



저작자표시 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.
- 이차적 저작물을 작성할 수 있습니다.
- 이 저작물을 영리 목적으로 이용할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#) 

공학박사학위논문

Supervised Feature Representations
for Document Classification

문서 분류를 위한 지도학습 기반의 특징 표현

2016 년 8 월

서울대학교 대학원
산업조선공학부 데이터마이닝 전공

박 은 정

Supervised Feature Representations for Document Classification

문서 분류를 위한 지도학습 기반의 특징 표현

지도교수 조 성 준

이 논문을 공학박사학위논문으로 제출함

2016 년 7 월

서울대학교 대학원

산업조선공학부

박 은 정

박은정의 박사학위논문을 인준함

2016 년 6 월

위 원 장	<u>박 종 현</u>	(인)
부위원장	<u>조 성 준</u>	(인)
위 원	<u>이 경 식</u>	(인)
위 원	<u>강 필 성</u>	(인)
위 원	<u>최 성 철</u>	(인)

Abstract

Supervised Feature Representations for Document Classification

Eunjeong park

Department of Industrial Engineering

The Graduate School

Seoul National University

While the traditional method for deriving representations for documents was bag-of-words, they suffered from high-dimensionality and sparsity. Recently, many methods to obtain lower-dimensional and dense distributed representations were proposed. Paragraph vectors is one of such algorithms, which extends the word2vec algorithm by assuming the paragraph as an additional word. However, it generates a single representation for all tasks, while different tasks may require different kinds of representations. In this work we propose supervised paragraph vectors, a task-specific variant of paragraph vectors for situations where class labels exist. Essentially, supervised paragraph vectors jointly trains class labels with words and documents so that representations for each class label, words, and documents are obtained with respect to the particular classification task. In order to prove the benefits of the proposed algorithm, three

performance criteria are used: interpretability, discriminative power, and computational efficiency. For interpretability, we find words that are close and far to class vectors, and demonstrate that such words are closely related to the corresponding class. We also use principal component analysis to visualize all words, documents and class labels in a joint space, and show that our method effectively displays the related words and documents for each class label. For discriminative power and computational efficiency, we perform document classification on four commonly used datasets with various classifiers, and achieve comparable classification accuracies to bag-of-words and paragraph vectors. This method is further extended to a semi-supervised version. Finally, a scored-based lexicon is extracted using supervised paragraph vectors, and are applied to short document classification tasks.

Keywords: data mining, text mining, document classification, distributional representations, representational learning

Student Number: 2011-30314

Contents

Abstract	i
Contents	v
List of Tables	viii
List of Figures	xi
Chapter 1 Introduction	1
1.1 Criteria of “good representations”	3
1.2 Distributed representations	5
1.3 Issues with general distributed representations	7
1.4 Overview	8
1.5 Notations	11
Chapter 2 Literature review	12
2.1 Distributed representations for words	13
2.1.1 Count-based methods	13
2.1.2 Prediction-based methods	15
2.2 Distributed representations for documents	24

2.2.1	Unordered methods	25
2.2.2	Ordered methods	29
2.2.3	Others	29
2.3	Document classification	31
2.3.1	Sentiment analysis	31
2.3.2	Others	32
2.4	Lexicon extraction	32
Chapter 3 Supervised representations of words and documents		34
3.1	Background	34
3.2	Proposed method	34
3.2.1	Representation learning	34
3.2.2	Inference and classification	37
3.3	Experiments	38
3.3.1	Datasets	38
3.3.2	Implementation	41
3.3.3	Evaluation	42
3.4	Summary and discussions	57
Chapter 4 Semi-supervised representations of words and documents		61
4.1	Background	61
4.2	Proposed method	62

4.3	Experiments	62
4.3.1	Datasets	64
4.3.2	Implementation	64
4.3.3	Evaluation	65
4.4	Summary and discussions	66
Chapter 5 Scored lexicon extraction for classification of short documents		68
5.1	Background	68
5.2	Proposed method	69
5.3	Experiments	71
5.3.1	Datasets	71
5.3.2	Implementation	72
5.3.3	Evaluation	73
5.4	Summary and discussions	73
Chapter 6 Conclusion		75
6.1	Summary and contributions	75
6.2	Future work	76
Bibliography		79
국문초록		89

List of Tables

Table 1.1	Methods covered in this dissertation	10
Table 3.1	Summary table for the datasets.	41
Table 3.2	Summary table for the hyperparameters for the comparison of PV and SPV.	42
Table 3.3	Ten closest and farthest words for each class vector in the IMDB dataset.	50
Table 3.4	Five closest and farthest words for each class vector in the yelp dataset.	51
Table 3.5	Test accuracies for each representation algorithm, per number of epochs.	55

Table 3.6	Test accuracies for each representation algorithm, per classifier. Document representations derived with each representation algorithm is classified with five different classifiers: decision tree (dt), logistic regression (lr), random forest (rf), neural network (nn) and naive Bayes (nb). Numbers in bold are the highest accuracies for each classifier (column). Numbers in parentheses indicate the number of epochs of the corresponding result.	56
Table 4.1	Summary table for the hyperparameters for the comparison of PV, SPV and SSPV.	65
Table 4.2	Held-out accuracies for each dataset. Numbers in bold are the best (highest) accuracies for each dataset (column). .	66
Table 5.1	Closest 20 word vectors from class vectors in the imdb dataset.	69
Table 5.2	Example of classifying a document with the scored lexicon.	71
Table 5.3	Lexicon experimental results for the imdb dataset	74
Table 6.1	Summary of algorithms. Note that bag-of-words(BOW) was removed from the comparison table due to lack of generalizations for unseen words.	77

List of Figures

Figure 1.1	Some examples of document classification.	2
Figure 1.2	The two stages of document classification.	3
Figure 1.3	A brief example of bag-of-words.	5
Figure 1.4	The distributional hypothesis	6
Figure 1.5	A brief example of distributed representations.	7
Figure 1.6	A 2-dimensional diagram to illustrate the difference between unsupervised representations and supervised representations.	9
Figure 2.1	Taxonomy for word representations.	13
Figure 2.2	Two word2vec architectures: (a) CBOW and (b) skip-gram [28].	19
Figure 2.3	Words mapped with word2vec and t-SNE using Korean restaurant related blog posts.	21
Figure 2.4	Words mapped with word2vec and t-SNE using North Korea's New Year messages.	22
Figure 2.5	Online webpages mapped with word2vec and t-SNE using user navigation logs.	23

Figure 2.6	Taxonomy for document representations.	24
Figure 2.7	Two paragraph vector architectures: (a) distributed memory (PV-DM) and (b) distributed bag-of-words (PV-DBOW).	27
Figure 3.1	Two supervised paragraph vector architectures: (a) distributed memory (SPV-DM) and (b) distributed bag-of-words (SPV-DBOW).	36
Figure 3.2	Joint visualization of words, documents, and classes for the IMDB dataset.	45
Figure 3.3	Joint visualization of words, documents, and classes for the Amazon dataset.	47
Figure 3.4	Joint visualization of words, documents, and classes for the Yelp dataset.	48
Figure 3.5	Joint visualization of words, documents, and classes for the 20news dataset.	49
Figure 3.6	The most negative document in the imdb dataset, according to the affinity towards the class vectors.	53
Figure 3.7	The most positive document in the imdb dataset, according to the affinity towards the class vectors.	53
Figure 3.8	The relation between number of documents $n(docs)$ and classification accuracy.	57

Figure 3.9	The relation between number of documents $n(docs)$ and training time.	58
Figure 3.10	The relation between number of vector dimensions $n(dims)$ and classification accuracy.	59
Figure 3.11	The relation between number of vector dimensions $n(dims)$ and training time.	60
Figure 4.1	Comparison diagrams of PV, SPV and SSPV.	63
Figure 4.2	Semi-supervised paragraph vectors.	64
Figure 5.1	Tweet lengths from a 1M sized sample ¹	68

Chapter 1

Introduction

Natural languages convey information from an individual to another. Languages are so powerful, that some scholars including Whorf and Wittgenstein have suggested **linguistic determinism**, a theory saying that language determines the range of possible cognitive processes of individuals [1]. But functionally, language is more of a medium of conversation, which allows humans to exchange their thoughts, assertions and emotions. For such needs, language normally takes the form of speech or writings, though it is also expressed in other forms, such as Morse codes, braille, or sign languages.

Both speech and writings have popular, eligible data formats. Speech, for instance, can be either manually transcribed, or recorded and transformed to text with speech recognition. Writings on the other hand, are inherently text. Either way, for any human idea to be processed by a machine, it is at some point, transformed to a written form, **text**. Hence, the ability to handle text implies the encapturing abundant communications among humans, and suggesting the possibility of myriad analyses.

Of the many possible analyses of text, **document classification**, the task

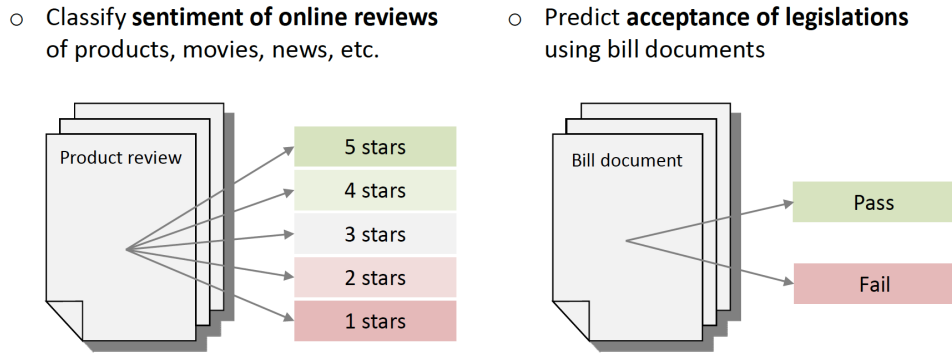


Figure 1.1: Some examples of document classification.

of classifying given documents into predefined classes as seen in [Figure 1.1](#), is one that has a big impact on many businesses. Some class labels regarded in document classification are as follows: sentiment of movie or product reviews [[2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [8](#)], the pass or fail of bills [[9](#)], the stock price change of 8K [[10](#)], or the subjectivity [[5](#), [11](#)], opinion polarity [[5](#)], or topics [[8](#), [6](#)] of any given document.

As with other classification tasks, document classification not only has the problem of training the particular classification algorithm, but also has the problem of deciding the representation of the documents, as shown in [Figure 1.2](#). In other words, document classification normally takes a two stage approach: the first step is to Find an appropriate representation for a given document, and the second step is to Train a classifier to classify the document to the most adequate class with the given representation.

There is a famous saying frequently used in data science, “Garbage in,

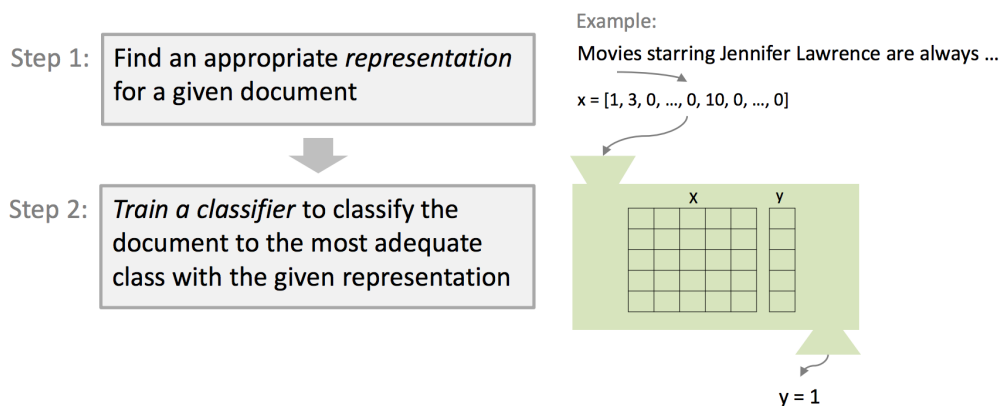


Figure 1.2: The two stages of document classification.

garbage out”. In a classification perspective, this can be interpreted that the performance depends on the quality of the representation. Hence, it is important to find “good representations” for given documents.

1.1 Criteria of “good representations”

A major difficulty of finding good representations is particularly hard for documents, because they are not structured to begin with, thus, there is no obvious way to encode and convey documents to computers.

However, we can still imagine and define what a “good representation” of a document should be. In this dissertation, we define a good representation in three aspects, among which were first suggested in [12].

Interpretability. A good representation should be interpretable, rather than latent. It is important that we can achieve insights from the document repre-

sentations themselves. This sort of interpretability is a qualitative measure, and is demonstrated by visualization and examples.

Discriminative power. A good representation should be able to discriminate a document from other distinguishable documents. Given some distance metric, documents with similar contents should have shorter distances whereas documents with different contents should have longer distances. We call this discriminative power. Classification accuracy can be used as a proxy measure.

Efficient computation. A good representation should be scalable, and computable in a reasonable amount of time. This can be measured by training time.

Other than the three predefined criteria stated above, **reconstruction quality**, which states whether a representation can be reversed to its original form, could also be a good criterion to evaluate representations according to [12]. However, because the algorithms compared in this paper basically use the “bag-of-words” assumption, reconstruction would be considered difficult.

Generalization ability can also be another criterion for good representations. We can consider that bag of words is not a good generalizer because it is in the discrete space – because in a discrete space, there is no general concept of distance or neighborhood, while in a continuous space, one can average over data points near the point of interest.

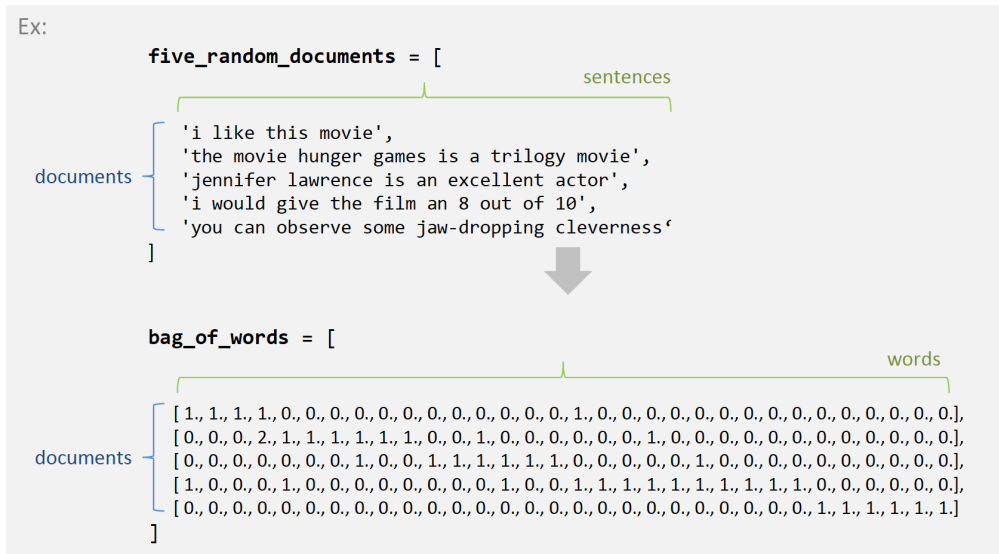


Figure 1.3: A brief example of bag-of-words.

1.2 Distributed representations

The traditional method for deriving representations for documents was a localized representation, **bag-of-words** (BOW). Bag-of-words considers words as atomic symbols, represented in the discrete space, as show in [Figure 1.3](#).

However, this scheme suffers from several issues. The first issue is that the representation is high-dimensional, therefore suffers from the curse of dimensionality. For example, [Figure 1.3](#) has only five very short documents, yet the number of dimensions ranges up to 32. The second issue is sparsity, therefore documents are near-orthogonal. For example, the similarity between two documents “d1” and “d2” are prone to be 0.0. Anohter issue is that it is not possible to calculate the similarity between words. For example, the inner product of

Ex: When the window size is 1;

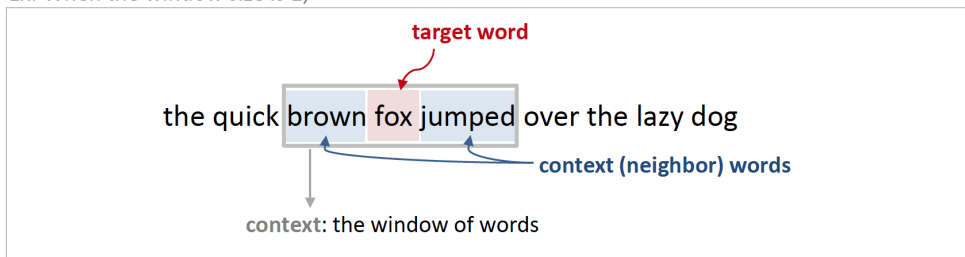


Figure 1.4: The distributional hypothesis

“wonderful” and “great”, and the inner product of “wonderful” and “awful” are both likely to be 0.0. However, in the sense of semantics, “wonderful” and “great” are closer to each other than “wonderful” and “awful”.

