Article

Long Text Truncation Algorithm Based on Label Embedding in Text Classification

Jingang Chen and Shu Lv *

School of Mathematical Sciences, University of Electronic Science and Technology of China, Chengdu 610000, China; 202021110309@std.uestc.edu.cn
* Correspondence: lvshu@uestc.edu.cn

Abstract: The long text classification task has become a hot research topic in the field of text classification due to its long length and redundant information. At present, the common processing methods for long text data, such as the truncation method and pooling method, are prone to the problem of too many sentences or loss of contextual semantic information. To deal with these issues, we present LTTA-LE (Long Text Truncation Algorithm Based on Label Embedding in Text Classification), which consists of three key steps. Firstly, we build a pretraining prefix template and a label word mapping prefix template to obtain the label word embedding, and we realize the joint training of long text and label words. Secondly, we calculate the cosine similarity between the label word embedding and the long text embedding, and we filter the redundant information of the long text to reduce the text length. Finally, a three-stage model training architecture is introduced to effectively improve the classification performance and generalization ability of the model. We conduct comparative experiments on three public long text datasets, and the results show that LTTA-LE has an average F_1 improvement of 1.0518% over other algorithms, which proves that our method can achieve satisfactory performance.

Keywords: long text classification; text truncation; label embedding; prefix template

Citation: Chen, J.; Lv, S. Long Text Truncation Algorithm Based on Label Embedding in Text Classification. *Appl. Sci.* **2022**, *12*, 9874. https://doi.org/10.3390/app12199874

Academic Editor: Valentino Santucci

Received: 31 August 2022
Accepted: 22 September 2022
Published: 30 September 2022

1. Introduction

Text classification refers to classifying text content according to certain rules or models [1]. As a basic task of natural language processing (NLP), it is widely used in applications such as emotion recognition, spam classification, news text classification, etc. At present, long text, especially document-level text data, shows an explosive growth trend. Long text classification has become a hot topic in text classification due to its rich information and long text length.

The traditional text classification algorithms mainly use n-gram [2], TF-IDF [3] to extract text features and then input the text features into the machine learning models for training. For example, Shahi et al. made use of hybrid features (FastText+TF-IDF) to represent Nepali COVID-19-related tweets for the sentiment classification [4]. The evaluation results on the NepCOV19Tweets demonstrated that their method achieved excellent performance. However, these methods rely heavily on manual feature engineering and ignore the contextual relationship and location information between words, which severely limits the model performance. With the rapid development of deep learning, neural network models such as convolutional neural network (CNN) [5], recurrent neural network (RNN) [6], and long short-term memory (LSTM) [7] are widely applied in the field of text classification. The rise of pretrained language models represented by bidirectional encoder representation from transformers (BERT) [8] indicates that the transformer [9] architecture can be fully applied to the field of text classification, enabling text context information to participate in the expression of word vectors in a more comprehensive and reasonable way. However, due to the limitation of the input data length, the pretrained language models perform poorly in the field of long text classification. Therefore, how to effectively reduce the text length while minimizing the loss of semantic information is the key to improving the performance of pretrained language models in the field of long text classification.

Long text data preprocessing methods can be classified into four categories: truncation method [10], pooling method [11], compression method [12,13] and transformer model improvement [14]. However, these methods generally have problems such as information loss, resource waste and insufficient precision. In long texts, there are often many redundant sentence segments that are not very relevant to the topic, which is usually considered as data noise. On the one hand, this noise information interferes with model learning; on the other hand, these redundant sentence segments occupy input positions that do not belong to them. Due to the length limitation, the BERT model must abandon other more important sentence segments, which results in poor performance of the BERT model in long text classification scenarios. Therefore, it is an important method to use label embedding to filter sentence segments irrelevant to the topic in long text, and how to better learn to obtain the label embedding is also a key step.

Label embedding represents the vector representation of label words obtained by model learning. At present, the mainstream label word learning methods include label-word joint embedding methods [15], multi-task label-embedding methods [16], interactive mechanism-based methods [17], and some improvements have also been made on the basis of these methods.

However, there are two main problems with these methods. The first problem is that these methods do not learn the textual information of domain-specific data, because they are all methods derived from the idea of word embedding. Word embedding refers to embedding a high-dimensional sparse word space into a low-dimensional dense word vector space. The training corpus of word embedding is mainly from general domain data, such as Wikipedia, Gigaword Dataset, etc. [18]. In the process of obtaining word vectors based on word embeddings, these methods are not further trained for specific domain data, so they cannot learn the unique text information in specific domain data. The second problem is that these methods do not alleviate the problem of "polysemy". For example, the word "apple" might represent both a fruit and a brand name. If the word "apple" is directly embedded without a prompt template, the obtained word vector is not accurate enough. However, if a prompt template is added in front of the word that needs to be represented, for example, a prompt template of "the brand of this electronic product is" is added in front of "apple", then the word mapping of "apple" we obtain is obviously related to electronic products. By introducing a prompt template, the problem of "polysemy" will be subtly alleviated.

For problem one, Howard and Ruder proposed a general paradigm ULMFiT [19]. The ULMFiT paradigm mainly divided model training into three stages. The first stage trained the language model on a large number of general corpora, the second stage continued to train the language model on the specific domain data, and the third stage carried out specific tasks on the specific domain data. After the first stage, we can obtain a pretrained model, which made the convergence of the second stage much faster. In the second stage, domain-specific data were used as the pretraining object, so the model can quickly adapt to the characteristics of the target data, so as to fully learn the context information of the specific domain data. For problem two, Schick and Schütze proposed a semi-supervised model: pattern-exploiting training (PET) [20]. By introducing a prompt template, PET reformulated the model input into cloze-style phrases, which can help the language model understand the given task. At the same time, PET added a prompt template before the user input, which was equivalent to limiting the semantic range corresponding to the user input in a prompt manner. This method enabled word embeddings to accurately represent the user input, thus effectively alleviating the problem of polysemy.

Inspired by this, we propose the LTTA-LE algorithm. First, two prefix templates are constructed to prompt user input, which effectively limits the semantic range corresponding to user input. Then, an adaptive filtering threshold equation is designed, which dynamically determines the filtering amount of each long text. Next, the similarity between the label embedding and each segment in the long text is calculated, effectively filtering out invalid information in the long text. Finally, a three-stage model training architecture is introduced, which significantly improves the classification performance and generalization ability of the model. The main contributions of this paper are as follows.

