Learning Class-specific Word Embeddings

Sicong Kuang   Brian D. Davison
Lehigh University, Bethlehem PA, USA
{sik211, davison} @cse.lehigh.edu

Bethlehem, Pennsylvania, United States of America

Abstract

Recent years have seen the success of applying word embedding algorithms to natural language processing (NLP) tasks. Most word embedding algorithms only produce a single embedding per word. This makes the learned embeddings indiscriminative since many words are polysemous. Some prior work utilizes the context in which the word resides to learn multiple word embeddings. However, context-based solutions are problematic for short texts, such as tweets, which have limited context. Moreover, existing approaches tend to enumerate all possible context types of a particular word regardless of their target applications. Applying multiple vector representations per word in NLP tasks can be computationally expensive because all possible combinations of senses of words in a snippet need to be considered.

Sometimes a word sense can be captured when the class information or label of the short text is presented. For example, in a disaster-related dataset, when a text snippet is labeled as "hurricane related", the word "water" in the snippet is more likely to be interpreted as rain and flood; when a snippet is labeled as "hurricane-unrelated", the word "water" can be interpreted as its more general meaning. In this work we propose to use class information to enhance the discriminativeness of words. Instead of enumerating all potential senses per word in the text, the number of vector representations per word should be a function of the future classification task. We show that learning the number of vector representations per word according to the number of classes in the classification task is often sufficient to clarify the polysemy.

Word embeddings learned from neural language models typically have the property of good linear compositionality. We utilize this property to encode class information into the vector representation of a word. We explore four approaches to train class-specific embeddings to encode class information by utilizing the label information and the linear compositionality property of word embeddings. We present a general framework consisting of a pair of convolutional neural networks to utilize the learned class-specific word embeddings as input for text classification tasks. We evaluate our approach and framework on topic classification of a disaster-focused Twitter dataset and a benchmark Twitter sentiment classification dataset from SemEval 2013. Our results show a relative accuracy improvement of 3-4% over a recent baseline.

Keywords: Word embeddings, Text classification, Polysemy

---

1. Introduction

Language is symbolic and discrete. To represent a word in human language in a form for machines to understand has always been a challenge in natural language processing (NLP). In the early (and simple) one-hot encoding approach, the vector representation of a word has the same length as the size of the vocabulary, thus naturally resulting in a sparse, high-dimensional word representation. Such a word representation approach cannot reflect the similarity or relatedness between words. The vector space model (VSM) of semantics addresses the shortcomings of the one-hot encoding approach by learning from the co-occurrence statistics from the word's context [1]. The center assumption here is the distributional hypothesis: the context surrounding a given word provides important information about its meaning [2]. The words' vector representations are constructed from the distributional patterns of co-occurrence with their neighboring words. In recent years, word vector representations learned from word embedding algorithms have demonstrated improvements both as inputs to other learning algorithms and as word features in NLP tasks, such as word similarity [3], part-of-speech tagging [4], named entity recognition (NER) [5], dependency parsing [6] and sentiment analysis [7]. Word embeddings typically learned from neural language models are well-known for capturing the semantics of words by learning dense low-dimensional vector representations [8, 3]. Since the introduction of the first efficient and effective word embedding algorithm by Mikolov et al. in 2013, multiple word embedding algorithms have been suggested, such as FastText and ELMo [9, 10]. However, the significant improvements have been made to unsupervised word embedding learning to generate universal embeddings. We show in this work, in addition to word co-occurrence patterns, short text labels can be a new source to provide semantics. Our work extends Mikolov et al.'s Word2Vec model.

There are two types of word embedding learning architectures in the Word2Vec model: the first one uses context words to predict the target word, such as the "continuous bag-of-words" (CBOW) model [3] and the "context-specific vector" (CSV) model [11]; the second one is to use the target word to predict the context words, such as the skip-gram model [3, 12]. These word embedding algorithms also follow the assumption that it is valuable to learn a word's meaning from its neighbors. Unlike VSM, which represents words from a co-occurrence matrix, word embedding algorithms represent words as dense vectors for input to a neural network model. The word embeddings are trained, taking the first type of word embedding algorithms for example, by maximizing the log likelihood of actual context versus random chosen context by using negative sampling [3].

Example 1	I decided to buy the **apple** without considering the others.
Example 2	This is the **case**.
Example 3	The **water** level is rising.

Table 1: Examples of Polysemous Words

However, despite the usefulness of word embeddings in NLP, most word embedding algorithms suffer from a significant drawback. That is, most models learn only a single embedding per word. The problem is that many words are polysemous (have multiple senses). For example, in Example 1 of Table 1, "apple" can be interpreted either as fruit or as computer brand; "case" in Example 2 of Table 1 is also ambiguous; "water" in Example 3 of Table 1 can be interpreted as referring to a flood or water in a sink or bathtub.[1] Thus in previous models such as the skip-gram and CBOW models, all the different meanings of a polysemous word will be combined into a single vector. In such a representation, quality of semantics will suffer.

Researchers try to solve the polysemy problem in word embedding algorithms mainly in two ways: the first is to process all the local contexts of a word in the corpus in a fine-grained manner and group contexts according to their semantic similarity [14, 15]; the second is to provide more information besides local contexts in the learning process to help interpret the sense of

[1]Example 3 in Table 1 is extracted from the disaster-focused Twitter corpus T6 [13] which we describe in Section 4.1.

the word [16, 17], such as an outside knowledge base [18, 19].

A short text snippet provides limited context. Moreover, a dataset of social communications, such as tweets, is full of newly-emerging words, acronyms and emojis, etc., which makes it hard to comprehend word meanings efficiently from context. We propose in this work to use label or class information as a type of context. We train the number-of-classes vector representations per word. We observe that the polysemy problem can be better managed when class information or label of the sentence is presented. Take Example 3 in Table 1 for example: in a disaster related classification task, when it is labeled as "hurricane related", the word "water" in the sentence is more likely to be correctly interpreted as rain and flood; when it is labeled as "hurricane-unrelated", the word "water" can be interpreted with its common meaning. Class information helps the system to interpret the correct sense of a word.

We adopt the linear compositionality property to encode the class or label information to learn class-specific word embeddings. Word embedding algorithms learned from neural language models typically have the property of good linear compositionality [3]. The linear compositionality property is best illustrated by the famous example

$$vector(\text{``}King\text{''}) - vector(\text{``}Man\text{''}) + vector(\text{``}Woman\text{''}) = vector(\text{``}Queen\text{''}).$$

We observed that vector("King")-vector("Man") results in a vector close to "Royalty". And the vector representation of "Queen" combines the semantic definitions of both "Royalty" and "Woman" through simple addition operation. Inspired by this observation, we generate class-specific word embeddings through the same operation.

