Optimal Word Embeddings for Text Classification Representations

Anonymous ACL submission

Abstract

Word embeddings have introduced a compact and efficient way of representing text for further downstream natural language processing (NLP) tasks. Most word embedding algorithms are optimized at the word level. However, many NLP applications require text representations of groups of words, like sentences or paragraphs. In this paper, we propose a supervised algorithm for finding text embeddings that uses labeled texts to produce an "optimal" weighted average of word embeddings for a given task. Our proposed text embedding algorithm combines the compactness and expressiveness of the word-embedding representations with the human-interpretability of a BoW-type model, where weights correspond to actual words. Numerical experiments across different domains show the competence of our algorithm.

1 Introduction

Word embeddings, or a learned mapping from a vocabulary to a vector space, are essential tools for state-of-the-art Natural Language Processing (NLP) techniques. Dense word vectors, like Word2Vec (Mikolov et al., 2013a) and GLoVE (Pennington et al., 2014), are compact representations of a word's semantic meaning, as demonstrated in analogy tasks (Levy and Goldberg, 2014) and part-of-speech tagging (Lin et al., 2015).

Most downstream tasks, like sentiment analysis and information retrieval (IR), are used to analyze groups of words, like sentences or paragraphs. For this paper, we refer to this more general embedding as a "text embedding".

In this paper we propose a supervised algorithm for finding text embeddings that uses labeled texts to produce an "optimal" weighted average of word embeddings for texts for that task.

Our algorithm computes the text embedding by performing weighted average of the words present in the given text, these weights are computed in a supervised manner by using labels provided with the text. The weights that we obtain define the importance of the words with respect to the supervised learning task at hand. Words with higher absolute weights have higher importance in the given supervised task. For example, in a supervised task of classifying movie reviews talking about action movies from romantic movies, words like "action", "romance", "love", and "blood", will get precedence over words, like "movie", "i", and "theater". This leads to the shifting of the text vector towards words with larger weights, as can be seen in Figure 1.

When we use unweighted averaged word embedding (**UAEm**) (Wieting et al., 2015) for representing the two reviews, we see that all the words get the same importance, due to which the reviews - **"I like action movies"** and **"I prefer romance flicks"** - end up close to each other in the vector space. Our algorithm, on the other hand, identifies "romance" and "action" as two important words in the vocabulary for the supervised task, and assigns weights with high absolute value to these words. This leads to shifting of the representation of the two reviews toward their respective important words in the vector space, increasing the distance between them. This indicates that, for the task of identifying an action movie review from a romantic movie review, there is less similarity between the sentences than what was indicated by **UAEm**.

Our algorithm has many advantages over simpler text embedding approaches, like bag-of-

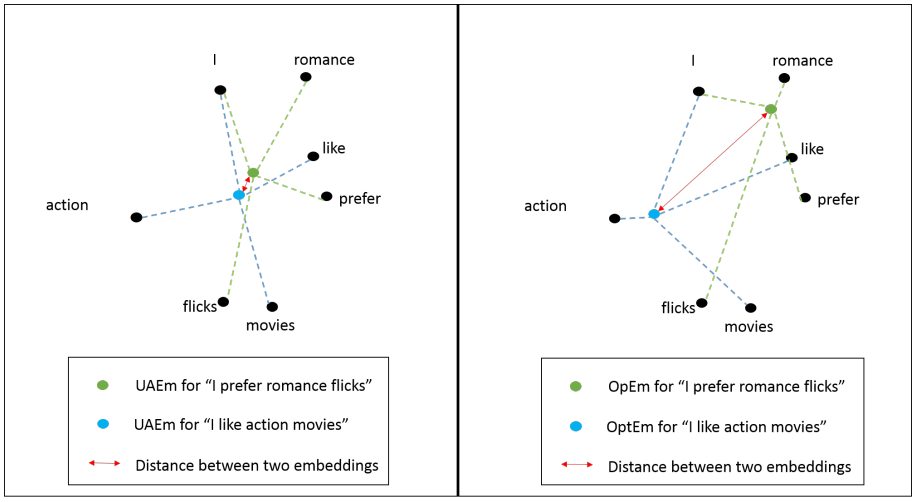

Figure 1: Unweighted (UAEm) (**left**) and Optimal embeddings (OptEm) (**right**) of two movie reviews in feature space. Distance between the two reviews increases for OptEm representation of the text.

words (BoW) and the averaged word-embedding schemes discussed in Section 2. In Section 4, we show results from experiments on different datasets. In general, we observed that our algorithm is competitive with other methods. Unlike the simpler algorithms, our approach finds a *task-specific* representation. While BoW and some weighting schemes, like tf-idf, rely only on word frequencies to determine word importance, our algorithm computes how important the word is to a specific task. We believe that for some applications, this task-specific representation is important for performance; one would expect the importance of words to be very different whether you are trying to do topic modeling or sentiment analysis.