1. **Unique prefix template design, which can alleviate the problem of polysemy.** This paper constructs a pretraining prefix template and a label word mapping prefix template. By constructing the pretraining prefix template, the joint training of label words and long texts is realized. During the pretraining process, the long text and the label words are fully interacted to avoid the gap between them. With the help of the label word mapping prefix template, the label word embedding can more accurately describe the real semantics expressed by the user input, which effectively alleviates the problem of polysemy.
2. **More efficient long text noise filtering scheme.** Considering that there are a large number of redundant sentence segments irrelevant to the topic in long texts, we can filter invalid information in long texts by calculating the similarity between the label embedding and each sentence segment in long texts. On the one hand, it can meet the limitation of the BERT model on the length of the input corpus. On the other hand, it can effectively filter the noise and improve the classification performance of the model.
3. **Adaptive filtering threshold equation.** By designing an adaptive filtering threshold equation, the filtering quantity of each long text is dynamically determined, which avoids the problem of incomplete information filtering caused by too long text or serious information loss caused by too short text after filtering.
4. **A general and efficient three-stage model training scheme.** This paper introduces a three-stage model training scheme, which mainly includes: in-domain pretraining, finetuning, and prediction. Among them, the in-domain pretraining stage can help the model learn the characteristics of specific domain data in advance and improve the model’s ability to represent sentences. The finetuning stage enables the model to be quickly applied to downstream tasks and improves the generalization ability of the model. In the prediction stage, the pseudo-label method is used to predict the test set, and the test set is filtered based on the obtained pseudo-label, which can avoid the interference of the model by the noise data in the test set. Through this three-stage model training scheme, the classification performance and generalization ability of the model can be significantly improved.

In the experimental phase, we verify and compare the algorithm performance on three public datasets. In terms of overall algorithm performance, our proposed LTTA-LE algorithm surpasses other long text processing methods.

In addition, we also design an experimental scheme to conduct an in-depth analysis of the main modules of the algorithm. We compare the performance of the algorithm on the Arxiv Dataset under different max_len, where max_len refers to the maximum text length of the model input. It is found that for most max_len, our algorithm achieves the best performance due to its superior filtering effect on long text, which means that we can reduce max_len as much as possible to save memory on the premise of fixed precision requirements. Meanwhile, we compare the impact of four label-embedding schemes on the final performance of the algorithm. The LTTA-LE algorithm uses the prefix template to jointly train the long text and the label word, and it fully learns the interaction information between them. Therefore, the LTTA-LE algorithm achieves the best result.

2. Related Work

2.1. Long Text Classification

The appearance of BERT has greatly improved the state-of-the-art performance of text classification tasks. However, in the face of chapter level or even longer text data, due to the limitations of input length and computational overhead, the text data input into the BERT model must be simplified to meet the requirements of the model. The maximum input length is required, and a lot of useful information will be lost if handled improperly. Therefore, how to effectively process long text data and avoid information loss has become the key to improving the performance of long text classification tasks.

Sheng and Yuan [10] truncated long text by combining the BERT model and BiGRU network, and they applied an attention mechanism to obtain the core information in long text, which improved the micro and macro F_1 scores of the baseline model by 1.68% and 1.53%, respectively. Zhang et al. [11] used a two-layer sliding window to model long texts, which achieved a 1.9-point improvement on the NQ long answer task and a 1.6-point improvement on the TyDi QA passage answer task. Beltagy et al. [21] proposed Longformer. On the one hand, Longformer can model local contextual representations by introducing local window attention. On the other hand, Longformer model enables sequence representation by introducing global attention. Therefore, this kind of model can achieve the purpose of processing long texts. Later, Dai et al. [14] put forward Transformer-XL in 2020, which can solve the problem of long text by learning dependencies beyond a fixed length without breaking temporal coherence. So far, the Transformer-XL has achieved the state-of-the-art results on some datasets such as enwiki8 and WikiText-103.

2.2. Label Embedding

Label embedding refers to converting label words in text classification into text vectors through vectorization learning. Wang et al. [15] proposed the label-embedding attentive model (LEAM), which can learn the representation of words and labels in the same space. Zhang et al. [16] developed multi-task label embedding (MTLE), which mapped the label of each task into a semantic vector, thereby transforming the traditional text classification task into a vector matching task. Du et al. [17] constructed the explicit interaction model (EXAM) and incorporated word-matching signals into text classification by introducing an interaction mechanism.

2.3. Prompt

Liu et al. [22] summarized the NLP technology into pre-neural network stage, neural network stage, pretraining–finetuning stage, and the pretraining–prompt stage. The prompt algorithm built a prefix template that can predict specific tasks without finetuning downstream tasks. Logan IV et al. [23] proposed prompt-based finetuning, which had two advantages: (1) it was robust to the selection of different prompt templates, and (2) prompt-based finetuning was 1000$\times$ more memory efficient with comparable performance to finetuning all parameters. Later, Hu et al. [24] proposed knowledgeable prompt-tuning (KPT), which expanded the verbalizer in prompt-tuning using the external knowledge bases. This method achieved good results in zero and few-shot text classification tasks. Han et al. [25] proposed prompt tuning with rules (PTR) for multi-class text classification. This method can encode prior human knowledge into prompt tuning by combining sub-prompts into task-specific prompts according to logical rules. Ultimately, the method achieved state-of-the-art performance.

In traditional long text classification algorithm, researchers often fail to make full use of the label word information when processing long text data, which may lead to an excessive loss of useful text information and reduce model performance.

Compared with the traditional pretraining–finetuning mode, the pretraining–prompt paradigm has the following main advantages:

1. Pretraining introduces a lot of prior knowledge into pretrained language models, and the prompt algorithm motivates this knowledge by constructing prefix templates.
2. Due to different training methods, there is usually a gap between pretraining and downstream tasks, and the prompt algorithm alleviates this gap to a large extent.
3. Compared with the traditional method, the prompt method enables the model to capture more contextual information by adding a template prefix to the beginning of the text.

In short, our method focuses on obtaining accurate label word embedding through the joint training of long text data and label words and then filtering the long text data to avoid an excessive loss of useful information to the greatest extent.

3. Method

As an improved compression method for the long text classification algorithm, the basic idea of the LTTA-LE algorithm is to first learn the similarity measure of each sentence segment and the label word after cutting the long text and then filter out the sentence segments with low similarity and recombine the remaining sentence segments into a new short text. Finally, the obtained new short text and original labels are input into the model for training. Through these steps, the purpose of compressing long text into the BERT model is achieved while minimizing the semantic loss. Figure 1 shows the whole framework. As can be seen from Figure 1, our proposed LTTA-LE can be summarized into three parts. The details are as follows.