A key problem remains as to how many vector representations per word should be learned to express the senses of a word. Existing models try to enumerate all possible senses of a word from the corpus while ignoring the application task of the trained embeddings. To apply multiple vector representations per word in future NLP tasks can be computationally expensive because all possible combinations of senses of words in a sentence need to be considered. For example, for a sentence of n words and l senses per word, we need to enumerate l^n sense combinations. Here we introduce the light polysemy problem; that is, instead of enumerating all potential senses of a word from the unlabeled corpus, we look to distinguish only a few vector representations per word as a function of the classification task. We present a framework which can input multiple vector representations per word for the classification task. Thus it runs in linear time. We only train the number of vector representations per word that is appropriate to the classification task. For example, in a disaster-related classification task, the task is to classify a sentence as “hurricane related” or “hurricane unrelated”; We train two embeddings per word: one is for “hurricane related” and the other one is for “hurricane unrelated”. Although a word like “water” might have more than two senses, we show in experiments that only two vector representations, one representing “hurricane related” context semantically close to rain and flood, and another one representing “hurricane unrelated” context which is trained from all non-hurricane related context, are able to capture enough sense information needed for the classification task. We call the embedding trained for each class per word, class-specific embedding. We show in the experiments that class-specific embeddings can address the light polysemy problem within the classification task.

In this work, we explore four approaches to learn class-specific word embeddings for classification using the linear compositionality property. We extend our prior work [20] by defining the light polysemy problem and additionally modifying the CBOW model to incorporate class information to learn class-specific word embeddings. We show in the experiments that class-specific word embedding is useful to address the light polysemy problem in classification tasks. We modified the skip-gram and CBOW models in Word2Vec [3] by introducing class information. For our classifier, we combined two convolutional neural network (CNN) models [21], which take the class-specific word embeddings from each class as input. Our contributions include:

1. Our work is the first to use the linear composition property to build class-specific embeddings.
2. We define the light polysemy problem in text classification tasks.
3. We propose the use of label information as global context to tackle the light polysemy problem.
4. We present a general framework consisting of two convolutional neural networks which take the class-specific word embeddings we trained as input for a binary text classification task.

We compare our approach with multiple baselines on a disaster-related Twitter dataset and a benchmark Twitter sentiment classification dataset from SemEval 2013.

2. Related Work

Many methods can obtain vector representation of words, such as Latent Semantic Analysis (LSA) [22] and Latent Dirichlet Allocation (LDA) [23]. Word embeddings trained by neural language models are well-known for their ability to represent words' general semantic meaning. More recently, Word2Vec, developed by Mikolov et al. [3], has been shown to provide a new state-of-the-art performance in NLP tasks. Many researchers have contributed to the neural language model-based word embedding literature [24, 7, 25, 26].

Most existing models only produce one vector representation per word, which is problematic for words with multiple meanings. A single word embedding does not address the polysemy problem. Several multiple-embedding models have been proposed to alleviate the problem caused by polysemy. Researchers typically address this issue by training multiple embeddings per word according to their multiple senses [27, 15]. Most existing work utilizes context-based models. That is, they learn various word embeddings per word by discriminating among distinguishable contexts in the corpus. Huang et al. [14] tackle this problem through k-means clustering. They heuristically pre-define k senses for each polysemous word and cluster all the local contexts of a word into k clusters. Local context is defined as 5 words before and after the target word. The local context limits the information we can use to learn to distinguish the word's sense, especially in a dataset consisting of short text snippets such as tweets. Twitter is known for having a short character limit.

Neelakantan et al. [15] further extend Huang et al.'s idea and apply a context-clustering schema on the skip-gram model. They notice that in Huang et al.'s work, the context-clustering schema is a pre-processing step; the context vectors are not updated in the learning process. They propose a joint model by concatenating the clustering algorithm and the skip gram model. Their approach clusters all the contexts the word has and finds the cluster centroid that is most close to the word's current context as its sense vector. Then the sense vector is sent to skip-gram model for learning and updating. The learned sense vector is updated as the new centroid for that cluster. Neelakantan et al.'s approach still suffers from the need to cluster contexts for every word, which makes training expensive.

Guo et al. [28] also propose a multiple embedding model. They combine the context-clustering schema with bilingual resources to learn multiple embeddings per word. Motivated by the intuition that the same word in the source language with different senses is supposed to have different translations in the foreign language, the authors obtain the senses of one word by clustering its translation words, exhibiting different senses in different clusters. Another bilingual word embedding (BWE) approach is proposed by Su et al. [29]. Different from traditional BWE approaches which either distinguish the correct bilingual alignments from the corrupted ones or model the joint bilingual probability, the authors introduce a latent variable to explicitly induce the underlying bilingual semantic space which generates word tokens in both two languages.

Pelevina et al. [30] generate multiple embeddings per word by clustering the related words in the ego-network. Similar to our approach, their method relies on existing single-prototype word embeddings, transforming them to sense vectors via ego-network clustering. An ego network consists of a single node (ego) together with the nodes they are connected to (alters) and all the edges among those alters. In their case, for each word w, they construct an ego network with word w as ego node and w's nearest neighbours calculated by vector similarities as other nodes with connections to word w. Then the authors use graph clustering method to obtain multiple senses for word w.

Other than context-clustering schema, other approaches have also proposed to generate multiple embeddings per word. The main idea is still to obtain distinguishable context vector representations through other learning models or outside expert annotators. Zheng et al. developed a convolutional neural network to learn a new sense vector for a word if the cosine similarity between the new context vector and every existing sense vector is less than a threshold [11]. Tian et al. extended the skip-gram model from Mikolov's work and generated multiple vector representations for each word in a probabilistic manner [12]. They added an item specifying the probability of the sense of the given word to the original skip-gram objective function and used the Expectation Maximization algorithm to train multi-sense vectors. Chen et al. rely on WordNet glosses, which have summarized each word's senses, to initialize multiple embeddings per word and update the multiple embeddings per word through a skip-gram model [19]. Instead of figuring out how many latent senses a word may have, Bollegala et al. [31] take a different path by directly learning the k-way co-occurrences embeddings. Most of the successful word embedding models, such as Word2Vec [3] and Glove [32], depend on word co-occurrences when $k = 2$. Bollegala et al. extend to the situation when $k \geq 2$; treat every context of size k as a bag-of-tokens and learn a vector representation for every context. Scheepers et al. [33] improve the semantics represented in the word embedding by using outside lexicographic definitions. All the context-based approaches suffers from same weakness. That is to learn all the distinguishable contexts to discriminate word senses regardless of the future application for the embedding and the computational cost.