Additionally, our algorithm yields a more interpretable result than looking at the weights corresponding to the word-embedding dimensions that have no human-interpretable meaning. Effectively, our text embedding algorithm combines the compactness and expressiveness of the word-embedding representations with the human-interpretability of a BoW-type model.

Rest of the paper is organized as follows: in section 2, we discuss related work. Later, in section 3, we present a detailed explanation and mathematical justification to support our proposed algorithm. In section 4, we present numerical experiments, followed by conclusions and future work in section 5.

2 Related Work

Various representation techniques for text have been introduced over the course of time. In the recent years, none of these representations have been as popular as the word embeddings, such as Word2Vec (Mikolov et al., 2013a) and GLoVE (Pennington et al., 2014), that took contextual usage of words into consideration. This has led to very robust word and text representations.

Text embedding has been a more challenging problem over word embeddings due to the variance of phrases, sentences, and text. Le (2014) developed a method to generate the embeddings that outperforms the traditional bag-of-words approach (Harris, 1954). More recently, deeper neural architectures have been developed to generate these embeddings. Some of these architectures involve sequential information of text, such as LSTMs (Palangi et al., 2016).

Methods have been developed that use word embeddings to generate text embeddings without having to train on whole texts. These methods are less costly than the ones that train directly on whole text, and can be implemented faster. Unlike the bag-of-words representation, most of these embeddings are label-independent.

Unweighted average word embedding (Wieting et al., 2015) generated text embeddings by computing average of the embeddings of all the words occurring in the text. This is one of the most popular methods of computing text embeddings from trained word embeddings, and, though simple, has been known to outperform the more com-

plex text embedding models especially in out-of-domain scenarios. Arora (2017) provided a simpler method to enhance the performance of text embedding generated from simple averaged embedding by the application of PCA.

The unsupervised text embedding methods face the problem of importance-allocation of words while computing the embedding. This is important, as word importance determines how biased the text embedding needs to be towards the more informative words. DeBoom (2016) introduced a method that would assign importance to the words based on their tf-idf scores in the text.

Our method generates weights based on the importance of the words perceived through a supervised approach. We use classifiers to determine the weights of the words based on their importance captured through the procedure. The advantage of this method over other methods is that we keep the simplicity of Wieting's algorithm (2015), while incorporating the semantically agreeable weights for the words.

3 Optimal Word Embeddings

A sentence, paragraph or document can be represented using word2vec as follows:

$$A_i = \sum_{j=1}^{k} \delta_{ij} \lambda_j v_j \tag{1}$$

where,
$A_i \in R^n$ is a vectorial representation (we will refer it as **text2vec** in rest of the paper) of *i*th sample; We will assume that A_i is the i^{th} row of a matrix $A \in R^{m \times n}$ containing a collection of m documents, k is the number of words in the word2vec corpus V;
$\lambda_j \in R$ is a weighting factor associated with the *j*th word $v_j \in V$. Note that for the widely used averaged text2vec representation (Wieting et al., 2015), $\lambda_j = 1, \forall j$; and
δ_{ij} is a normalized occurrence count. It is the number of times *j*th word appears in the document i divided by the total number of words in the document i.

Our proposed algorithm assumes that we have a supervised classification problem for which we want to find an optimal representation at the document (text) level from the word embeddings.

More concretely, we consider the problem of classifying m points in the n-dimensional real space R^n, represented by the $m \times n$ matrix A, according to membership of each point A_i in the classes +1 or -1 as specified by a given $m \times m$ diagonal matrix D with ones or minus ones along its diagonal.

In general this linear classification problem can formulated as follows:

$$\begin{aligned} \min_{(w,\gamma,y\geq 0)} \quad & cL(y) + R(w) \\ \text{s.t.} \quad & D(Aw - e\gamma) + y \quad \geq \quad e \end{aligned} \tag{2}$$

where $e \in \mathbb{R}^{m \times 1}$ is a column of ones; $y \in \mathbb{R}^{m \times 1}$ is a slack vector; and $(w, \gamma) \in \mathbb{R}^{(n+1) \times 1}$ represents the separating hyperplane. L is a loss function that is used to minimize the misclassification error, R is a regularization function used to improve generalization, and c is a constant that controls the trade-off between error and generalization.