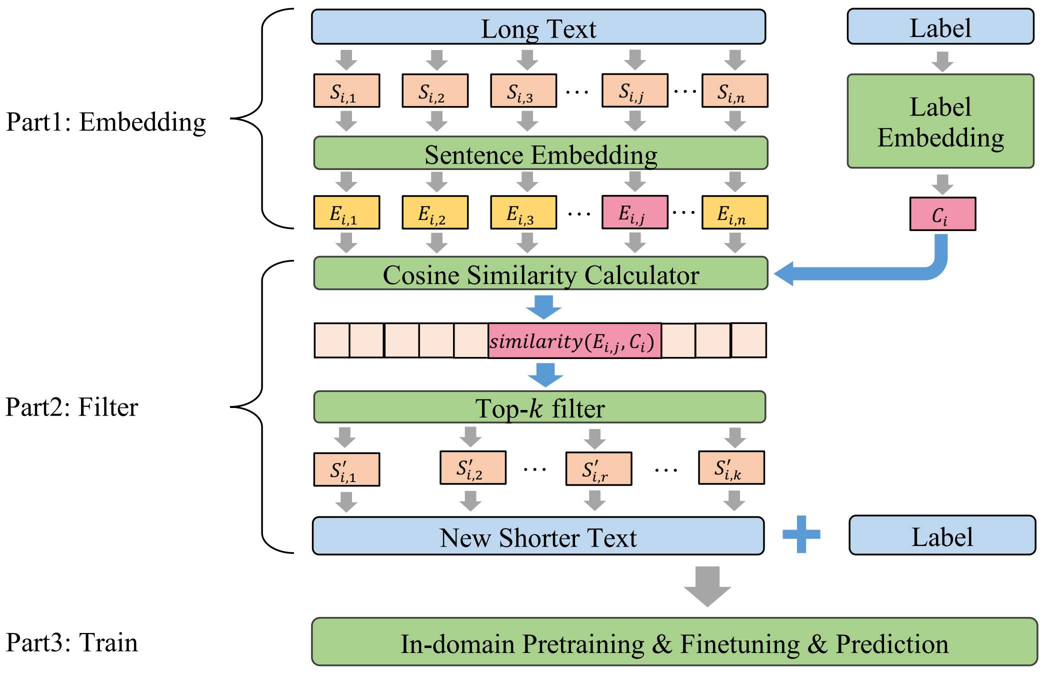
Part1: Embedding
Long Text
Label
$S_{i,1}$ $S_{i,2}$ $S_{i,3}$ ··· $S_{i,j}$ ··· $S_{i,n}$
Sentence Embedding
Label Embedding
$E_{i,1}$ $E_{i,2}$ $E_{i,3}$ ··· $E_{i,j}$ ··· $E_{i,n}$
C_i
Part2: Filter
Cosine Similarity Calculator
$similarity(E_{i,j}, C_i)$
Top-k filter
$S'_{i,1}$ $S'_{i,2}$ ··· $S'_{i,r}$ ··· $S'_{i,k}$
New Shorter Text
Label
Part3: Train
In-domain Pretraining & Finetuning & Prediction

Figure 1. The whole framework of LTTA-LE. In this figure, $S_{i,j}, i \in [1, m], j \in [1, n]$ represents the sentence segment after the original text data $x_i, i \in [1, m]$ is divided by the text segmentation dictionary Text_SD (Text_SD=[',', ':', ';', '.', '?', '!', '\t', '\n']). $E_{i,j}, i \in [1, m], j \in [1, n]$ indicates the sentence vector representation obtained after sentence embedding. C_i means the label word vector representation. The $similarity(E_{i,j}, C_i)$ is used to calculate the similarity between $E_{i,j}$ and C_i. $S'_{i,r}, i \in [1, m], r \in [1, k]$ refers to the sentence segment obtained after similarity filtering.

- The first part is the text encoding layer, which includes two sub-modules: the vectorized representation of long text and the vectorized representation of label words. To fully learn the contextual semantic information of the label words, the pretrained corpus data are reconstructed by constructing the prefix template, and the label word vector is retrained and extracted based on the mask language model (MLM).
- The second part is the long text filtering module. This module first calculates the cosine similarity between each sentence segment of the long text and the label word; then, it filters and retains the most similar top-k sentence segments and finally recombines the top-k sentence segments to form a new shorter text. With the help of this module, the purpose of compressing the long text is realized.
- The third part is the model training strategy module, which inputs the compressed text data together with the text classification labels into the model for training and

prediction. This module consists of three parts: in-domain pretraining, finetuning and prediction. By learning the semantic information of a specific dataset, the gap between upstream and downstream tasks is eliminated.

3.1. Label Embedding

Label word representation methods mainly include one-hot embedding representation, static word vector representation (such as word2vec [26]), and word vector extraction based on the pretrained BERT model. In comparison, one-hot embedding and static word vectors find it difficult to capture contextual information, while the word vector extraction method based on pretraining model finds it difficult to capture the interaction information between long text and label words. In order to better obtain the vector representation of label words and alleviate the problem of polysemy, the method of constructing a prefix template is improved on the basis of the BERT model. Figure 2 shows the whole framework of label embedding.

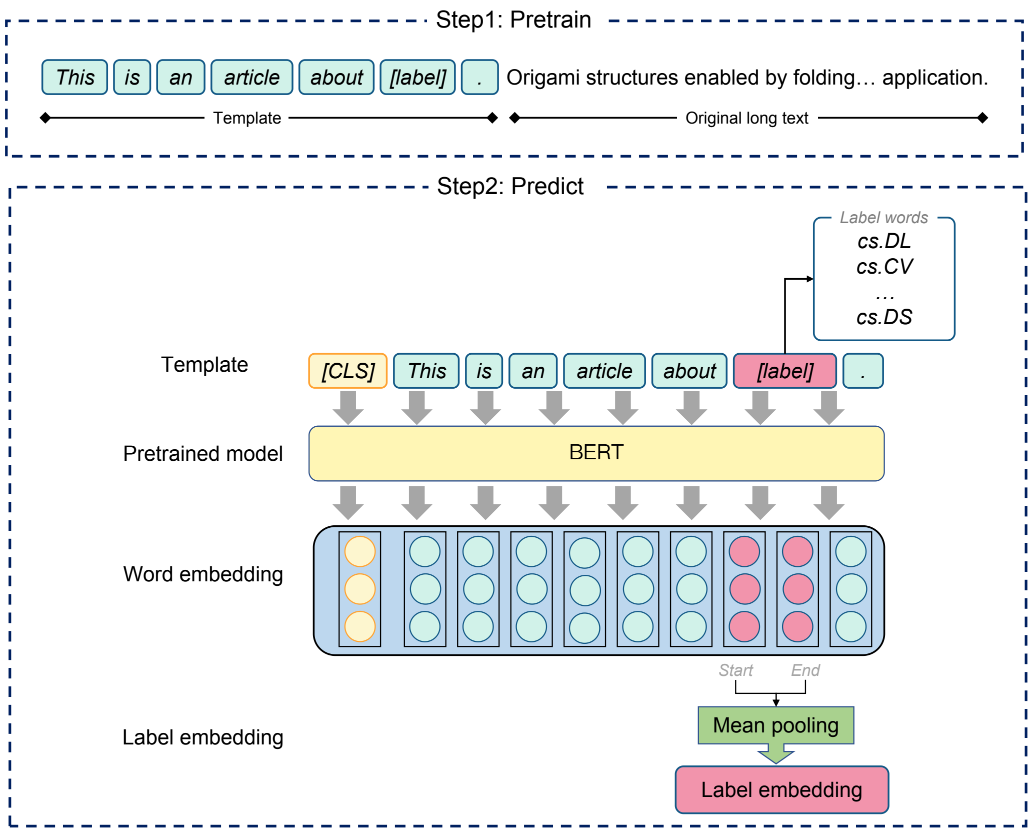

Figure 2. The whole framework of label embedding. In this figure, [CLS] is the sentence vector representation slot, which outputs the sentence vector representation. [label] is the label word filling slot, which fills the label word to associate the label word with the long text.

As can be seen from Figure 2, this framework consists of two steps.

Step 1: Label word pretraining based on prefix template constructing

Firstly, we construct training sample pairs which shows as:

$$T = [(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots, (x_m, y_m)],$$

where x_i is the long text, and y_i is the corresponding label.