Unlike the above word-level construction of word embeddings, some research work focuses on morphology, that is, the sub-word level, to learn multiple embeddings per word. Bojanowski et al. also extended the skip-gram model [34], targeting the morphology of words. Unlike previous approaches which train a single word vector for each word and ignore the internal structure of words, they modified the skip-gram model to represent each word as a bag of character n-grams. Each character n-gram is trained to associate with a vector representation. The vector for the word is the sum of the n-grams' vectors. Athiwaratkun, Wilson and Anandkumar [35] combine Bojanowski et al.'s FastText with a Gaussian mixture model [36]. They initialize each word with a hyper-parameter of k Gaussian components. Each Gaussian component represents a different sense of a word.

The polysemy problem not only exists in words but also in entity disambiguation. Chen et al. try to solve the challenging task of finding the correct referential entity in a knowledge base (KB) [37]. The authors learn word and entity embeddings by training a bilinear joint learning model. Their embedding learning model is the same as the skip-gram model. The only difference is that they propose a bilinear model to learn the semantic gap (a projection) between word embedding and entity embedding.

Our approaches utilizes class labels as resource to comprehend word's sense. We think that a word's sense such as "water" can be better interpreted with the aid of class information as global context. For example, when the tweet "The water level is rising" is labeled as "hurricane-related tweet", we would know that the "water" here means flood. We introduce the light polysemy problem: instead of making efforts to enumerate all potential senses per word from the unlabeled text, the number of vector representations per word should be closely related to the number of classes in the future task.

In this work we address the light polysemy problem by utilizing the linear compositionality property in four approaches. In the first approach, we build separate word embeddings using data filtered by class label and feed the embeddings into the classification framework for a single class label prediction. In the second approach, we build class-specific word embeddings by directly

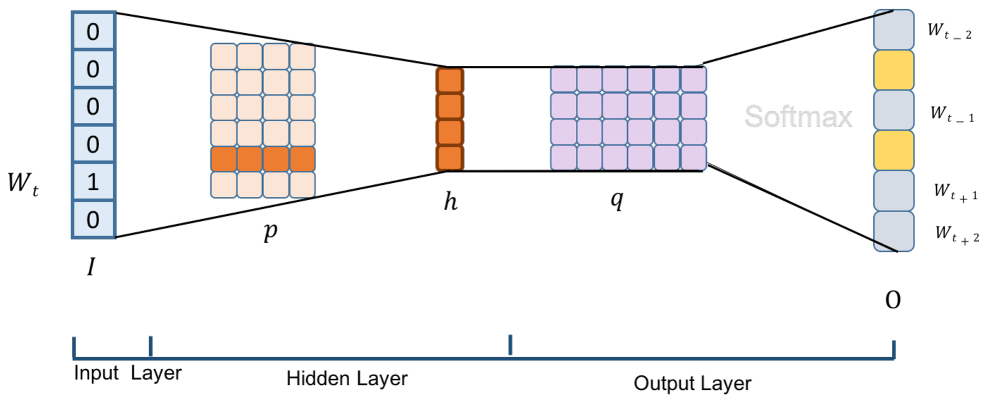


Figure 1: Skip-gram model architecture with word embedding dimension $n = 4$, vocabulary size $|V| = 6$, window size 5 ($c = 2$). The input layer is a one-hot encoding $I \in |V| \times 1$ denoting the target word in the context window. In the hidden layer, after multiplying I with the vocabulary matrix $\mathcal{P} \in \mathbb{R}^{|V| \times n}$, the resulting vector is $h \in n \times 1$. After multiplying h with the output weight matrix $q \in n \times |V|$ in the output layer and sending the result vector to softmax function, a vector of probabilities $O \in |V| \times 1$ in the output layer specifies the likelihood of each word to appear in the context window.

adding the vector representation of the classification polarity to the vector representing the general meaning of the word. In the third approach, we modify the skip-gram architecture to train a class-specific word embedding. In the fourth approach, we modify the CBOW model architecture to train a class-specific word embedding.

3. Methodology

In this section, we introduce the details of generating class-specific word embeddings. We incorporate class information into word embeddings by utilizing the linear compositionality property shown by the word embeddings learned from neural network based language models [3, 32]. Our work directly extends the Word2Vec model architecture [3]. In the following sections, we present three model architectures to generate class-specific word embeddings. We then describe the use of the class-specific word embeddings in a framework consisting of convolutional neural networks for text classification.

3.1. Skip-gram Model

Mikolov et al. [3] introduce the skip-gram model, which learns continuous vector representations of words from the context in which the word resides.

Figure 1 shows the skip-gram model's architecture. The model takes word w_t as input and predicts the c words ahead of and behind w_t by maximizing the log likelihood function:

$$\mathcal{L} = \sum_{w_t \in \mathcal{C}} \texttt{log}\, p(Context(w_t) \mid w_t)) \tag{1}$$

Where $\mathcal{C}$ is the corpus, the function tries to maximize the conditional probability of words appearing within a certain range of w_t given the target word w_t. To address computational complexity, Mikolov et al. adopts hierarchical softmax and negative sampling [3] to implement the skip-gram model. In this paper, we address the skip-gram model trained with hierarchical softmax in the advanced model I. The vocabulary in the skip-gram model with hierarchical softmax is initialized as a Huffman tree.

Besides the architecture difference in the skip-gram model and the CBOW model, the skip-gram model generally performs better in semantic tests [3] in terms of accuracy though more slowly than CBOW. Compared to CBOW, skip-gram model trains over more data since each word in the corpus can be a training tuple.