Alternatively, methods to find denser representations for documents were proposed. Many of these are based on the distributional hypothesis [13], which states that if two words often co-occur, they are similar in meaning.

“Words that appear in the same context share semantic meaning.”

– Harris, *Distributional Structure* (1954)

In contrast to the bag-of-words approach, distributed representations characterize an object by its context, and therefore discrete objects such as words and documents are represented in the continuous space as seen in

What’s so great about distributed representations for documents? First, they result in denser and usually shorter vectors, which allow less data for training. They can also calculate similarities between object pairs, and show better generalization for unseen objects. While no general concept of distance

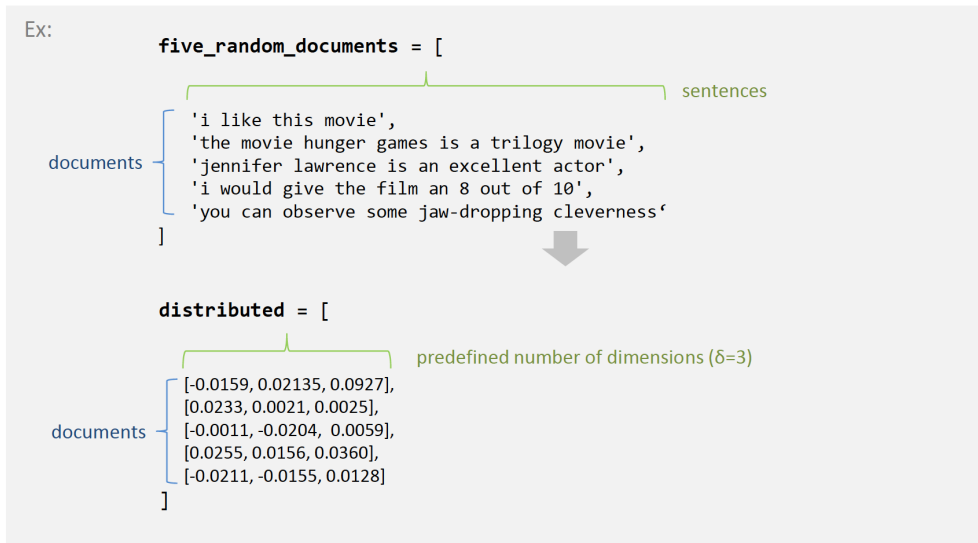


Figure 1.5: A brief example of distributed representations.

or neighborhood can be defined in discrete space, in a continuous space, one can average over data points near the point of interest.

1.3 Issues with general distributed representations

However, distributed representations are not silver bullets. In this dissertation, we consider the issues of distributed representations as three-fold.

While most representations generate a single representation regardless of the task, whether they are localized or distributed, different tasks may require different kinds of representations. As in [Figure 1.6 \(a\)](#), word2vec normally generates word vectors so that semantically and syntactically similar words are located adjacently in the embedding space. Yet, if the task is sentiment classi-

fication, it would be preferred to separate words that have opposite sentiment polarities such as “amazing” and “awful”, desirably into different locations of the space as in [Figure 1.6](#) (b).

In order to derive distributed representations, one also needs sufficient training data. Therefore, such representations may not be suitable for low-frequency words and very short documents.

Another shortcoming is that distributed representations are normally a fixed-length representation, regardless of the document length. For example, Whether there are 200 words, or 20000 words in a document, it is represented in 100 dimensions.

1.4 Overview

This dissertation coherently describes methods that find “good representations” in order to overcome the previously stated shortcomings of distributed representations, and employ them for document classification. We demonstrate that augmenting label information to the training phase of finding distributed representations, not only enhances document classification performance, but also provides the potential to explore the following. First, I present a novel method to obtain the representations of word, document, and class labels, by jointly training them in an extended word2vec architecture. Second, this method is extended in the manner of semi-supervised learning. Third, the embedding space derived from this method is fully exploited, by calculating similarities between

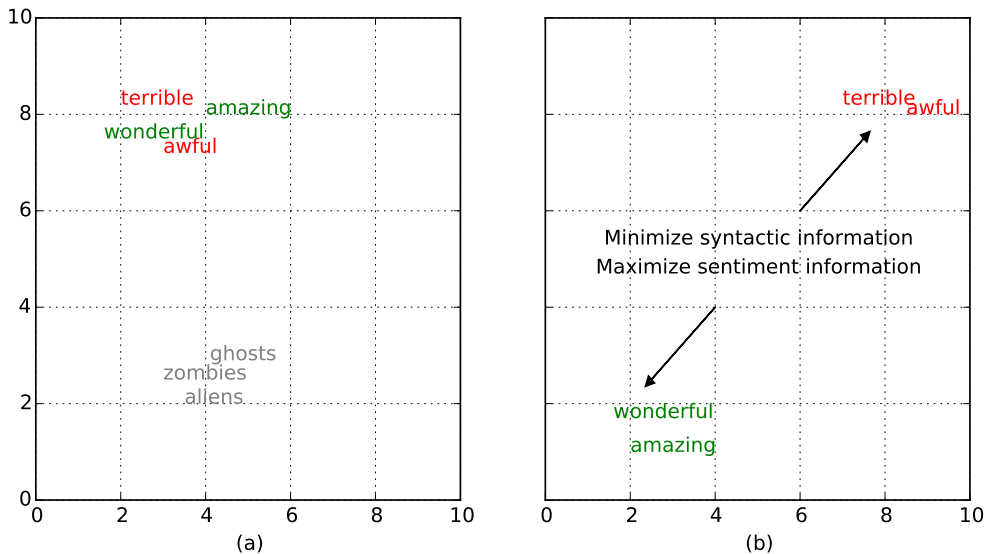


Figure 1.6: A 2-dimensional diagram to illustrate the difference between unsupervised representations and supervised representations. While most word vectors would have tens to thousands of dimensions, this diagram was plotted in 2-dimensions for demonstration. (a) Words with opposite sentiment polarity could be located nearby since unsupervised word vectors considers context words only. (b) When not only context words, but also the class label information is concerned, we can expect to separate words with opposite sentiment polarity to different parts of the semantic space.

Chapter	Target documents	Method
3	Any length	Supervised paragraph vectors
4	Any length	Semi-supervised paragraph vectors
5	Short-length	Scored lexicon extraction

Table 1.1: Methods covered in this dissertation

the representations of words, documents, and class labels. Fourth, this method is applied to the areas of popular applications such as movie or product reviews.

The main chapters, Chapters 3–5, each include the following. In Chapter 3, I present supervised paragraph vectors (SPV), a method fit for medium length documents to obtain the representations of word, document, and class labels, by jointly training them in an extended word2vec architecture. I further extend this to a semi-supervised version in Chapter 4. However, SPV may not be suitable for extremely short documents. So finally in Chapter 6, we use the class and word vectors derived from SPV to build a sentiment lexicon, because lexicon methods are known to work better for short documents.

1.5 Notations

symbol	description
γ	number of context words
δ	dimension of vectors
$\mathcal{V} = \{w_1, \dots, w_V\}$	vocabulary
$\mathcal{D} = \{d_1, \dots, d_N\}$	documents in training data
$\mathcal{C} = \{c_1, \dots, c_M\}$	unique class labels in training data
V	number of words in vocabulary
N	number of documents in training data
M	number of unique class labels in training data
$\mathbf{x}_i \in \mathbb{R}^{V \times 1}$	i th one-hot vector for words in input layer
$\mathbf{p} \in \mathbb{R}^{N \times 1}$	one-hot vector for documents in input layer
$\mathbf{z} \in \mathbb{R}^{M \times 1}$	one-hot vector for classes in input layer
$\mathbf{h} \in \mathbb{R}^{\delta \times 1}$	hidden layer
$\mathbf{y}_j \in \mathbb{R}^{V \times 1}$	j th one-hot vector in output layer
\mathbf{X}	co-occurrence matrix
$\mathbf{W}^{(1)} \in \mathbb{R}^{V \times \delta}$	input word embedding matrix
$\mathbf{W}^{(2)} \in \mathbb{R}^{\delta \times V}$	output word embedding matrix
$\mathbf{D} \in \mathbb{R}^{N \times \delta}$	document embedding matrix
$\mathbf{C} \in \mathbb{R}^{M \times \delta}$	class label mbedding matrix
$\mathbf{v}(w) \in \mathbb{R}^{\delta \times 1}$	vector representation of word w

Chapter 2

Literature review

Like other classification tasks, document classification involves finding the appropriate representation of the documents before training a classifier. While the traditional method for deriving such representations was **bag-of-words**, they suffered from high-dimensionality and sparsity. For this reason, such representations were not suitable to calculate similarities – by which cosine similarities are considered in many cases – between document pairs due to their near-orthogonality [14, 15], and were difficult to generalize to unobserved words. Some alleviations were suggested to overcome this difficulty, for example to consider word similarities when calculating the distance between two vectors in the vector space [16].

Alternatively, methods to find denser representations for documents were proposed. Many of these are based on the distributional hypothesis [13], which states that if two words often co-occur, they are similar in meaning.

In this chapter, we explore some distributional representation methods for documents, as well as words, and observe their effectiveness in document classification problems.

2.1 Distributed representations for words

Because they are atomic symbols of the human language, finding representation of words, and some notion of similarity between them helps to perform better not only on document classification tasks, but also on other general text mining and natural language processing applications. While the localized representation is simply to represent a word as a one-hot, or 1-of- V encoded vector, there are other well known methods that find distributed representations of words as shown in [Figure 2.1](#): count-based methods, and prediction-based methods.

2.1.1 Count-based methods

Count-based methods map count-statistics of neighbors to a smaller, denser vector. The most common approach is to first obtain global co-occurrence count statistics over a massive training dataset, then apply dimensionality reduction techniques such as latent semantic analysis.

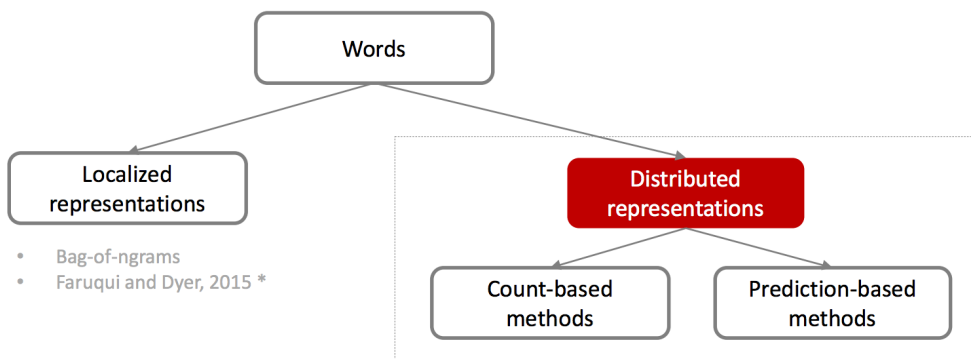


Figure 2.1: Taxonomy for word representations.

Latent semantic analysis (LSA). Latent semantic analysis [17], also known as **latent semantic indexing (LSI)** in the context of information retrieval, first accumulates a term-document matrix $\mathbf{X} \in \mathbb{R}^{V \times N}$ by looping over a set of documents, where V is the size of the vocabulary and N is the number of documents. Then it performs **singular value decomposition (SVD)** on \mathbf{X} to get a $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ decomposition:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \begin{matrix} & N & & & & \\ & & & N & & \\ & & & & & V \\ V & \begin{bmatrix} | & | & | \\ u_1 & u_2 & \cdots \\ | & | & | \end{bmatrix} & N & \begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} & N & \begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{bmatrix} \end{matrix} \quad (2.1)$$

where $\mathbf{U} \in \mathbb{R}^{V \times N}$, $\mathbf{V} \in \mathbb{R}^{V \times N}$ are orthogonal matrices and $\mathbf{\Sigma} \in \mathbb{R}^{N \times N}$ is a diagonal matrix containing the singular values. We can then select the best k rank approximation of \mathbf{X} :

$$\mathbf{X} \approx \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top = \begin{matrix} & k & & & & \\ & & & k & & \\ & & & & & V \\ V & \begin{bmatrix} | & | & | \\ u_1 & u_2 & \cdots \\ | & | & | \end{bmatrix} & k & \begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} & k & \begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{bmatrix} \end{matrix} \quad (2.2)$$

where $\mathbf{U}_k \in \mathbb{R}^{V \times k}$, $\mathbf{V}_k \in \mathbb{R}^{V \times k}$ and $\mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}$.

and the rows of U can then be considered as a k -dimensional vector for each word. Hence, words and documents are represented as points in the Euclidean space. It has also been reported that removing the first 50-dimensions before selecting the k -dimensions *helps* [18].

However, note that the size of the matrix X depends on the number of documents; billions of documents will result in an enormous matrix. Using a term-term matrix instead will alleviate this problem.

Limitations of count-based methods. Count-based methods are intuitive and easy to calculate, but there are some limitations. First, it is difficult to include unobserved words to the existing matrix X . Second, the matrix is extremely high-dimensional due to the large vocabulary, and sparse because not many words actually co-occur. Third, we must introduce some **smoothing** method to handle the extreme values of word frequencies. Finally, since SVD has quadratic cost, training becomes intractable as the matrix co-occurrence matrix X becomes larger.

2.1.2 Prediction-based methods

Instead of computing an enormous co-occurrence matrix, **prediction-based methods** predict a word from its neighboring words with each example document, normally using a neural network. Essentially, these methods combine vector space semantics with the prediction of probabilistic models, and has the advantage that no supervised information is necessary.

Neural network language model (NNLM) [19, 20], log-bilinear models [21, 22], Turian et al. [23], recurrent neural network language model [24], Collobert and Weston [25], recursive neural network language model [26], and word2vec [27] are some popular examples of prediction-based methods to find distributed representations for words.

Neural network language model (NNLM). The neural network language model (NNLM) [19, 20] is a pioneering work of unsupervised neural network based embedding for learning distributed representations of words.

The objective of NNLM is to perform **language modeling**, which computes the probability of a particular word sequence. For a sequence of T words $[w_1, w_2, \dots, w_T]$, the probability of its occurrence, according to the chain rule of probability is as follows:

$$p(w_1, w_2, \dots, w_T) = \prod_{i=1}^T p(w_i | w_1, \dots, w_{i-1}) \quad (2.3)$$

This can be further approximated by the Markov assumption. For example, using the γ th-order Markov assumption, Equation 2.3 reduces to:

$$p(w_1, w_2, \dots, w_T) \approx \prod_{i=1}^T p(w_i | w_{i-\gamma+1}) \quad (2.4)$$

In NNLM, the probability $p(w_i | w_{i-\gamma+1})$ is obtained as follows. First, each word w_i is mapped to a δ -dimensional feature vector $v(w_i) \in \mathbb{R}^{\delta \times 1}$, which is

row i of the input embedding matrix $\mathbf{W}^{(1)} \in \mathbb{R}^{\delta(\gamma-1) \times h}$, where h is the number of hidden units. Then, if we let x denote the concatenation of the vectors of $\gamma - 1$ previous words:

$$x = [v(w_{i-\gamma+1}); \dots; v(w_{i-2}); v(w_{i-1})] \quad (2.5)$$

the next word is predicted using a standard neural network architecture with x as input, and the softmax activation function for the output units.

$$p(w_i | w_{i-\gamma+1}) = \frac{\exp(a_i)}{\sum_{k \in V} \exp(a_k)} \quad (2.6)$$

where $a_k = b_k^{(2)} + \sum_{i=1}^h \mathbf{W}_{ki}^{(2)} \tanh(b_i^{(1)} + \sum_{j=1}^{(n-1)d} \mathbf{W}_{ij}^{(1)} x_j)$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times V}$ is the output embedding matrix.

Then, the log-likelihood of the above probability is maximized:

$$L(\theta) = \sum_i \log p(w_i | w_{i-\gamma+1}) \quad (2.7)$$

using gradient descent and backpropagation.

However, due to the intractability of the softmax function with very large vocabularies, it took days and days to train NNLM, which called for the needs of a faster algorithms.

Recurrent neural network language model (RNNLM). In contrast to other modern language models where the model employs the Markov assump-

tion, the **recurrent neural network language model (RNNLM)** [24] conditions on all previous words, with the time complexity of $O(T)$, where T is the length of the given document.

Recursive neural network language model (RecNNLM). The recursive neural network language model [26] represents the semantic of a sentence via a tree structure.