Secondly, for each sample pair (x_i, y_i), define the pretraining prefix template pt_1 as:

“This is an article about: $[y_i].x_i$”.

Thirdly, fill the training sample pair data into the prefix template to obtain the new text data $Text_{new}$, which contains the original long text and label words.

Finally, the new text data $Text_{new}$ are input into the BERT model for in-domain pretraining to fully learn the interaction information between long text and label words. Through in-domain pretraining, we obtain a pretrained model H, and the steps of in-domain pretraining will be described in detail in Section 3.3.1.

Step 2: Label word representation

In order to obtain the vector representation of each label word, the forward propagation method is used to predict the label word vector based on the BERT model trained in the previous step.

Firstly, construct a new label word mapping prefix template pt_2 as below:

"This is an article about: $[y_i]$."

In this step, by constructing a label word mapping prefix template, richer context information is introduced into the label word, and the semantic range corresponding to the label word is limited, so that the problem of polysemy can be alleviated.

Secondly, as shown in Figure 2, the prefix template containing the label word is input into the pretrained model H. Then, we can obtain the embedding of each word in the prefix template.

Finally, due to the unpredictability of the text category during word segmentation, a label word may occupy multiple characters, so we simultaneously record the start and end positions of the label word: $label_{start}$ and $label_{end}$. For all vectors in the range of $label_{start}$ and $label_{end}$, mean pooling is conducted to obtain the embedding of each label word.

In addition, in order to make the reader understand this process more clearly, we summarize the process of label embedding in Algorithm 1.

Algorithm 1 Label-embedding algorithm.

```
Input: The traning sample pairs $T = [(x_1, y_1), (x_2, y_2), \dots, (x_i, y_i), \dots, (x_m, y_m)]$;
       The pretraining prefix template $pt_1$ and the label word mapping prefix template $pt_2$.
Output: Label word vector $C = [C_1, C_2, \dots, C_i, \dots, C_m]$.
1: Fill the sample pair consisting of $x_i$ and $y_i$ into the $pt_1$ to obtain new text data $Text_{new}$.
2: Input $Text_{new}$ into the BERT model for in-domain pretraining, and obtain the pretrained
   model $H$.
3: Fill $y_i$ into $pt_2$ to obtain label prompt data, and record the start position $label_{start}$ and
   end position $label_{end}$ of the label word.
4: Input the label prompt data in step 3 into $H$ to obtain the corresponding embedding
   vector.
5: For all vectors in the range of $label_{start}$ and $label_{end}$ in step 4, perform mean pooling to
   obtain the label embedding $C_i$.
```

3.2. Long Text Filtering Based on Similarity Calculation

Since there are a large number of expressions unrelated to label words in long texts, we propose a long text filtering scheme based on text and label similarity calculation to filter more valuable texts in long texts, so as to shorten the text length and improve the long text classification performance. The encoding of the label word has been introduced in the previous section, and the label word vector is defined as C_i.

3.2.1. Similarity Calculation between Sentence Vector and Label Word Vector

In the encoding stage of the long text, considering that the length of the long text far exceeds the length limit of the model, our method adopts the form of sentence vectors for representation, as shown in Figure 1. Firstly, sentence $S_{i,j}$ is obtained by truncating the long text according to the text truncation dictionary Text_SD (Text_SD = [',', ':', ';', '.', '?', '!', '\t', '\n']). Secondly, in order to ensure the model performance and objectivity of the sentence vector, this scheme uses the pipeline function [27] in the Hunging Face open source transformer package to characterize the sentence segment $S_{i,j}$ to obtain the sentence vector $E_{i,j}$. Finally, based on the obtained sentence vector $E_{i,j}$ and label word vector C_i, the similarity between the two is calculated to filter sentences with more relevant label information. A cosine similarity equation is adopted for similarity calculation, which is expressed as:

$$similarity\left(E_{i,j}, C_i\right) = \cos(\theta) = \frac{\sum_{k=1}^{d} E_{i,j,k} \times C_{i,k}}{\sqrt{\sum_{k=1}^{d}\left(E_{i,j,k}\right)^2} \times \sqrt{\sum_{k=1}^{d}(C_{i,k})^2}}, \tag{1}$$

where $E_{i,j,k}$ represents the k-th dimension value in the $E_{i,j}$ vector, $C_{i,k}$ represents the k-th dimension value in the C_i vector, d represents the BERT model output vector dimension, and d = 768 is set in this paper.

3.2.2. Sentence Vector Filtering and Text Reorganization

According to the value of cosine similarity, sort all sentence vectors from large to small, and select the top-k most similar sentence vectors. Then, the corresponding index is obtained according to the selected sentence vectors. Next, keep the corresponding sentence segments based on these indexes, and finally recombine these sentence segments into a new short text. When choosing the value of k, we comprehensively consider the number of sentences in the long text, the number of words and the maximum text length in the input model for modeling. We specify the maximum text length of the input model as max_len, the number of tokens in the long text as len(tokens), and the number of sentences as len(sentences). By calculating the ratio of the maximum text length to the number of tokens in the long text, the k value of each long text data is obtained. The adaptive filtering threshold equation is:

$$k = \frac{\text{max\_len}}{\text{len(tokens)}} \times \text{len (sentences)}. \tag{2}$$

According to the equation, the value of k corresponding to each long text data is calculated, the sorted sentence vectors are extracted, and the top-k sentence segments with the highest correlation with the label words are selected. Finally, the sentence segments are reorganized according to the original order of the sentences, so as to shorten the length of the long text while preserving the original contextual semantics of the original long text. In general, Algorithm 2 shows the detailed steps of the LTTA-LE algorithm.

Algorithm 2 LTTA-LE algorithm.

```
Input: The traning sample pairs $T = [(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots, (x_m, y_m)]$;
       Text segmentation dictionary Text_SD = [',', ':', ';', '.', '?', '!', '\t', '\n'].
Output: New shorter text.
1: According to Text_SD, truncate $x_i$ to obtain the truncated set $S_i = [S_{i,1}, S_{i,2}, \ldots, S_{i,j}, \ldots, S_{i,n}]$.
2: Use the pipeline function [27] to encode $S_i$ to obtain the sentence vector $E_i = [E_{i,1}, E_{i,2}, \ldots, E_{i,j}, \ldots, E_{i,n}]$.
3: Utilize label-embedding algorithm to embed the label to obtain label word vector $C_i$.
4: Based on Equation (1), calculate the similarity between $E_{i,j}$ and $C_i$.
5: Calculate the threshold $k$ according to Equation (2).
6: According to the similarity value, keep the indexes corresponding to the top-$k$ $E_{i,j}$.
7: According to the index of $E_{i,j}$ retained in step 6, filter out the corresponding $S_{i,j}$ (i.e., $S'_{i,r}$ in Figure 2), and splice it into a new short text.
```

3.3. Model Training

The training process of our model can be divided into three stages: in-domain pre-training, finetuning and prediction.