3.2. Continuous Bag-of-words Model (CBOW)

Another neural language based model is the CBOW model, which is also introduced by Mikolov et al. [3]. Figure 2 shows the architecture of the CBOW model. It consists of three layers: an input layer, a projection layer (also known as a hidden layer) and an output layer. Unlike skip-gram, the CBOW model predicts the target word given the context words both c words preceding and following the target word; in the input layer, the vocabulary is represented as an input vocabulary matrix $\mathcal{P} \in \mathbb{R}^{|V| \times n}$; each column in $\mathcal{P}$ is the vector representation of a word in the vocabulary; $\mathcal{P}$ is randomly initialized from the uniform distribution in the range $[-1, 1]$. In the hidden layer, the vector representation of the context, $\texttt{g}$, is calculated as the arithmetic mean of the vector representation of all words h_i in the context window with c words before and after the target word, as shown in Formula 2.

$$\texttt{g} = \frac{1}{2c} \sum_{i \in [-c,-1] \cup [1,c]} w_i \tag{2}$$

$p(w_t|\texttt{g})$ is used to calculate the probability of the target word given context vector representation $\texttt{g}$ as shown in Formula 3, which is represented as a

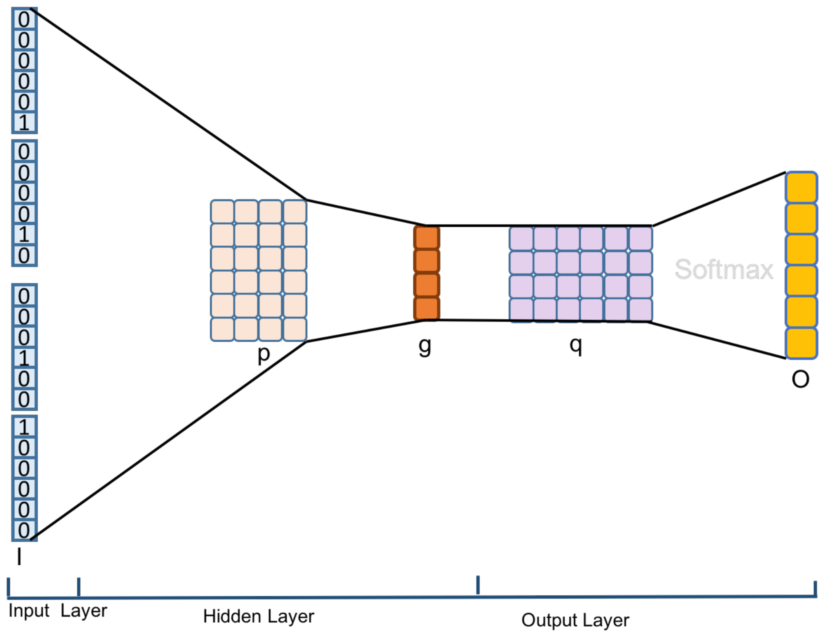


Figure 2: CBOW model architecture with word embedding dimension $n = 4$, vocabulary size $|V| = 6$ and window size 5 ($c = 2$). It consists of three layers: input layer, hidden layer and output layer. In the input layer, each $I_i \in |V| \times 1$ is a one-hot encoding vector of a context word in the context window surrounding the target word; in the hidden layer, each one-hot encoding vector I_i^T multiplied against the vocabulary matrix $\mathcal{P} \in \mathbb{R}^{|V| \times n}$ to select the matrix row that represents the context word; $\mathsf{g} \in n \times 1$ is the average of the context word vectors. After multiplying g with the output weight matrix $q \in n \times |V|$ and sending the result to a softmax function, a vector of probabilities $O \in |V| \times 1$ in the output layer specifies the likelihood of each word to be the target word in the context window.

softmax function over the dot product of the vector representation of the context $\mathtt{g}$ and target word w_t.

$$p(w_t|\mathtt{g}) = \frac{e^{w_t \cdot \mathtt{g}}}{\sum\limits_{w_i \in Vocab} e^{w_i \cdot \mathtt{g}}} \tag{3}$$

Finally we can depict the loss function of the CBOW model in Formula 4.

$$\mathcal{L} = \sum_{w_t \in \mathbb{C}} \mathtt{log}\ p(w_t|\mathtt{g}) \tag{4}$$

where over all training tuples in the corpus $\mathbb{C}$, we are maximizing the probability of finding the target word w_t given $\mathtt{g}$, its context. However, to go over all the words in the vocabulary in Formula 3 is expensive. Instead of comparing all words in the vocabulary, to only distinguish the target word from several noise words largely reduces the computation load. This is called negative sampling. The window size $2c + 1$ and the word embedding dimension n are all hyperparameters. In the advanced model II, we modified the CBOW model to train class-specific word embedding.

3.3. Class-Specific Word Embedding

As described earlier, the standard word embedding approach is problematic for words with multiple senses. In this section, we describe our proposed models and frameworks based on the linear compositionality property of modern word embeddings.

3.3.1. Linear compositionality property

Word embeddings learned from the skip-gram model show good linear compositionality [3, 38]. A famous example would be that

$$vector(\text{“}King\text{”}) - vector(\text{“}Man\text{”}) + vector(\text{“}Woman''))$$

results in a vector which is the closest to the vector representation of the word “Queen” [3]. One interpretation is that the operation of $vector(\text{“}King'') - vector(\text{“}Man'')$ results in a vector which is close to the semantic definition of “Royalty”; thus $vector(\text{“}King'') - vector(\text{“}Man'') = vector(\text{“}Royalty'')$; then we add $vector(\text{“}Royalty'')$ to $vector(\text{“}Woman'')$, we get the semantic information from both words, which is $vector(\text{“}Queen'')$. Based on this observation, the semantic information in the vector representation trained from a neural language model satisfies linear composition. Thus the vector representation of a word, such as “Queen”, which combines the semantic meaning of “Woman” and “Royalty”, could be obtained directly through the addition operation. We observe that the new vector representation combines the semantic definitions of both sides. In our work, we use this linear compositionality property to encode the class information into the word embeddings by adding a vector that represents the class information.

3.3.2. Vector Representation of Class

Based on the linear composition property, we propose to obtain the class-specific word embedding by adding the vector representation of the class to the vector that represents the general meaning of the word. In this section, we describe how we find the vector representation of the class information.

