Unenhanced Sparse Vector-based Embedding Method for Sentiment Analysis

G. R. Kishore
Department of Information Science and Engineering, JSS Science and Technology University, Mysuru, Karnataka, India
kkishorkumar12@gmail.com

B. S. Harish
Department of Information Science and Engineering, JSS Science and Technology University, Mysuru, Karnataka, India.
bsharish@jssstuniv.in (corresponding author)

C. K. Roopa
Department of Information Science and Engineering, JSS Science and Technology University, Mysuru, Karnataka, India.
ckr@jssstuniv.in

Received: 31 December 2024 | Revised: 23 January 2025 | Accepted: 7 February 2025

 DOI: https://doi.org/10.48084/etasr.10098

ABSTRACT

Natural language processing is one of the most trending fields in research, with sentiment analysis being one of the well-known problems in the field. Many methods have been proposed to handle text-based sentiment data, with social networks acting as one of the main data sources and research targets. An important step in designing a text-based model is the embedding method, which helps in the representation of the inputs. This study presents a novel static text embedding method to represent text inputs and compares its sentiment classification performance with some well-known text embedding methods. The results are on par with existing embedding methods, achieving a promising classification accuracy of 90.66%.

Keywords-sentiment analysis; representation; word2vec; fast-text; word embedding; sparse vector; contextual embedding model

I. INTRODUCTION

Identifying the sentiment in a given text has become a widely researched area, as most industries are working on models to automate the process of understanding the market sentiment to make appropriate decisions. With an appropriate model, sentiment analysis can be used to understand the intent or opinion of a person behind a particular statement. The problem can be described as identifying sarcasm or the polarity of a statement. The problem can be anything that deals with identifying the actual meaning behind the targeted sentence. During text analysis, one of the most important steps is the representation, and word embedding is one of the widely known text representation methods. Word embeddings are methods where each word is represented by a vector. There are multiple ways to obtain the embedding vector, which can be classified as static embedding and contextual embedding methods. In static embedding, a vector is generated for each word and remains the same irrespective of the context of a sentence, whereas in contextual embedding the vector of the same word changes depending on the context of the sentence. Some of the most popular static word embedding methods are Word2Vec, Glove, and Fast-text, and some of the well-known methods to obtain contextual embeddings are Embeddings from Language Models (ELMo), Bidirectional Encoder Representations from Transformers (BERT), Transformer-XL, etc. As this work focuses mainly on static embedding methods, the following definitions give the basic idea behind each static embedding algorithm.

Word2Vec embedding uses a shallow network to obtain the representation on a large corpus. It has two approaches, namely Skip-gram and Continuous Bag of Words (CBOW). Glove stands for Global Vectors for Word Representation, where the representation is obtained by factorizing the co-occurrence matrix of the words. It also considers global word co-occurrence and, hence, the name global vectors. Fast-text embedding generates the vectors even at the subword levels (n-grams), thus helping in handling the words that are out of vocabulary.

II. CURRENT STATE ANALYSIS

In [1], the performance of word2vec embedding was examined when combined with Convolutional Neural Networks (CNNs) for classification. Comparing CBOW and skip-gram showed that the former performs better on texts that follow a uniform format, such as news articles, while the latter performs well for non-uniform scenarios that are best seen in social media posts such as Twitter. In [2], a new approach was proposed, adding Bidirectional LSTM (BiLSTM) and attention layers with CNNs to improve classification performance. In [3], word2vec was combined with Long-Short-Term Memory (LSTM) for sentiment analysis. The skip-gram approach outperformed CBOW, as the data was not in a uniform format. In contrast to classic word2vec, a novel method was proposed in [4], called W2C-CL, to scale the effective training of large corpora with an adjustable number of iterations and a large batch size, while treating high-frequency and low-frequency words equally. In [5], a combination of word2vec hyperparameters was presented, which can contribute to improved performance of sentiment analysis, while an increase in dimensions can reduce performance. However, this study concluded that hyperparameter combinations are mostly task-specific.

