Neurocomputing 363 (2019) 366–374

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Improving text classification with weighted word embeddings via a multi-channel TextCNN model

Bao Guo[a], Chunxia Zhang[a], Junmin Liu[a,*], Xiaoyi Ma[b]

[a] School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China
[b] School of Art and Science, University of Colorado Boulder, Boulder, CO 80310, USA

ARTICLE INFO

Article history:
Received 6 March 2019
Revised 3 June 2019
Accepted 20 July 2019
Available online 24 July 2019

Communicated by Dr Fenza Giuseppe

Keywords:
Text classification
Term weighting
Word embedding
Convolutional neural network
Term frequency-inverse document frequency (TF-IDF)

ABSTRACT

In recent years, convolutional neural networks (CNNs) have gained considerable attention in text classification because of the remarkable good performance they achieved in various situations. The usual practice is to first perform word embedding (i.e., mapping each word into a word vector), and then employ a CNN to perform classification. To improve classification accuracy, term weighting approaches have been proven to be quite effective. But to the best of our knowledge, almost all these methods assign only *one* weight to each term (word). Considering the fact that one term generally has different importance in documents with different class labels, we propose in this paper a novel term weighting scheme to be combined with word embeddings to enhance the classification performance of CNNs. In the novel method, *multiple* weights are assigned to each term and these weights are applied to the word embeddings of the words separately. Subsequently, the transformed features are fed into a multi-channel CNN model to predict the label of the sentence. By comparing the novel method with several other baseline methods with five benchmark data sets, the results manifest that the classification accuracy of the proposed method exceeds that of other methods by an amazing margin. Moreover, the weights assigned by different weighting schemes are also analyzed to get more insights of their working mechanism.

1. Introduction

Text classification, aiming at assigning documents into one or more predefined categories, is an important and interesting task in natural language processing (NLP). It has many useful applications in the fields of sentiment analysis [1], spam detection [2], subject categorization [3] and so on. With the great success achieved by deep learning in visual and speech recognition, many deep architecture models have been applied to cope with various problems in NLP so that better performance can be reached. The popular models include but are not limited to word vector learning through neural language models [4], text classification with recurrent neural networks (RNNs) [6] and convolutional neural networks (CNNs) [7].

Regarding the deep learning algorithms for text classification, one traditional but successful preprocessing method is one-hot representation [5]. Nevertheless, it encounters the curse of dimensionality due to the huge vocabulary when dealing with a large corpus. In past several years, *word embedding* has gradually taken its place and become a standard approach. Originally introduced by Bengio et al. [4] with an unsupervised neural language model in 2003, a word embedding scheme $\mathcal{E}: words \to \mathbb{R}^p$ plays an important role in transforming words into low-dimensional dense vectors, where p is generally between 100 and 500. For example, we might obtain the representations

$$\mathcal{E}(\text{“apple”}) = (0.064, -0.160, -0.012, \cdots),$$
$$\mathcal{E}(\text{“paper”}) = (-0.089, 0.096, -0.219, \cdots),$$

by a word embedding scheme. Because the computing power and algorithms do not yet allow the training of a large vocabulary, the research of word embedding has stagnated for some time. It was not until 2013 that Mikolov et al. proposed the Word2Vec model [8,9] to help removing the above barrier. In 2014, Pennington et al. [10] released Global Vectors (GloVe), a competitive set of pre-trained word embeddings, which signals that word embedding had turned into the main stream. So far, Word2Vec [8,9] and GloVe [10] are still among the most popular and efficient word embedding methods. More importantly, some pre-trained vectors which are extracted from very large corpora by these two models have been made publicly available and convenient to use.

Since good representations are a key component in many NLP models, some effective pre-training embedding methods have been developed in recent years. For example, Peters et al. [11]

* Corresponding author.
E-mail address: junminliu@mail.xjtu.edu.cn (J. Liu).

https://doi.org/10.1016/j.neucom.2019.07.052
0925-2312/

introduced a general approach named ELMo (Embeddings from Language Models) to learn high-quality deep context-dependent representations through two-layer biLMs(bidirectional language models). The prominent advantages of ELMo lies in that it can capture polysemy (i.e., the meaning of a word or phrase varies across linguistic contexts) and can model complex characteristics of word use such as syntax and semantics. To make full use of unlabeled text corpora in pre-training embeddings, Radford et al. [12] proposed a semi-supervised fine-tuning approach, generative pre-training transformer (OpenAI GPT), to learn a universal representation that transfers with little adaptations to a wide range of tasks. To overcome the limitation of unidirectional language models, BERT (bidirectional encoder representation from transformers) was put forward by Devlin et al. [13] to pre-train deep directional representations by jointly conditioning on both left and right context in all layers. As reported in literature, BERT is conceptually simple, empirically powerful and allows the same pre-trained model to successfully tackle a broad set of NLP tasks.

In text classification, many scholars have explored the use of deep learning methods and reported notable performance [6,7,14–16]. Generally speaking, the existing techniques can be divided into two categories, i.e., sequence-based RNN models and CNN models. With the special feedback loop structure where each node at a time step takes an input from the previous node, RNNs such as LSTM (long short-term memory, [17]) and GRU (gated recurrent unit, [18]) are suitable for storing past information with the internal state (memory). As a result, RNNs can process sequential data better and they have achieved significant results in text classification [6,14]. Although CNNs were originally invented and developed for computer vision, recently they have been shown to be effective in capturing informative semantic features of n-grams (word or character embeddings) with convolutional operations. Meanwhile, CNNs have exhibited promising performance in many NLP tasks such as text classification [7], language modeling [15] and others [16].

In comparison with RNNs, CNNs have attracted more attention in text classification since they require less training data to learn fewer parameters. Based on word embeddings, Kim [7] made the first attempt to propose an improved effective CNN model named TextCNN, which achieves excellent performance on several benchmark data sets. Since then, more and more CNN-based models are developed in text classification. For instance, Conneau et al. [19] presented a very deep character-level CNN (it has 29 layers) and showed that the performance improves with the depth of the CNN. Li et al. [20] proposed a simple but effective method to initialize convolutional layers (filters) instead of directly using random values. Chen et al. [21] proposed a two-step approach which first classifies sentences into different types, and then performs text classification separately on sentences from each type. To speed up classification process, Yenigalla et al. [22] designed methods to combine both character- and word-based CNNs. Zhang et al. [23] introduced a special piecewise pooling technology into CNNs for text classification. Rezaeinia et al. [24] proposed a novel architecture, Multiple Block Convolutional Highways (MBCH), which demonstrates improved accuracy on multiple benchmarks. By utilizing traditional feature-based methods as the enhance model, Zhang et al. [25] developed a three-way enhanced CNN model called 3W-CNN which further improves the accuracy of sentiment classification. Moreover, Liu et al. [26] proposed a new CNN model to learn two context vectors through integrating attention mechanism into CNNs.