3.3.1. In-Domain Pretraining

In order to improve the model's ability to learn in-domain knowledge, we need to pretrain the model based on the experimental dataset. Therefore, the first step of the in-domain pretraining stage is to construct an in-domain pretraining dataset.

As shown in Figure 1, at the end of part 2, we obtain the filtered new shorter text. Based on the new shorter text, we build the in-domain pretraining dataset $P = [x'_1, x'_2, \ldots, x'_i, \ldots, x'_m]$, where x'_i represents the i-th filtered text.

The in-domain pretraining stage includes two subtasks, namely next sentence prediction (NSP) and MLM. In the NSP task, two sentences x'_i and x'_j are randomly selected from the dataset P to construct the positive and negative splicing sample pairs according to (3). We define that if x'_i and x'_j are adjacent sentences, the spliced sample is a positive sample with label "1"; otherwise, the sample is negative with label "0". By constructing sentence-level positive and negative samples, the model can learn sentence-level contextual information in the experimental dataset.

$$[\text{CLS}] + \text{token}(x'_i) + [\text{SEP}] + \text{token}\left(x'_j\right) + [\text{SEP}], \tag{3}$$

where [CLS] is the sentence vector representation slot used to output the sentence vector representation. [SEP] is the representation notation used to separate two sentences in NSP tasks.

In the MLM task, for each sentence x'_i in the dataset P, we first randomly mask 15% of the words; then, the masked words are used as labels. Finally, the unmasked context information is used to predict the masked words. By re-predicting the masked words, the model can learn word-level contextual information in the experimental dataset.

In general, with the help of NSP and MLM, we have completed in-domain pretraining, and the model can fully learn the sentence-level and word-level context information from the experimental dataset.

3.3.2. Finetuning

The input of the finetuning stage consists of the pretrained model and the finetuning dataset. Specifically, we obtain the pretrained model H of the experimental dataset after in-domain pretraining. Combining the dataset P and the original category label data, we can construct the finetuning dataset $D = [(x'_1, y_1), (x'_2, y_2), \ldots, (x'_i, y_i) \ldots, (x'_m, y_m)]$.

There are three steps in the model finetuning:

1. Input finetuning dataset into BERT model loaded with the pretrained model H.
2. Calculate the cross-entropy loss between the output [CLS] of the model hidden layer and the real category label.
3. Update model parameters according to step 2.

In addition, the specific calculation equation of cross-entropy loss is:

$$H(p, q) = -\sum_{i=1}^{n} p(x_i) \log(q(x_i)), \tag{4}$$

where $p(x_i)$ and $q(x_i)$ refer to the probability distributions of the true category label and predicted category label of the i-th text, respectively, and $H(p, q)$ is the value of cross-entropy.

After finetuning, we obtain the finetuned model N.

3.3.3. Prediction

The LTTA-LE algorithm needs to filter long text according to label data. However, due to the lack of label data in the test set, the label data cannot be used to filter the noise information in the prediction stage. Therefore, we need to take advantage of the pseudo-label method [28] to make predictions on the test set and then complete the filtering of the test set based on the obtained pseudo-labels. The specific implementation steps are as follows.

1. In order to make the length of the test set meet the input requirements of the BERT model, we first shorten the test set by the head + tail truncation method, i.e., keep the fixed-length words in the head and tail of the test set as the input of the BERT model. Then, we use the model N trained in the finetuning stage to make predictions on the test set to obtain the predicted pseudo-labels.
2. Based on our proposed LTTA-LE algorithm, the test set is filtered according to the predicted pseudo-labels, and new shorter texts are obtained after filtering.
3. Use the model N trained in the finetuning stage to predict the new shorter text and obtain the final predicted category label of the test set.

Through the above steps, the problem of missing labels in the test set can be solved and the performance of the model can be significantly improved. Furthermore, the model training process including the above three stages is summarized in Algorithm 3.

Algorithm 3 Model training algorithm.

```
Input: New shorter text $P = [x'_1, x'_2, \dots, x'_i, \dots, x'_m]$;
       Label dataset $Y = [y_1, y_2, \dots, y_i, \dots, y_m]$.
Output: The predict label.
1: Input $P$ into the BERT model for in-domain pretraining, and obtain the pretrained model $H$.
2: According to $P$ and $Y$, build the finetuning dataset $D = [(x'_1, y_1), (x'_2, y_2), \dots, (x'_i, y_i) \dots, (x'_m, y_m)]$.
3: Input $D$ into model $H$ for finetuning, and obtain the finetuned model $N$.
4: Use $N$ to make predictions on the test set and obtain pseudo-labels.
5: Based on the LTTA-LE algorithm and the predicted pseudo-labels in step 4, the long texts in the test set are filtered to obtain new shorter texts.
6: Use $N$ to predict the new shorter text in step 5 to obtain the final predict label of the test set.
```

4. Experiments

4.1. Datasets

In order to verify the performance of our proposed model, we conduct algorithm validation on three public datasets: Arxiv Dataset [29], 20NewsGroup [1], and IMDB Review [30].

4.1.1. Arxiv Dataset

ArXiv Dataset is the largest open-source academic paper sharing platform, which has collected more than 1.7 million scientific and technological papers. We randomly select 50,000 sample data from it (90% for training and 10% for testing), and the number of data categories is 39. In order to verify the performance of our proposed model under the condition of long text, we only keep the abstract of each academic paper and its corresponding category label.

4.1.2. 20NewsGroup

20NewsGroup includes 18,000 news text documents covering 20 different news text categories. Since we pay more attention to the performance of the model on the long text dataset, we filter 8000 long text data from it, and the ratio of training set and test set in the dataset is 9:1.

4.1.3. IMDB Review

IMDB Review is a well-known movie evaluation dataset, which contains a total of 50,000 annotated text data. The dataset has two types of labels: positive and negative. We filter 20,000 long texts with annotations. The ratio of the training set and test set in the dataset is 9:1.

4.2. Experimental Details

We have completed the key part of the code development with the help of the transformer toolkit. The hyperparameter settings of the model are shown in Table 1. Some symbols and setting are explained as follows:

- Epoch: one epoch refers to training once with all the samples in the training set.
- Batch_size: the number of samples grabbed for one training iteration.
- Learning rate: the rate at which the model learns the parameters.
- Model optimization: Adam optimizer [31].

Table 1. Hyperparameter settings for our proposed model.

Stage	**Epoch**	**Batch_Size**	**Learning Rate**
Pretraining	50	64	2×10^{-5}
Finetuning	10	32	2×10^{-5}

In addition, in order to minimize the interference caused by randomness, five-fold cross-validation is used for model training and performance evaluation.

4.3. Model Performance Evaluation

4.3.1. Evaluation Metrics

We utilize three evaluation metrics, precision, recall, and F_1, to measure the performance of long text classification methods. Among them, precision refers to the proportion of correctly predicted positive samples to all predicted positive samples. Recall represents the proportion of correctly predicted positive samples to all actual positive samples. F_1 is the harmonic mean of precision and recall, which can take into account both precision and recall. Furthermore, they can be described by the following equation.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{5}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{6}$$

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{7}$$

where TP = true positive, FP = false positive, TN = true negative, and FN = false negative.