Note that, if $L(.)=\|(.)_+\|_2^2$ and $R(.)=\|.\|_2^2$, then equation (2) corresponds to an SVM formulation (Lee and Mangasarian, 2001). The corresponding unconstrained convex optimization problem is given as:

$$\min_{\omega,\gamma} c\|(e - D(Aw - e\gamma))_+\|_2^2 + \|w\|_2^2 \tag{3}$$

which we will denote by

$$(w, \gamma) = SVM(A, D, c) \tag{4}$$

From (1), we can rewrite A as:

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{k} \delta_{1j} \lambda_j v_j \\ \sum_{j=1}^{k} \delta_{2j} \lambda_j v_j \\ \vdots \\ \sum_{j=1}^{k} \delta_{mj} \lambda_j v_j \end{bmatrix} \tag{5}$$

That is,

$$A = \Delta \Lambda V \tag{6}$$

where $\Delta \in \mathbb{R}^{m \times k}$; is a matrix of occurrences count with δ_{ij} in the (i, j) position. $\Lambda = diag((\lambda_1, \ldots, \lambda_k)) \in \mathbb{R}^{k \times k}$, and $V \in \mathbb{R}^{k \times n}$ is the matrix whose rows are all the word2vec vectors considered in the word2vec corpus or dictionary.

From (3) and (6),

$$\min_{w,\gamma,\boldsymbol{\lambda}} c\|(e - D((\Delta \Lambda V)w - e\gamma))_+\|_2^2 + \|w\|_2^2 \tag{7}$$

where $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_k)$.

Formulation (7) is a biconvex optimization problem, which can be solved using alternate optimization (Bezdek and Hathaway, 2003). By

solving this problem, not only do we obtain an SVM-type classifier, but also learn the optimal importance weights for each word in our corpus $(\lambda_1, \ldots, \lambda_k)$. Though we could have restricted the λ_i to be positive, we choose to leave them unconstrained in order to make our algorithm more scalable and computationally efficient. Another interesting consideration would be to add a relative importance constrained on addition to the non negativity bounds of the form:

$$\lambda_1 + \lambda_2, \ldots, \lambda_k = 1 \tag{8}$$

but again, we choose not too for computationally efficiency. We will explore this option in the future.

In (7), if we fix Λ to a constant $\bar{\Lambda}$, we have:

$$\tilde{A} = \Delta\bar{\Lambda}V \tag{9}$$

We can obtain the corresponding optimal solution for (w, γ) by solving $(w*, \gamma*) = SVM(\tilde{A}, D, c)$.

On the other hand, if we fix $(w, \gamma) = (\tilde{w}, \tilde{\gamma})$, we get

$$A\tilde{w} = (\Delta\Lambda V)\tilde{w} = (\Delta\tilde{W})\boldsymbol{\lambda} \tag{10}$$

where $\tilde{W} \in \mathbb{R}^{k\times k} = diag(\tilde{w})$.

Similarly, from (7) and (10) and making $\tilde{M} = \Delta\tilde{W}$, we have

$$\begin{aligned} & \min_{\gamma,\lambda_i} c\|(e - D(\tilde{M}\lambda - e\gamma))_+\|_2^2 + \|\tilde{w}\|_2^2 \\ \equiv\ & \min_{\gamma,\lambda_i} c\|(e - D(\tilde{M}\lambda - e\gamma))_+\|_2^2 \end{aligned} \tag{11}$$

since $\tilde{W}$ is a constant.

We can obtain an approximate optimal $(\boldsymbol{\lambda}, \gamma)$ by solving $(\boldsymbol{\lambda}^*, \gamma^*) = SVM(\tilde{M}, D, c)$. Note that this solution will consider a regularization term for $\boldsymbol{\lambda}$.

We are ready now to describe our proposed alternate optimization (AO) algorithm to solve formulation (7).

One of the advantages of the algorithm is that it can be easily implemented by using existing open-source SVM libraries, like the ones included in scikit-learn (scikit-learn, 2017).

The optimal text embedding algorithm, then, inherits the convergence properties and characteristics of the AO problems (Bezdek and Hathaway, 2003). It is important to note that the set of possible solutions to which Algorithm 1 can converge can include certain type of saddle points (i.e. a point that behaves like a local minimizer only when projected along a subset of the variables). However, it is stated in the paper (Bezdek and Hathaway, 2003) that it is extremely difficult to find examples where converge occurs to a saddle point rather than to a local minimizer.

Algorithm 1: Optimal Text Embedding

Input : Training vocabulary matrix (V);
scaled word occurrence matrix (Δ);
vector of labels $diag(D)$;
max number of iterations *maxiter*;
tolerance *tol*;
regularization parameters c_1 and c_2;

Output: optimal word weight vector $\boldsymbol{\lambda}^*$;
classification hyperplane (w^*, γ^*);

```
Initialize ∀j λ_j=1; Λ_0=diagonal(λ)
i = 0;
while i ≤ maxiter or ‖Λ_i − Λ_{i−1}‖ > tol do
    iter++;
    Given Λ_{i−1}, calculate Ã = ΔΛ̄_{i−1}V;
    Solve (w_i, γ) = SVM(Ã, D, c_1);
    Given (w_i, γ), calculate ΔW̃_i as
      described in equation (10);
    Solve (λ_i, γ) = SVM(M̃, D, c_2);
end
λ* = λ_i;
(w*, γ*) = (w_i, γ_i);
```