Term weighting, a preprocessing strategy to assign appropriate weights to terms so that the weights represent the relevancy of terms to documents, plays an important role in boosting the performance of text classification. Traditional term weighting schemes, such as term frequency (TF) and term frequency-inverse document frequency (TF-IDF) [27], are unsupervised, which means that they do not utilize the label information of documents to compute weights. Because it is natural to take labels into consideration, supervised term weighting schemes [28–30] have been proposed recently.

It is noteworthy that all the above-mentioned weighting methods follow the same rule, i.e., one term just has one weight, which implicitly assumes that the same term also has the same importance in different collections. However, it is very common that a term occurs in documents with different labels. In each collection, its importance is usually different. Hence, the trade-off among the scores of different collections loses a lot of useful information. To overcome this shortcoming of existing term weighting approaches, we propose in this paper a novel weighting strategy for text classification which gives each term multiple weights. And the number of weights (at least two) depends on the number of text classes. Even though any term weighting strategy can be used to compute each single weight of a term, we choose TF-IDF to achieve the purpose because of its simplicity but good performance. Given a corpus, we first create a vocabulary of terms. Subsequently, each single weight assigned to a term is computed as the TF-IDF estimated by each collection of the documents with the same label. These weights are applied to the word embeddings of sentence separately, which are then taken as the input of a multi-channel CNN model [7] to execute text classification. By evaluating and comparing our method with several other baseline methods on some benchmark data sets, the experimental results show that our method achieves the state-of-art performance on all the considered data sets and improves the classification accuracy of other methods by a large margin.

The remainder of this paper is organized as follows. Section 2 first gives a brief introduction to the CNN model proposed by Kim [7], and then presents the details of our proposed method. Section 3 conducts experiments to evaluate and compare the novel method with several other baseline methods. Finally, some conclusions and discussions are offered in Section 4 to conclude the paper.

2. Methodology

Because we adopt the CNN model proposed by Kim [7] in our method to perform text classification, it is introduced briefly in section 2.1. Section 2.2 presents the details of our proposed term multi-weighting scheme to obtain more efficient word embeddings. In particular, an unsupervised term weighting method TF-IDF [27] is first described to facilitate later discussions. The design, algorithm and implementation of the novel term weighting scheme are then followed.

2.1. TextCNN: a CNN model for classification with word embeddings

TextCNN, proposed by [7], is a very useful and effective deep learning algorithm for short text classification tasks. Due to its promising performance, TextCNN has become a baseline model for text classification [19–26].

Here, we would like to make a remark on the input of TextCNN. Initially, all the words in a given corpus need to be transformed into low-dimensional dense vectors. Particularly, a vocabulary composed of the words occurring in the corpus needs to be first created. The word embeddings for these words can be initialized with random values or the pre-trained vectors learned by the Word2Vec [8,9] or GloVe [10] model. In general, each word in the vocabulary can be represented as a vector $\mathbf{e} \in \mathbb{R}^l$, where l is the embedding size. If these word vectors are fixed during training, it's called *CNN-static*. Otherwise, the corresponding model is called *CNN-non-static* since the word embeddings are updated

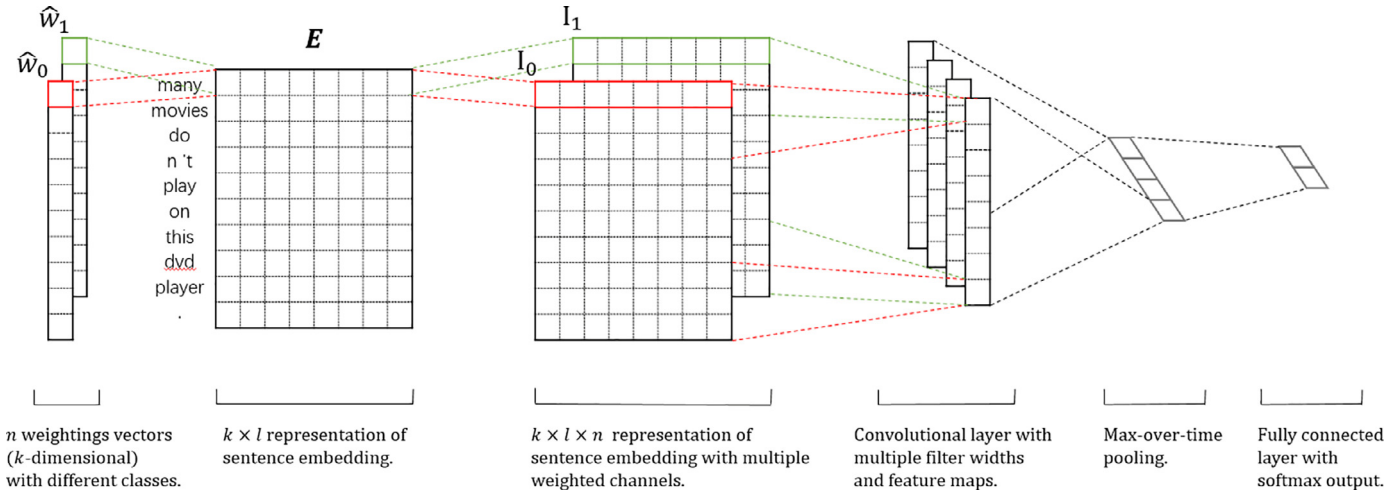


Fig. 1. The architecture of our model.

via an algorithm like backpropagation in the training process. For a sentence of length k (padded or truncated when necessary), $\mathbf{s} = (s_1, \ldots, s_k)^{\mathrm{T}}$, the word embedding of $\mathbf{s}$ can be stacked in a matrix $\mathbf{E} = (\mathbf{e}_1^{\mathrm{T}}, \ldots, \mathbf{e}_k^{\mathrm{T}})^{\mathrm{T}} \in \mathbb{R}^{k \times l}$ where $\mathbf{e}_i$ is the embedding of word s_i. In this way, $\mathbf{E}$ is like an image which is then input into a CNN to perform classification.

With the matrix $\mathbf{E}$ as the input of a CNN, convolutional operations are followed. In the convolutional layer, many filters with varying window sizes (here, only the height is different) slide over the full rows of $\mathbf{E}$, i.e., the width of filters is usually the same as the width of $\mathbf{E}$. Every filter performs convolution on $\mathbf{E}$ and different feature maps are generated. For the elements that locate in the same feature map, a max-over-time pooling operation is applied to extract the most important feature. Notice that a non-linear activation is usually added in the feature extraction step. Finally, these features from different feature maps are concatenated as the penultimate layer and passed to a fully connected softmax layer to predict the probability distribution over class labels. Generally speaking, some regularization techniques such as dropout [31] and batch normalization [32] can be imposed on the penultimate layer and the weight vectors to prevent the model from over-fitting.