4.3.2. Experimental Results

In order to verify the excellent performance of the LTTA-LE algorithm, this section will compare different algorithms on three public datasets, setting the hyperparameter of max_len as 256. An overview of each model approach involved in the comparison is shown below.

BERT-base + head truncation: Keep sentence segments of max_len length from the beginning of the text.

BERT-base + tail truncation: Keep sentence segments of max_len length from the end of the text.

BERT-base + head & tail truncation: Keep a part at the beginning and end of the text; in this experiment, keep half at the beginning and end of the text.

BERT-base + max pooling: The long text is first segmented into multiple short sentence segments using the same method as LTTA-LE, and then, the sentence vector information for each short sentence segment is extracted based on the BERT-based model. Finally, max pooling is applied to the obtained [CLS] modules. After splicing, the text classification results are obtained through a layer of a fully connected network.

CogLTX: Identify key sentences by training a judgment model, concatenate key sentences for reasoning, and enable multi-step reasoning via rehearsal and decay [32].

BERT-base + LTTA-LE (ours): The LTTA-LE method proposed in this paper.

Table 2 shows the performance of different methods on three public datasets, and the last three columns of Table 2 show the average scores of precision, recall, and F_1 on the three datasets. According to the experimental results in Table 2, our proposed LTTA-LE algorithm achieves the best results on both the Arxiv Dataset and IMDB Review, and it achieves the second best results on 20NewsGroup. For the average F_1 on the three datasets, the LTTA-LE algorithm is 0.8698% higher than BERT-base + head truncation, 1.2289% higher than BERT-base + tail truncation, 0.5705% higher than BERT-base + head & tail truncation, 2.5441% higher than BERT-base + max pooling, and 0.0795% higher than the state-of-the-art method CogLTX.

The experimental results demonstrate that the LTTA-LE algorithm proposed by this paper can optimize the performance of the model on long text datasets. The specific reasons can be summarized as the following three points: (1) The truncation method violently roughly intercepts part of the information, which can result in the loss of a lot of useful information. The longer the text, the more serious the loss of information. The LTTA-LE algorithm proposed by us filters out the meaningless sentence segments according to the similarity between the label word vector and each sentence segment of the long text. On the one hand, the length of the text is shortened, and on the other hand, the noise information in the long text is filtered, which can improve the performance of the model. (2) The max pooling method divides the text into different sentence segments for processing, which loses the connection between the sentence segments. The proposed LTTA-LE algorithm filters the segmented sentence segments and then splices them again according to the original word order, which can not only reduce the text length but also retain the contextual semantic information between different sentence segments. (3) CogLTX does not use label information when processing long texts, which may mistakenly delete sentence segments related to the topic of the text, resulting in the loss of useful information. On the one hand, our method introduces a pretraining prefix template, which enables full interaction between long texts and label words. On the other hand, our method introduces a label word mapping prefix template, which limits the semantic range of the label word in a prompt manner and effectively alleviates the problem of polysemy.

4.3.3. Comparison under Different max_len

Limited by the memory size and device computing power, in some cases, the model can only accept a smaller max_len input to save resources. To verify the performance of the algorithm in this case, a comparative experiment is designed to evaluate the performance of the model on the Arxiv Dataset with different max_len. Considering the sub-optimal performance of BERT-base + head & tail truncation in previous experiments, BERT-base + head & tail truncation is chosen as the comparison model.

Table 3 shows the performance of the LTTA-LE algorithm on the Arxiv Dataset with different max_len hyperparameter settings. From Table 3, we have the following findings: (1) with the decrease of max_len, the F_1 of the model decreases to a certain extent. (2) In most cases, the F_1 of BERT-base + LTTA-LE (ours) is better than BERT-base + head & tail truncation. (3) When max_len = 32, the F_1 of BERT-base + LTTA-LE (ours) is lower than that of BERT-base + head & tail truncation. When max_len = 128, BERT-base + LTTA-LE (ours) has the best performance, which is 0.88% higher than BERT-base + head & tail truncation.

Table 2. Experimental results on different datasets (±indicates standard deviation).

Method	**Arxiv Dataset**			**20NewsGroup**			**IMDB Review**			**Average**		
	Precision	**Recall**	F_1	**Precision**	**Recall**	F_1	**Precision**	**Recall**	F_1	**Precision**	**Recall**	F_1
BERT-base + head truncation	0.8419 ± 0.0176	0.8139 ± 0.0221	0.8276 ± 0.0197	0.8612 ± 0.0091	0.8392 ± 0.0122	0.8500 ± 0.0105	0.9588 ± 0.0039	0.9292 ± 0.0071	0.9437 ± 0.0051	0.8873	0.8608	0.8738
BERT-base + tail truncation	0.8393 ± 0.0199	0.8121 ± 0.0239	0.8254 ± 0.0211	0.8577 ± 0.0129	0.8321 ± 0.0134	0.8447 ± 0.0133	0.9567 ± 0.0028	0.9276 ± 0.0089	0.9419 ± 0.0042	0.8846	0.8573	0.8707
BERT-base + head & tail truncation	0.8477 ± 0.0139	0.8199 ± 0.0177	0.8335 ± 0.0152	0.8633 ± 0.0113	0.8411 ± 0.0098	0.8520 ± 0.0106	0.9579 ± 0.0019	0.9299 ± 0.0067	0.9436 ± 0.0029	0.8896	0.8636	0.8764
BERT-base + max pooling	0.8386 ± 0.0162	0.8101 ± 0.0267	0.8241 ± 0.0202	0.8482 ± 0.0219	0.8208 ± 0.0295	0.8343 ± 0.0253	0.9391 ± 0.0106	0.9019 ± 0.0186	0.9201 ± 0.135	0.8753	0.8443	0.8595
CogLTX	0.8487 ± 0.0137	0.8271 ± 0.0181	0.8377 ± 0.0155	**0.8701 ± 0.0162**	**0.8493 ± 0.0076**	**0.8595 ± 0.0104**	0.9591 ± 0.0016	0.9309 ± 0.0063	0.9448 ± 0.0025	0.8926	0.8691	0.8807
BERT-base + LTTA-LE (ours)	**0.8512 ± 0.0116**	**0.8292 ± 0.0163**	**0.8401 ± 0.0133**	0.8679 ± 0.0132	0.8489 ± 0.0081	0.8582 ± 0.0101	**0.9601 ± 0.0017**	**0.9321 ± 0.0057**	**0.9458 ± 0.0020**	**0.8931**	**0.8701**	**0.8814**

Table 3. Experimental results on the Arxiv dataset of different methods (±indicates standard deviation).