In [6], an optimization of Glove embedding was proposed with the help of constraints by incorporating more information, so that neighboring and similar words are as close as possible in the resultant vector space. In [7], the performance of Glove embedding was compared with the well-known Bidirectional Encoder Representations from Transformers (BERT) model for sentiment analysis, showing that embedding methods such as Glove outperform BERT, although the latter provides better contextual-based vectors. In [8], Glove was combined with LSTM to identify emotions in tweets. In [9], Glove was combined with CNN to identify features, which were then fed to BiLSTM, showing how a word embedding method is used in brief text for sentiment analysis. In [10], an LSTM-Gated Recurrent Unit (LSTM-GRU) model was combined with Glove for sentiment analysis, enhancing LSTM capability with one of the positive aspects of GRU to avoid resource waste.

In [11, 12], Fast-text embedding with CNN was examined along with its impact on text processing and sentiment analysis, evaluating its performance against benchmark datasets. In [13], the ability of Fast-text embedding to extract features even in non-standard word representations was shown and combined with SVM for sentiment analysis. In [14], a new hybrid method was presented, where the ability of Fast-text embedding was extended with parts-of-speech tagging and word position information for word embedding. In [15], the performance of the Word2Vec, Glove, and Fast-text embedding methods was compared, concluding that Fast-text was more accurate than the other two. In [16], dense uninterpretable word embeddings were transformed into a sparse and interpretable space using sparse coding, resulting in human-understandable embeddings. In [17] the word embeddings of some well-known methods, such as word2vec, glove, and LSA, were augmented into four feature sets, each representing the similarity and dissimilarity of the pair of words, and then feeding an SVM for classification. This approach had some shortcomings while providing decent classification performance.

During the implementation of these embedding techniques, it was observed that current methods have embedded vectors with a dimension of at least 100 and most of them must be loaded before use. This presents an issue when the system has limited memory available. The following are the contributions of the proposed work:

- The proposed method considers a lesser-dimensional vector compared to some of the existing embedding methods, which in turn reduces the model complexity.
- Obtains the embedding dynamically instead of working on pre-trained embeddings.
- Achieves better sentiment classification results.

III. PROPOSED WORK

A. Dataset

A Twitter dataset is created for the experiment by combining two distinct labeled datasets: one is a sentiment dataset with labels that have positive or negative polarities [19], while the labels in the other dataset are essentially non-polar by nature, such as sarcasm, figurative language, and regular statements [20]. The samples from both datasets were randomly sampled so that each category can have close to 17000 to 20000 samples. The final dataset consists of 97000 samples with 5 labels, resulting in the combination of polarity and emotion-based labels. The challenge expected in this dataset is to identify and distinguish the ambiguity offered by the polarity and emotion in a given text statement.

TABLE I. DATASET DETAILS

Class	Number of samples
Positive	20000
Negative	20000
Regular	17000
Sarcasm	20000
Figurative	20000

B. Sentiment Weights

Weights are used to obtain the degree of sentiment possessed by a word related to a particular class sentiment [18]. The sentiment weight has two approaches:

- Odds Ratio (OR): This metric ranks the words based on their relevance to a particular class with the help of their occurrence frequency. Positive odds ratio indicates a stronger tendency of that word's usage in a class. Frequency calculation and OR are represented as follows:

$$Fr^{*}OR(w_i, R^k) \approx log \frac{a_i^k * \left(t_1 + t_2 - t_i - b_i^k\right)}{(t_k - a_i^k) * b_i^k} \quad (1)$$

where R^k denotes the document, k is a class, a_i^k is the number of documents that contain the word w_i and belong to a class, b_i^k is the number of documents that contain the word w_i but do not belong to the class, t_k is the total

number of documents in the class, and the sentiment weight $SW(w_i)$ can be obtained by:

$$SW(w_i) = \max[OR(w_i, R^1), OR(w_i, R^2)] \quad (2)$$

- Weighted Odds (WO): This metric applies weight to odds ratio to obtain the amount of variance in word frequency across the full collection of documents. In some cases, the WO outperforms OR [18]. Frequency calculation and WO is given by:

$$Fr^*WO(w_i, R^k) \approx \left(\frac{a_i^k}{t_i}\right)^z log\left(\frac{a_i^k(t_1+t_2-t_i)}{b_i^k t_i}\right)^{1-z} \quad (3)$$

and the sentiment weight $SW(w_i)$ is:

$$SW(w_i) = \max[WO(w_i, R^1), WO(w_i, R^2)] \quad (4)$$

C. Method

The proposed method generates a sparse vector for each given word. To generate the sparse vector, each character is associated with its own predefined indices, which can be seen in Table II.