With the aim to prevent overfitting by ensuring that the learned vectors do not deviate too far from the original values, Kim [7] had designed a multi-channel CNN model. In this model, two word embedding matrices with one being kept static throughout training (CNN-static) and the other being fine-tuned via backpropagation (CNN-non-static) constitute its input. To get an intuitive understanding of the above explanation, we would like to use the architecture shown in Fig. 1 to make an explanation. By ignoring the first layer, one can image that there are two word embedding matrices $\mathbf{E}_1$, $\mathbf{E}_2$ which are obtained with CNN-static and CNN-non-static schemes, respectively. Put in another way, each embedding matrix $\mathbf{E}_i (i = 1, 2)$ is taken as a channel in the multi-channel CNN model [7]. In next convolutional layer, each filter is applied to $\mathbf{E}_i$ that corresponds to each channel and the results are added to compose a feature. For the other issues in the multi-channel model, it is equivalent to the single channel architecture.

2.2. A novel term weighting scheme for word embedding

In the study of text classification, some evidence has shown that the use of suitable methods for term weighting can bring a great boost in accuracy and effectiveness. It is worth mentioning that our proposed term multi-weighting method can be used to improve any existing weighting scheme. But for simplicity and convenience, we employ the TF-IDF, an unsupervised method which has been proved to be an effective scheme, to compute each single weight of a term in our method.

In information retrieval, TF-IDF [27] is a statistical measure being widely used to evaluate how important a word is to a document in a corpus. Given a vocabulary $\mathcal{V}$ of terms and a collection of documents $\mathcal{D}$, the TF-IDF weight for a term $t \in \mathcal{V}$ and a document $d \in \mathcal{D}$ is computed as

$$\begin{aligned} \mathrm{TF}(t, d) &= f_{td}, \\ \mathrm{IDF}(t, d) &= \log\left(\frac{|\mathcal{D}|}{m}\right) + 1, \\ \text{TF-IDF}(t, d) &= \mathrm{TF}(t, d) \times \mathrm{IDF}(t, d), \end{aligned} \tag{1}$$

where f_{td} is the frequency (i.e., the number of occurrences) of term t in document d, $|\mathcal{D}|$ indicates the number of documents in $\mathcal{D}$ and m represents the number of documents in $\mathcal{D}$ which contains the term t. In this manner, we can compute the TF-IDF weights of each term in $\mathcal{V}$ for document d. For the terms appearing in d, the weight can be easily computed by Eq. (1). While for the terms not in d, the weight is simply zero because its corresponding TF(t, d) is zero. For each document d in $\mathcal{D}$, therefore, we can obtain a TF-IDF weight vector of length $|\mathcal{V}|$. To make all weight vectors fall within the same subspace, they are usually normalized by their own Euclidian norms. With these TF-IDF vectors, we can create a TF-IDF term-document matrix $\mathcal{M} \in \mathbb{R}^{|\mathcal{D}| \times |\mathcal{V}|}$, where the ith row corresponds to the ith document in $\mathcal{D}$, and the jth column corresponds to the jth term in $\mathcal{V}$.

As argued in introduction, most existing term weighting schemes [27–30] just follow the same rule, that is, one term only has one weight for all documents. This hypothesis is often violated in practice since one term may occur in documents which belong to different classes. In the documents of each class, it usually has different importance. Based on this fact, we propose a novel term weighting scheme to assign multiple weights to a term. The number of weights of a term depends on the number of text classes, that is, each term is assigned n weights when given a corpus consisting of n categories. Fig. 1 shows the architecture of our proposed model for text classification with two classes. Here, it is worthwhile to point out the difference between the two multi-channel models proposed by us and Kim [7]. In our model, the input of each channel is obtained by applying the weight vectors corresponding to each class to the word embedding matrix. In Kim's model, however, the inputs of two channels

are two word embedding matrices that is obtained by CNN-static and CNN-non-static schemes, respectively.

The following Algorithm 1 lists the main steps about how our term weighting scheme works. The whole process can be broadly divided into two phases, i.e., the computation of term weights and the creation of weighted word embeddings.

Algorithm 1 The algorithm to compute the weighted word embeddings.

Input: a training set $\mathcal{C}$ of n classes.
Output: weighted word embeddings $\mathbf{I}_1, \ldots, \mathbf{I}_n$.

```
Use C to create a vocabulary V of words.
Cluster the documents in C with the same label as {C_1, ..., C_n}.
for each i ∈ {1, ..., n} do
    Predefine a weight vector w_i ∈ R^{|V|};
    Obtain a TF-IDF term-document matrix M_i with C_i;
    For the words occurring in C_i, assign the maximum value of each column of M_i to corresponding entry of w_i;
    For the words not occurring in C_i, assign the minimum value of all maximum values to the rest entries of w_i;
end for
For a sentence s in C, pad or truncate it to length k;
Get the word embedding matrix E ∈ R^{k×l} of s;
for each i ∈ {1, ..., n} do
    for each j ∈ {1, ..., k} do
        For the jth word in s, find its corresponding weight ŵ_ij from w_i;
    end for
    Denote ŵ_i = (ŵ_i1, ..., ŵ_ik)^T, i.e., a weight vector computed from the collection C_i;
    Let W_i be a k × l matrix whose each column is ŵ_i;
    Let I_i = W_i ⊙ E, where ⊙ denotes the element-wise multiplication of two matrices.
end for
```

Given a corpus $\mathcal{C}$ consisting of n classes, we first tokenize it into terms which refer to words in our method. Based on the obtained terms, we create a vocabulary $\mathcal{V}$ in which every term is unique. To simplify notations, we represent $\mathcal{C}$ as the collections $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, where $\mathcal{C}_i$ includes all the documents with the same label i in the corpus. For each collection $\mathcal{C}_i$, the TF-IDF term-document matrix $\mathcal{M}_i \in \mathbb{R}^{|\mathcal{D}_i| \times |\mathcal{V}_i|}$ can be computed according to Eq. (1). Notice that $|\mathcal{D}_i|$ denotes the number of documents in $\mathcal{C}_i$ and $|\mathcal{V}_i|$ is the number of unique terms in $\mathcal{C}_i$, $i = 1, \ldots, n$. Subsequently, we can generate a weight vector $\mathbf{w}_i \in \mathbb{R}^{|\mathcal{V}|}$ with $\mathcal{M}_i$ in the following manner. With respect to the terms in $\mathcal{C}_i$ and $\mathcal{V}$, their weights are assigned as the maximum value of each column of $\mathcal{M}_i$. As for the terms not in $\mathcal{C}_i$ but in $\mathcal{V}$, let their weights be the minimum value of the maximum values which have been computed with $\mathcal{M}_i$. By repeating this process over the collections $\mathcal{C}_1, \ldots, \mathcal{C}_n$, n weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_n$ can be obtained.

