to finalize the representation for a text. Via intensive experimental studies, our approach shows promising results on multiple benchmark datasets and *Sensitive*, a newly published dataset.

We have utilized Stanford Dependency-Parser and conducted tests by using dependency trees instead. The results show that incorporating dependency trees does not significantly improves performances. While, we will keep working on

this direction. Another direction worth exploring is multi-label classification. Extending the model to incorporate edge features (rather than occurrence frequency) would be another interesting topic.

**Acknowledgement.** This work is supported by Sichuan Scientific Innovation Fund (No. 2022JDRC0009) and the National Key Research and Development Program of China (No. 2017YFA0700800).

## References

1. Cheng, J., Dong, L., Lapata, M.: Long short-term memory-networks for machine reading. In: EMNLP, pp. 551–561 (2016)
2. Ding, K., Wang, J., Li, J., Li, D., Liu, H.: Be more with less: hypergraph attention networks for inductive text classification. In: EMNLP, pp. 4927–4936 (2020)
3. Forman, G.: BNS feature scaling: an improved representation over TF-IDF for SVM text classification. In: CIKM, pp. 263–270 (2008)
4. Huang, L., Ma, D., Li, S., Zhang, X., Wang, H.: Text level graph neural network for text classification. In: EMNLP-IJCNLP, pp. 3442–3448 (2019)
5. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. In: EACL, pp. 427–431 (2017)
6. Kim, S.B., Han, K.S., Rim, H.C., Myaeng, S.H.: Some effective techniques for Naive Bayes text classification. *IEEE TKDE* **18**(11), 1457–1466 (2006)
7. Kim, Y.: Convolutional neural networks for sentence classification. In: EMNLP, pp. 1746–1751 (2014)
8. Lai, S., Xu, L., Liu, K., Zhao, J.: Recurrent convolutional neural networks for text classification. In: AAAI, pp. 2267–2273 (2015)
9. Li, X., Roth, D.: Learning question classifiers. In: COLING (2002)
10. Lin, Y., Xu, G., Xu, G., Chen, Y., Sun, D.: Sensitive information detection based on convolution neural network and bi-directional LSTM. In: TrustCom, pp. 1614–1621 (2020)
11. Liu, P., Qiu, X., Huang, X.: Recurrent neural network for text classification with multi-task learning. In: IJCAI, pp. 2873–2879 (2016)
12. Liu, Y., Guan, R., Giunchiglia, F., Liang, Y., Feng, X.: Deep attention diffusion graph neural networks for text classification. In: EMNLP, pp. 8142–8152 (2021)
13. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: ICLR (2013)
14. Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., Khudanpur, S.: Recurrent neural network based language model. In: ISCA, pp. 1045–1048 (2010)
15. Peng, H., et al.: Large-scale hierarchical text classification with recursively regularized deep graph-CNN. In: Proceedings of the 2018 World Wide Web Conference, pp. 1063–1072 (2018)
16. Pennington, J., Socher, R., Manning, C.D.: GloVe: global vectors for word representation. In: EMNLP, pp. 1532–1543 (2014)
17. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. *IEEE Trans. Neural Netw.* **20**(1), 61–80 (2009)
18. Tan, S.: An effective refinement strategy for KNN text classifier. *Elsevier ESWA* **30**(2), 290–298 (2006)
19. Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)

20. Wang, Y., Huang, M., Zhu, X., Zhao, L.: Attention-based LSTM for aspect-level sentiment classification. In: EMNLP, pp. 606–615 (2016)
21. Was, T., Skibski, O.: An axiomatization of the eigenvector and Katz centralities. In: AAAI, pp. 1258–1265 (2018)
22. Wu, F., de Souza, A.H., Zhang, T., Fifty, C., Yu, T., Weinberger, K.Q.: Simplifying graph convolutional networks. In: ICML, pp. 6861–6871 (2019)
23. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. *IEEE TNNLS* **32**(1), 4–24 (2021)
24. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A.J., Hovy, E.H.: Hierarchical attention networks for document classification. In: NAACL-HLT, pp. 1480–1489 (2016)
25. Yao, L., Mao, C., Luo, Y.: Graph convolutional networks for text classification. In: AAAI, pp. 7370–7377 (2019)
26. Zhang, Y., Yu, X., Cui, Z., Wu, S., Wen, Z., Wang, L.: Every document owns its structure: inductive text classification via graph neural networks. In: ACL, pp. 334–339 (2020)
27. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: a survey. *IEEE TKDE* **34**(1), 249–270 (2022)
28. Zhou, J., et al.: Graph neural networks: a review of methods and applications. *AI Open* **1**, 57–81 (2020)
29. Zhou, X., Wan, X., Xiao, J.: Attention-based LSTM network for cross-lingual sentiment classification. In: EMNLP, pp. 247–256 (2016)