In the procedure to compute the vector representation of class, our first step is to manually define the classification polarity of the task. Some classification tasks have one polarity while others have two or more polarities. For example, in a basic sentiment analysis task, there are typically two polarities, namely positive and negative; in a task to classify hurricane related Tweets from general Tweets stream, there is only one polarity, namely hurricane. Because for tweets that are labeled as hurricane-unrelated, we treat them as ordinary tweets which have no semantic polarity inside the sentences in terms of this task. In the second step, we manually select the word that is most representative of classification polarity of the task. We define the word as polarity word, such as “hurricane”. In this work, the polarity words are defined with the help of the label information of the dataset. For example, in a disaster-related dataset, the labeling task of the dataset is to classify a sentence as hurricane-related or hurricane-unrelated. Thus we manually choose the polarity word to be hurricane. If a dataset is used for sentiment analysis, now the labeling task of the dataset becomes labeling positive sentences and negative sentences. By briefly examining the labeled dataset, we found that the polarity word good can be used for positive class and the polarity word bad can be chosen for negative class. It is true that we generally need two steps to manually decide the polarity word. One is to know the labeling task of the dataset. The other one is to manually decide the polarity words.

In the third step, we adopt a heuristic approach to find the vector representations of the classification polarities. We first use the original skip-gram model from Word2Vec on our dataset to obtain class-independent word embeddings, providing a vector representation for each word in our vocabulary.

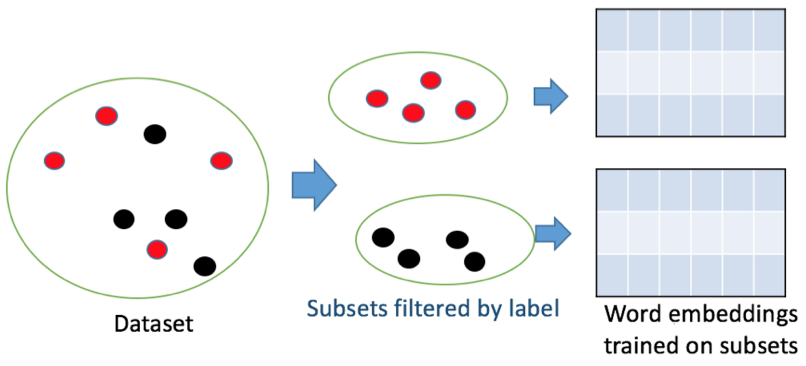


Figure 3: Diagram of Basic Approach I.

From the class-independent word embeddings we retrieve the polarity words' vector representations. Next, for each polarity word we use cosine similarity to select the top n words' vector representations that are most similar to the polarity word's vector representation from the vocabulary of the dataset:

$$\texttt{similarity\_score} = \frac{\texttt{vector}(w_{polarity}) \cdot \texttt{vector}(w)}{\|\texttt{vector}(w_{polarity})\| \| \texttt{vector}(w)\|} \tag{5}$$

where w is a word in the vocabulary; $w_{polarity}$ is the polarity word. According to the similarity score, we choose n $\texttt{vector}(w)$ which have highest similarity scores. Then we calculate the arithmetic mean of the top n $\texttt{vector}(w)$ as the vector representation of the class:

$$\mathcal{V}(class) = \frac{1}{n}(\texttt{vector}(w_1) + \texttt{vector}(w_2) + \cdots \texttt{vector}(w_n)) \tag{6}$$

where $vector(\cdot)$ denotes the embedding's vector representation of a word; $\mathcal{V}(\cdot)$ denotes the vector representation of class information. In our work n is 100. Here n is a hyper-parameter that we set to 100. The number n is to make sure that the vector representation of the class is general enough to be representative when we calculate the arithmetic mean of the top n vectors.

3.3.3. Basic Approach I

In basic approach I, we do not use the vector representation of class to build class-specific word embeddings. Our idea is simple as shown in Figure 3: we first divide the training set into subsets according to the class label. For example, in a sentiment analysis dataset, the training set is divided into two

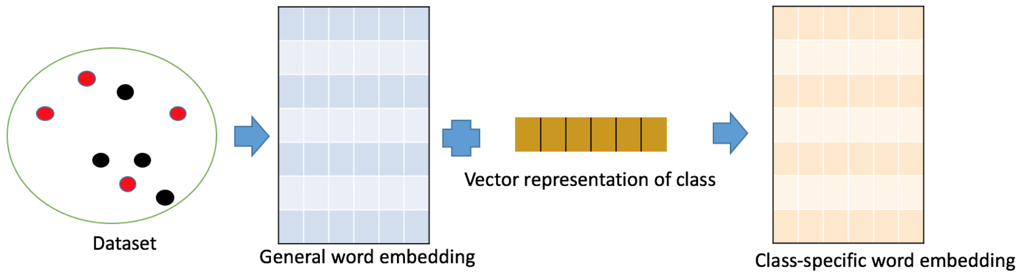


Figure 4: Diagram of Basic approach II.

subsets according to class label, namely positive and negative; next we train a skip-gram model over the data in each subset to generate a particular set of word embeddings for a specific class. In the sentiment analysis example, we generate one set of word embeddings on the positive set and we generate another set of word embeddings on the negative set. So for each class, we have a separate set of word embeddings. We then apply the two sets of word embeddings to our parallel CNN classification framework.

This approach is designed to allow us to test the effectiveness of the linear compositionality property. We expect that on the same dataset training the embedding without utilizing the linear compositionality property would dampen the classification framework's performance. On the other hand, it is the simplest of the four proposed approaches. We separate the dataset into subsets, and build a Word2Vec model for each subset. For disadvantages, we reduce the dataset to subsets separated by class labels. In this approach, we do not really use the class label information to build word vector representations. A larger dataset results in more training data and thus leads to higher accuracy; similarly, a smaller dataset results in less training data and leads to lower accuracy [39].

3.3.4. Basic Approach II

Considered an unsupervised approach, the skip-gram model does not utilize class information to learn word embeddings. For a binary classification task, we aim to train two vector representations for each word; one for each class. In basic approach II, we use linear composition to encode the class information into general word embeddings.

In basic approach II, we integrate the class information into the general word embedding by directly adding the vector representation of the classification polarities to the vector representing the general meaning of the word based on the linear compositionality property. For example, in a task to classify hurricane related tweets from a general tweet stream, we have elements of the training dataset labeled as “hurricane-related” or “hurricane-unrelated”. Since there is only one polarity word “hurricane”, we obtain the vector representation of the class hurricane according to Section 3.3.2; then the class-specific word embedding is defined as:

$$h = \mathcal{V}(hurricane) + w \tag{7}$$

where h denotes the class-specific word embedding of word w; w denotes the vector representing the general meaning of word w trained from skip-gram model.