With the above obtained weight vectors, the weighted word embeddings generated as follows can be fed into a multi-channel CNN model to execute classification. For a document (sentence) in the corpus, first pad or truncate it to length k, i.e., $\mathbf{s} = (s_1, \ldots, s_k)^{\mathrm{T}}$. Then, we can get the word embedding matrix of $\mathbf{s}$ as $\mathbf{E} = (\mathbf{e}_1^{\mathrm{T}}, \ldots, \mathbf{e}_k^{\mathrm{T}})^{\mathrm{T}} \in \mathbb{R}^{k \times l}$, where l is the embedding size and $\mathbf{e}_i$ is the word embedding of s_i which can be gotten from random values or the pre-trained vectors created by the Word2Vec or GloVe model. Next, we need to find the corresponding weights for the words in $\mathbf{s}$ from each weight vector $\mathbf{w}_1, \ldots, \mathbf{w}_n$. Let $\hat{\mathbf{w}}_i = (\hat{w}_{i1} \ldots, \hat{w}_{ik})^{\mathrm{T}}$ $(i = 1, \ldots, n)$ denote the weights for the words in $\mathbf{s}$ from the ith weight vector $\mathbf{w}_i$, and $\hat{w}_{ij} (j = 1, \ldots, k)$ stands for the weight for the jth word of $\mathbf{s}$ from $\mathbf{w}_i$. In doing so, we can acquire a collection of weighted word embeddings $\{\mathbf{I}_1, \ldots, \mathbf{I}_n\}$ in which $\mathbf{I}_i \in \mathbb{R}^{k \times l}$. Here, $\mathbf{I}_i$ can be calculated by first getting a weight matrix $\mathbf{W}_i$ of order $k \times l$, whose each column is $\hat{\mathbf{w}}_i$. And then let $\mathbf{I}_i = \mathbf{W}_i \odot \mathbf{E}$ where $\odot$ denotes the element-wise multiplication of two matrices. By concatenating $\mathbf{I}_1, \ldots, \mathbf{I}_n$ in the third dimension, we can get a three-dimensional array, say, $\mathcal{I}$. Note that $\mathcal{I}$ is just like a multi-channel image, where each $\mathbf{I}_i$ can be viewed as one of the channels. In our term multi-weighting scheme, the concatenated $\mathcal{I}$ is thus considered as the input of a multi-channel TextCNN model.

To illustrate the application of our method, we take the sentiment analysis to determine the polarity (i.e., positive or negative) of a document as an example. After creating the vocabulary $\mathcal{V}$ of words, we can obtain a weight vector $\mathbf{w}_{\mathrm{pos}} \in \mathbb{R}^{|\mathcal{V}|}$ from the TF-IDF term-document matrix with all the positive documents. Similarly, a weight vector $\mathbf{w}_{\mathrm{neg}} \in \mathbb{R}^{|\mathcal{V}|}$ can be generated with all the negative documents. Provided a sentence of length k, therefore, we can get two weighted word embeddings by multiplying the embedding of the words in $\mathbf{s}$ by their corresponding weights from $\mathbf{w}_{\mathrm{pos}}$ and $\mathbf{w}_{\mathrm{neg}}$ separately. Consequently, $\mathbf{I}_{\mathrm{pos}}$ and $\mathbf{I}_{\mathrm{neg}}$ can be obtained and taken as the input of a two-channel CNN model to classify whether the document is positive or negative.

Table 1
The main characteristics of the used data sets.

Data set	k	n	N	$\lvert\mathcal{V}\rvert$
MR	20	2	10662	18760
Subj	23	2	10000	21319
CR	19	2	3775	5338
MPQA	3	2	10606	6245
TREC	10	6	5952	9594

Note: Here, k indicates the average length of sentences, n means the number of classes, N stands for the number of documents(sentences) and $|\mathcal{V}|$ represents the number of words in vocabulary.

3. Experimental study

In this section, we employ various benchmark data sets to examine the performance of our proposed method for addressing text classification tasks. Meanwhile, it is also compared with several other baseline methods. In what follows, we sequentially describe the experimental data sets, baseline methods, experimental setup as well as the results followed by discussions.

3.1. Experimental data sets

Here, we give a brief description to the used text classification data sets which are publicly available. Table 1 lists the main characteristics of these data sets.

MR: Movie review sentence polarity dataset v1.0. It contains 5331 positive snippets and 5331 negative snippets extracted from website. The task is to determine whether the review is positive or negative[1].

Subj: This is a subjectivity data set where the task is to classify a sentence as being subjective or objective. This corpus has 5000 subjective sentences and 5000 objective sentences.[1]

CR: This set is composed of some annotated customer reviews of 14 products from Amazon. The purpose is to categorize each customer review into positive or negative. In particular, it contains

[1] https://www.cs.cornell.edu/people/pabo/movie-review-data/.

2405 positive sentences and 1366 negative sentences. Evidently, this data set is class imbalanced.[2]

MPQA: It is an opinion polarity data set with two classes. Specifically, there are 3311 positive documents and 7293 negative documents in this corpus. This is also an imbalanced data set.[3]

TREC: This is a multi-class data set. The main task is to classify a question into one of six question types (i.e., person, location, numeric information and etc.).[4]

3.2. Baseline methods

To demonstrate the effectiveness of our proposed method, the following CNN-based models are included into the comparison. Notice that these methods have been reported to have achieved the state-of-the-art text classification performance.

CNN-static: One-layer CNN with pre-trained word vectors that are kept static during training proposed by Kim [7].

CNN-non-static: One-layer CNN with word vectors that are fine-tuned via backpropagation during training proposed by Kim [7].

CNN-multichannel: One-layer CNN with static and non-static channels [7].

BiLSTM-CRF: A novel two-step pipeline framework by first classifying sentences into different types, and then training 1d-CNNs separately for the sentences in each type [21].

CNN+BI: One-layer CNN with a novel initialization method for the weights of convolutional filters [20].

CNN-1: a new CNN architecture for addressing the issues with both word and character embeddings [22].

MBCH-6F: A novel architecture based on highway networks, DenseNet, batch normalization and bottleneck layers [24].

PPCNN: A piecewise pooling mechanism in CNN proposed by Zhang et al. [23].

ACNN(BiLSTM): A text representation and classification model which integrates attention mechanism into a CNN model [26].

CNN-VE: A CNN model combined with the self-attention mechanism to obtain contextual word embeddings [33].

3W-CNN: A three-way enhanced CNN model, which is an ensemble method proposed by Zhang et al. [25].