However, constructing such a tree has a time complexity of $O(T^2)$ for a sentence with length T .

word2vec. word2vec [27] is a distributed word representation algorithm that proposed a simple neural network with one hidden layer, which showed we can obtain word vectors that demonstrate linguistic regularities, such as $v(\text{"Berlin"}) - v(\text{"Germany"}) + v(\text{"France"}) = v(\text{"Paris"})$. As presented in Figure 2.2, there are two different architectures in word2vec: continuous bag-of-words (CBOW) and skip-gram. $x \in \mathbb{R}^{V \times 1}$, $h \in \mathbb{R}^{d \times 1}$, $y \in \mathbb{R}^{V \times 1}$ are input, hidden, output vectors, respectively, where V is the size of the vocabulary, d is the predefined size of word vectors, and C is the number of context words. $\mathbf{W}^{(1)} \in \mathbb{R}^{V \times d}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times V}$ are each input and output word vector matrices. When C is a predefined window size, CBOW accepts C contiguous context words as input and predicts a target word, while skip-gram uses the target word as input, and predicts C surrounding context words.

Each layer in both word2vec architectures have the following characteristics.

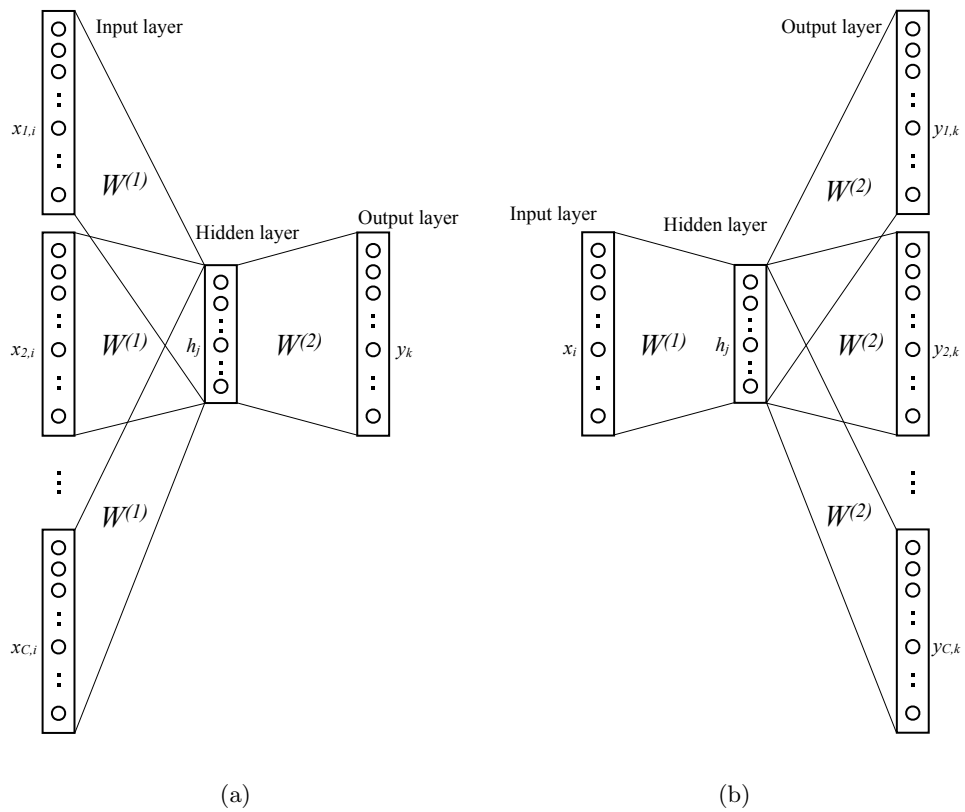


Figure 2.2: Two word2vec architectures: (a) CBOW and (b) skip-gram [28].

First, the input layer consists of 1-of- V encoded vectors, where the i th row represents the presence of the i th word in the vocabulary. Then, each unit in the hidden layer h_j is simply the weighted sum of its inputs with a linear activation function, where the weights are $\mathbf{W}^{(1)}$. In the output layer, weights $\mathbf{W}^{(2)}$ are multiplied to the hidden layer, and target words are predicted with a softmax function. Then if we encode each target word as a 1-of- V vector, the difference of the predicted and actual occurring vectors, or errors can be derived and used for backpropagation. Finally through backpropagation, the input weight $\mathbf{W}^{(1)}$ and output weight $\mathbf{W}^{(2)}$ are updated, and word vectors can be obtained from selecting either one of the matrices, or composing them by sum or concatenation.

word2vec is powerful in the sense it is not only language-independent, as can be seen in [Figure 2.3](#), or in [Figure 2.4](#) [16]. It can also be applied to model online webpages using user navigation logs as seen in [Figure 2.5](#).

Many variants have been introduced for word2vec. For example, context words can be selected based on dependency parsing [29], or attention [30]. Hierarchical word vectors that were optimized to show hierarchical relations among dimensions of a vector [31], Multi-sense word vectors which used clustering for multiple sense-specific vectors [32], enriched word vectors using document level contextual information [33], induced multiple sense specific vector representations [34]. Probabilistic word embeddings, that map words into a density rather than a point were also proposed [35]. They are known to capture uncertainty

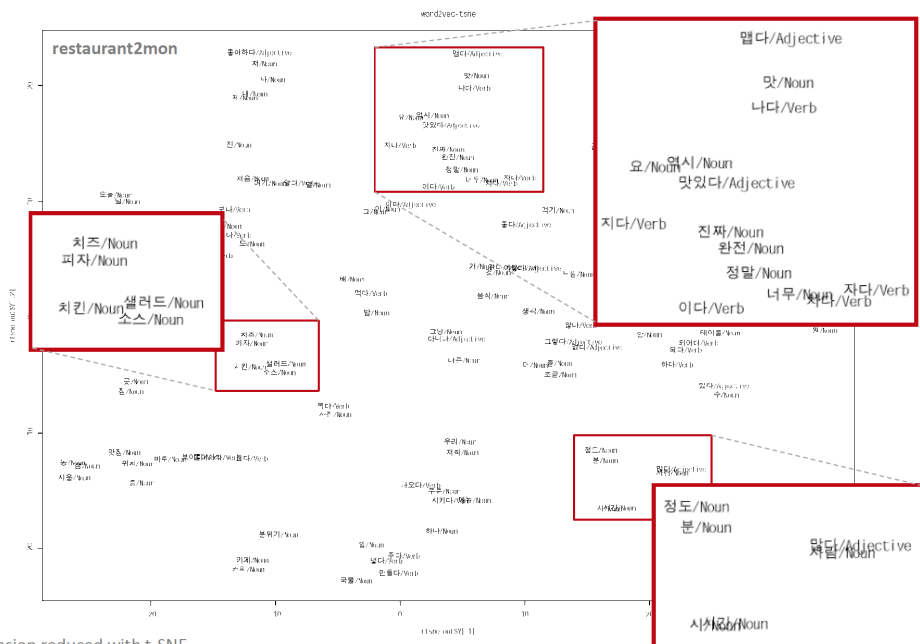


Figure 2.3: Words mapped with word2vec and t-SNE using Korean restaurant related blog posts.



Figure 2.4: Words mapped with word2vec and t-SNE using North Korea's New Year messages.

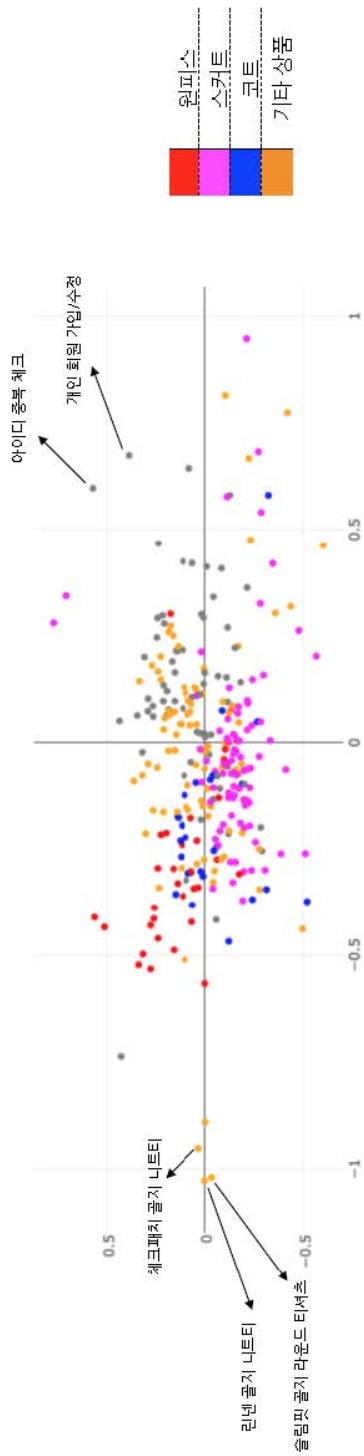


Figure 2.5: Online webpages mapped with word2vec and t-SNE using user navigation logs. Modeled with 500K user logs by considering each browsers' "pagestreams" as sentences. Used skip-gram of 100 dimensions, with a window of 5 webpages and minimum count of 2.

about a representation and its relationships and expresses asymmetries more naturally than dot product or cosine similarity. Finally, Word embeddings enhanced with ensembles were proposed [36].

However, there has been a study asserting that there is no qualitative difference between predictive neural network models and count-based distributional semantics models [37]. It was also found that much of the performance gains of word embeddings are due to certain system design choices and hyperparameter optimizations, rather than the embedding algorithms themselves [38].

2.2 Distributed representations for documents

For several decades, the common approach of document representation was a bag-of-words (BOW) [13], where the features are words, and total number of features equals the size of the vocabulary. In this case, the feature values can be filled with either the term presence, term frequency, or weighted term frequen-

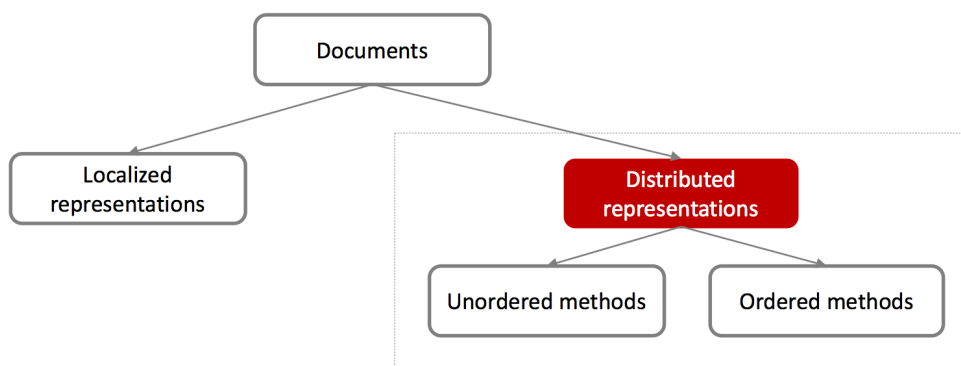


Figure 2.6: Taxonomy for document representations.

cies such as **TF-IDF**. This approach is well known for its intuitiveness, and is known to perform well in classification tasks [39]. However, because the unique number of terms used in the human language is known to range up to 13M tokens [40] and most documents certainly do not employ most of the terms, document vectors created using this approach are normally very high-dimensional and sparse. It is also difficult to calculate similarities between document pairs with such representations because they are approximately orthogonal [14, 15].

Localized representations are not suitable to calculate similarities between document pairs due to their near-orthogonality [41, 14, 15].

Methods to find distributed representations for larger bodies of text, such as documents were also proposed.

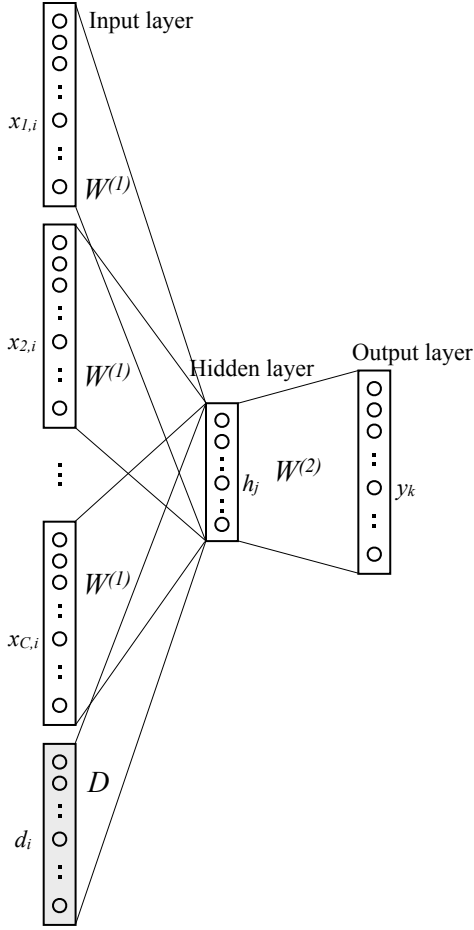
2.2.1 Unordered methods

Unordered composition. Some approaches choose to combine word embeddings to document representations with **compositionality**. Compositionality uses simpler elements to construct more complex meaning via combination. Some examples of compositionality from word vectors to document vectors are as follows: summation and multiplication of vectors [42, 43], averaging vectors, weighted averaging vectors (“importance factor approach”) [44], maximum pooling [25, 45, 4], clustering vectors (“bag-of-means”) [6, 46] and tensor-based composition [47, 48].

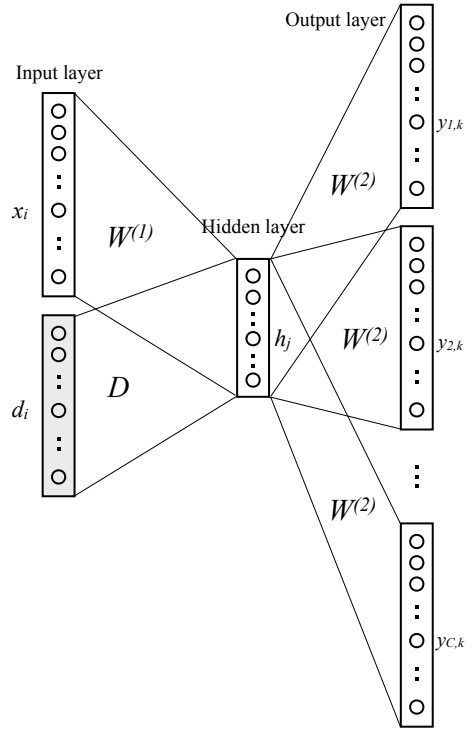
Paragraph vectors [49]. Particularly, paragraph vectors (PV) [49] is an architecture that extends word2vec by adding a 1-of- V encoded vector of documents d to the input layer, and hence enables the joint training of word and document vectors. Figure 2.7 presents the two architectures of PV: distributed memory (PV-DM) and distributed bag-of-words (PV-DBOW). Documents are represented as $d \in \mathbb{R}^{N \times 1}$, 1-of- V encoded vectors in the input layer, where N is the number of documents, and corresponding document vectors are obtained as rows in the document embedding matrix $D \in \mathbb{R}^{N \times d}$. As with word2vec, $x \in \mathbb{R}^{V \times 1}$, $h \in \mathbb{R}^{d \times 1}$, $y \in \mathbb{R}^{V \times 1}$ are input, hidden, output vectors, V is the size of the vocabulary, d is the size of word vectors, and C is the number of context words. $\mathbf{W}^{(1)} \in \mathbb{R}^{V \times d}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times V}$ are input and output word vector matrices. Interestingly, when word and document vectors are jointly trained, it was shown that meaningful relations between word and document vectors could be found with vector computations [50].

However, word2vec and paragraph vectors both generate a single representation regardless of the task, while different tasks may require different kinds of representations. Word2vec normally generates word vectors so that semantically and syntactically similar words are located adjacently in the embedding space. Yet, if the task is sentiment classification, it would be preferred to separate words that have opposite sentiment polarities such as “amazing” and “awful”, desirably into different locations of the space.

While it is impossible, or at least extremely difficult to adapt word vectors



(a) caption



(b) caption

Figure 2.7: Two paragraph vector architectures: (a) distributed memory (PV-DM) and (b) distributed bag-of-words (PV-DBOW).

to a particular task with only context words, we can expect to obtain task-adaptive word vectors by employing class labels in the training phase. For this, we can assume that the more frequently a word occurs with a particular class label, the more it is related to that class. A way to incorporate this belief is by including a class term to word2vec, as paragraph vectors did by adding a document term.