TABLE II. CHARACTERS AND THEIR INDEXES

Character	0	1	2	3	4	5	6	7	8	9
Index	0	1	2	3	4	5	6	7	8	9
Character	a	b	c	d	e	f	g	h	i	j
Index	10	11	12	13	14	15	16	17	18	19
Character	k	l	m	n	o	p	q	r	s	t
Index	20	21	22	23	24	25	26	27	28	29
Character	u	v	w	x	y	z	-	’	,	.
Index	30	31	32	33	34	35	36	37	38	39

Since the proposed experiment focuses only on the English language, only the English alphanumeric characters are considered. The steps involved in generating the vector are presented in Algorithm 1.

```
Algorithm 1: Creating Sparse Embedding
Vector
function GENERATE_SPARSE_VECTOR(word);
Input: word w, consisting of n number of
characters with c being the character
Output: A sparse vector of dimension 40
n_normalized_positions ←
  get_normalize_positions(word)
for c, c_pos_norm in word,
  n_normalized_positions do
  onehot_encoding ←
    get_onehot_encoding(c)
    augmented_vector ← onehot_encoding
      * index(c) * c_pos_norm
    w_vector ← w_vector +
      augmented_vector
end for
word_vector ← w_vector +
  sentiment_polarity(word)
weighted_word_vector ← word_vector *
  sentiment_weight(word)
return weighted _word_vector
```

As can be seen in the algorithm, a word w is considered a collection of characters c, and the number of characters in each word is represented by n. For any given word, the working principle of the proposed algorithm is explained as follows:

- The normalized vector of character positions is obtained. The positions are normalized to scale down the respective positional coefficient values to range between 0 to 1.
- The one-hot encoding of each character is obtained, whose length is equal to the number of characters under consideration. It can be observed from Table II that the number of characters is 40. Hence the dimension of the vector will be 40, and in the resultant vector, the value at the index corresponding to the respective character will be 1 (one).
- The scalar product of one-hot encoded vectors of each character, the index of the respective character, and their corresponding normalized positional coefficients are obtained, and the resulting vectors are summed up to form a single vector to get a vector of dimension 40. One of the main reasons for scalar products of the character's normalized positional coefficients is to avoid any duplication of the vectors for different words with the same characters, such as "bat" and "tab".
  i. For better understanding, let us consider a 3-dimensional vector for the characters, where $a = [1,0,0]$, $b = [0,1,0]$, and $t = [0,0,1]$. Now, vector summation is performed, and $[1,1,1]$ is obtained for both "bat" and "tab".
  ii. Considering the scalar product approach, the resultant vectors will be different for both words. For example, let's consider the positional coefficients without applying normalization, i.e., $[1,2,3]$ for first, second, and third characters, respectively, then the resultant vector for "bat" will be $[2,1,3]$, whereas for "tab" will be $[3,1,2]$.
- Then the polarity of the word is added to the resultant vector. Python offers various libraries to obtain the polarity of the word, such as VADER and SpaCy. In the proposed method, polarity is obtained using the SpaCy library.
- Finally, scalar multiplication is performed, where the resultant vector is multiplied by the Sentiment Weight (SW) of the word.

1) Example

Let's consider the word "sad" for demonstration purposes. The vector is obtained as follows:

- The normalized position values for each letter in the word are:

Letter	Position	Normalized position values
s	1	0.26726124
a	2	0.53452248
d	3	0.80178373

- The index vector of dimension 40 for each letter in the word, based on Table II, for the letter will be:

Letter	Index vector for the letter
s	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 28, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
a	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
d	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

- The resultant vector obtained by the product of the index vector and the normalized positional values of each letter will be:

Word	Resultant sparse vector
sad	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.345224, 0.0, 0.0, 10.423188, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 7.483314, 0.0, 0.0, 0 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

In this implementation, the decimal value is considered up to 6 positions.