3.3. Experimental setup

To obtain good classification performance, it is crucial to adopt good word embeddings to preprocess the given data. In general, the word representations created by the Word2Vec and GloVe algorithms are good choices. In our experiments, we used the publicly available Word2Vec[5] vectors that were trained on 100 billion words from Google News by using the continuous bag-of-words model [9]. The vectors have dimensionality of 300, that is, $l = 300$. Regarding the words not present in the set of pre-trained words, they were initialized as random values which are generated from a uniform distribution between -0.25 and 0.25.

As for the hyperparameters in our multi-channel TextCNN model, we followed the proposal provided by Kim [7] to set their values. In the experiments, the pre-trained word vectors were kept static throughout training. The activation function was taken as the ReLU (rectified linear units) to get nonlinear transformations. To extract various features, we used convolutional filters with window sizes 3, 4, 5, respectively. For each filter type, 100 feature maps were generated. Aiming at avoiding over-fitting, the dropout and weight decay techniques were applied. In particular, dropout was applied to the fully connection layer and the dropout rate was set as 0.5. The L_2-norm constraints of 3 were imposed on all weight vectors. In each experiment, we calculated the weighting vectors with only training data, and the multi-channel TextCNN model was trained with mini-batches with each batch consisting of 50 documents. Since the used data sets have no separate training and test sets, we repeated 10-fold cross validation ten times and utilized some average metrics to evaluate the performance of each approach.

As for the evaluation metrics, the *classification accuracy* (i.e., the proportion that sentences are classified correctly) estimated on a test set is the most natural choice. But for class imbalanced learning scenarios, it cannot provide a comprehensive assessment of a classification algorithm any more. Thus, the F_1-*score* and *AUC* were adopted to further assess the behavior of a method. The definition of the F_1-score is

$$F_1 - \text{score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \quad \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{2}$$

where TP, FP and FN denote the number of true positives, false positives and false negatives, respectively. Moreover, AUC is the area under the ROC curve. It provides a compound measure of a classifier's performance for evaluating which method is better on average.

Table 2
The classification accuracies of each method on all considered data sets.

Method	MR	Subj	CR	MPQA	TREC
CNN-static	81.0	93.0	84.7	89.6	92.8
CNN-non-static	81.5	93.4	84.3	89.5	93.6
CNN-multichannel	81.1	93.2	85.0	89.4	92.2
BiLSTM-CRF	82.3	–	85.4	–	–
CNN+BI	82.2	93.7	85.8	89.5	94.6
CNN-1	81.8	93.3	85.5	90.0	–
MBCH-6F	–	94.1	85.7	90.4	–
PPCNN	81.1	93.4	84.5	89.5	–
ACNN(BiLSTM)	83.4	94.2	87.5	90.8	**95.8**
CNN-VE	81.9	93.9	–	90.5	95.0
3W-CNN	82.3	93.5	85.8	90.3	–
our method	**86.6**	**95.6**	**87.5**	**93.0**	91.9

3.4. Results and discussions

3.4.1. Performance comparison

In Table 2, we listed the classification accuracies for all the baseline methods and our proposed approach on the used data sets. Since the experimental setup was made similar to those used by Kim [7], the results for the baseline methods are directly copied from the related papers [7,26]. It can be noticed that the accuracies for some methods are missing because the corresponding results cannot be found anywhere in relevant literature.

As shown in Table 2, our proposed method achieves the best accuracy on all binary classification data sets and improves the results of all baseline methods. Specifically, our method gives a substantial improvement (about 3.8%) of the second best technique ACNN on MR, which is promising. In addition, it improves the accuracy of ACNN, the best baseline method, by 2.4%, 1.5% on MPQA, Subj data sets, respectively. On the CR data set, both our method and ACNN achieve the highest classification accuracy. As far as the results attained on TREC, our method seems to be defeated. Nevertheless, we must point out that our used data format for this set is most closest to the description presented in the original paper. On GitHub, there is also a version of this data set while it has

[2] http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html.
[3] http://www.cs.pitt.edu/mpqa/.
[4] http://cogcomp.cs.illinois.edu/Data/QA/QC/.
[5] https://code.google.com/p/word2vec/.