Sachan and Kumar [51] did just that, and suggested an architecture for embedding word vectors alongside with class vectors, by replacing the document vector with a class vector in the DBOW architecture of paragraph vectors. After deriving word and class vectors, they used three scoring models in order to decide the predicted class of a test document. The first model was to find the class with the maximum probabilistic score $\hat{c} = \arg \max_{i=1,\dots,M} \sum_{j=1}^V \log s(w_j|c_i)$, where M is the number of unique class labels, and s is the softmax function. The second and third model each computed the difference of probabilistic scores given class labels for each word $f(w) = \log s(w|c_{pos}) - \log s(w|c_{neg})$ and the difference of similarities with class labels for each word $g(w) = v(c_{pos})^\top v(w) - v(c_{neg})^\top v(w)$, used the calculated values as features for the word in a bag-of-words representation, and trained a logistic regression classifier.

However, there are some limitations to their approach. First, they employed linear combinations of word probability scores for classification instead of directly embedding documents, while representing the document itself can be insightful. Second, the scoring models are limited to binary classification except

for their first classification model.

2.2.2 Ordered methods

Ordered composition. Some hierarchical approaches, such as the **adpative hierarchical sentence model (AdaSent)** [11], consider the order of elements. In this approach, adjacent words are composed to phrases and phrases are composed to sentences through a recursive gated network. The intermediate hidden layers allow representations in multiple levels, such as words, phrases and sentences.

Convolutional neural networks (CNN). CharSCNN [4] is a character to sentence CNN.

Recurrent neural networks (RNN). RNNs can also be used to find representations of documents. However, because RNN models tend to be biased towards the end of a document [52], bi-directional variants, or unbiased methods such as CNNs can be employed.

RNNLM has been applied to opinion mining [53].

2.2.3 Others

Semi-supervised methods

The traditional methods for conducting semi-supervised learning can be explained in two perspectives: generative models and discriminative models. First,

for the semi-supervised learning of generative models, unlabeled data is used to either modify or reprioritize hypotheses obtained from labeled data alone. Given the Bayesian formula this is straightforward:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

With this equation, we can easily discover that $p(x)$ influences $p(y|x)$. Mixture models with EM, self-training, are included in this group.

Semi-supervised learning for discriminative models, on the other hand, is not so trivial since $p(y|x)$ is estimated ignoring $p(x)$. To solve this problem, $p(x)$ dependent terms are often brought into the objective function, which amounts to assuming that $p(y|x)$ and $p(x)$ share parameters. Transductive SVMs, Gaussian processes, are included in this group.

The number of literature for semi-supervised distributed representations is limited, but there are deep implications in the methods.

For example, Turian et al. [23] suggested appending unsupervised word representations as features to supervised word representations.

Distance-based methods

Sometimes, it is not necessary to find a representation of each document, but only distances or similarities between documents are needed. In this sense, A simple method to measure the similarity between documents based on the minimal distance word embeddings have to travel from one document to another

in [15]. In this paper, only non stopwords are considered, and the distance is measured using k-NN classification.

2.3 Document classification

Document classification is the task of classifying an entire text by assigning it a class label \hat{c} drawn from a set of predefined labels $\mathcal{C} = c_1, c_2, \dots, c_M$. Training a function to compute labels for an input is a typical case of supervised learning. If one provides a large corpus of rich inputs along with human-provided class labels, we then follow the gradient over the free parameters of the function of the function’s output and the desired class labels.

Some class labels regarded in document classification are as follows: sentiment of movie or product reviews [2, 3, 4, 5, 6, 7, 8], the pass or fail of bills [9], the stock price change of 8K [10], or the subjectivity [5, 11], opinion polarity [5], or topics [8, 6] of any given document. Text categorization is the task of classifying an entire text by assigning it a class label \hat{c} drawn from a set of predefined labels $\mathcal{C} = c_1, c_2, \dots, c_M$.

2.3.1 Sentiment analysis

Sentiment analysis regards the task of judging the polarity of a given document.

The polarity can be either binary (positive and negative), ternary (positive, neutral and negative), or n-ary (one star to five stars). Methods are normally

lexicon based or machine learning based. Lexicon based approaches use a dictionary or corpus to identify the sentiment orientation of a document [?, ?], while the machine learning based approaches train a classifier such as naive Bayes classifier [39], support vector machines [?, 39], and neural networks [?]. The use of machine learning methods for sentiment classification on the document level were pioneered by Pang and Lee [2, 3], mainly using bag-of-words for document representation. Afterwards, approaches using distributed representations, such as recursive neural networks [26], convolutional neural networks [4, 54, 6, 8] were proposed.

A more difficult variant of sentiment analysis is **stance detection**, where the task is to judge the polarity of a text, but towards a particular target. In this case, the target may or may not be existant in the text. For example, given a text “I like Obama”, can be disfavor rather than favor towards the Republican Party. This aspect is not explored in this dissertation.

2.3.2 Others

Some other tasks involving document classification are subjectivity analysis, spam detection, language identification, stock prediction [10].

2.4 Lexicon extraction

There are various levels of methods for extracting a lexicon.

Firstly, and obviously, one can annotate words manually. However, building

such sentiment lexica usually require man-hours.

Also, there can be dictionary-based automatic annotations. However, this method also has the shortcoming that it is almost impossible to list up sentiments for all words, and their derived forms.

Then there are corpus-based automatic annotations, where one measures a word’s polarity based on a set of labeled documents. Hatzivassiloglou and McKeown [55] predicted the semantic orientation of adjectives. Turney [56] was also one of the first to propose methods to measure a word’s polarity using pointwise mutual information (PMI) between words. According to Turney, The PMI between words w_1 , w_2 can be calculated as follows:

$$PMI(w_1, w_2) = \log_2 \frac{p(w_1, w_2)}{p(w_1)p(w_2)}$$

Then, the semantic orientation (SO) of a phrase or document p , can be calculated with the following equation:

$$SO(p) = PMI(p, "excellent") - PMI(p, "poor")$$

After Hatzivassiloglou and McKeown, and Turney’s pioneering work, various methods to automatically annotated words in a corpus. Some examples include methods to first find a proposition’s polarity then search for negations nearby, or methods to find pairs of adjective-nouns that imply polarity (ex: “light laptop”).

Chapter 3

Supervised representations of words and documents

3.1 Background

In this chapter, we describe **supervised paragraph vectors (SPV)**, a method to obtain supervised representations of words and documents, as well as to obtain a representation for the class labels.

3.2 Proposed method

3.2.1 Representation learning

Supervised paragraph vectors (SPV) is a word, document, class label embedding algorithm using a simple neural network with one of each input, hidden, and output layers, and linear activation functions. Each neuron in the input and layer indicate actual words. The main difference of SPV-models from PV-models is that the class label information of documents is utilized as shown in **Figure 3.1**. In this figure, class labels are represented as one-hot vectors in the input layer $\mathbf{z} \in \mathbb{R}^{M \times 1}$, where M is the number of unique class labels. Corre-

sponding class vectors are rows in the class label embedding matrix $\mathbf{C} \in \mathbb{R}^{M \times \delta}$. As with word2vec and paragraph vectors, $\mathbf{x} \in \mathbb{R}^{V \times 1}$, $\mathbf{h} \in \mathbb{R}^{\delta \times 1}$, $\mathbf{y} \in \mathbb{R}^{V \times 1}$, $\mathbf{p} \in \mathbb{R}^{N \times 1}$ are input, hidden, output, document vectors, respectively, where V is the size of the vocabulary, δ is the size of word vectors, N is the number of documents and γ is the number of context words. $\mathbf{W}^{(1)} \in \mathbb{R}^{V \times \delta}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{\delta \times V}$ and $\mathbf{D} \in \mathbb{R}^{N \times \delta}$ are the input, output word embedding matrices, and the document embedding matrix, respectively.

Similarly to paragraph vectors, which assumes that a document vector can be derived by predicting sequences of its constituent words, supervised paragraph vectors also assumes that a class vector can be derived by predicting sequences of its constituents. This is because in a broader sense, a class can be considered as a larger document – one which is created by merging all the individual documents of the corresponding class.

In this perspective, the objective functions of supervised paragraph vectors can be defined by employing an additional term to the paragraph vectors' objective function. Specifically, the objective function for SPV-DM is as follows:

$$L_{SPV-DM} = \frac{1}{T} \sum_{t=1}^T \left\{ \sum_{j=1}^{\gamma} \log p(w_t | w_{t+j}) + \log p(w_t | d) + \log p(w_t | c) \right\} \quad (3.1)$$

where c indicates the class of the document and all other notations are equal to those of paragraph vectors. Similarly, for SPV-DBOW, the objective function

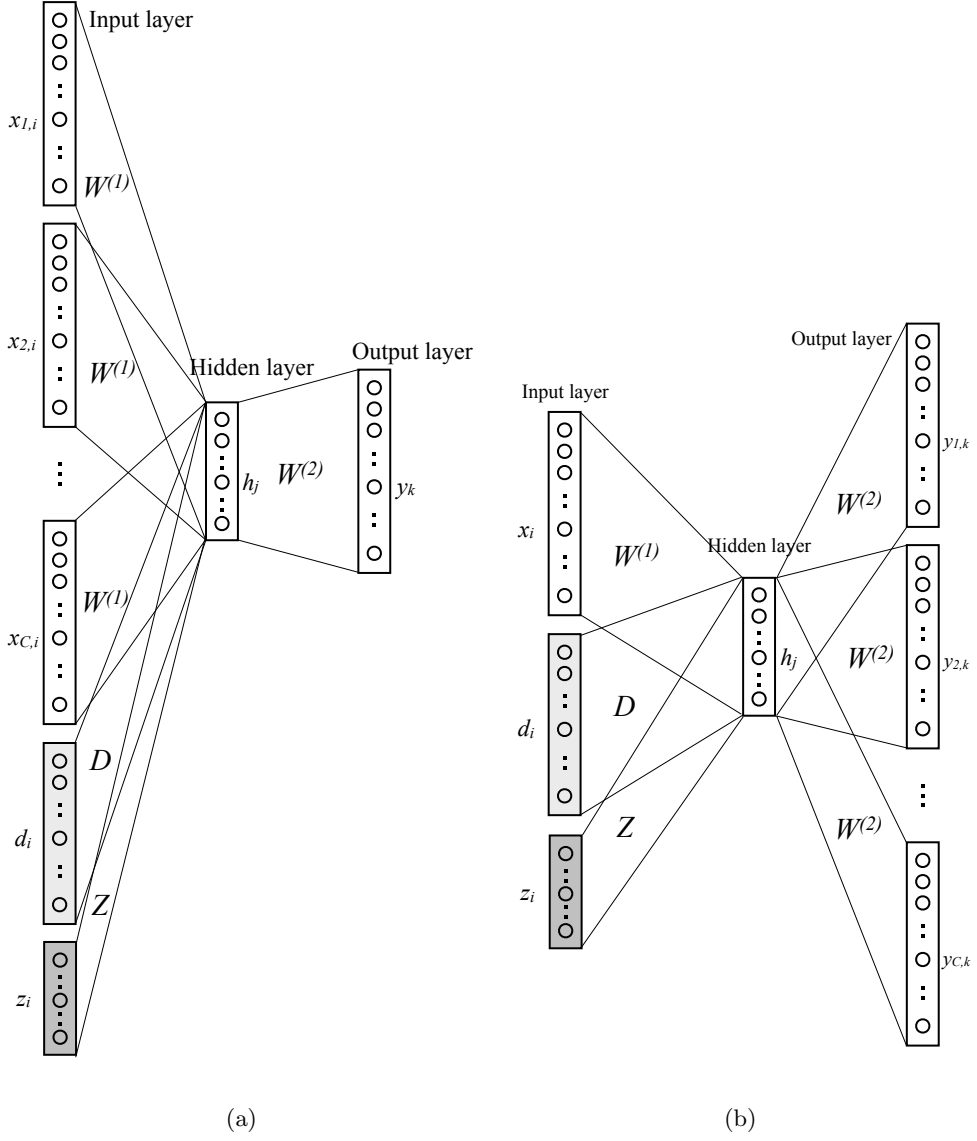


Figure 3.1: Two supervised paragraph vector architectures: (a) distributed memory (SPV-DM) and (b) distributed bag-of-words (SPV-DBOW).

is as follows:

$$L_{SPV-DBOW} = \frac{1}{T} \sum_{t=1}^T \sum_{j=1}^{\gamma} \left\{ \log p(w_{t+j}|w_t) + \log p(w_{t+j}|d) + \log p(w_{t+j}|c) \right\} \quad (3.2)$$

As with word2vec and paragraph vectors, parameters for supervised paragraph vectors are learned with stochastic gradient descent, and the gradients are obtained with backpropagation. Context and target word vectors, along with document vectors and class vectors, can be selected either randomly or sequentially into the learning process.

3.2.2 Inference and classification

After the representation model is trained, one can obtain the word and class vectors, as well as the vectors for each document in the training set. But in order to classify documents in the test set, we should obtain representations for unseen documents as well. This is called the inference step.

In the inference step, we calculate document vectors for each test document using our previously trained representation model, with the exact same process. However, the difference is that the class conditional term from the objective function is removed because we do not know the true class of each document for the test set. Also, class and word vectors, softmax weights obtained from the training stage are fixed, which means that they will not be updated during the test phase.

After we obtain representations for the test documents through this process,

we can finally classify them in two ways. The first approach is to use document vectors as an input for another classifier, since we already have δ dimensional document vectors for the training and test documents. Then we can input these into a separate classifier algorithm, for example, logistic regression. The second approach is to calculate the cosine similarity or any other appropriate similarity measure between a document vector and all class vectors, and choose the closest class as the document’s predicted class.

The whole process of document classification with supervised paragraph vectors is presented in [Algorithm 1](#).

3.3 Experiments

In this section we evaluate the proposed algorithm supervised paragraph vectors, with four different classification datasets.

3.3.1 Datasets

The datasets used for experiments are as follows.

imdb. ¹ A binary class dataset that contains 100K movie reviews from IMDB, with no more than 30 reviews per movie. Original reviews were rated between 1-10, but further merged into three groups: negative (1-4), neutral (5-6), positive (7-10). Among these groups, neutral reviews were not included in the final dataset, and therefore only the reviews highly polarized towards negativity or

¹<http://ai.stanford.edu/~amaas//data/sentiment/>

Algorithm 1 Document classification with supervised paragraph vectors

Given a training dataset $\mathcal{D} = \{(d^{(i)}, c^{(i)})\}_{i=1}^N$ ($c^{(i)} \in \{c_1, \dots, c_M\}$), test dataset $\mathcal{D}' = \{d^{(i)}\}_{i=1}^{N'}$, and a predefined vector dimension δ , where $d^{(i)}$ is a document, $c^{(i)}$ is a corresponding class label, N, N' are the total number of documents in each dataset, and V, N, M are number of unique words, documents and class labels, the detailed procedure for classifying documents with supervised paragraph vectors (SPV) is as follows:

- 1: Learn representation matrices of words $\mathbf{W}^{(1)} \in \mathbb{R}^{V \times \delta}$, documents $\mathbf{D} \in \mathbb{R}^{N \times \delta}$, and class labels $\mathbf{C} \in \mathbb{R}^{M \times \delta}$ with SPV, using documents in the training dataset \mathcal{D} .
 - 2: Train logistic regression or any appropriate classifier, with the derived document vectors \mathbf{D} as input variables and corresponding class labels as output.
 - 3: Infer representations $\mathbf{D}' \in \mathbb{R}^{N' \times \delta}$ of the test dataset \mathcal{D}' , using the representation model trained in Step 1.
 - 4: Classify the test documents \mathcal{D}' , using the classifier trained in Step 2, with \mathbf{D}' as input.
-

positivity were left. Positive and negative reviews were selected evenly, so that a random guess yields 50% accuracy. [57]

amazon. ² A binary class dataset of 200K electronic product reviews from Amazon.com. The number of positive and negative reviews are also equal in this dataset. The dataset used here is was originally gathered by [58] and later modified by [8].

yelp. ³ A multiclass dataset that contains five ordinal document classes from 1 to 5. This dataset was obtained from Kaggle, and preprocessed with the code written by Matt Taddy. ⁴ It is to note that this dataset is distinguishable from other datasets noted here in the sense that the classes are ordinal – the classes are ranked in a sequential order from class 1 to class 5.

20news. ⁵ A multiclass dataset of 20K documents that are of 20 nominal document categories, each corresponding to a different topic. Some classes are closely related to each other by sharing a larger domain (ex: `comp.graphics`, `comp.sys.mac.hardware`), while others are almost completely unrelated (ex: `alt.atheism`, `rec.sport.hockey`).

Table 3.1 shows the summary of datasets used for experiments.