In order to further reduce the computational complexity of the proposed algorithm, we can consider a simplified loss function $L(.)=\|.\|_2^2$ and $R(.)=\|.\|_2^2$. Then formulation (7) becomes the corresponding unconstrained convex optimization problem:

$$\min_{w,\gamma,\boldsymbol{\lambda}} c\|e - (D(\Delta\Lambda V)w - e\gamma)\|_2^2 + \|w\|_2^2 \tag{12}$$

Fixing $\Lambda = \tilde{\Lambda}$, from (9) and (12), we have

$$c\|(e - D(\tilde{A}w - e\gamma))\|_2^2 + \|w\|_2^2 \tag{13}$$

This formulation corresponds to a least-squares or Proximal SVM formulation (Fung and Mangasarian, 2001; Suykens and Vandewalle, 1999), and its solution can be obtained by solving a simple system of linear equations. We will denote formulation (13) by

$$(w, \gamma) = LSSVM(\tilde{A}, D, c) \tag{14}$$

If $\bar{A} = \begin{bmatrix} \tilde{A} & \vdots - e \end{bmatrix}$ then the solution to (13) is given by

$$(w, \gamma) = (\bar{A}^T\bar{A} + \frac{1}{c}I)^{-1}\bar{A}^T De \tag{15}$$

On the other hand, fixing $(w, \gamma) = (\tilde{w}, \tilde{\gamma})$, we have

$$\begin{aligned} & \min_{\boldsymbol{\lambda}} c\|e - (D(\Delta\tilde{W}\boldsymbol{\lambda}) - e\gamma)\|_2^2 + \|\tilde{w}\|_2^2 \\ \equiv \; & \min_{\boldsymbol{\lambda}} c\|e - (D(\Delta\tilde{W}\boldsymbol{\lambda}) - e\gamma)\|_2^2 \end{aligned} \tag{16}$$

since $\tilde{w}$ is a constant. Hence,

$$\boldsymbol{\lambda} = ((\Delta\tilde{W})^T(\Delta\tilde{W}))^{-1}(\Delta\tilde{W})^T De(1-\gamma) \tag{17}$$

Furthermore,

$$(\Delta\tilde{W})^T(\Delta\tilde{W}) = \tilde{W}^T\Delta^T\Delta\tilde{W} \tag{18}$$

From (17) and (18),

$$\begin{aligned} \boldsymbol{\lambda} &= (\tilde{W}^T\Delta^T\Delta\tilde{W})^{-1}(\Delta\tilde{W})^T De(1-\gamma) \\ &= (\Delta^T\Delta\tilde{W})^{-1}(\tilde{W})^{-1}\tilde{W}\Delta^T De(1-\gamma) \\ &= \mathbf{diag}(\tfrac{1}{\tilde{w}})(\Delta^T\Delta)^{-1}\Delta^T De(1-\gamma) \end{aligned} \tag{19}$$

For some problems, $\Delta^T\Delta$ can be ill-conditioned, which may lead to incorrect values for $\boldsymbol{\lambda}$. In order to improve conditioning we add a Tikhonov regularization perturbation (Neumaier, 1998). (19) becomes

$$\boldsymbol{\lambda} = \mathbf{diag}(\frac{1}{\tilde{w}})(\Delta^T\Delta + \epsilon I)^{-1}\Delta^T De(1-\gamma) \tag{20}$$

where ϵ is a very small value.

Note that $(\Delta^T\Delta + \epsilon I)^{-1}$ involves calculating the inverse of a $k \times k$ matrix, where k is the number of words in the word2vec dictionary. In some cases, k can be much larger than m, the number of training set examples. If this is the case, we can use the Sherman-Morrison-Woodbury formula (Golub and Van Loan, 1996):

$$(Z + uv^T)^{-1} = Z^{-1} - Z^{-1}u(I + v^T Z^{-1}u)^{-1}v^T Z^{-1} \tag{21}$$

with $Z = \epsilon I$, $u = v = \Delta^T$. Then $(\Delta^T\Delta + \epsilon I)^{-1}$ becomes

$$(\Delta^T\Delta + \epsilon I)^{-1} = \frac{1}{\epsilon}(I - \Delta^T(\Delta\Delta^T + \epsilon I)^{-1}\Delta) \tag{22}$$

which involves inverting an $m \times m$ matrix with $m << k$.

The $\boldsymbol{\lambda}$ we obtained is a vector of weights of the words that would be used in (1) to calculate text2vec of a given sample.