Method	**max_len = 32**			**max_len = 64**			**max_len = 128**			**max_len = 256**		
	Precision	**Recall**	F_1	**Precision**	**Recall**	F_1	**Precision**	**Recall**	F_1	**Precision**	**Recall**	F_1
BERT-base + head & tail truncation	**0.8145 ± 0.0101**	**0.7982 ± 0.0239**	**0.8062 ± 0.0143**	0.8221 ± 0.0177	0.8029 ± 0.0241	0.8123 ± 0.0201	0.8279 ± 0.0112	0.8112 ± 0.0198	0.8194 ± 0.0146	0.8477 ± 0.0139	0.8199 ± 0.0177	0.8335 ± 0.0152
BERT-base + LTTA-LE (ours)	0.8121 ± 0.0173	0.7977 ± 0.0301	0.8048 ± 0.0218	**0.8268 ± 0.0149**	**0.8091 ± 0.201**	**0.8178 ± 0.0277**	**0.8372 ± 0.0136**	**0.8165 ± 0.0171**	**0.8267 ± 0.0152**	**0.8512 ± 0.0116**	**0.8292 ± 0.0163**	**0.8401 ± 0.0133**
Improvement	−0.29%	-0.06%	−0.18%	0.57%	0.77%	0.67%	1.12%	0.65%	0.88%	0.41%	1.12%	0.77%

The reasons for the differences in model performance under different max_len settings can be summarized as follows:

1. For the same model, with the increase of max_len, the effective information that the model can receive is also increasing, which leads to the improvement of the model performance.
2. For long texts, especially document-level text, there is a lot of useless redundant text information, which will inevitably bring noise to model learning and restrict the performance of the model. The LTTA-LE algorithm proposed by us makes use of sentence vector filtering and text reorganization technology to filter out a large number of redundant texts by calculating the similarity between text fragments and topics so as to improve the performance of the model.
3. For max_len = 32 or even smaller, since our experiment focuses on long text, the data we select are all long text data. When max_len is extremely small, only a few important sentence segments can be retained and part of the head and tail information is ignored, which results in slightly worse performance of the BERT-base + LTTA-LE (ours) than that of the BERT-base + head & tail truncated method.

4.3.4. Comparison of Different Label-Embedding Methods

The core improvement of the LTTA-LE algorithm is the introduction of a pretraining prefix template and a label word mapping prefix template. The pretraining prefix template is used to realize the interaction between long text and label words, and the label word mapping prefix template is used to limit the semantic scope of label words. Compared with other traditional label word embedding methods, the LTTA-LE algorithm can better utilize context information to extract the vector representation of label words, and it can effectively alleviate the problem of word polysemy. In order to verify the performance of the LTTA-LE algorithm, we conduct some experiments to compare the long text classification effects under different label word embedding algorithms in this section. The label word embedding methods compared are shown as follows.

One-hot embedding: Use the one-hot embedding method to encode the label words, and filter the long text according to the label encoding vector to obtain shorter text data, which are then input into the model for training and prediction.

Word2vec embedding: Use the word2vec method to vectorize label words and then filter long text data.

Pretraining without prefix: A pretraining parameter package is used to encode the label word vector directly without introducing the prefix template.

LTTA-LE (ours): The LTTA-LE method proposed in this paper.

The comparison experiments in this section will be performed on the Arxiv dataset with max_len = 256. Table 4 shows the performance of various label-embedding strategies on the Arxiv dataset. As shown in Table 4, LTTA-LE (ours) achieved the highest F_1, which is improved by 3.65% (one-hot embedding), 1.35% (word2vec embedding), and 1.14% (pretraining without prefix), respectively.

Table 4. Experimental results of different label-embedding methods on the Arxiv dataset (±indicates standard deviation).

Method	**Precision**	**Recall**	F_1
One-hot embedding	0.8201 ± 0.0227	0.8012 ± 0.0318	0.8105 ± 0.0267
Word2vec embedding	0.8471 ± 0.0132	0.8116 ± 0.0337	0.8289 ± 0.0191
Pretraining without prefix	0.8489 ± 0.0122	0.8131 ± 0.0297	0.8306 ± 0.0175
LTTA-LE (ours)	**0.8512** ± **0.0116**	**0.8292** ± **0.0163**	**0.8401** ± **0.0133**

The reasons for the differences in the scores of different label-embedding strategies on the Arxiv dataset can be summarized into the following four points.

1. One-hot embedding uses a simple and independent one-hot vector to represent label word information, resulting in the loss of potential label information. Filtering and extracting long text data based on the results of one-hot label embedding will mistakenly miss important information.
2. Word2vec embedding is a static word vector, which cannot be dynamically adjusted and optimized for specific datasets and tasks, so it cannot capture rich semantic information between label words and long texts.
3. Pretraining without a prefix can capture contextual semantic information to a certain extent. However, such models are only pretrained on a general corpus, so they are not capable of representing text data in specific fields, especially professional datasets such as the Arxiv dataset. Therefore, the ability of pretraining without a prefix to represent label words is also insufficient.
4. On the one hand, the LTTA-LE algorithm builds a pretraining prefix template to realize the interaction between long text and label words, so as to use the label word information to filter the noise information in the long text. On the other hand, the LTTA-LE algorithm builds a label word mapping prefix template to limit the semantic scope of label words so as to alleviate the problem of polysemy.

4.3.5. The Results of iFLYTEK Text Classification Challenge

iFLYTEK is a well-known intelligent voice service listed company in the Asia-Pacific region mainly researching intelligent voice, NLP, computer vision and other directions. It held an academic paper classification challenge in July 2021. The task of the competition was to use the title and abstract to predict the specific category of the paper. The competition dataset consisted of 60,000 papers in total, including 50,000 in the training set and 10,000 in the test set. The competition used the accuracy as the evaluation index, and a total of 473 teams participated in the competition. We applied the LTTA-LE algorithm to this academic paper classification competition and achieved an accuracy of 86.32%, which was 0.76 percentage points higher than the second place.

5. Conclusions

In this paper, we propose a more efficient method to deal with the long text classification task, namely the LTTA-LE algorithm. Compared with other mainstream long text classification methods, our method makes four major contributions.

1. Two prefix templates are designed and used: a pretraining prefix template and a label word mapping prefix template. With the help of the pretraining prefix template, the model can better learn the interaction information between the long texts and labels. With the help of the label word mapping prefix template, label embedding with more accurate and rich semantic information can be obtained. In addition, through the limitation of prefix templates, the problem of polysemy can be further alleviated.
2. By calculating the cosine similarity between the label embedding and the sentence segment, the sentence segments that are not related to the semantics of the label words are filtered out, avoiding the interference of the noise data to the model.
3. An adaptive filtering threshold equation is designed and used, which can dynamically determine the filtering amount of each long text to avoid the problem of information loss caused by excessive text filtering.
4. A model training method including three stages of "in-domain pretraining, finetuning and prediction" is introduced, so that the model can fully learn the context information of specific domain data, thereby improving the classification performance of the model.

We test the performance of our LTTA-LE algorithm on three public datasets and find that our proposed method is superior to some commonly used existing algorithms. Meanwhile, we design some experiments to discuss the performance of our algorithm with different max_len settings and label-embedding strategies, and we further quantify the advantages of the LTTA-LE algorithm. Our method was applied to the iFLYTEK text classification challenge and won the first prize. In addition, in order to facilitate readers to have a clearer understanding of the LTTA-LE algorithm, we summarize its advantages and disadvantages in Table 5.

Table 5. Advantages and disadvantages of the LTTA-LE algorithm.