²http://riejohnson.com/cnn_data.html

³<http://www.kaggle.com/c/yelp-recruiting/data>

⁴<http://github.com/TaddyLab/deepir/blob/master/code/parseyelp.py>

⁵<http://qwone.com/~jason/20Newsgroups/>

item	imdb	amazon	yelp	20news
# training documents	25,000	200,000	229,910	11,311
# test documents	25,000	25,000	22,960	7,529
# words in document (average)	256.4	117.1	113.3	279.9
# word types	120,407	237,776	178,509	288,873
# unknown words in the test set	187,756	37,537	20,203	292,072
# classes	2	2	5	20

Table 3.1: Summary table for the datasets.

3.3.2 Implementation

We first compare the proposed algorithm to bag-of-words with term frequency (BOW-TF) and term frequency with inverse document frequency (BOW-TFIDF). The number of terms in the BOW models are limited to 1000 in all datasets, for efficient calculation and better generalization. Also, we compare performance of both architectures of paragraph vectors: PV-DM and PV-DBOW.

Hyperparameters for both paragraph vectors and supervised paragraph vectors are set equally. Specifically, vector dimensions (δ) are 100, trained with 20 context words (γ) (or window size 10), and the words with term frequencies lower than 5 were ignored. Frequent terms are discarded by the probability $P(w_i) = 1 - \sqrt{\tau/f(w_i)}$ where $f(w_i)$ is the term frequency and τ is the threshold value set to 0.001. Negative sampling is employed, with five noise words.

After representations are found for each document, Five classifiers are applied: decision tree (dt), logistic regression (lr), random forest (rf), neural network (nn) and naive Bayes (nb). Decision tree uses the Gini impurity to determine splits. Logistic regression uses L2 penalty. Random forest also uses the

hyperparameter	value
vector dimensions (δ)	100
number of context words (γ)	20
minimum word count	5
downsampling threshold (τ)	0.001
# noise words for negative sampling	5

Table 3.2: Summary table for the hyperparameters for the comparison of PV and SPV.

Gini impurity with 10 trees. Neural network uses a one hidden layer with 100 neurons and a rectified linear unit (ReLU) activation function. There are no particular hyperparameters used for naive Bayes.

Classification accuracies on a held-out test dataset are used to measure performance.

3.3.3 Evaluation

Interpretability. First, in order to check the interpretability of the proposed algorithm, we find the closest and farthest words to each class vector. Here, we use the `imdb` dataset, embed class labels and words into a 100-dimensional embedding space, and use cosine similarity as the similarity measure. We also remove words with a frequency lower than 100. [Table 3.3](#) shows that words with negative sentiment are close to the negative class vector (`c0.0`), and words with positive sentiment are distant. Similarly, words that are closest to the positive class vector (`c1.0`) show positive sentiment, whereas the farthest are non-positive words. This method is not applicable to the unsupervised version of paragraph vectors, because no vector for the class labels exist.

Next, the difference between interclass similarities and intraclass similarities for paragraph vectors and supervised paragraph vectors are compared in order to empirically prove the desired results shown in [Figure 1.6](#). Interclass similarity indicates the average distance between words in two distinct classes, and intraclass similarity means the average distance between words in the same class. Each similarity measure is shown in [Equation 3.3](#) and [Equation 3.4](#), respectively:

$$s_{interclass} = \frac{1}{N_{c_i \neq c_j}} \sum_{w \in c_i} \sum_{v \in c_j} \mathbb{1}_{c_i \neq c_j} s(w, v) \quad (3.3)$$

$$s_{intraclass} = \frac{1}{N_{c_i = c_j}} \sum_{w \in c_i} \sum_{v \in c_j} \mathbb{1}_{c_i = c_j} s(w, v) \quad (3.4)$$

where $\mathbb{1}$ is the indicator function, c_i is the set of words in a given class i , N_k is the total number of qualified word pairs for a given condition k , and $s(w, v)$ is the cosine similarity between two words w and v . Then, it is desirable that the intraclass similarities are large, and interclass similarities are small, and hence the difference, $s_{diff} = s_{intraclass} - s_{interclass}$ is large.

Let us consider the positive set of words:

$$c_{pos} = \{\text{wonderful, awesome, fascinating}\}$$

and the negative set of words:

$$c_{neg} = \{\text{lame}, \text{awful}, \text{bad}\}.$$

Then in the `imdb` dataset, s_{diff} for paragraph vectors equals 0.0077, and supervised paragraph vectors is 0.3398. We can therefore conclude that supervised paragraph vectors is indeed better than paragraph vectors for the purpose of separating words with different sentiment in the embedding space.

We also use visualization for better description and interpretation of the embedding space. For this purpose, principal component analysis (PCA) with two principal components is first performed on the class vectors in order to find the principal components that best describe the space of class vectors. Then the same principal components are reused to plot the word and document vectors on the same space. Though the well-known t-SNE [59, 60] is generally used to visualize the embedding space [50, 5], using PCA in this fashion gives the effect of dispersing the data points according to each class vector. In our experiments, some hand-picked words that indicate sentiment, are plotted in green: love, hate, awesome, wonderful, awful, great, scary, horror, lovely, funny.

Specifically in [Figure 3.2](#), where the IMDB dataset is plotted, the negative class vector (`c0.0`) and the positive class vector (`c1.0`) are located in the left and right side of the principal component space, respectively. Then, on this same space, word vectors and document vectors are drawn. Negative words such as “awful” are located close to the negative class vector, while positive words such as “wonderful” or “lovely” are located close to the positive class vector.

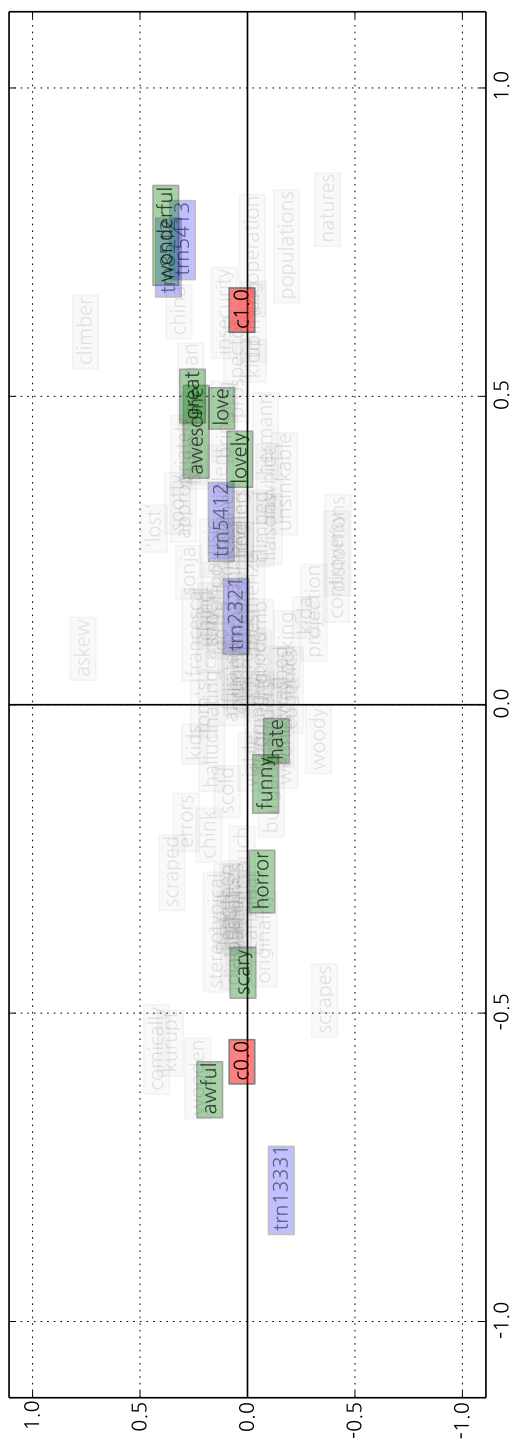


Figure 3.2: Joint visualization of words, documents, and classes for the IMDB dataset. Randomly selected words are gray-colored, and words selected for demonstration purposes are colored in green. Points in blue, red are each document vectors and class vectors, respectively. c1.0, c0.0 each indicate the positive and negative class vector.

In contrast to the imdb dataset, the amazon dataset is consisted of product reviews as seen in [Figure 3.3](#). Therefore it is sensible that the words such as “scary”, which normally depict a sentiment towards a movie, is considered more neutral in a product review perspective.

The Yelp dataset visualization at [Figure 3.4](#) shows even more interesting results. In this figure, the class vectors, indicating ratings for electronic products, are plotted in sequential order from left to right rather than in random order. Words that have strong negative sentiment are plotted on the left side of the diagram while words with strong positive sentiment are plotted to the right. According to this diagram, users at Amazon.com use the word “awful” rather than “hate” to indicate strong negativity.

The 20news dataset, being multiclass dataset of 20 nominal document categories with some classes closely related to each other are drawn in [Figure 3.5](#). As seen here, classes sharing a larger domain are more closely located with each other in the embedding space.

Note that the visualizations with PCA plot the class vectors so they surround the origin, but this does not necessarily divide the words and documents into negative space and the positive space. In other words, words and documents located at the right side of the diagram, are not necessarily positive, and vice versa. Methods can be devised to adjust the coordinates, so that the absolute values in the principal component space have specific meanings.

The words that are closest and farthest to each class vector based on cosine

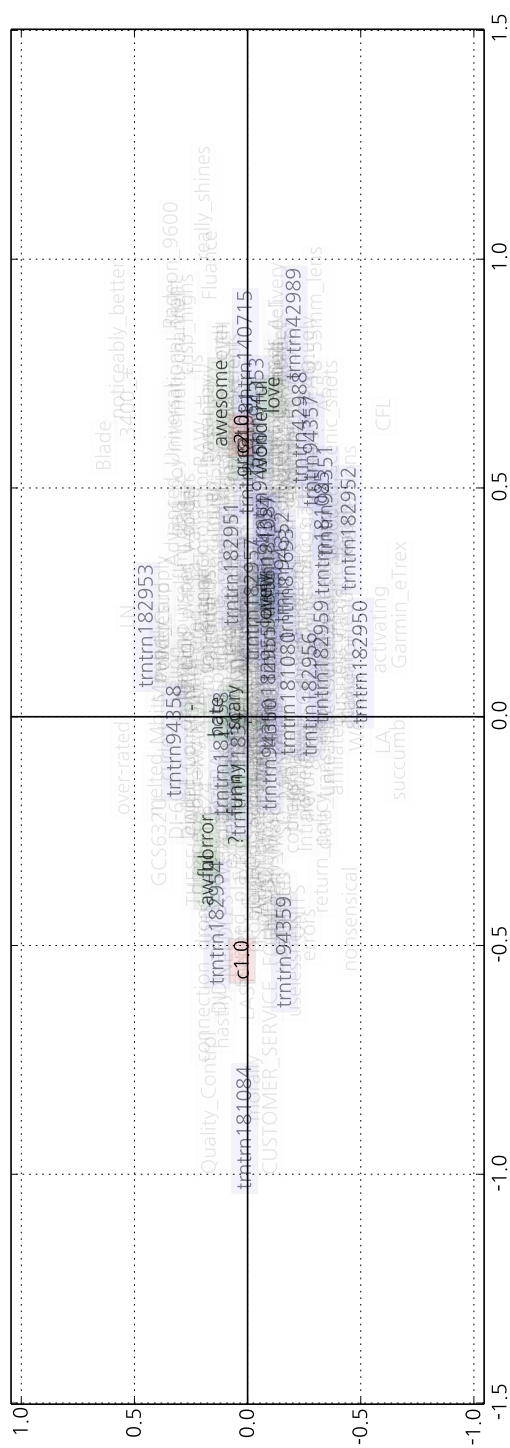


Figure 3.3: Joint visualization of words, documents, and classes for the Amazon dataset. Randomly selected words are gray-colored, and words selected for demonstration purposes are colored in green. Points in blue, red are each document vectors and class vectors, respectively. c1.0, c2.0 each indicate the negative and positive class vector.

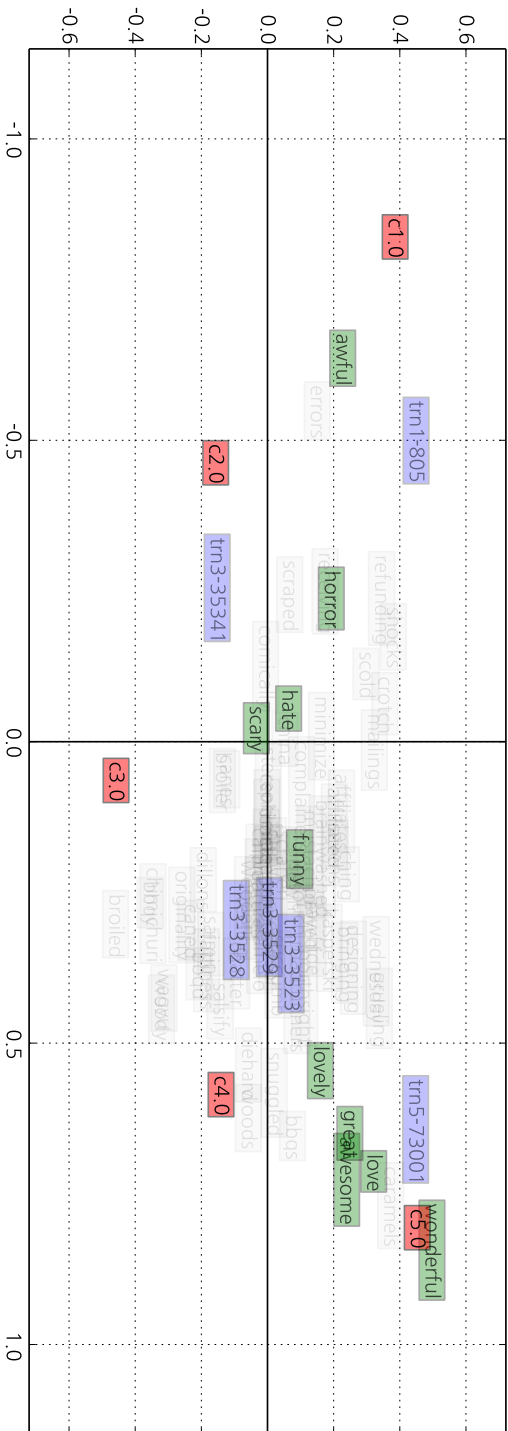
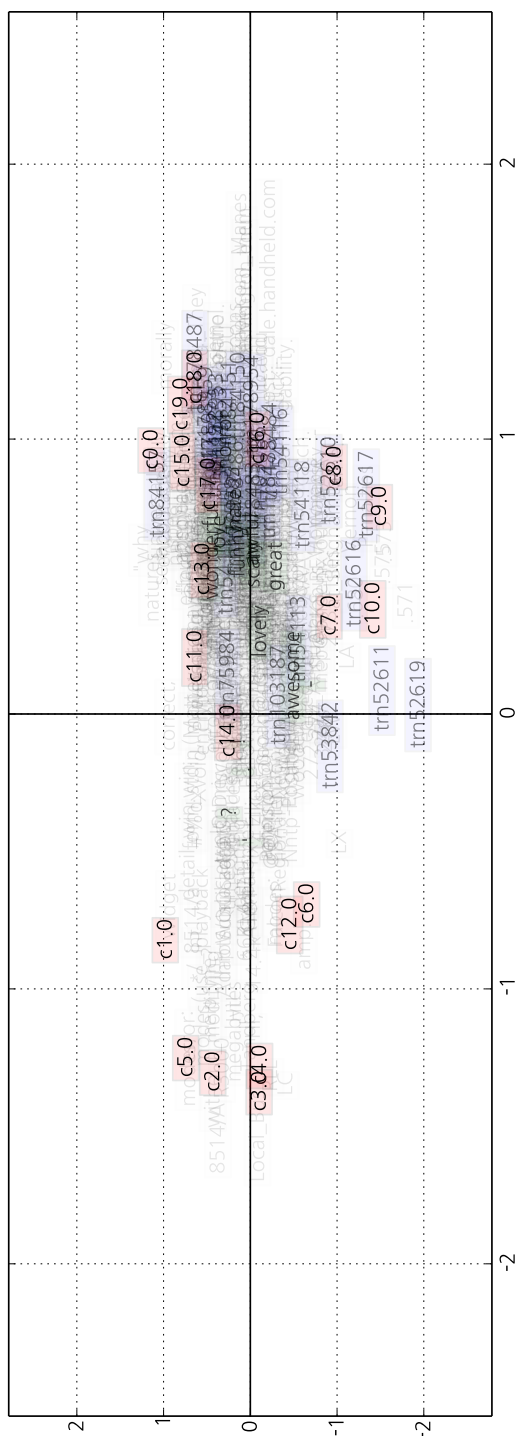


Figure 3.4: Joint visualization of words, documents, and classes for the Yelp dataset. As with IMDB, randomly selected words are gray-colored, and words selected for demonstration purposes are colored in green. Points in blue, red are each document vectors and class vectors, respectively. c1.0 to c5.0 each represent one star to five star labels.