- The polarity for the word "sad" is -0.5, and the scalar addition will result in the following vector:

Word	Resultant vector after adding word polarity
sad	[-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, 4.84522484, -0.5, -0.5, 9.92318843, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, 6.98331477, -0.5, -0.5, -0.5.0, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5]

- Finally the sentiment weights for the word "sad" are:

Sentiment weight type	Odds ratio	Weighted odds
Sentiment weights	1.1887	0.067

The final resultant embedding vectors for the word "sad" after performing the scalar product are as follows,

Sentiment weight	Resultant embedding vectors
Odds ratio	[-0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, 5.759518, -0.59435, -0.59435, 11.795694, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, 8.301066, -0.59435, -0.59435, -0.59435.0, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435, -0.59435]
Weighted odds	[-0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, 0.328021, -0.03385, -0.03385, 0.671799, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, 0.472770, -0.03385, -0.03385, -0.03385.0, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385, -0.03385]

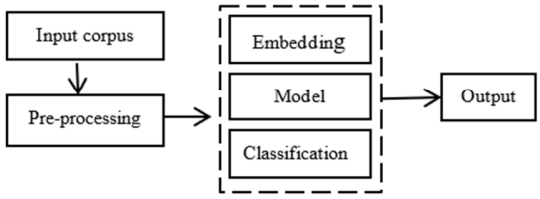

Fig. 1. Workflow of the model.

Figure 1 shows the workflow of the experimental implementation. The implementation follows a straightforward flow, where the input first undergoes preprocessing to obtain the clean text, with commonly known steps such as stop-word removal, lemmatization, etc., and then it is fed to the core model consisting of embedding and classification steps.

IV. RESULTS AND DISCUSSION

The proposed sparse vector-based embedding was compared with some well-known static embedding algorithms, such as Glove, Word2Vec, and Fast-Text. The results showed that the proposed method achieved fairly good training and validation accuracies compared to those of the aforementioned embedding methods. Figure 2 shows that existing methods start to overfit after a certain point. The performance of the proposed embedding method was verified using three well-known machine learning models, namely SVM, Radial Basis Function Neural Network (RBF-NN), and LSTM.

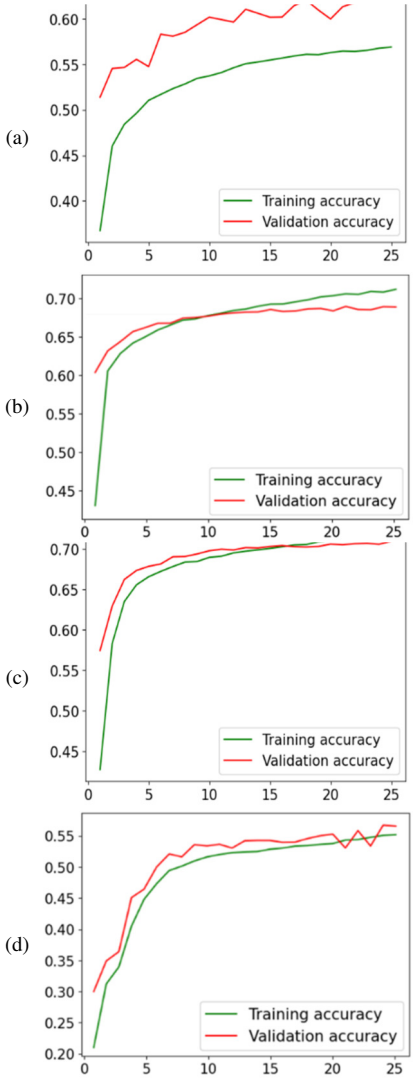

Fig. 2. Training and validation accuracies of: (a) Proposed method, (b) Glove embedding, (c) Fast-text embedding, (d) Word embedding.

A. Classification Performance using SVM

SVM is a well-known linear classification model. Table III presents the classification performance of each embedding method using SVM. With Term Frequency (Fr) calculation of Odds Ratio (OR) and Weighted Odds (WO) being represented as Fr*OR and Fr*WO respectively, it is observed that the proposed embedding method outperforms the existing ones.

TABLE III. CLASSIFICATION PERFORMANCE OF THE PROPOSED APPROACH USING SVM

Classifier	Embedding approach	Embedding dimensions	Sentiment types	Accuracy using Fr*OR	Accuracy using Fr*WO
SVM (Liner Kernel) Value of C = 0.9 and Gamma = 1.0 – Empirically set	Fast-Text (FT) embedding	300	Positive	69.83	78.54
			Negative	71.14	79.34
			Figurative	65.58	72.34
			Sarcasm	71.76	79.10
			Regular	73.25	76.42
			Overall Accuracy	70.31	77.14
	Glove embedding	100	Positive	75.91	78.35
			Negative	78.66	76.52
			Figurative	71.23	74.56
			Sarcasm	80.19	81.22
			Regular	79.59	81.48
			Overall Accuracy	77.11	78.42
	Word2vec	100	Positive	82.35	83.13
			Negative	81.52	81.96
			Figurative	80.25	81.24
			Sarcasm	83.19	85.51
			Regular	81.54	82.68
			Overall Accuracy	81.77	82.90
	Proposed Method	40	Positive	83.12	84.42
			Negative	81.91	82.34
			Figurative	84.54	84.91
			Sarcasm	88.67	89.56
			Regular	79.54	80.53
			Overall Accuracy	83.55	84.35