Algorithm 1 can be modified to consider formulation (3) instead of (13) by making two simple changes:

1. Substitute line 6 of Algorithm 1 by: Solve equation (15) to obtain (w_i, γ);
2. Substitute line 8 of Algorithm 1 by: Solve equation (19) to obtain $\boldsymbol{\lambda}$;

4 Experiments

We used binary classification as the task for evaluating our text representation against unweighted averaged (**UAEm**) (Wieting et al., 2015) and weighted averaged text (**WAEm**) representations. We computed WAEm using tf-idf weights as the weights (De Boom et al., 2016). The main reason behind the choice of representations is that our model does not require re-training the entire word embedding models. We used the same text-processing pipeline for all the representations.

We implemented two versions of our Algorithm 1: SVM-based (**SVM-OptEm**) (Formulation 3) and least square SVM-based (**LSSVM-OptEm**) (Formulation 12).

In SVM-OptEm, we used a support vector machine (SVM) (Cortes and Vapnik, 1995) as the classifier. We used a scikit-learn (Pedregosa et al., 2011) implementation of SVM for the experiments.

In LSSVM-OptEm, we used a least square support vector machine (LS-SVM) (Cortes and Vapnik, 1995) as the classifier.

4.1 Datasets

To showcase the performance of our model, we chose fifteen different binary classification tasks over the subsets of different datasets. Twelve public datasets are briefly described in Table 1.

We also performed experiments on three datasets belonging to the insurance domain.

- **BI-1** and **BI-2**: These datasets consist of the claim notes with binary classes based on topic of phone conversation. These notes were taken by call representative of the company after the phone call was completed. For BI-1, we classified the call notes into two categories based on claim complexity: simple and complex. For BI-2, we wanted to identify notes that documented a failed attempt made by the call representative to get in touch with the customer.

- **TRANSCRIPTS**: These datasets consist of the phone transcripts with two classes: pay-by-phone calls and others. These transcripts were generated inside the company for the calls received at the call center. Each call would be assigned a class based on the purpose of the call.

Dataset	Description	Positive Class	Reference
20NEWSGRP-SCI	20 Newsgroup documents	Science-related documents	(Lang, 1995)
AMZN-EX	Amazon reviews	Electronics review	(Blitzer et al., 2007)
AMZNBK-SENT	Amazon book reviews	Positive review	(McAuley et al., 2015; He and McAuley, 2016)
BBC	BBC news articles	Sports article	(Greene and Cunningham, 2006)
BLOG-GENDER	Blog articles	Male Writer	(Mukherjee and Liu, 2010)
DBPEDIA	Wikipedia articles	Artist article	(DBPedia, 2018; Lehmann et al., 2015; Zhang et al., 2015)
IMDB	IMDB movie reviews	Positive review	(Maas et al., 2011)
SCIPAP	Sentences from scientific papers	Owner-written sentence	(Lichman, 2013)
SST	Movie reviews	Positive sentiment	(Socher et al., 2013)
YAHOO-ANS	Questions from Yahoo's question-answer dataset	Health-related question	(Zhang et al., 2015)
YELP-REST	Yelp Restaurant Reviews	Restaurant-related review	(Yelp, 2018)
YELP-STAR	Yelp Reviews	Positive review	(Yelp, 2018)

Table 1: Public datasets used for experiment.

4.2 Word Embeddings

We chose to work on different word2vec-based word embeddings. These word embeddings have either been pre-trained models or in-house trained models. These embeddings were used on the datasets based on their contextual relevance.

- **wikipedia** (Bojanowski et al., 2016): The skip-gram model was trained on English articles in Wikipedia by FastText (Facebook, 2018).
- **google-news** (Mikolov et al., 2013b): The model was trained on Google News Data, and is available on the Google Code website (Google, 2018).
- **amzn**: The skip-gram model was trained in-house on amazon reviews (McAuley et al., 2015; He and McAuley, 2016). Gensim (Řehůřek and Sojka, 2010) was used to train the model.
- **yelp**: The skip-gram model was trained in-house on yelp reviews (Yelp, 2018). Gensim was used to train the model.
- **transcript**: The continuous bag-of-words model was trained in-house on the transcripts generated in the of the calls from call centers. Gensim was used to train the model over approximately 3 million transcripts.
- **claim-notes**: The continuous bag-of-words model was trained in-house on the notes taken by call representatives after the call was completed. C-based code from Google Word2vec website (Google, 2018) was used to train the model over approximately 100 million notes.

We used different word2vec models to verify that our models works well independently of the underlying embedding representation. Moreover, it also gives better contextual representation of words for these datasets.

4.3 Text processing

The method of processing employed on text was similar to the one done for training the word2vec models. This ensured the consistency of word-occurrence in the dataset in lieu to the model that would be used for mapping the words.

Different word2vec models had different processing procedures, such as substitutions based on regular expressions, removal of non-alphabetical words, and lowercasing the text. Accordingly, text-processing was done for the training data.

4.4 Results

To compare performance of the algorithms tested, we decided to use area under curve (AUC) for evaluation. This metric was chosen in order to remove the possibility of unbalanced datasets affecting the efficacy of the accuracy of the models. We also recorded the accuracy scores.