An advantage of this approach is that we use the linear compositionality property to build word vector representations for each class training on the whole dataset compared to Basic Approach I. For disadvantages, we calculate the vector representation of the class $\mathcal{V}(class)$. Then $\mathcal{V}(class)$ is added to the word’s general vector representation for each word appeared in that class. The linear shift for every word in that class might be a problem: some words have polysemy problems can be clarified in this process; some words that have no polysemy problems might be shifted away from its position in the latent semantic space. We next use non-linear models to solve the polysemy problem.

3.3.5. Advanced Model I

Based on the linear compositionality property of word embeddings trained on a neural language model, our basic approach II generates a class-specific word embedding by adding $\mathcal{V}(\cdot)$ directly, the vector representation of class information to w, the vector representation of the general meaning of word w_t. The main issue with the original skip-gram model is that only a single vector representation per word is not enough to tackle the polysemy problem. In the proposed advanced models, we utilize the neural language model to train class-specific word embeddings for each class in the corpus. In the advanced model I, instead of adding the class information vector linearly, we utilize skip-gram’s neural language model shown in Figure 5 to predict the context words’ embeddings from a class-specific word embedding of the target word.

Figure 5 shows the architecture of the advanced model I. For each class in the corpus, we use the approach introduced in Section 3.3.2 to represent

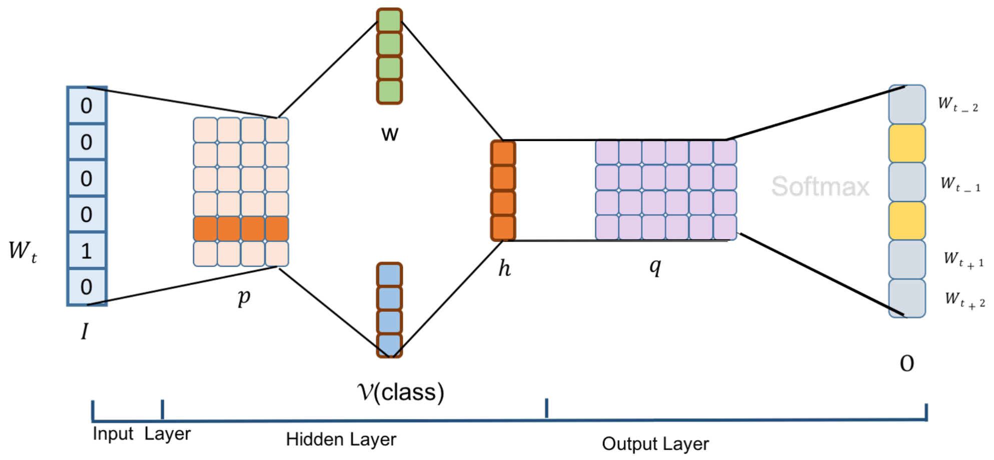


Figure 5: The architecture of advanced model I. Based on the linear compositionality property, the class-specific word embedding of the target word is obtained by adding directly $\mathcal{V}(class)$, the vector representation of class information to w, the vector representation of the general meaning of word w_t.

the class information in vector space denoted as $\mathcal{V}(class)$. We add $\mathcal{V}(class)$ to the general vector representation of the target word, w as shown in Equation 8. Based on the property of linear compositionality, the summation of $\mathcal{V}(class)$ and w should capture the semantic meaning from both sides. We use the class-specific word embedding of the target word to update the word embeddings of its context words in the modified skip-gram model.

$$h_{w,\mathcal{V}(class)} = w + \mathcal{V}(class) \tag{8}$$

Equation 9 is the objective function of the modified skip-gram model.

$$\mathcal{L} = \sum_{w \in \mathbb{C}} \texttt{log}\ p(Context(w) \mid h_{w,\mathcal{V}(class)}) \tag{9}$$

where over all training tuples in the corpus $\mathbb{C}$, we are maximizing the probability of finding the context words around w given w and its class information.

We adopt the hierarchical softmax based skip-gram model [3], which uses a binary Huffman tree to organize the words in the vocabulary. Each leaf in the Huffman tree represents a word. The path from root to leaf represents the Huffman encoding of the word. For each non-leaf node in the tree, a binary classifier produces a probability to decide which path to take. As in Mikolov et al.'s skip-gram model, we choose a logistic regression classifier for each non-leaf node. Thus the conditional probability in Equation 9 can be further written as:

$$p(Context(w) \mid w, \mathcal{V}(class)) = \prod_{j=2}^{l^w} p(d_j^w | h_{w,\mathcal{V}(class)}, \theta_{j-1}^w) \tag{10}$$

$$p(d_j^w | h_{w,\mathcal{V}(class)}, \theta_{j-1}^w) = \begin{cases} \sigma(h_{w,\mathcal{V}(class)}), & d_j^w = 0 \\ 1 - \sigma(h_{w,\mathcal{V}(class)}), & d_j^w = 1 \end{cases} \tag{11}$$

where $h_{w,\mathcal{V}(class)}$ denotes the output of the projection layer in the advanced model I, which is the summation of w and $\mathcal{V}(class)$; d_j^w is the binary Huffman code at jth node of word w; θ_{j-1}^w is the vector representation of the $(j-1)th$ non-leaf node of word w; l^w is the number of non-leaf nodes for word w; $\sigma(\cdot)$ is the sigmoid activation function of the logistic regression classifier at non-leaf nodes. We use SGD (Stochastic Gradient Descent) to maximize $\mathcal{L}$ and update $h_{w,\mathcal{V}(class)}$ and θ_{j-1}^w.

3.3.6. Advanced Model II

In advanced model II we modify the original CBOW model to train class-specific word embeddings. The original CBOW model uses the averaged word embeddings of the context words to predict the target word. Although CBOW model is demonstrated to capture semantic information in the single vector representation per word, it is problematic for polysemous words. We take advantage of the architecture of the CBOW model to train class-specific word embeddings to tackle the light polysemy problem existing in the classification task.