B. Classification Performance using RBF-NN

RBF-NN is known to have a strong tolerance to input noise and the ability to generalize. Table IV shows the classification performance of each embedding method with RBF-NN. It can be observed that the proposed embedding method surpassed all others using the Fr*OR approach, with an overall classification accuracy of 84.55%. It also performed on par with word2vec (w2v), using the Fr*WO approach, where the w2v embedding method achieved 87.66% classification accuracy. These results indicate that the proposed method has promising performance in combination with RBF-NN.

C. Classification Performance using LSTM

LSTM is an RNN that is specifically designed to handle sequential data. As shown in Table V, the proposed embedding method outperformed the other existing methods.

TABLE IV. CLASSIFICATION PERFORMANCE OF THE PROPOSED APPROACH USING RBF-NN

Classifier	Embedding approach	Embedding dimensions	Sentiment types	Accuracy using Fr*OR	Accuracy using Fr*WO
RBF-NN Parameters being $\alpha i = 0.1$, $\beta i = 1.0$ (Empirical)	Fast-Text (FT) embedding	300	Positive	84.86	87.85
			Negative	83.22	86.75
			Figurative	80.68	86.80
			Sarcasm	81.96	85.50
			Regular	83.74	84.40
			Overall Accuracy	82.89	86.26
	Glove embedding	100	Positive	83.23	86.18
			Negative	81.87	86.69
			Figurative	83.16	86.34
			Sarcasm	81.25	85.91
			Regular	84.06	86.88
			Overall Accuracy	82.71	86.40
	Word2vec	100	Positive	81.80	87.91
			Negative	86.95	87.14
			Figurative	81.43	87.20
			Sarcasm	85.67	88.81
			Regular	86.15	87.24
			Overall Accuracy	84.40	87.66
	Proposed Method	40	Positive	84.16	85.21
			Negative	86.58	85.00
			Figurative	83.75	88.51
			Sarcasm	83.27	88.58
			Regular	85.01	85.76
			Overall Accuracy	84.55	86.61

TABLE V. CLASSIFICATION PERFORMANCE OF THE PROPOSED APPROACH USING LSTM

Classifier	Embedding approach	Embedding dimensions	Sentiment types	Accuracy using Fr*OR	Accuracy using Fr*WO
LSTM Parameters are Empirically set	Fast-Text (FT) embedding	300	Positive	84.18	83.97
			Negative	82.19	83.66
			Figurative	84.14	84.75
			Sarcasm	82.99	83.29
			Regular	84.88	85.35
			Overall Accuracy	83.67	84.20
	Glove embedding	100	Positive	87.66	86.81
			Negative	84.02	88.96
			Figurative	84.28	84.50
			Sarcasm	85.45	87.64
			Regular	87.87	88.19
			Overall Accuracy	85.85	87.22
	Word2vec	100	Positive	86.51	87.28
			Negative	88.68	87.29
			Figurative	88.99	88.22
			Sarcasm	86.89	88.94
			Regular	85.34	89.70
			Overall Accuracy	87.28	88.28
	Proposed Method	40	Positive	90.26	90.43
			Negative	89.07	92.83
			Figurative	88.62	89.20
			Sarcasm	88.39	90.85
			Regular	90.95	90.01
			Overall Accuracy	89.45	90.66

Figure 3 shows the overall performance of the proposed method across the three models. It can be observed that the proposed approach performs very well with the LSTM model,

achieving an overall maximum classification accuracy of 90.66. The proposed approach drastically reduces the number of trainable parameters, thus effectively using the available resources. Unlike the other existing methods that require loading an embedding dictionary, in the proposed approach, the embedding can be generated dynamically, thus making it both a CPU- and storage-efficient approach.