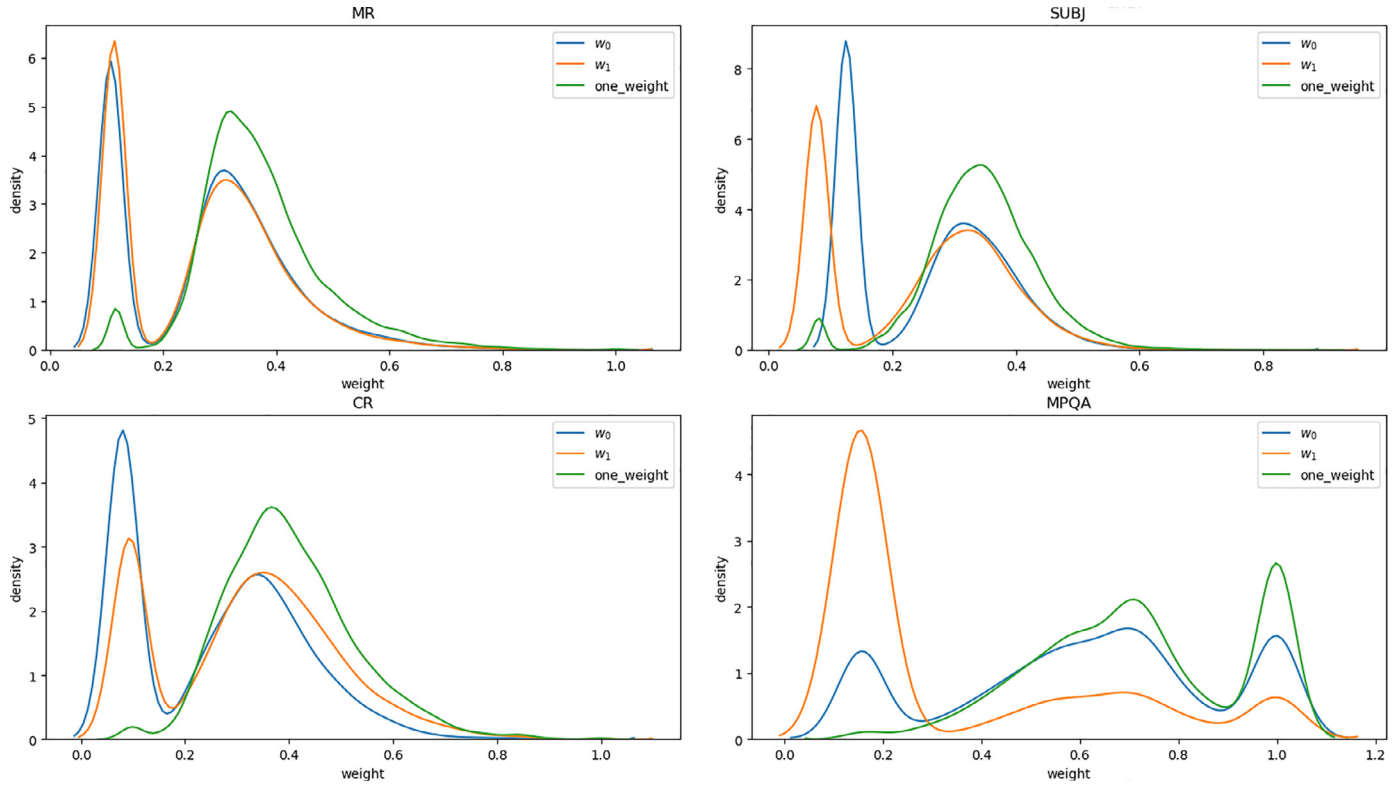


Fig. 2. The density curves of the weights on different data sets.

Table 3
The experimental results on imbalanced data sets.

Data set	Method	Accuracy	F_1 score	AUC
CR	CNN-static	0.847	0.884	0.843
	Our method	0.875	0.904	0.864
MPQA	CNN-static	0.896	0.832	0.874
	Our method	0.930	0.886	0.917

Table 4
The accuracies of methods with different weighting schemes.

Method	MR	CR	MPQA	Subj
CNN-static	81.0	84.7	89.6	93.0
One-weighting	80.5	83.2	89.9	92.8
Our method	86.6	87.5	93.0	95.6

one more dimension than the original data. If using this version of the TREC data to do experiments, our method achieves an accuracy 98.4%. In the other papers, however, the authors have not indicate which version they have adopted. Thus, it should be careful when comparing all the considered techniques on this data set.

With respect to the imbalanced data sets CR and MPQA, the metrics F_1-score and AUC were also employed to evaluate the performance of a method. Table 3 summarizes the results of CNN-static and our method on these two sets. It can be observed that our method still possesses a significant advantage over CNN-static in terms of these two metrics.

As we know, good term weighting strategies are usually of great importance for delivering high classification accuracy. With the purpose to verify the usefulness of our proposed term multi-weighting scheme, we did more experiments to compare it with two other approaches. The first one is CNN-static without using any term weighting scheme, that is, the word embeddings of a corpus were directly input into a single-channel CNN-static model to execute classification. The second one is similar to our method but assigning a unique weight to each term. We call this method one-weighting because a TF-IDF term-document matrix is first estimated with *all* documents in the given corpus, and then the maximum value of each column of the matrix is considered as the weight for the corresponding word to generate a weight vector. We then get the weighted word embeddings of a sentence with the word embedding multiplying its weight vector, which are then fed into to a single-channel CNN-static model. Table 4 lists the accuracies of CNN-static, one-weighting and our proposed method.

The results reported in Table 4 reveal that one-weighting has little effect to enhance the classification accuracy of CNN-static. In some cases, it even decreases the performance of CNN-static. In contrast, our multi-weighting scheme significantly improves the accuracy of CNN-static. This unequivocally demonstrates the value of our multi-weighting scheme to be combined with word embeddings to boost classification accuracy.

3.4.2. Further investigation of weighting schemes

In view of the better performance of our proposed multi-weighting method than one-weighting, we would like to find out the underlying reasons. To achieve this goal, Fig. 2 shows the density curves of weights which are computed by multi-weighting and one-weighting schemes, respectively. Here, the TF-IDF term-document matrixes were estimated with the whole corpus. Because the number of weights assigned by multi-weighting to one term is equal to the number of classes, only four binary classification data sets were considered here to better visualize the results. In each plot, w_0 stands for the weights that are computed with the negative examples and w_1 means the weights that are computed with the positive examples. It can be observed in Fig. 2 that all curves are bimodal distributions, except for MPQA, where the weights are multi-modally distributed. Among the

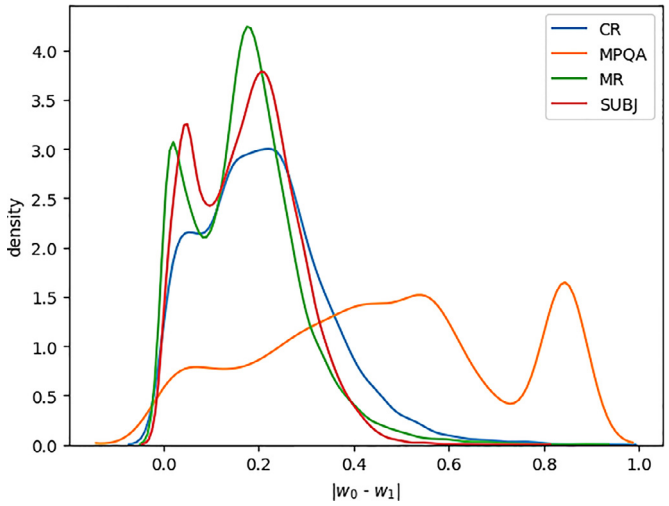


Fig. 3. The density curves of $|w_0 - w_1|$ on different data sets.

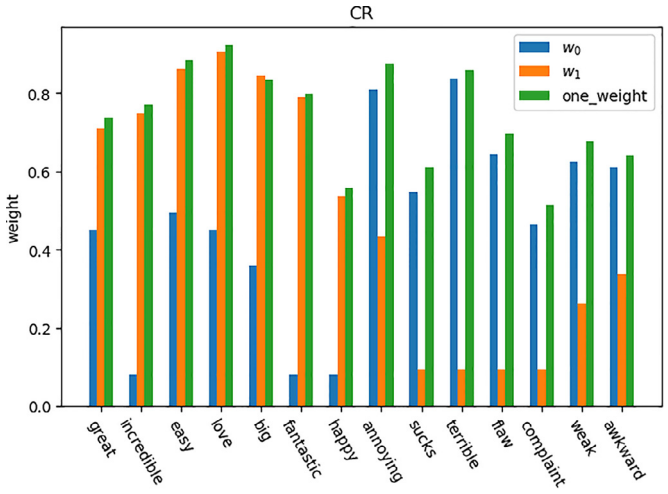


Fig. 4. The bar plots of different weights for some words in the CR set.