The performance of our models for the experiments can be seen in Table 2.

Our algorithm provides better or comparable performance against **UAEm** and **WAEm**. This performance is achieved over multiple iterations, as seen in Figure 2. The number of iterations required to reach the best performance for our model varies with the dataset and training size.

Our algorithm is found to approach an "equilibrium" stage for the vector $\boldsymbol{\lambda}$, as seen in Figure 3. In other words, with the iterations, the mean difference between the current weights and weights from previous iteration of the words approaches zero. This behavior is seen consistently for all the experimental cases. This shows that our algorithm exhibits good convergence behavior as expected.

Dataset	Word2Vec Model	Data Size		Area Under Curve				Accuracy			
		Train	Test	UAEm	WAEm	SVM-OptEm	LSSVM-OptEm	UAEm	WAEm	SVM-OptEm	LSSVM-OptEm
20NEWSGRP-SCI	google-news	3000	2000	0.9017	0.8587	**0.9242**	0.9104	0.8280	0.8005	**0.8555**	0.8380
AMZN-EX	wikipedia	10,000	10,000	0.9913	0.9879	**0.9918**	**0.9918**	0.9633	0.9616	0.9609	**0.9653**
AMZNBK-SENT	amzn	10,000	10,000	0.9289	0.9000	**0.9374**	0.9264	0.8538	0.8237	**0.8702**	0.8570
BBC	google-news	1,850	500	**0.9966**	0.9826	0.9963	0.9960	0.9700	0.9300	**0.9780**	0.9700
BLOG-GENDER	wikipedia	2,000	1,000	0.7680	0.7413	**0.7905**	0.7545	0.7110	0.6950	**0.7280**	0.6980
DBPEDIA	wikipedia	10,000	10,000	0.9925	0.9915	**0.9936**	0.9920	0.9589	0.9576	**0.9656**	0.9588
IMDB	wikipedia	5,000	2,500	0.9155	0.8642	**0.9340**	0.9100	0.8439	0.7863	**0.8599**	0.8343
SCIPAP	wikipedia	1,500	750	0.8617	0.8432	0.9142	**0.9252**	0.8000	0.7920	0.8600	**0.8787**
SST	google-news	10,000	10,000	0.8868	0.8776	**0.8923**	0.8558	0.8026	0.8031	**0.8070**	0.8032
YAHOO-ANS	wikipedia	20,000	10,000	**0.9249**	0.9195	0.9230	0.8987	0.8464	0.8467	**0.8471**	0.8222
YELP-REST	yelp	40,000	40,000	0.9731	0.9700	**0.9750**	0.9736	0.9198	0.9176	**0.9274**	0.9245
YELP-STAR	yelp	20,000	10,000	0.9705	0.9629	**0.9727**	0.9700	0.9135	0.9041	**0.9165**	0.9111
BI-1	claim-notes	1,508	561	0.8779	0.8646	0.8926	**0.8960**	**0.9198**	0.9180	0.9144	0.9109
BI-2	claim-notes	1,081	238	0.7695	0.7701	**0.8115**	0.7749	0.9118	0.9118	**0.9244**	**0.9244**
TRANSCRIPTS	transcript	5,000	3,000	0.9608	0.9294	**0.9676**	0.9670	0.9150	0.8580	**0.9280**	0.9237

Table 2: Binary text classification AUC and accuracy results for test data for SVM-based implementation (**SVM-OptEm**) and least square SVM-based implementation (**LSSVM-OptEm**). The highest score for each evaluation metric is in boldface.

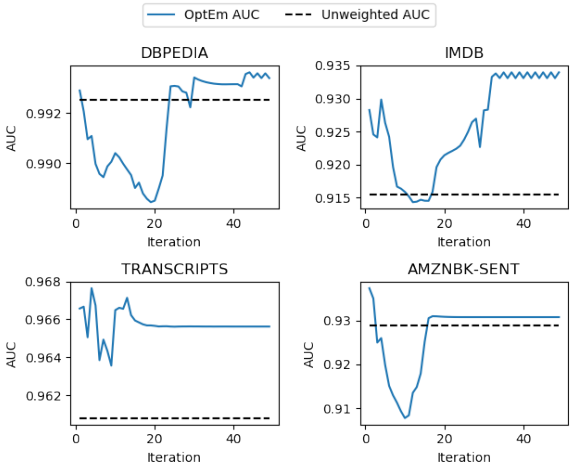

Figure 2: AUC Scores of test data over iterations

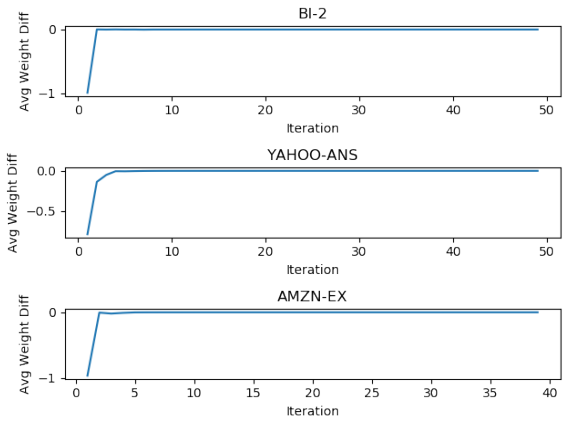

Figure 3: Average of Weight Difference over iteration for 3 datasets. This difference approaches zero over iterations.