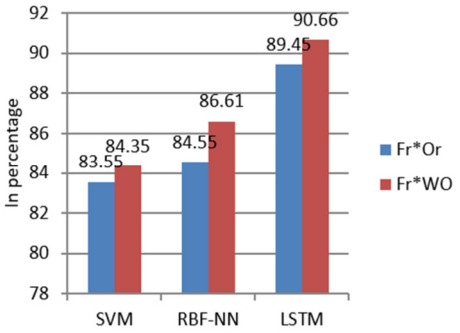

Fig. 3. Classification performance of the proposed method with different models.

D. Embedding Dimensions and Performance

This section provides an overview of how the size of a dimension might affect classification by comparing the performance of several embedding algorithms and their corresponding embedding dimensions. Figure 4 depicts the different models' behavior with different embedding dimensions.

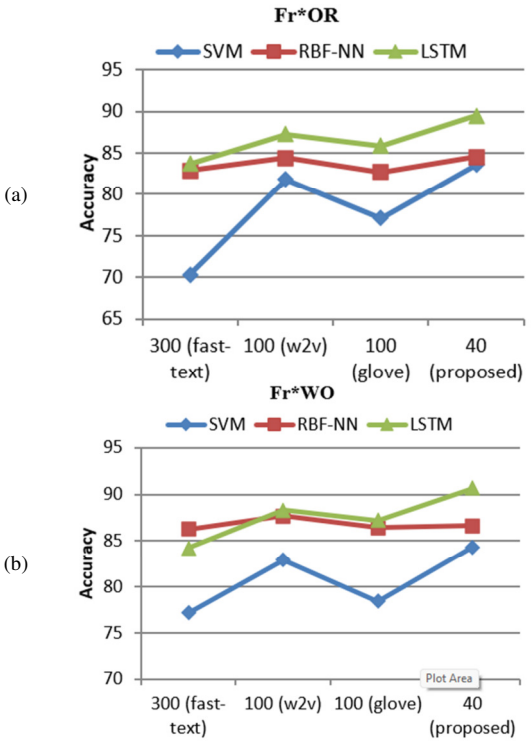

Fig. 4. Classification performance of the proposed method with: (a) Fr*OR-based sentiment weights, (b) Fr*WO-based weights.

V. CONCLUSION

This study presented a sparse vector-based embedding technique, comparing its classification performance with some popular embedding techniques. Essentially, each word's vector is created by combining one-hot encoded vectors with mathematical calculations. The results showed that despite the proposed embedding technique using a lesser-dimensional vector, it provides a higher classification accuracy compared to existing embedding methods. The primary goal of this study was to develop an approach in which vectors and their values are interpretable while having an efficient vector dimension. This is in contrast to the traditional non-interpretable embedding methods that are thought to be generated by a black-box method, where the vectors are not human-understandable. The experimental results are promising, showing that an embedding vector can be created in an interpretable manner with appropriate traditional mathematical calculations. Future work will progressively increase the accuracy of the vector to accommodate considerably more grammatical and linguistic information and better manage the uncertainty that develops during sentiment classification.

REFERENCES

[1] B. Jang, I. Kim, and J. W. Kim, "Word2vec convolutional neural networks for classification of news articles and tweets," *PLOS ONE*, vol. 14, no. 8, Aug. 2019, Art. no. e0220976, https://doi.org/10.1371/journal.pone.0220976.

[2] B. Jang, M. Kim, G. Harerimana, S. Kang, and J. W. Kim, "Bi-LSTM Model to Increase Accuracy in Text Classification: Combining Word2vec CNN and Attention Mechanism," *Applied Sciences*, vol. 10, no. 17, Aug. 2020, Art. no. 5841, https://doi.org/10.3390/app10175841.

[3] P. F. Muhammad, R. Kusumaningrum, and A. Wibowo, "Sentiment Analysis Using Word2vec And Long Short-Term Memory (LSTM) For Indonesian Hotel Reviews," *Procedia Computer Science*, vol. 179, pp. 728–735, 2021, https://doi.org/10.1016/j.procs.2021.01.061.

[4] B. Li, A. Drozd, Y. Guo, T. Liu, S. Matsuoka, and X. Du, "Scaling Word2Vec on Big Corpus," *Data Science and Engineering*, vol. 4, no. 2, pp. 157–175, Jun. 2019, https://doi.org/10.1007/s41019-019-0096-6.