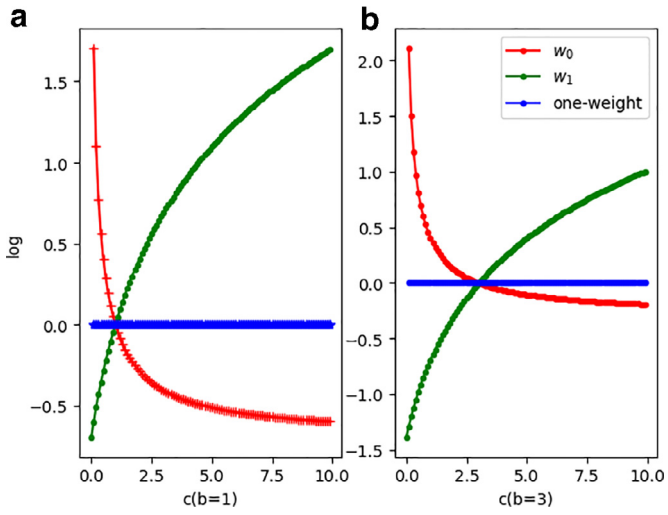


Fig. 5. The first term in the IDF factors I_0, I_1 and I as a function of c.

bimodal distributions, it is obvious that the larger mode of multi-weighting (w_0, w_1) is less than the larger mode of one-weighting, while the smaller mode of multi-weighting is far greater than the smaller mode of one-weighting. This phenomenon indicates that a larger proportion of the weights of multi-weighting are clustered around a small number (e.g., 0.1), in contrast to those of one-weighting. Since the magnitude of a weight directly reflects the importance of a word, it can be deduced that multi-weighting assigns larger weighs to the terms which really matter.

Notice that multi-weight is inspired from the intuition that in the documents of each class, the same words usually have different importance (i.e., weights). To demonstrate the difference between the weights of different classes, we consider the absolute values of $w_0 - w_1$. Fig. 3 plots the density curves of $|w_0 - w_1|$ on different data sets. It can be seen that the largest proportion of $|w_0 - w_1|$ on all but MPQA data sets are clustered around 0.2. On the set MPQA, the difference is more significant. These observations confirm our initial conjecture, i.e., it is the contrast of weights between classes that makes the features extracted by a CNN more distinguishable. Hence, the classification performance of textCNN is greatly improved.

In order to illustrate the specific difference between the weights obtained by two weighting schemes, we chose some words from the vocabulary of the CR set and plotted their weights estimated by different schemes into Fig. 4. Surprisingly, the weights of one-weighting are always close to the larger one of multi-weighting (i.e., w_0, w_1). In fact, this is not a coincidence. To get more insights about this phenomenon, please reconsider the computation of the weights from Eq. (1). First, let us consider the factor IDF(t, d). To ease expositions, we denote the IDF factors in w_0, w_1 and one-weighting by I_0, I_1, I, respectively. Assume the factor IDF(t, d) in one-weighting be $I = \log(\frac{|\mathcal{D}|}{m})$. With the purpose to facilitate discussions, suppose $\frac{|\mathcal{D}_0|}{|\mathcal{D}_1|} = b$ and $\frac{m_0}{m_1} = c$ $(b, c > 0)$, where $|\mathcal{D}_i|$ refers to the number of documents in class i and m_i is the number of documents containing term t in $\mathcal{D}_i$. By some derivations, we can obtain the IDF factor in w_0 and w_1 as $I_0 = \log(\frac{b}{b+1}\frac{c+1}{c}\frac{|\mathcal{D}|}{m})$ and $I_1 = \log(\frac{c+1}{b+1}\frac{|\mathcal{D}|}{m})$, respectively. Obviously, these two terms I_0 and I_1 can be rewritten as $I_0 = \log(\frac{b}{b+1}\frac{c+1}{c}) + \log(\frac{|\mathcal{D}|}{m})$ and $I_1 = \log(\frac{c+1}{b+1}) + \log(\frac{|\mathcal{D}|}{m})$. In view of the IDF factor for one-weighting as $I = 0 + \log(\frac{|\mathcal{D}|}{m})$, we only need to consider the difference between the first terms of IDF for w_0, w_1 and one-weighting, that is, $\log(\frac{b}{b+1}\frac{c+1}{c})$, $\log(\frac{c+1}{b+1})$ and 0.

For simplicity, we first assume $b = 1$ which indicates the observed data in two classes are balanced. Denote the IDF factors in w_0, w_1 and one-weighting by I_0, I_1, I, respectively. With $b = 1$, we can see that the first term in I_0, I_1, I are $\log(\frac{c+1}{2c})$, $\log(\frac{c+1}{2})$ and 0, respectively. Fig. 5(a) plots how these three values vary with the change of c. We can conclude that the relationship between the IDF factors I_0, I_1, I is $I_0 > I > I_1 (c < 1)$ and $I_0 < I < I_1 (c > 1)$. Then, we considered the case with $b = 3$, i.e., the data are imbalanced distributed in two classes. Similar to Fig. 5(a), Fig. 5(b) depicts how the first term of IDF in three weights changes when $b = 3$. It can be seen that the relationship of IDF in I_0, I_1, I remains the same except for the change of position of the intersection. Secondly, assume the factor TF(t, d) of w_0, w_1 are f_0, f_1, respectively. Then, the TF(t, d) in one-weighting will be $f = \max\{f_0, f_1\}$, that is, $f \geq f_0$ and $f \geq f_1$. Based on the above analysis, we can conclude that the value of the weight calculated by one-weighting is close to the larger weight among the two values of multi-weighting by multiplying TF with IDF. This explains the phenomenon that we observe in Figs. 2 and 4.

4. Conclusions and future work

It is crucial to choose a proper term weighting scheme to improve the performance of a CNN model in text classification. For the existing term weighting schemes, we notice that all of them

assign one weight to one term although the term occurs in documents with different labels. This is obviously unreasonable since the term in different documents generally has different importance to the corresponding documents. In view of this fact, this paper proposes a novel term weighting strategy to overcome the above-mentioned shortcoming. The core idea of our method is to assign multiple weights to a term so that each weight can appropriately reflect its importance to the documents coming from different classes. Thus, the number of weights assigned to each term equals to the number of classes. By computing each single weight as the TF-IDF which is estimated from the documents with the same class label, multiple weighted word embeddings of a sentence can be obtained by multiplying its word embeddings with the corresponding weights. These weighted word embeddings are then concatenated as a multi-channel image which is input into a multi-channel CNN model to implement classification. The experiments conducted with several benchmark data sets show that in comparison with some other baseline methods, our approach achieves state-of-the-art performance on all the investigated data sets.

It is worthwhile that the novel multi-weighting strategy is actually a general one even though we took TF-IDF as an example in our experiments due to its simplicity. In future work, we plan to explore some more advanced data-driven term weighting schemes to estimate each single weight of a term so that more performance improvement can be gained in text classification. Due to the limitations of our computational resources, the size of our considered data sets is not large. It is also an interesting direction to investigate how the proposed method will perform on large corpora.