4.5 Text Representation

One of the advantages our model holds over **UAEm** and **WAEm** is that our model can be used to extract the most important words in the training set. As our model reconfigures the weights of the words at each iteration, it also indirectly reassigns the degree of importance to these words. We can obtain these words by taking the absolute values of the weights assigned to these words at the end of the iteration. This information can be used for improving different algorithms, such as visual representation of text and topic-discovery, and as features for other models.

Figure 4 shows weights of top 15 words for three of our datasets. Weights assigned to the words are based on the role they play in helping the classifier determine the class of any given sample.

Table 3 shows the top 10 words for three of the datasets. For a human eye, these words clearly have discriminative power with respect to the given classification task. We also found that words that are least informative about the given task have weights close to zero.

5 Conclusions And Future Work

Our paper provides an alternative way of sentence/document representation, based on the realignment of the weights of words in the text. This approach takes labels into consideration while generating the weights of these words. Evaluation of this algorithm was compared against unweighted and weighted average text embedding, and we found that our model performs better or comparatively against them.

Our model also brings additional benefits to the table. It provides a ranking of the relevance of the words with respect to the text classification prob-

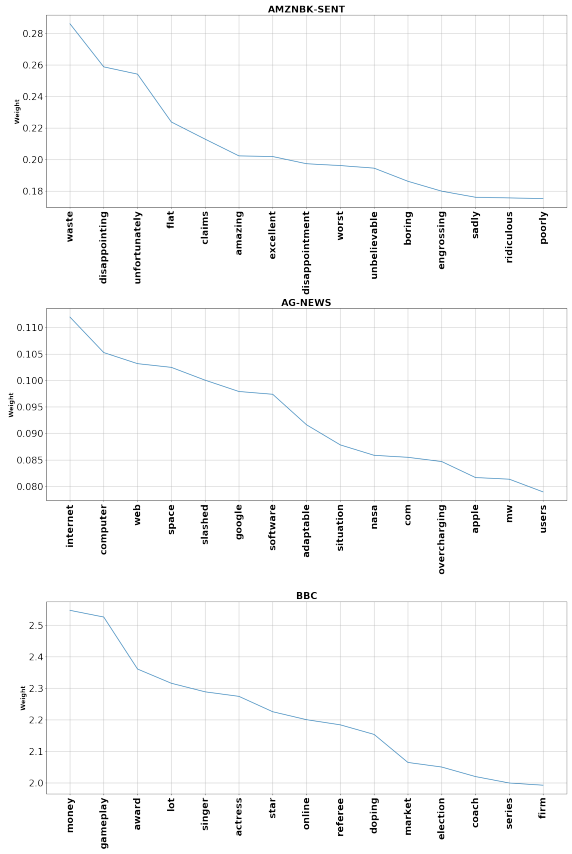

Figure 4: Weights of top 15 words identified by OptEm for three of the datasets used in our experiments. The words appear to be very informative; some can be easily associated to corresponding class.

AMZN-EX	YAHOO-ANS	DBPEDIA
book	period	born
sound	profile	author
product	mushrooms	singer
player	medicare	directed
use	daily	album
unit	youngest	artist
price	longest	writer
quality	anger	known
lens	aerobics	musician
radio	confirm	novelist

Table 3: 10 Words with highest absolute weights for AMZN-EX, YAHOO-ANS, AND DBPEDIA. Most of these words are clear indicators of their class.

lem at hand. This ranking of words by importance can be used for different NLP applications, such as extraction-based summarization, context-matching, and text cleaning. By learning the optimal weights of the words, our model also tends to remove or ignore less informative words, thus performing its own version of feature selection. Our text embedding algorithm combines the compactness and expressiveness of the word-embedding representations with the human-interpretability of a BoW-type model.

We intend to extend this work to make the proposed algorithm more scalable in order to incorporate larger, more complex classification models and tasks, such as multi-label and multi-class classification and summarization.

We want to explore using other normalizations and constraints to the weight vector. One possibility is to explore 1-norm regulation for the weight vector to make it more sparse and have a more aggressive feature (word) selection. Another interesting direction is toconsider group regularization similar to Fung's paper (2007), where the groups of words are suggested by a graph naturally defined by the distances between the words provided by the word embedding. In this way, semantically similar words would be weighted similarly and the result of the algorithm would be a clustering of terms by semantic meaning or topics that are relevant to the classification problem at hand.