Class	Closest 10 words	farthest 10 words
c0.0	atrocious (194)	unique (627)
	just (17364)	touching (431)
	poor (1818)	captures (216)
	unfunny (244)	dark (1190)
	stupid (1639)	vivid (108)
	awful (1538)	achievement (113)
	lame (711)	joy (273)
	porn (236)	future (825)
	bad (8640)	wonderful (1648)
	fake (438)	delight (147)
c1.0	wonderful (1648)	stupid (1639)
	perfection (140)	suit (228)
	sadness (111)	crap (952)
	fascinating (388)	garbage (428)
	powerful (609)	fat (220)
	quiet (281)	fake (438)
	also (9145)	apparently (910)
	captivating (118)	porn (236)
	joy (273)	plastic (119)
	poignant (155)	looks_like (779)

Table 3.3: Ten closest and farthest words for the negative class vector (c0.0) and the positive class vector (c1.0) in the IMDB dataset. Word with frequency under 100 were removed. The number in the parenthesis next to each word is the frequency of the word in the training data.

Class	Closest 10 words	Furthest 10 words
1	unacceptable (292) denied (144) refunded (128) incident (244) supervisor (123)	nothing_beats (115) bruschettas (124) sweet_savory (143) awesomely (113) im_sucker (266)
2	terrible (3556) poor (3079) horrible (3633) pathetic (307) unacceptable (292)	highly_recommended (1143) bravo (209) yum (128) shrimp_ceviche (116) perfect_combination (162)
3	ok (15770) average (5583) okay (8089) odd (2331) boring (1525)	tony (292) karen (108) todd (120) greg (149) mary (191)
4	plus (8766) must_say (1325) :) (8956) nice (49526) especially (10474)	lied (184) supervisor (123) acted_like (213) mgr (122) unacceptable (292)
5	amazing (24147) wonderful (10545) incredible (3142) fantastic (10387) fabulous (3525)	lame (953) compensate (139) uninspired (218) deal_breaker (101) acceptable (563)

Table 3.4: Five closest and farthest words for each class vector in the yelp dataset, where the minimum count of each word is above 100. The number in the parenthesis next to each word is the frequency of the word in the training data, and the decimal number is the cosine similarity of the word towards the class vector.

similarity in the 100-dimension embedding space are observed, as shown in [Table 3.3](#). Words with a frequency lower than 100 were removed. As seen in this table, words with negative sentiment are closest to the negative class vector (c0.0), and words with positive sentiment are farthest. Similarly, words that are closest to the positive class vector (c1.0) show positive sentiment, whereas the farthest are non-positive words.

Finally, closest documents to each class vector in the imdb dataset, also show a tendency for strong sentiment. Specifically, the closest document to the negative class contains many negative words such as “worst”, “stupid” or “shitty” as seen in [Figure 3.6](#). On the other hand, the closest document to the positive class contains many positive words such as “best”, “great” or “superb” as seen in [Figure 3.7](#).

Discriminative power. Next, in order to evaluate discriminative power, we perform document classification and measure the classification accuracies as a proxy measure. [Table 3.5](#) shows the classification accuracies for bag-of-words (BOW), paragraph vectors (PV), supervised paragraph vectors (SPV) per dataset. Note that the accuracies for yelp and 20news are lower than imdb and amazon, due to the number of unique class labels.

As seen in the table, SPV-DBOW outperforms bag-of-words and paragraph vectors, for imdb, amazon, and 20news while the bag-of-words strategy is best for yelp with a marginal difference. SPV-DM and SPV-DBOW outperformed PV-DM and PV-DBOW, respectively.

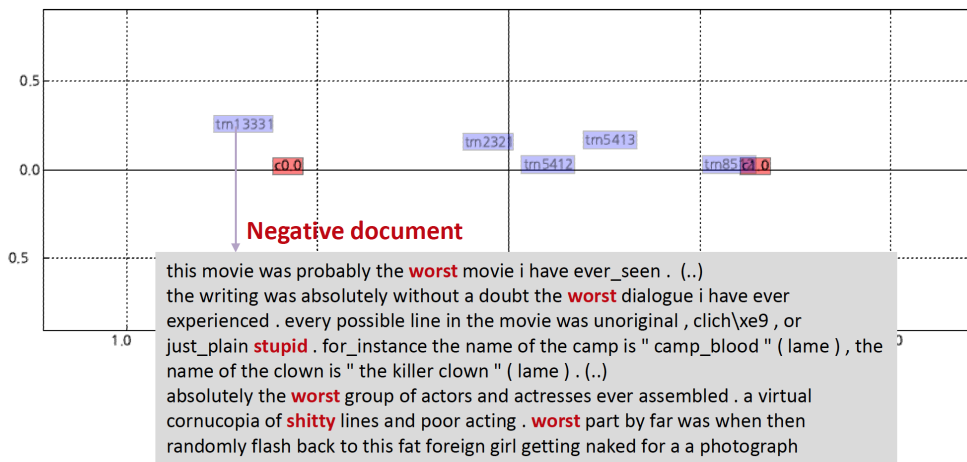


Figure 3.6: The most negative document in the imdb dataset, according to the affinity towards the class vectors.

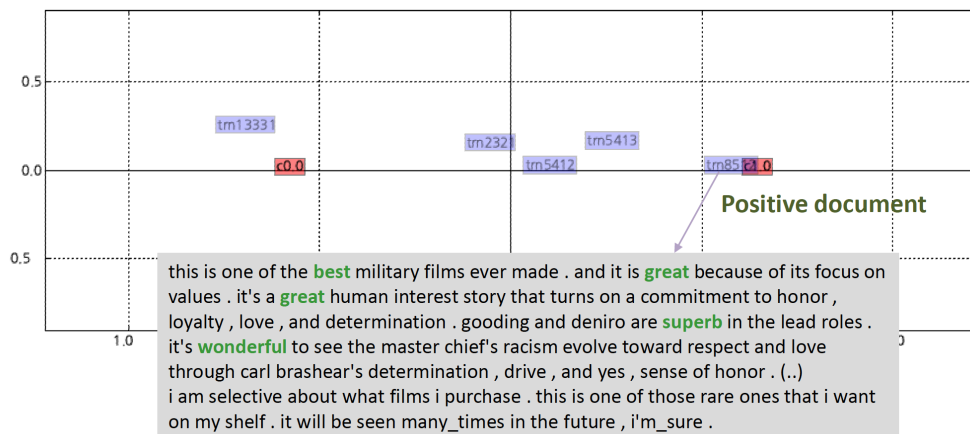


Figure 3.7: The most positive document in the imdb dataset, according to the affinity towards the class vectors.

PV and SPV are run for a total of 100 epochs, where an epoch indicates one forward pass and one backward pass of all the training examples. Before the algorithms terminate at 100 epochs, classification accuracies using logistic regression are measured at 1, 5, 10, 30, 50, 70, 100 epochs.

In order to justify the robustness of the proposed method, we also provide experimental results using various simple classifiers in [Table 3.6](#). PV and SPV were run for 100 epochs, and the best result was chosen for each representation algorithm and classifier pair. For `imdb`, SPV gave the best classification accuracies. For `amazon`, `yelp`, and `20news`, BOW showed better results with the tree algorithms - decision tree and random forest - while SPV showed better results with logistic regression, neural network, and naive Bayes. However, it is to note that SPV proved higher classification accuracies compared to the corresponding PV architecture in all cases, regardless of the classifier.

Computational efficiency. [Figure 3.9](#) shows the relation between the number of documents and classification accuracy, using the `imdb` dataset and a logistic regression classifier. As the number of training documents increase, the classification accuracy increases in the shape of a logarithmic function ?? shows the relation between the number of documents and the representation algorithm training time. Here, the training time increases in a linear manner.

[Figure 3.10](#) shows the relation between the number of vector dimensions and classification accuracy. As the number of dimensions increase, the classification accuracy increase until approximately 100 dimensions, and decreases by a small

	imdb						
n(epochs)	1	5	10	30	50	70	100
BOW-TF	85.30	-	-	-	-	-	-
BOW-TFIDF	85.55	-	-	-	-	-	-
PV-DM	77.06	80.78	80.84	79.68	81.22	81.49	82.16
PV-DBOW	85.89	88.19	88.47	88.27	88.22	88.12	88.04
SPV-DM	82.57	81.68	82.05	82.66	82.42	82.53	82.61
SPV-DBOW	87.58	*88.87	88.69	88.53	88.51	88.56	88.49
	amazon						
n(epochs)	1	5	10	30	50	70	100
BOW-TF	85.91	-	-	-	-	-	-
BOW-TFIDF	85.97	-	-	-	-	-	-
PV-DM	76.47	77.38	78.86	79.26	75.83	77.00	78.57
PV-DBOW	86.87	87.96	88.30	88.34	88.31	88.42	88.18
SPV-DM	79.05	78.35	78.66	80.26	80.36	80.62	80.70
SPV-DBOW	88.58	89.21	89.16	*89.34	89.08	89.06	89.29
	yelp						
n(epochs)	1	5	10	30	50	70	100
BOW-TF	58.42	-	-	-	-	-	-
BOW-TFIDF	58.93	-	-	-	-	-	-
PV-DM	50.59	51.70	52.67	52.97	51.73	51.81	52.70
PV-DBOW	58.53	59.37	58.91	59.13	59.10	59.06	59.21
SPV-DM	51.48	51.42	52.40	53.14	53.77	53.75	53.84
SPV-DBOW	60.13	*60.21	60.04	59.93	59.85	59.77	59.93
	20news						
n(epochs)	1	5	10	30	50	70	100
BOW-TF	58.58	-	-	-	-	-	-
BOW-TFIDF	63.43	-	-	-	-	-	-
PV-DM	24.45	41.17	45.36	49.04	52.34	53.85	54.27
PV-DBOW	53.51	69.16	72.51	74.76	75.22	75.16	75.40
SPV-DM	44.76	64.18	67.01	67.84	68.32	68.34	67.70
SPV-DBOW	69.41	76.97	77.95	78.93	78.93	79.47	*79.59

Table 3.5: Test accuracies for each representation algorithm, per number of epochs. Logistic regression is used as the classifier. Each representation algorithm is run for a maximum number of 100 epochs, where an epoch indicates one forward pass and one backward pass of all the training examples. Numbers in bold are the highest accuracies for each representation algorithm (row). The highest accuracy among all algorithms is marked with an asterisk. The rightmost column named t is the training time per epoch in seconds.

	imdb				
classifier	dt	lr	rf	nn	nb
BOW-TF	69.75	86.90	76.20	84.40	79.10
BOW-TFIDF	69.82	86.93	77.34	84.90	80.44
PV-DBOW	70.06 (75)	82.16 (100)	77.65 (30)	84.01 (30)	85.76 (75)
PV-DM	67.44 (100)	88.47 (10)	73.94 (100)	80.99 (100)	78.84 (10)
SPV-DBOW	71.50 (1)	82.66 (30)	79.00 (1)	85.12 (5)	86.12 (3)
SPV-DM	68.14 (50)	88.87 (5)	75.82 (30)	83.02 (1)	80.97 (3)
	amazon				
classifier	dt	lr	rf	nn	nb
BOW-TF	73.96	85.92	80.75	83.67	77.95
BOW-TFIDF	74.44	85.97	81.72	84.20	80.40
PV-DBOW	71.97 (1)	79.26 (30)	79.45 (30)	88.15 (30)	83.28 (70)
PV-DM	65.12 (100)	88.42 (70)	72.68 (100)	81.74 (100)	72.14 (100)
SPV-DBOW	74.42 (3)	80.70 (100)	81.01 (10)	89.06 (75)	84.26 (30)
SPV-DM	67.66 (70)	89.34 (30)	74.62 (75)	84.00 (50)	75.02 (75)
	yelp				
classifier	dt	lr	rf	nn	nb
BOW-TF	41.34	59.06	47.68	50.43	45.94
BOW-TFIDF	42.02	59.25	48.98	50.64	48.28
PV-DBOW	39.20 (5)	52.97 (30)	46.08 (1)	60.42 (70)	55.78 (50)
PV-DM	38.55 (10)	59.37 (5)	46.08 (10)	59.37 (10)	53.82 (10)
SPV-DBOW	40.16 (1)	53.84 (100)	48.04 (1)	60.82 (70)	56.60 (75)
SPV-DM	38.55 (10)	60.21 (5)	46.08 (10)	59.74 (10)	53.82 (10)
	20news				
classifier	dt	lr	rf	nn	nb
BOW-TF	44.16	58.29	52.26	58.14	40.76
BOW-TFIDF	44.71	63.86	51.94	58.63	45.11
PV-DBOW	24.82 (5)	54.27 (100)	36.09 (10)	66.58 (70)	68.32 (100)
PV-DM	22.22 (75)	75.40 (100)	29.22 (70)	55.12 (50)	49.61 (100)
SPV-DBOW	30.03 (3)	68.34 (70)	45.08 (5)	72.64 (70)	74.64 (30)
SPV-DM	35.99 (10)	79.59 (100)	49.53 (10)	71.94 (5)	66.78 (50)

Table 3.6: Test accuracies for each representation algorithm, per classifier. Document representations derived with each representation algorithm is classified with five different classifiers: decision tree (dt), logistic regression (lr), random forest (rf), neural network (nn) and naive Bayes (nb). Numbers in bold are the highest accuracies for each classifier (column). Numbers in parentheses indicate the number of epochs of the corresponding result.

amount. Also, Figure 3.11 shows the relation between the number of vector dimensions and training time. The training time increases linearly, according to the number of dimensions.

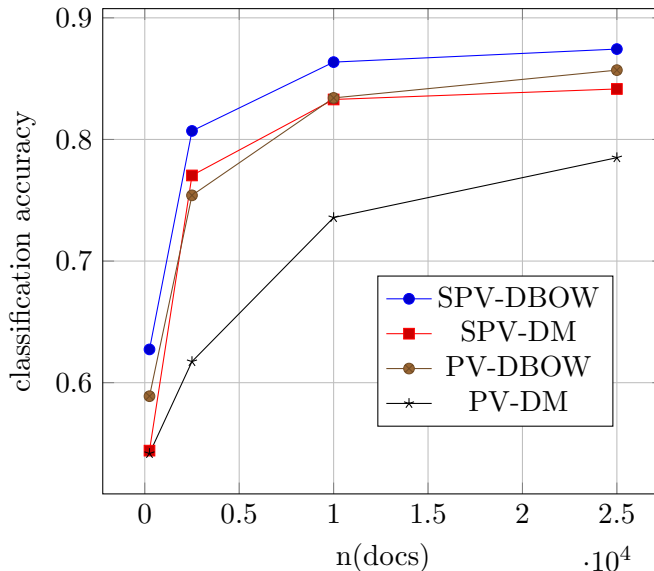


Figure 3.8: The relation between number of documents $n(docs)$ and classification accuracy.

3.4 Summary and discussions

We proposed supervised paragraph vectors, a method to jointly embed words, document and classes into the same space. We proved the benefits of this algorithm by finding close and far words to the class vectors, calculating the difference between intraclass and interclass similarities, and visualizing the joint space of words, documents and classes. This is an advantage in the sense of representation interpretability, and one that was not possible with previous

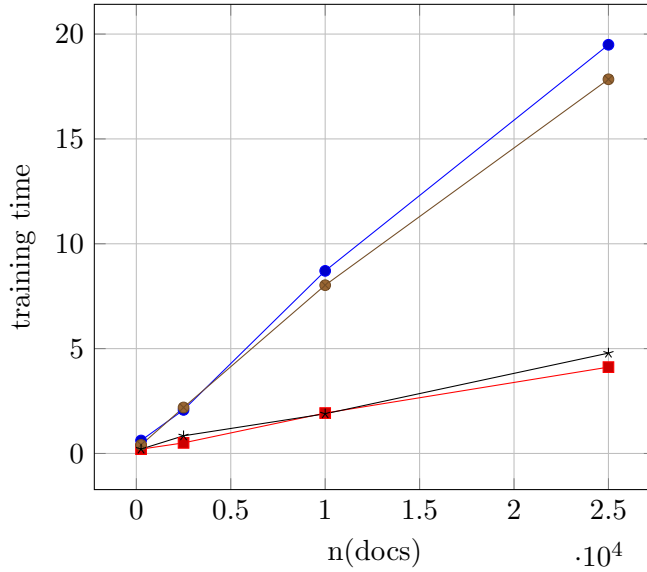


Figure 3.9: The relation between number of documents $n(docs)$ and training time.

unsupervised methods. We also showed that the supervision of class labels increased the discriminative power of document vectors, with a slight increase in training time.