At present, RNN-based deep models also perform well in text classification. However, we haven't consider the RNN framework with the weighted word embeddings created by the novel term weighting scheme. It is well-known that the prominent advantage of RNN is to deal with sequential data. Notice that there are multiple word embedding matrices in our situation, it is hence difficult to extract or pose how each embedding matrix is related to each other. Thus, we reserve this issue as another work deserved to do in the future.

Declaration of competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

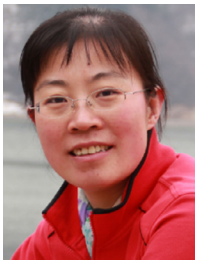
Acknowledgements

The authors would like to thank the Editor and the reviewers for their useful suggestions which have helped to improve the paper substantially. This research was supported by the National Natural Science Foundation of China (Nos. 11671317, 61877049).

References

[1] K. Liu, Sentiment Analysis: Mining Opinions, Sentiments, and Emotions, Cambridge University Press, Cambridge, 2015.
[2] A. Heydari, M. ali Tavakoli, N. Salim, Z. Heydari, Detection of review spam: a survey, Expert Syst. Appl. 42 (7) (2015) 3634–3642.
[3] X. Chen, Y. Zhang, J. Xu, C. Xing, H. Chen, Deep learning based topic identification and categorization: mining diabetes-related topics on Chinese health websites, in: Proceedings of the International Conference on Database Systems for Advanced Applications, Springer, 2016, pp. 481–500.
[4] Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, A neural probabilistic language model, J. Mach. Learn. Res. 3 (2003) 1137–1155.
[5] D.Y. Harris, S.L. Harris, Digital Design and Computer Architecture, Morgan Kaufmann, 2010.
[6] P. Liu, X. Qiu, X. Huang, Recurrent neural network for text classification with multi-task learning, 2016, arXiv:1605.05101.
[7] Y. Kim, Convolutional neural networks for sentence classification, 2014, arXiv:1408.5882.
[8] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, 2013, arXiv:1301.3781.
[9] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Proceedings of the Advances in Neural Information Processing Systems, 2013, pp. 3111–3119.
[10] J. Pennington, R. Socher, C. Manning, Glove: global vectors for word representation, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543.
[11] M.E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettemoyer, Deep contextualized word representations, 2018, arXiv:1802.05365v2.
[12] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving language understanding by generative pre-training, 2018, Preprint, Available at: https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf.
[13] J. Devlin, M.W. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, 2018, arXiv:1810.04805v1.
[14] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Proceedings of the Advances in Neural Information Processing Systems, 2014, pp. 3104–3112.
[15] N. Kalchbrenner, E. Grefenstette, P. Blunsom, A convolutional neural network for modelling sentences, 2014, arXiv:1404.2188.
[16] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, J. Mach. Learn. Res. 12 (2011) 2493–2537.
[17] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[18] K. Cho, D. Bahdanau, F. Bougares, H. Schwerk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, 2014, arXiv:1406.10783v3.
[19] A. Conneau, H. Schwenk, L. Barrault, Y. Lecun, Very deep convolutional networks for text classification, 2016, arXiv:1606.01781.
[20] S. Li, Z. Zhao, T. Liu, R. Hu, X. Du, Initializing convolutional filters with semantic features for text classification, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2017, pp. 1884–1889.
[21] T. Chen, R. Xu, Y. He, X. Wang, Improving sentiment analysis via sentence type classification using BiLSTM-CRF and CNN, in: Expert Systems with Applications, 72, 2017, pp. 221–230.
[22] P. Yenigalla, S. Kar, C. Singh, A. Nagar, G. Mathur, Addressing unseen word problem in text classification, in: Proceedings of the International Conference on Applications of Natural Language to Information Systems, Springer, 2018, pp. 339–351.
[23] Y. Zhang, Q. Wang, Y. Li, X. Wu, Sentiment classification based on piecewise pooling convolutional neural network, Comput. Mater. Contin. 56 (2) (2018) 285–297.
[24] S.M. Rezaeinia, A. Ghodsi, R. Rahmani, Text classification based on multiple block convolutional highways, 2018, arXiv:1807.09602.
[25] Y. Zhang, Z. Zhang, D. Miao, J. Wang, Three-way enhanced convolutional neural networks for sentence-level sentiment classification, Inf. Sci. 477 (2019) 55–64.
[26] T. Liu, S. Yu, B. Xu, H. Yin, Recurrent networks with attention and convolutional networks for sentence representation and classification, Appl. Intell. 48 (10) (2018) 3797–3806.
[27] K. Sparck Jones, A statistical interpretation of term specificity and its application in retrieval, J. Doc. 28 (1) (1972) 11–21.
[28] F. Debole, F. Sebastiani, Supervised term weighting for automated text categorization, in: Text Mining and its Applications, Springer, 2004, pp. 81–97.
[29] H. Wu, X. Gu, Y. Gu, Balancing between over-weighting and under-weighting in supervised term weighting, Inf. Process. Manag. 53 (2) (2017) 547–557.
[30] H.J. Escalante, M.A. García-Limón, A. Morales-Reyes, M. Graff, M. Montes-y Gómez, E.F. Morales, J. Martínez-Carranza, Term-weighting learning via genetic programming for text classification, Knowl. Based Syst. 83 (2015) 176–189.
[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.
[32] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Proceedings of the Thirty Second International Conference on Machine Learning. PMLR, 2015, pp. 448–456.
[33] X. Wu, Y. Cai, Q. Li, J. Xu, H. Leung, Combining contextual information by self-attention mechanism in convolutional neural networks for text classification, in: Proceedings of the International Conference on Web Information Systems Engineering, Springer, 2018, pp. 453–467.

Bao Guo received his Bachelor of Science degree in Statistics from Xian Jiaotong University, Xian, China, in 2017. Currently, he is pursuing the Master degree in Statistics at Xian Jiaotong University. His main research interest is text classification with CNN- and RNN-based deep neural networks.



Chun-Xia Zhang received her Ph.D. degree in Applied Mathematics from Xian Jiaotong University, Xian, China, in 2010. Currently, she is an associate professor in School of Mathematics and Statistics at Xian Jiaotong University. She has authored and coauthored about 40 journal papers on ensemble learning techniques, nonparametric regression and etc. Her main interests include ensemble learning, high-dimensional data analysis and deep learning.

Jun-Min Liu received his Ph.D. degree in Applied Mathematics from Xian Jiaotong University, Xian, China, in 2013. He is now an associate professor in School of Mathematics and Statistics at Xian Jiaotong University. His current research interests include hyperspectral unmixing, remotely sensed image fusion and deep learning.

Xiao-Yi Ma is an undergraduate student in School of Art and Science in University of Colorado Boulder, U.S.A. She is interested in text classification based on deep learning.