References

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings .

James C. Bezdek and Richard J. Hathaway. 2003. Convergence of alternating optimization. *Neural, Parallel Sci. Comput.* 11(4):351–368. http://dl.acm.org/citation.cfm?id=964885.964886.

John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the Association for Computational Linguistics (ACL)*.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606* .

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Mach. Learn.* 20(3):273–297. https://doi.org/10.1023/A:1022627411411.

DBPedia. 2018. Dbpedia. http://wiki.dbpedia.org/.

Cedric De Boom, Steven Van Canneyt, Thomas Demeester, and Bart Dhoedt. 2016. Representation learning for very short texts

using weighted word embedding aggregation. *Pattern Recogn. Lett.* 80(C):150–156. https://doi.org/10.1016/j.patrec.2016.06.012.

Facebook. 2018. Fasttext. https://fasttext.cc/.

G. Fung and O. L. Mangasarian. 2001. Proximal support vector machine classifiers. In F. Provost and R. Srikant, editors, *Proceedings KDD-2001: Knowledge Discovery and Data Mining, August 26-29, 2001, San Francisco, CA*. Asscociation for Computing Machinery, New York, pages 77–86. Ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-02.ps.

Glenn Fung and Jonathan Stoeckel. 2007. Svm feature selection for classification of spect images of alzheimer's disease using spatial information. *Knowledge and Information Systems* 11(2):243–258. https://doi.org/10.1007/s10115-006-0043-5.

Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

Google. 2018. Word2vec. https://code.google.com/archive/p/word2vec/.

Derek Greene and Pádraig Cunningham. 2006. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proc. 23rd International Conference on Machine learning (ICML'06)*. ACM Press, pages 377–384.

Zellig Harris. 1954. Distributional structure. *Word* 10(23):146–162.

Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, WWW '16, pages 507–517. https://doi.org/10.1145/2872427.2883037.

Ken Lang. 1995. Newsweeder: Learning to filter netnews. In *Proceedings of the 12th International Machine Learning Conference*. pages 331–339.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. JMLR.org, ICML'14, pages II–1188–II–1196. http://dl.acm.org/citation.cfm?id=3044805.3045025.

Yuh-Jye Lee and O.L. Mangasarian. 2001. Ssvm: A smooth support vector machine for classification. *Computational Optimization and Applications* 20(1):5–22. https://doi.org/10.1023/A:1011215321374.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2015. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal* 6(2):167–195. http://jens-lehmann.org/files/2015/swj_dbpedia.pdf.

Omer Levy and Yoav Goldberg. 2014. Linguistic regularities in sparse and explicit word representations. In *CoNLL*.

M. Lichman. 2013. UCI machine learning repository. http://archive.ics.uci.edu/ml.

Chu-Cheng Lin, Waleed Ammar, Chris Dyer, and Lori S. Levin. 2015. Unsupervised POS induction with word embeddings. *CoRR* abs/1503.06760. http://arxiv.org/abs/1503.06760.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, pages 142–150. http://www.aclweb.org/anthology/P11-1015.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, USA, SIGIR '15, pages 43–52. https://doi.org/10.1145/2766462.2767755.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In *Proceedings of ICLR Workshop 2013*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. Curran Associates Inc., USA, NIPS'13, pages 3111–3119. http://dl.acm.org/citation.cfm?id=2999792.2999959.

Arjun Mukherjee and Bing Liu. 2010. Improving gender classification of blog authors. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Stroudsburg, PA, USA, EMNLP '10, pages 207–217. http://dl.acm.org/citation.cfm?id=1870658.1870679.

Arnold Neumaier. 1998. Solving ill-conditioned and singular linear systems: A tutorial on regularization. *SIAM Review* 40(3):636–666. https://doi.org/10.1137/S0036144597321909.

Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. 2016. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 24(4):694–707. http://dl.acm.org/citation.cfm?id=2992449.2992457.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. pages 1532–1543. http://www.aclweb.org/anthology/D14-1162.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, pages 45–50. `http://is.muni.cz/publication/884893/en`.

scikit-learn. 2017. Support vector machines. http://scikit-learn.org/stable/modules/svm.html.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Parsing with compositional vector grammars. In *EMNLP*.

J. A. K. Suykens and J. Vandewalle. 1999. Least squares support vector machine classifiers. *Neural Process. Lett.* 9(3):293–300. https://doi.org/10.1023/A:1018628609742.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2015. Towards universal paraphrastic sentence embeddings. *CoRR* abs/1511.08198.

Yelp. 2018. Yelp dataset challenge. https://www.yelp.com/dataset/challenge.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. MIT Press, Cambridge, MA, USA, NIPS'15, pages 649–657. http://dl.acm.org/citation.cfm?id=2969239.2969312.