However, when one attempts to reconstruct the original document from the representations obtained with supervised paragraph vectors, the quality can be low because the order of words are lost due to the bag-of-words assumption. For the same reason, supervised paragraph vectors is not an algorithm that achieves the best accuracy compared to state-of-the-art algorithms that include the sequential information of words.

Also, when compared to BOW, SPV sometimes does not show better results, specifically in the `yelp` dataset. This may be due to the characteristics of

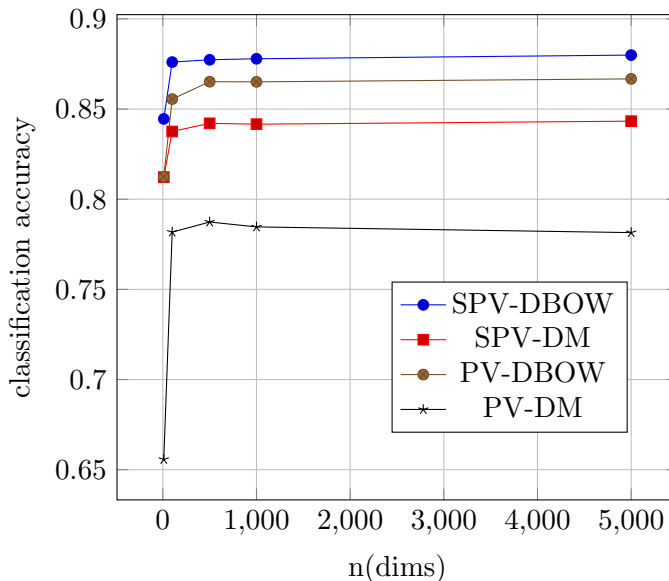


Figure 3.10: The relation between number of vector dimensions $n(dims)$ and classification accuracy.

the `yelp` dataset, in the sense that the classes are ordinal, but this must be investigated to a further level to be confirmed. If SPV is extended so that it can consider word sequences into account, then it can be expected that SPV could show better results than SPV.

SPV can also be applied to other various domains. The “class labels” in this algorithm does not necessarily have to be the label for a classification task, but could also be multiple “tags” that describe a document. For example, we could attempt to find “user vectors” by aggregating all reviews written by one user, and further use such vectors for user profiling, or calculating user similarities, and apply the algorithm to recommendation tasks.

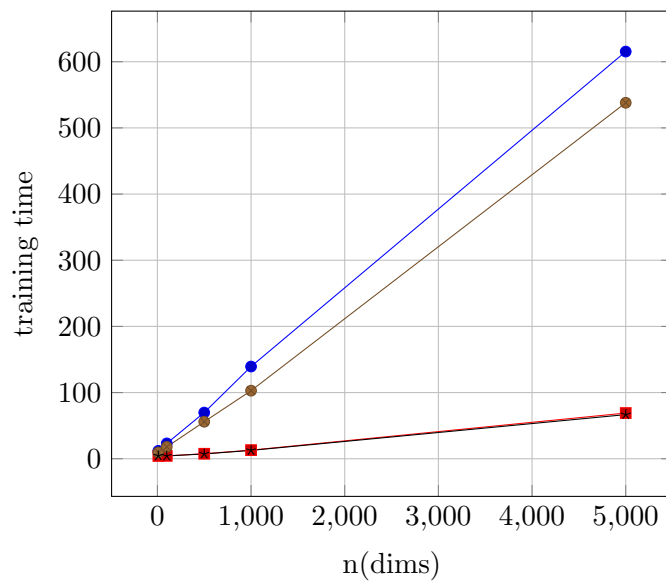


Figure 3.11: The relation between number of vector dimensions $n(\text{dims})$ and training time.

Chapter 4

Semi-supervised representations of words and documents

4.1 Background

Supervised paragraph vectors can be further extended a semi-supervised version, that makes it possible to employ unlabeled documents into the training process.

Statistical learning requires a large amount of training data. But in most cases, we only have a limited amount of labeled data. Specifically, in text mining, a small amount of training data may result in many out-of-vocabulary (OOV) words – words that were unobserved in the training data but exist in the test data. Can we figure out a way for our learning algorithms to take advantage of additional unlabeled data?

While PV-DM and PV-DBOW are fully unsupervised models, the proposed SPV-DM and SPV-DBOW are fully supervised models. But then we could also consider the semi-supervised case, where documents with no class labels can be augmented to the training data..

This can be achieved either by assigning no label or simply by assigning a

dummy class label to the unlabeled documents. For example, if a classification task has two labels 1 (positive) and -1 (negative) for each document, we can create another class, say class 0 for neutral or unlabeled documents. Then, the rest of the training process is identical to the original SPV.

A comparison table for PV, SPV and SSPV are shown in [Figure 4.1](#).

4.2 Proposed method

SSPV is a semi-supervised extension of paragraph vectors that employs unlabeled data as well as labeled data.

In this method, we assign a dummy class label to the unlabeled documents. For example, if a task has two labels 1 (positive) and -1 (negative), create another class, say class 0 for neutral or unlabeled documents. The rest of the training process is identical to the original SPV.

The rationale behind this method is that it will improve test accuracies because out-of-vocabulary (OOV) can be included in the trained model via unlabeled data.

4.3 Experiments

SSPV-DBOW, the semi-supervised version of SPV-DBOW is experimented with the same settings with SPV-DBOW. except for the fact that 10K unlabeled documents were randomly selected from Wikipedia ¹ in each experiment, and labeled with a dummy class.

¹<https://dumps.wikimedia.org/enwiki/20150901/>




- **General representations:** Train general vectors with unlabeled data *only*

- **Task-specific representations**
 1. Train task-specific vectors with labeled data (joint learning)

 2. Train task-specific vectors with labeled *and* unlabeled data


Figure 4.1: Comparison diagrams of PV, SPV and SSPV.

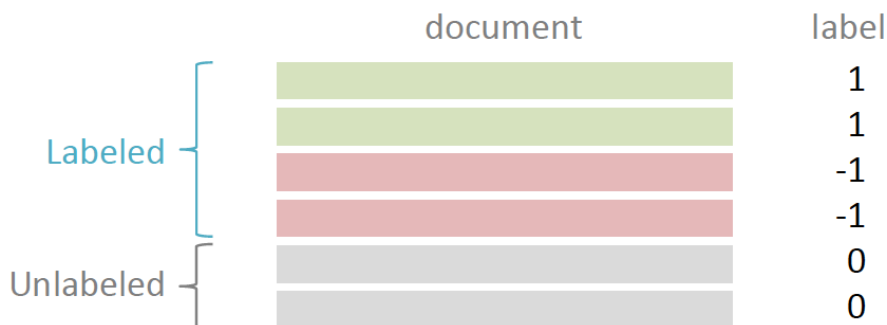


Figure 4.2: Semi-supervised paragraph vectors.

4.3.1 Datasets

The four datasets described in Chapter 3 – imb, amazon, yelp, and 20news – were reused. Refer to Chapter 3 for specifics of the datasets.

4.3.2 Implementation

Specifically for SSPV, 10K unlabeled documents were selected from Wikipedia.

² These documents are labeled as a separate class – namely, class 0 – to be distinguished from preexisting classes.

The rest of the experimental settings are almost identical to those of SPV. We first compare the proposed algorithm to bag-of-words with term frequency (BOW-TF) and term frequency with inverse document frequency (BOW-TFIDF). The number of terms in the BOW models are limited to 1000 in all datasets, for efficient calculation and better generalization. Also, we compare performance of both architectures of paragraph vectors: PV-DM and PV-DBOW.

Hyperparameters for PV, SPV, SSPV are all set equally. Specifically, vector

²<https://dumps.wikimedia.org/enwiki/20150901/>

hyperparameter	value
vector dimensions (δ)	100
number of context words (γ)	20
minimum word count	5
downsampling threshold (τ)	0.001
# noise words for negative sampling	5

Table 4.1: Summary table for the hyperparameters for the comparison of PV, SPV and SSPV.

dimensions (δ) are 100, trained with 20 context words (γ) (or window size 10), and the words with term frequencies lower than 5 were ignored. Frequent terms are discarded by the probability $P(w_i) = 1 - \sqrt{\tau/f(w_i)}$ where $f(w_i)$ is the term frequency and τ is the threshold value set to 0.001. Negative sampling is employed, with five noise words.

After representations are found for each document, logistic regression with L2 penalty is trained on the training set. Finally, classification accuracies on a held-out test dataset are used to measure performance.

4.3.3 Evaluation

As seen in the table, SSPV-DBOW outperforms bag-of-words and paragraph vectors, as well as SPV for imdb, amazon, and 20news, while the bag-of-words strategy is best for yelp with marginal difference. Also, when comparing SPV-DM and SPV-DBOW to each counterpart PV-DM and PV-DBOW, we can see that the performance is enhanced with SPV-DM in all cases. SSPV shows better performance than SPV for most cases except for the 20newsgroup dataset, where the number of documents is small.

Experimental results for PV, SPV and SSPV shown in this table were all trained with one epoch for fair comparison, but it is important to note that for some experiments, accuracies increased by up to 3% if trained with more epochs.

algorithm	imdb	amazon	yelp	20news
BOW-TF	85.3	85.91	58.42	58.58
BOW-TFIDF	85.55	85.97	58.93	63.43
PV-DM	77.24	76.46	49.98	23.38
PV-DBOW	85.77	86.57	57.34	53.3
SPV-DM	82.25	78.66	50.18	42.65
SPV-DBOW	87.67	88.7	58.81	68.16
SSPV-DBOW	87.95	88.8	58.85	68.08

Table 4.2: Held-out accuracies for each dataset. Numbers in bold are the best (highest) accuracies for each dataset (column).

4.4 Summary and discussions

SSPV shows better performance than SPV for all cases except for 20news. For 20news, we consider that SSPV did not show better results than SPV because the number of documents are too small compared to the number of vocabulary used – the domain of the documents are not homogeneous as in other datasets. We can expect better results in other corpora if the documents are more coherent to a specific domain.

As previously pointed out in Chapter 3, SPV does not show better results than BOW specifically in the `yelp` dataset. Neither does SSPV. The results should be further investigated in order to configure whether it was the algorithm or the dataset’s characteristics that derived such results.

Other methods to enable semi-supervised learning can be investigated as well. For example, as in Turian et al. [23], we can try extending the dimension of word vectors with pretrained vectors. Another method is to simply include unlabeled documents into the training set, and create a novel sampling method that can effectively take the implications of classes in to account.

Chapter 5

Scored lexicon extraction for classification of short documents

5.1 Background

Short documents don't have enough word pairs to fully derive a reasonable document representation. For example, Twitter Tweets are known to be 28 characters in average, as seen in [Figure 5.1](#).

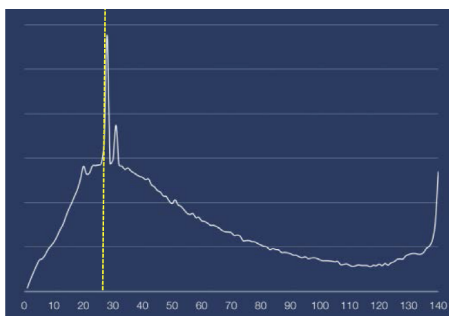


Figure 5.1: Tweet lengths from a 1M sized sample¹

In such cases, it is better to use sentiment lexica for the classification of extremely short-documents instead of directly computing representations for each document. Also, even for average length or long documents, the needs

¹<http://thenextweb.com/twitter/2012/01/07/interesting-fact-most-tweets-posted-are-approximately-30-characters-long/>

for sentiment lexica are high, because they provide a simple way to annotate sentiments.

However, as previously covered in Chapter 2, it is normally expensive to obtain a lexicon, or otherwise difficult to annotated all words respective to a corpus. Would there we a way to automatically calculate sentiment scores for all words, including their derived forms?

5.2 Proposed method

SPV provides a way to automatically extract a score-based sentiment lexicon, by calculating the distance, or similarity between a word and class vectors. Any real-valued function that can act as a similarity measure can be employed, in

Negative (c0)			Positive (c1)		
word	freq	sim	word	freq	sim
atrocious	194	0.4641	wonderful	1648	0.4269
just	17364	0.4545	perfection	140	0.4245
poor	1818	0.4509	sadness	111	0.4241
unfunny	244	0.4327	fascinating	388	0.4094
stupid	1639	0.4300	powerful	609	0.4040
awful	1538	0.4211	quiet	281	0.4030
lame	711	0.4185	also	9145	0.3958
porn	236	0.4093	captivating	118	0.3918
bad	8640	0.4062	joy	273	0.3834
fake	438	0.4033	poignant	155	0.3724
horrid	114	0.4026	still	5519	0.3569
retarded	122	0.4023	brilliantly	235	0.3536
non-existent	119	0.4015	strongly	125	0.3530
useless	126	0.3989	wonderfully	323	0.3504
fat	220	0.3986	daniel	142	0.3494
terrible	1628	0.3873	dramas	126	0.3484
horrible	1192	0.3849	captures	216	0.3466
just_plain	332	0.3765	appreciated	173	0.3441
boring	1803	0.3734	victoria	130	0.3409
idiotic	144	0.3702	emotions	368	0.3390

Table 5.1: Closest 20 word vectors from class vectors in the imdb dataset.

order to measure the affinity of a word toward each class, then the calculated similarity values can be used as a negativity or positivity score of each word as seen in [Table 5.1](#). Here, we use cosine similarity as the similarity measure, denoted as sim , which is calculated between two vectors \mathbf{a} and \mathbf{b} as follows:

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$$

Such scores are calculated between all words in a document and each class. Then, scores are averaged or summed for each class, and the class with the maximum value is selected according to the following equation:

$$\arg \max_{c \in C} \sum_{i=1}^K \text{sim}(w_i, c)$$

where sim is the similarity measure, namely cosine similarity, K is the number of words in sentence, and C is the set of classes such as {Positive, Negative}.

Though cosine similarity is normally not a preferred similarity measure in the vector space due to sparsity issues, the issue can be relieved when we take the similarities of vectors into consideration as observed in [\cite{park2010vector}](#). Such scores are calculated between all words in a document and each class. Then, scores are averaged or summed for each class, and the class with the maximum value is selected according to the following equation:

An example of employing this equation to a sentence or document is demonstrated in [Table 5.2](#). In this example, one can observe that the sentence "**this**

	Negative score	Positive score
this	0.1691	0.1403
is	-0.0098	-0.0049
a	-0.0215	-0.0162
wonderful	-0.0784	0.1202
movie	0.1706	0.1229
Score sum	0.2300	0.3624

Table 5.2: Example of classifying a document with the scored lexicon. The sentence "this is a wonderful movie" is classified into the positive class according to the similarity scores.

"is a wonderful movie" is classified as a positive sentence, because the overall sum of scores favors the positive class over the negative class.

5.3 Experiments

5.3.1 Datasets

We experiment on the imdb dataset, described in Chapter 3. The imdb dataset is a binary class dataset that contains 100K movie reviews from IMDB, with no more than 30 reviews per movie. Original reviews were rated between 1-10, but further merged into three groups: negative (1-4), neutral (5-6), positive (7-10). Among these groups, neutral reviews were not included in the final dataset, and therefore only the reviews highly polarized towards negativity or positivity were left. Positive and negative reviews were selected evenly, so that a random guess yields 50% accuracy.

5.3.2 Implementation

The imdb dataset is not used as is, but is preprocessed in order to conduct this experiment. Specifically, words exceeding $n(\text{words})$ were truncated from each document, where $n(\text{words})$ are in 5, 10, 30, 50 as shown in [Table 5.3](#). If a document length is smaller than $n(\text{words})$, it is left as is. The minimum value 5 was chosen because it can be considered the average length of Tweets. If we can obtain good results with such short documents, we may conclude that a method is effective for extremely short documents. The maximum value 50 was chosen because the average number of words in the imdb dataset is approximated 100.

After each document was truncated into $n(\text{words})$, the previously denoted classification method is applied to each truncated document.

For example, if there was a sentence “brilliant over-acting by lesley ann_warren . best dramatic hobo lady i have ever_seen , and love scenes in clothes warehouse are second to none .”, the truncated versions would be as follows:

- $n(\text{words})=5$: brilliant over-acting by lesley ann_warren
- $n(\text{words})=10$: brilliant over-acting by lesley ann_warren . best dramatic hobo lady
- $n(\text{words})=30$: brilliant over-acting by lesley ann_warren . best dramatic hobo lady i have ever_seen , and love scenes in clothes warehouse are second to none .
- $n(\text{words})=50$: brilliant over-acting by lesley ann_warren . best dramatic hobo lady i have ever_seen , and love scenes in clothes

warehouse are second to none .