For each class in the corpus, we first use the approach introduced in Section 3.3.2 to represent the class information in vector space denoted as $\mathcal{V}(class)$. Instead of representing the context using the average of the vector representations of all the words in the context window, we learn class-specific context $\mathsf{g}_{\mathtt{class}}$ by adding the class vector $\mathcal{V}(class)$ to the context vector representation g as shown in Equation 12 and 13.

$$\mathsf{g} = \frac{1}{2c} \sum_{i \in [-c,-1] \cup [1,c]} w_i \tag{12}$$

$$\mathsf{g}_{\mathtt{class}} = \mathsf{g} + \mathcal{V}(class) \tag{13}$$

Figure 6 illustrates the architecture of the advanced model II. As opposed to the architecture of the original CBOW model in Figure 2, we generate

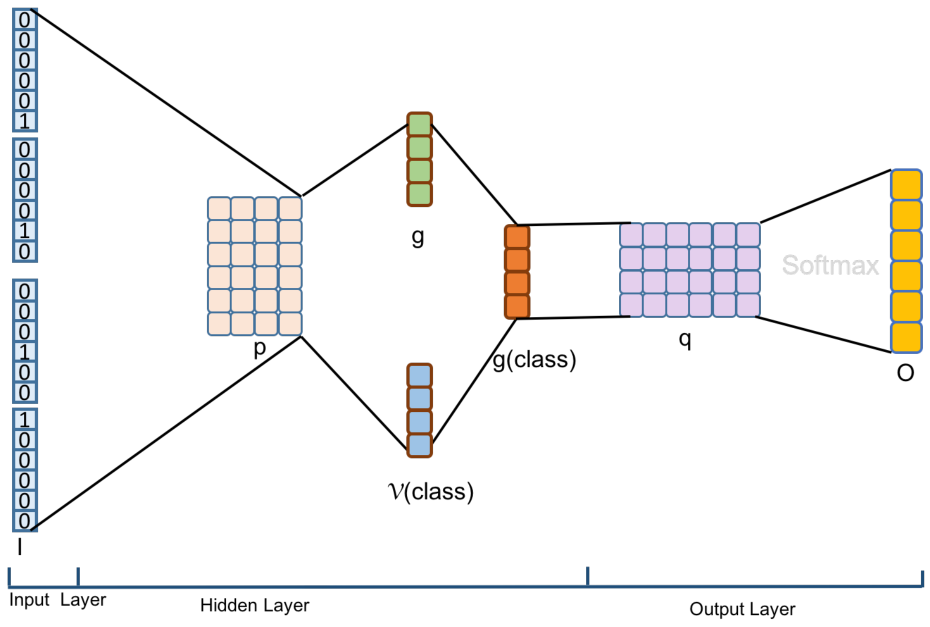


Figure 6: Architecture of Advanced model II. Based on the linear compositionality property of wording embedding algorithm, we modify the CBOW model by adding $\mathcal{V}(class)$ the class vector to the context vector representation $\mathtt{g}$. The result, class-specific context $\mathtt{g}_{\mathtt{class}}$, combines the local context $\mathtt{g}$ as well as global context, $\mathcal{V}(class)$.

class-specific context $\mathtt{g_{class}}$, which combines the local context $\mathtt{g}$ as well as global context, $\mathcal{V}(class)$ to tackle the light polysemy problem.

$$\mathcal{L} = \sum_{w \in \mathbb{C}} \mathtt{log}\, p(w|\mathtt{g_{class}}) \tag{14}$$

Equation 14 is the objective function of the advanced model II. The objective function tries to maximize the conditional probability of the target word given the class-specific context $\mathtt{g_{class}}$ for all the training tuples in corpus $\mathbb{C}$.

$$p(w \mid \mathtt{g_{class}}) = \prod_{j=2}^{l^w} p(d_j^w|\mathtt{g_{class}}, \theta_{j-1}^w) \tag{15}$$

$$p(d_j^w|\mathtt{g_{class}}, \theta_{j-1}^w) = \begin{cases} \sigma(\mathtt{g_{class}}), & d_j^w = 0 \\ 1 - \sigma(\mathtt{g_{class}}), & d_j^w = 1 \end{cases} \tag{16}$$

where $\mathtt{g}$ is the context representation; $\mathtt{g_{class}}$ denotes the output of the projection layer in the advanced model II, which is the summation of $\mathtt{g}$ and $\mathcal{V}(class)$; d_j^w is the binary Huffman code at jth node of $\mathtt{g}$; θ_{j-1}^w is the vector representation of the $(j-1)th$ non-leaf node of $\mathtt{g}$; l^w is the number of non-leaf nodes for word w; $\sigma(\cdot)$ is the sigmoid activation function of the logistic regression classifier at non-leaf nodes. We use SGD (Stochastic Gradient Descent) to maximize $\mathcal{L}$ and update $X_{\mathtt{g_{class}}}$ and θ_{j-1}^w.

In summary, compared with Basic Approach II, instead of adding the class information vector linearly, the advanced approaches utilize skip-gram and CBOWs neural language models to generate the class-specific word embeddings. The skip-gram model trains over more data since each word in the corpus can be a training tuple. Thus the skip-gram model favors small datasets. In our work, we use a labeled dataset to encode the class information into embeddings. Labeled datasets are usually smaller (because of the cost to acquire the labels), and thus an Advanced Model I that uses a skip-gram model has advantages over an Advanced Model II that uses a CBOW model.

3.4. Classification Framework

We apply class-specific word embedding for text classification under a supervised learning framework. Our framework extends Kim's work [21] which introduced the use of convolutional neural networks (CNN) for sentence classification. In Kim's work, the input is a sentence and for each word in the

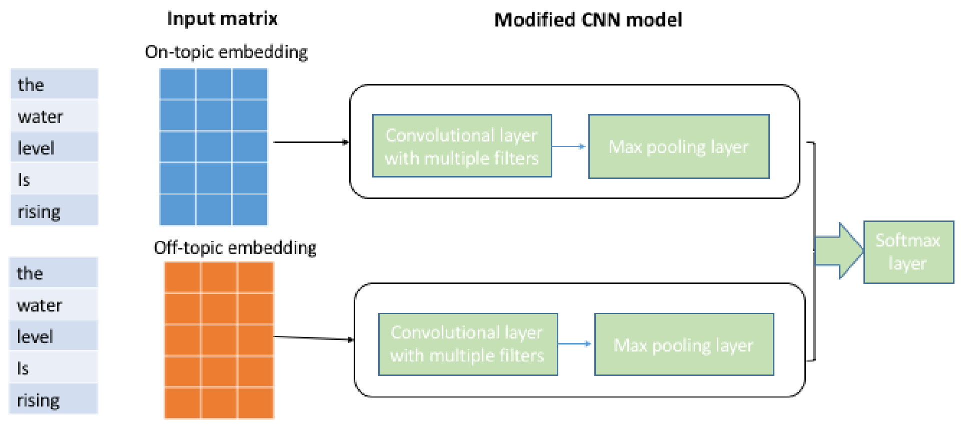


Figure 7: A general binary classification framework that takes two embeddings, namely on-topic embedding and off-topic embedding as input.

sentence, Kim's CNN takes one fixed length word embedding trained from Word2Vec. It consists of a convolutional layer with multiple filters in different widths, a max-pooling layer and a softmax output layer.