[5] T. Adewumi, F. Liwicki, and M. Liwicki, "Word2Vec: Optimal hyperparameters and their impact on natural language processing downstream tasks," *Open Computer Science*, vol. 12, no. 1, pp. 134–141, Mar. 2022, https://doi.org/10.1515/comp-2022-0236.

[6] F. Sakketou and N. Ampazis, "A constrained optimization algorithm for learning GloVe embeddings with semantic lexicons," *Knowledge-Based Systems*, vol. 195, May 2020, Art. no. 105628, https://doi.org/10.1016/j.knosys.2020.105628.

[7] A. Khatri and P. P, "Sarcasm Detection in Tweets with BERT and GloVe Embeddings," in *Proceedings of the Second Workshop on Figurative Language Processing*, 2020, pp. 56–60, https://doi.org/10.18653/v1/2020.figlang-1.7.

[8] P. Gupta, I. Roy, G. Batra, and A. K. Dubey, "Decoding Emotions in Text Using GloVe Embeddings," in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, Greater Noida, India, Feb. 2021, pp. 36–40, https://doi.org/10.1109/ICCCIS51004.2021.9397132.

[9] A. Pimpalkar and J. R. Raj, "MBiLSTMGloVe: Embedding GloVe knowledge into the corpus using multi-layer BiLSTM deep learning model for social media sentiment analysis," *Expert Systems with Applications*, vol. 203, Oct. 2022, Art. no. 117581, https://doi.org/10.1016/j.eswa.2022.117581.

[10] R. Ni and H. Cao, "Sentiment Analysis based on GloVe and LSTM-GRU," in *2020 39th Chinese Control Conference (CCC)*, Shenyang, China, Jul. 2020, pp. 7492–7497, https://doi.org/10.23919/CCC50068.2020.9188578.

[11] M. Umer *et al.*, "Impact of convolutional neural network and FastText embedding on text classification," *Multimedia Tools and Applications*, vol. 82, no. 4, pp. 5569–5585, Feb. 2023, https://doi.org/10.1007/s11042-022-13459-x.

[12] I. N. Khasanah, "Sentiment Classification Using fastText Embedding and Deep Learning Model," *Procedia Computer Science*, vol. 189, pp. 343–350, 2021, https://doi.org/10.1016/j.procs.2021.05.103.

[13] D. A. Wibowo and A. Musdholifah, "Sentiments Analysis of Indonesian Tweet About Covid-19 Vaccine Using Support Vector Machine and Fasttext Embedding," in *2021 4th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, Yogyakarta, Indonesia, Dec. 2021, pp. 184–188, https://doi.org/10.1109/ISRITI54043.2021.9702871.

[14] F. Alotaibi and V. G. Gupta, "Sentiment Analysis System using Hybrid Word Embeddings with Convolutional Recurrent Neural Network," *The International Arab Journal of Information Technology*, vol. 19, no. 3, 2022, https://doi.org/10.34028/iajit/19/3/6.

[15] S. Khomsah, R. D. Ramadhani, and S. Wijaya, "The Accuracy Comparison Between Word2Vec and FastText On Sentiment Analysis of Hotel Reviews," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 6, no. 3, pp. 352–358, Jun. 2022, https://doi.org/10.29207/resti.v6i3.3711.

[16] A. Templeton, "Word Equations: Inherently Interpretable Sparse Word Embeddingsthrough Sparse Coding." arXiv, 2020, https://doi.org/10.48550/ARXIV.2004.13847.

[17] S. Selva Birunda and R. Kanniga Devi, "A Review on Word Embedding Techniques for Text Classification," in *Innovative Data Communication Technologies and Application*, vol. 59, J. S. Raj, A. M. Iliyasu, R. Bestak, and Z. A. Baig, Eds. Springer Singapore, 2021, pp. 267–281.

[18] S. Prakash, T. Chakravarthy, and E. Kaveri, "Statistically weighted reviews to enhance sentiment classification," *Karbala International Journal of Modern Science*, vol. 1, no. 1, pp. 26–31, Sep. 2015, https://doi.org/10.1016/j.kijoms.2015.07.001.

[19] M. Michailidis, "Sentiment140 dataset with 1.6 million tweets." Kaggle, [Online]. Available: https://www.kaggle.com/datasets/kazanova/sentiment140.

[20] J. Nikhil, "Tweets with Sarcasm and Irony." Kaggle, [Online]. Available: https://www.kaggle.com/datasets/nikhiljohnk/tweets-with-sarcasm-and-irony.