5.3.3 Evaluation

As shown in [Table 5.3](#), the proposed method shows better classification results for extremely short sentences.

A strength of this method is that no separate classifier is necessary, and only calculates and averages similarities. This is a big difference from the methods described in previous chapters, where a separate classifier had to be trained separately.

5.4 Summary and discussions

Similarities between each word-class pairs were calculated using SPV, and were used as each word’s sentiment score. Using a scored sentiment lexicon showed better results than PV or SPV for short documents.

This method can be further explored in various ways. First, various normalization methods may be applied. For example, the similarity scores of each word for each class may be normalized, so that the sum of similarity scores are set to 1. Second, attention-based methods to each word can be applied, by identifying a word’s sentiment importance. Third, additional information such as part-of-speech (POS) tags can be augmented.

n(words)*	BOW-TF	BOW-TFIDF	PV-DM	PV-DBOW	SPV-DM	SPV-DBOW	Lexicon extraction
5	58.03	58.33	57.11	56.55	56.44	57.56	58.72
10	61.78	62.57	59.97	61.69	62.00	64.35	60.65
30	68.15	68.88	64.03	69.47	66.38	71.14	63.61
50	71.79	72.47	66.68	73.53	70.39	75.13	65.96

In conjunction with a separate classifier
(ex: Logistic regression)

No classifier needed

Table 5.3: Lexicon experimental results for the imdb dataset

Chapter 6

Conclusion

6.1 Summary and contributions

Three criteria of good representations for documents were proposed: interpretability, discriminative power, efficient computation. Under such criteria, a supervised and semi-supervised method to derive distributed representations for documents was proposed, for better task adaptation and interpretability. A scored-based lexicon was extracted using SPV, and applied to short document classification tasks.

Supervised paragraph vectors (SPV), is a method to jointly embed words, document and classes into the same space. We proved the benefits of this algorithm by finding close and far words to the class vectors, calculating the difference between intraclass and interclass similarities, and visualizing the joint space of words, documents and classes. This is an advantage in the sense of representation interpretability, and one that was not possible with previous unsupervised methods. We also showed that the supervision of class labels increased the discriminative power of document vectors, with a slight increase in training time.

SPV was further extended a semi-supervised version, semi-supervised paragraph vectors (SSPV), that made it possible to employ unlabeled documents into the training process. SSPV showed better performance than SPV for all cases except for 20news.

Then a lexicon scoring method using SPV was proposed. Similarities between each word-class pairs were calculated using SPV, and were used as each word’s sentiment score. Using a scored sentiment lexicon showed better results than PV or SPV for short documents.

Table 6.1 shows the summary of the benchmark and proposed algorithms.

6.2 Future work

Theoretical foundations of how each method works should be further studied. The efficiency of the methods could be enhanced by introducing sampling methods for context-target word pairs. The effectiveness of the methods should be also checked in large scale data analysis. We could also try using applying the proposed methods to other domains and check the transferability of domains, such as the online webpage modeling example seen in **Figure 2.5**. We can also expect to visualize online reviews on the Web, for better user experience for a product or movie.

Specifically for each method, improvements are suggested as follows.

SPV can also be applied to other various domains. The “class labels” in this algorithm does not necessarily have to be the label for a classification task, but could also be multiple “tags” that describe a document. For example, we could

	Interpretability	Discriminative power	Computational efficiency
PV	++	+	++
SPV	+++	++	++
SSPV	+++	+++	++
Lexicon extra ction	+++	++ (for short docs)	++

Proposed

Table 6.1: Summary of algorithms. Note that bag-of-words(BOW) was removed from the comparison table due to lack of generalizations for unseen words.

attempt to find “user vectors” by aggregating all reviews written by one user, and further use such vectors for user profiling, or calculating user similarities, and apply the algorithm to recommendation tasks.

With SSPV, we can try extending the dimension of word vectors with pre-trained vectors. Another method is to simply include unlabeled documents into the training set, and create a novel sampling method that can effectively take the implications of classes in to account.

Lexicon scoring can be improved by applying various normalization methods. For example, the similarity scores of each word for each class may be normalized, so that the sum of similarity scores are set to 1. Also, applying attention-based methods or adding information such as part-of-speech (POS) tags can be considered.

Bibliography

- [1] B. L. Whorf, *The relation of habitual thought and behavior to language*. 1956.
- [2] B. Pang and L. Lee, “Seeing stars: Exploiting class relationships for sentiment categorization with respect,” in *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05)*, (Morristown, NJ, USA), pp. 115–124, jun 2005.
- [3] B. Pang and L. Lee, “Opinion Mining and Sentiment Analysis,” *Foundations and Trends in Information Retrieval*, vol. 2, pp. 1–135, jan 2008.
- [4] C. N. dos Santos and M. Gatti, “Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts,” in *Proceedings of the 25th International Conference on Computational Linguistics (COLING’14)*, pp. 69–78, 2014.
- [5] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, “Skip-Thought Vectors,” in *Advances in Neural Information Processing Systems (NIPS’15)*, no. 786, pp. 1–11, 2015.

- [6] X. Zhang, J. Zhao, and Y. LeCun, “Character-level Convolutional Networks for Text Classification,” in *Advances in Neural Information Processing Systems (NIPS’15)*, pp. 1–9, sep 2015.
- [7] M. Taddy, “Document Classification by Inversion of Distributed Language Representations,” in *Proceedings of the 53rd meeting of the Association for Computational Linguistics (ACL’15)*, pp. 45–49, apr 2015.
- [8] R. Johnson and T. Zhang, “Effective Use of Word Order for Text Categorization with Convolutional Neural Networks,” in *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL’15)*, pp. 103–112, 2015.
- [9] T. Yano, N. A. Smith, and J. D. Wilkerson, “Textual predictors of bill survival in congressional committees,” in *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT ’12)*, pp. 793–802, Association for Computational Linguistics, jun 2012.
- [10] H. Lee, M. Surdeanu, B. Maccartney, and D. Jurafsky, “On the Importance of Text Analysis for Stock Price Prediction,” in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC’14)*, pp. 1170–1175, 2014.
- [11] H. Zhao, Z. Lu, and P. Poupart, “Self-Adaptive Hierarchical Sentence Model,” in *Proceedings of the 24th International Joint Conferences on Ar-*

- tificial Intelligence (IJCAI'15)*, pp. 4069–4076, 2015.
- [12] S. Kim, *Novel document representations based on labels and sequential information*. PhD thesis, 2015.
 - [13] Z. S. Harris, “Distributional structure,” *Word*, vol. 10, pp. 146–162, 1954.
 - [14] D. Greene and P. Cunningham, “Practical solutions to the problem of diagonal dominance in kernel document clustering,” in *Proceedings of the 23rd International Conference on Machine Learning (ICML'06)*, pp. 377–384, 2006.
 - [15] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, “From Word Embeddings To Document Distances,” in *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, vol. 37, pp. 957–966, 2015.
 - [16] J. Park, B.-C. Choi, and K. Kim, “A vector space approach to tag cloud similarity ranking,” *Information Processing Letters*, vol. 110, no. 12, pp. 489–496, 2010.
 - [17] T. K. Landauer and S. T. Dumais, “A solution to Plato’s problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge.,” 1997.
 - [18] G. Lapesa, S. Evert, and S. Schulte Im Walde, “Contrasting Syntagmatic and Paradigmatic Relations: Insights from Distributional Semantic Mod-

- els,” *In Proceedings of the Third Joint Conference on Lexical and Computational Semantics SEM*, Dublin, Ireland, August -, pp. 160–170, 2014.
- [19] Y. Bengio, R. Ducharme, and P. Vincent, “A neural probabilistic language model,” in *NIPS*, vol. 3, pp. 1137–1155, 2001.
- [20] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, “A neural probabilistic language model,” *The Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.
- [21] A. Mnih and G. Hinton, “Three new graphical models for statistical language modelling,” in *Proceedings of the 24th International Conference on Machine Learning (ICML’07)*, vol. 62, pp. 641–648, 2007.
- [22] A. Mnih, Z. Yuecheng, and G. Hinton, “Improving a statistical language model through non-linear prediction,” *Neurocomputing*, vol. 72, no. 7-9, pp. 1414–1418, 2009.
- [23] J. Turian, L. Ratinov, and Y. Bengio, “Word representations: A simple and general method for semi-supervised learning,” in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL’10)*, pp. 384–394, 2010.
- [24] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur, “Recurrent Neural Network based Language Model,” *Interspeech*, no. September, pp. 1045–1048, 2010.

- [25] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural Language Processing (Almost) from Scratch,” *Journal of Machine Learning Research (JMLR)*, vol. 12, pp. 2493–2537, feb 2011.
- [26] R. Socher, J. Pennington, and E. Huang, “Semi-supervised recursive autoencoders for predicting sentiment distributions,” *Conference on Empirical Methods in Natural Language Processing, EMNLP*, no. i, pp. 151–161, 2011.
- [27] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” in *Advances in Neural Information Processing Systems (NIPS’13)*, pp. 3111—3119, 2013.
- [28] X. Rong, “word2vec Parameter Learning Explained,” *arXiv:1411.2738*, pp. 1–19, 2014.
- [29] O. Levy and Y. Goldberg, “Dependency-Based Word Embeddings,” *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 302–308, 2014.
- [30] W. Ling, L. C.-c. Yulia, T. Silvio, A. Chris, D. Alan, and W. B. Isabel, “Not All Contexts Are Created Equal : Better Word Representations with Variable Attention,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP’14)*, 2014.

- [31] D. Yogatama, M. Faruqui, C. Dyer, and N. a. Smith, “Learning Word Representations with Hierarchical Sparse Coding,” *Icml 2015*, 2015.
- [32] J. Reisinger and R. J. Mooney, “Multi-prototype vector-space models of word meaning,” *NAACL HLT 2010 - Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Proceedings of the Main Conference*, pp. 109–117, 2010.
- [33] E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng, “Improving Word Representations via Global Context and Multiple Word Prototypes,” 2012.
- [34] A. Neelakantan, J. Shankar, A. Passos, and A. McCallum, “Efficient Non-parametric Estimation of Multiple Embeddings per Word in Vector Space,” *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP’14)*, 2014.
- [35] L. Vilnis and A. McCallum, “Word Representations via Gaussian Embedding,” *Iclr*, p. 12, 2015.
- [36] W. Yin and H. Schuetze, “Learning Word Meta-Embeddings by Using Ensembles of Embedding Sets,” *arXiv: 1508.04257*, 2015.
- [37] O. Levy, Y. Goldberg, and I. Ramat-Gan, “Linguistic regularities in sparse and explicit word representations,” *CoNLL-2014*, pp. 171–180, 2014.

- [38] O. Levy, Y. Goldberg, and I. Dagan, “Improving Distributional Similarity with Lessons Learned from Word Embeddings,” *Transactions of the Association for Computational Linguistics*, vol. 3, pp. 211–225, 2015.
- [39] S. Wang and C. D. C. Manning, “Baselines and bigrams: Simple, good sentiment and topic classification,” in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2 (ACL’12)*, vol. 94305, pp. 90–94, 2012.
- [40] LDC, “Web 1t 5-gram version 1,” 2006.
- [41] B. Schölkopf, J. Weston, E. Eskin, C. Leslie, and W. S. Noble, “A kernel approach for learning from almost orthogonal patterns,” in *European Conference on Machine Learning*, pp. 511–528, Springer, 2002.
- [42] J. Mitchell and M. Lapata, “Composition in Distributional Models of Semantics,” *Cognitive Science*, vol. 34, no. 8, pp. 1388–1429, 2010.
- [43] A. Yessenalina and C. Cardie, “Compositional Matrix-Space Models for Sentiment Analysis,” *Computational Linguistics*, pp. 172–182, 2011.
- [44] C. De Boom, S. V. Canneyt, S. Bohez, T. Demeester, and B. Dhoedt, “Learning Semantic Similarity for Very Short Texts,” *arXiv:1512.00765*, 2015.

- [45] J. Weston and K. Adams, “#TagSpace: Semantic Embeddings from Hash-tags,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP’14)*, pp. 1822–1827, 2014.
- [46] Z. Cui, X. Shi, and Y. Chen, “Sentiment Analysis via Integrating Distributed Representations of Variable-length Word Sequence,” *Neurocomputing*, 2015.
- [47] B. Coecke, M. Sadrzadeh, and S. Clark, “Mathematical Foundations for a Compositional Distributional Model of Meaning,” *Linguistic Analysis*, vol. 36, no. 1-4, pp. 345–384, 2011.
- [48] E. Grefenstette, G. Dinu, Y.-Z. Zhang, M. Sadrzadeh, and M. Baroni, “Multi-Step Regression Learning for Compositional Distributional Semantics,” *Proceedings of the 10th International Conference on Computational Semantics (IWCS’13)*, no. 2010, p. 10, 2013.
- [49] Q. Le and T. Mikolov, “Distributed Representations of Sentences and Documents,” in *Proceedings of the 31st International Conference on Machine Learning (ICML’14)*, vol. 32, pp. 1188–1196, 2014.
- [50] A. M. Dai, C. Olah, and Q. V. Le, “Document Embedding with Paragraph Vectors,” in *Advances in Neural Information Processing Systems (NIPS’14)*, pp. 1–8, 2014.
- [51] D. S. Sachan and S. Kumar, “Class Vectors: Embedding representation of Document Classes,” *arXiv:1508.00189*, 2015.

- [52] S. Lai, L. Xu, K. Liu, and J. Zhao, “Recurrent Convolutional Neural Networks for Text Classification,” *Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 2267–2273, 2015.
- [53] O. Irsoy and C. Cardie, “Opinion Mining with Deep Recurrent Neural Networks,” *Emnlp-2014*, pp. 720–728, 2014.
- [54] Y. Kim, “Convolutional Neural Networks for Sentence Classification,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP’14)*, pp. 1746–1751, 2014.
- [55] V. Hatzivassiloglou and K. R. McKeown, “Predicting the semantic orientation of adjectives,” in *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pp. 174–181, Association for Computational Linguistics, 1997.
- [56] P. D. Turney, “Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews,” 2002.
- [57] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, “Learning word vectors for sentiment analysis,” in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL HLT’11)*, pp. 142–150, Association for Computational Linguistics, jun 2011.
- [58] J. McAuley, R. Pandey, and J. Leskovec, “Inferring Networks of Substitutable and Complementary Products,” in *Proceedings of the 21st*

ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'15), p. 12, 2015.

- [59] L. V. D. Maaten and G. Hinton, “Visualizing Data using t-SNE,” *Journal of Machine Learning Research (JMLR)*, vol. 9, pp. 2579–2605, 2008.
- [60] L. V. D. Maaten, “Accelerating t-SNE using Tree-Based Algorithms,” *Journal of Machine Learning Research (JMLR)*, vol. 15, pp. 1–21, 2014.

국문초록

전통적으로 문서의 표현을 찾는 데는 bag-of-words (BOW) 방법론이 주로 사용되었다. 그러나 BOW는 고차원이고, 데이터 자체가 성기다는 단점이 있어 최근에는 분산 표현(distributed representation)이라는 더 저차원의 조밀한 표현을 찾는 방법들이 조망 받고 있다. 그 중 하나는 paragraph vector인데, 기존의 word2vec 알고리즘에 문단을 별개의 단어로 간주해서 추가하여 문서의 표현법까지 찾는 방법이다. 하지만 이런 표현법은 모든 문제에 대해 단 하나의 표현만 찾는다는 문제가 있다. 이 논문에서는 클래스 레이블 정보를 추가하여 supervised paragraph vectors(SPV)를 제안하는 것으로 시작한다. SPV는 기존의 단어, 문서와 더불어 클래스 레이블까지 학습하여 해당 문제에 더 적합한 표현을 찾을 수 있다는 장점을 가진다. 이후 SPV는 semi-supervised paragraph vectors(SSPV)로 확장되어, 클래스 레이블이 존재하지 않는 문서까지 학습에 활용하여 문서 분류 정확도를 높이려 한다. 마지막으로, 짧은 문서에 대한 문서 분류 정확도를 높기 위해 본 논문에서는 SPV를 활용한センチメント 사전 점수 부여 방법론을 제안한다. 제안한 방법론들의 효과를 증명하기 위해 해석용이성(interpretability), 분리효과성(discriminative power), 그리고 계산효율성(computational efficiency) 등 세 가지의 “좋은 표현법”의 기준을 정립하였으며, 해당 기준들에 입각하여 제안한 방법론들의 효과를 측정하였다.

주요어: 데이터마이닝, 텍스트마이닝, 문서 분류, 분산 표상, 표상 학습

학번: 2011-30314