Advantages	Disadvantages
Unique prefix template design, which can alleviate the problem of polysemy More efficient long text noise filtering scheme Adaptive filtering threshold equation A general and efficient three-stage model training scheme	Not suitable for multi-label long text classification tasks

6. Future Work

In future work, we plan to apply the LTTA-LE algorithm to the multi-label long text classification task. Based on the idea of the LTTA-LE algorithm, in the multi-label text classification scenario, we need to filter the sentence segments that are not related to the label. Since different label categories correspond to different task scenarios, the key point of the problem is to take into account the needs of each scenario. Therefore, first of all, a specific prompt template is constructed for the task scene corresponding to each type of label to obtain the corresponding label representation. Then, the cosine distance between the sentence segment and each label is calculated separately. Next, a weighted sum operation is performed on these cosine distances, and the obtained value is used as the retention coefficient of the sentence segment. Finally, according to our proposed adaptive filtering threshold equation, the top-k sentence segments are retained and reorganized so that the retained information takes into account the task scenarios corresponding to various labels, thereby improving the model performance of multi-label classification tasks.

Author Contributions: Conceptualization, S.L. and J.C.; methodology, S.L.; validation, S.L. and J.C.; formal analysis, S.L.; investigation, J.C.; resources, S.L.; writing—original draft preparation, J.C.; writing—review and editing, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available in [1,29,30].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lang, K. Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*; Elsevier: Amsterdam, The Netherlands, 1995; pp. 331–339.
2. Cavnar, W.B.; Trenkle, J.M. N-gram-based text categorization. In Proceedings of the SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, NV, USA, 11–13 April 1994; Citeseer: Princeton, NJ, USA, 1994; Volume 161175.
3. Joachims, T. *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization*; Technical report; CMU Computer Science Department: Pittsburgh, PA, USA, 1996.
4. Shahi, T.; Sitaula, C.; Paudel, N. A Hybrid Feature Extraction Method for Nepali COVID-19-Related Tweets Classification. *Comput. Intell. Neurosci.* **2022**, *2022*, 5681574. [CrossRef] [PubMed]
5. Gu, J.; Wang, Z.; Kuen, J.; Ma, L.; Shahroudy, A.; Shuai, B.; Liu, T.; Wang, X.; Wang, G.; Cai, J.; et al. Recent advances in convolutional neural networks. *Pattern Recognit.* **2018**, *77*, 354–377. [CrossRef]
6. Liu, P.; Qiu, X.; Huang, X. Recurrent neural network for text classification with multi-task learning. *arXiv* **2016**, arXiv:1605.05101.
7. Li, F.; Zhang, M.; Fu, G.; Qian, T.; Ji, D. A Bi-LSTM-RNN model for relation classification using low-cost sequence features. *arXiv* **2016**, arXiv:1608.07720.

8. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
9. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*; Curran Associates: Montreal, QC, Canada, 2017; Volume 30.
10. Sheng, D.; Yuan, J. An Efficient Long Chinese Text Sentiment Analysis Method Using BERT-Based Models with BiGRU. In Proceedings of the 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Dalian, China, 5–7 May 2021; IEEE, 2021, pp. 192–197.
11. Zhang, H.; Gong, Y.; Shen, Y.; Li, W.; Lv, J.; Duan, N.; Chen, W. Poolingformer: Long document modeling with pooling attention. In Proceedings of the International Conference on Machine Learning, Online, 18–24 July 2021; PMLR, 2021; pp. 12437–12446.
12. Tagarelli, A.; Karypis, G. A segment-based approach to clustering multi-topic documents. *Knowl. Inf. Syst.* **2013**, *34*, 563–595. [CrossRef]
13. Barrow, J.; Jain, R.; Morariu, V.; Manjunatha, V.; Oard, D.W.; Resnik, P. A joint model for document segmentation and segment labeling. In Proceedings of the the 58th Annual Meeting of the Association for Computational Linguistics, Online, 5–10 July 2020, pp. 313–322.
14. Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.; Le, Q.V.; Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv* **2019**, arXiv:1901.02860.
15. Wang, G.; Li, C.; Wang, W.; Zhang, Y.; Shen, D.; Zhang, X.; Henao, R.; Carin, L. Joint embedding of words and labels for text classification. *arXiv* **2018**, arXiv:1805.04174.
16. Zhang, H.; Xiao, L.; Chen, W.; Wang, Y.; Jin, Y. Multi-task label embedding for text classification. *arXiv* **2017**, arXiv:1710.07210.
17. Du, C.; Chen, Z.; Feng, F.; Zhu, L.; Gan, T.; Nie, L. Explicit interaction model towards text classification. In Proceedings of the the AAAI Conference on Artificial Intelligence, Atlanta, GA, USA, 8–12 October 2019; Volum 33, pp. 6359–6366.
18. Pennington, J.; Socher, R.; Manning, C.D. Glove: Global vectors for word representation. In Proceedings of the the 2014 conference on empirical methods in natural language processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1532–1543.
19. Howard, J.; Ruder, S. Universal language model fine-tuning for text classification. *arXiv* **2018**, arXiv:1801.06146.
20. Schick, T.; Schütze, H. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv* **2020**, arXiv:2001.07676.
21. Beltagy, I.; Peters, M.E.; Cohan, A. Longformer: The long-document transformer. *arXiv* **2020**, arXiv:2004.05150.
22. Liu, P.; Yuan, W.; Fu, J.; Jiang, Z.; Hayashi, H.; Neubig, G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv* **2021**, arXiv:2107.13586.
23. Logan IV, R.L.; Balažević, I.; Wallace, E.; Petroni, F.; Singh, S.; Riedel, S. Cutting down on prompts and parameters: Simple few-shot learning with language models. *arXiv* **2021**, arXiv:2106.13353.
24. Hu, S.; Ding, N.; Wang, H.; Liu, Z.; Li, J.; Sun, M. Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification. *arXiv* **2021**, arXiv:2108.02035.
25. Han, X.; Zhao, W.; Ding, N.; Liu, Z.; Sun, M. Ptr: Prompt tuning with rules for text classification. *arXiv* **2021**, arXiv:2105.11259.
26. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv* **2013**, arXiv:1301.3781.
27. Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. Transformers: State-of-the-art natural language processing. In Proceedings of the the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Online, 16–20 November 2020; pp. 38–45.
28. Lee, D.H. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In Proceedings of the ICML 2013 Workshop : Challenges in Representation Learning (WREPL), Atlanta, GA, USA, 16–21 June 2013; Volume 3, p. 896.
29. Clement, C.B.; Bierbaum, M.; O'Keeffe, K.P.; Alemi, A.A. On the Use of ArXiv as a Dataset. *arXiv* **2019**, arXiv:1905.00075.
30. Maas, A.; Daly, R.E.; Pham, P.T.; Huang, D.; Ng, A.Y.; Potts, C. Learning word vectors for sentiment analysis. In Proceedings of the the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Portland, Oregon, 19–24 June 2011; pp. 142–150.
31. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
32. Ding, M.; Zhou, C.; Yang, H.; Tang, J. CogLTX: Applying bert to long texts. In *Advances in Neural Information Processing Systems*; Curran Associates: Montreal, QC, Canada, 2020; Volume 33, pp. 12792–12804.