We aim to build a classifier framework which can take multiple sets of class-specific word embeddings we trained as inputs. Since for each test sentence its class label is not revealed yet, it is not known which word embedding, for example class-specific word embedding or general meaning word embedding, should be applied to a classifier. We design a classification framework that takes multiple sets of word embeddings as input. The number of word embeddings per word depends on the class polarities of the classification task. For example, for sentiment analysis we have two class polarities; thus we have two word embeddings per word: one embedding learned from positive class and the other embedding learned from negative class. For a topic-related classification task, such as a task to classify hurricane-related tweets, we also have two word embeddings per word: one on-topic embedding trained from hurricane-related tweets and one off-topic embedding trained from hurricane-unrelated tweets.

In a multi-class text classification problem, for each word, we generate one embedding per class using the proposed approaches; instead of a classification framework takes exactly two embeddings per word, we would need to build a text classification framework that takes in the number of embeddings per word corresponding to the number of classes in the dataset. If we consider binary text classification, the proposed classification framework is illustrated in Figure 7. We combine two CNNs with a softmax layer which takes concatenated feature vectors from the two max pooling layers and outputs the probability distribution over class labels.

4. Experiment

We conduct experiments to evaluate the proposed four approaches to learn class-specific word embeddings. We apply class-specific word embeddings to the supervised classification framework described in Section 3.4.

4.1. Experiment Setup and Datasets

We conduct experiments on two publicly available datasets. The first dataset is a disaster-related Twitter dataset [13], called T6. T6 is labeled by crowdsourcing workers according to disaster relatedness (as “on-topic”, or “off-topic”) [13]. T6 contains 6 crisis events in 2012 and 2013. We choose to test our approach on the hurricane Sandy dataset. The statistics of the hurricane Sandy dataset is shown in Table 2. The other dataset is the benchmark Twitter sentiment classification dataset in SemEval 2013[2]. Each tuple in the SemEval dataset has three class label options: positive, negative and neutral. Since we focus on the binary text classification task and we aim to use the same classification framework for both datasets, we filter out the tuples in the SemEval 2013 dataset which are labeled as neutral. We also do a pre-processing step: we first eliminate all tweets in the two datasets that are non-English, and then we eliminate tweets that contain fewer than five words. Our pre-processing step is in line with Olteanu et al.'s work on the same dataset [13]. For the parameters of our experiments, we choose a window size of 5 and word embedding dimension of 50. To reduce the randomness and the stochasticity in the experiments, we conduct each experiment 30 times and report the mean results of the 30 runs for each experiment. The characteristics of the SemEval 2013 dataset are shown in Table 3.

4.2. Baseline Methods

To compare the quality of the class-specific word embedding, we implement the following baselines:

[2]https://www.cs.york.ac.uk/semeval-2013/

Data	**On-topic**	**Off-topic**	**Total**
Train	4911	3098	8009
Test	1227	772	1999

Table 2: Hurricane Sandy dataset characteristics

Data	**Positive**	**Negative**	**Total**
Train	2256	849	3104
Test	330	172	502

Table 3: SemEval 2013 dataset characteristics

1. Sentiment-specific word embedding (SSWE): Tang et al. [7] introduce a supervised method to learn sentiment-specific word embeddings based on Collobert et al.'s unsupervised approach [24]. We build a word embedding according to Tang's method and test the embedding on our classification framework. We use Attardi's NLP pipeline to generate this baseline [40].
2. Word embeddings trained using the skip-gram model: we train our own embedding using Word2Vec's original skip-gram model [3]. We apply the word embeddings as features of a convolutional neural network [21]. A single embedding per word is trained on all training data without use of the training labels.

Tang et al.'s approach [7] is the research work that is closest to our own. Although their method is to generate a sentiment-specific embedding, we found their method could be extended to any labeled dataset that has contrasting polarities.

4.3. Results and Analysis

Table 4 shows the results of the experiments on different approaches. We choose convolutional neural networks over other classifiers. The reasons are: since our proposed classification framework consists of two CNNs, it is reasonable to compare our framework's performance with a single CNN; secondly, a CNN has achieved the state-of-art result in sentiment analysis [21].

For the SSWE baseline, we use Attardi's implementation [40] of SSWE [7] to generate hurricane-specific and sentiment-specific word embeddings. Although our work and SSWE are both derived from neural language models,

Method	**Hurricane Sandy**	**SemEval 2013**
SSWE+CNN [7]	85.54	69.92
Skip-gram [3] created embedding using unlabeled text + single CNN [21]	86.00	70.72
Basic Approach I: Two skip-gram-generated embeddings from class-filtered text + parallel CNN framework	88.15	71.35
Basic Approach II: Addition of class vector and general meaning vector + parallel CNN framework	87.72	71.60
Advanced Model I: Modified skip-gram + parallel CNN framework	**88.19**	**73.15**
Advanced Model II: Modified CBOW + parallel CNN framework	87.91	71.99

Table 4: Comparison of classification accuracy across the two datasets using word embeddings from various models.

our model extends Mikolov's skip-gram model, while SSWE extends Collobert's C&W model [24]. The skip-gram model has a simple architecture, while C&W model keeps a look-up table for all the words in the vocabulary and a fully-connected hidden layer, which makes SSWE slower to compute and hard to scale to large datasets. In Basic Approach II, we use the tweets in the training set to generate a general meaning word embedding w. We then calculate $\mathcal{V}(hurricane)$ and add $\mathcal{V}(hurricane)$ to w to produce a class-specific embedding for the second CNN. We then use the two sets of embeddings in the classification framework. In the advanced model, we use the same $\mathcal{V}(hurricane)$ from Basic Approach II and added to the input word for each training tuple in the input layer to train the class-specific word embedding.

In the SemEval dataset experiments, a slight difference is in the choice of the polarity word when we try to calculate $\mathcal{V}(positive)$ and $\mathcal{V}(negative)$. We choose the polarity word "good" for positive class and "bad" for negative class for use in generating two sets of class-specific word embeddings for Basic Approach II and the Advanced Model.

In both sets of experiments, the SSWE+CNN result is relatively weak compared to skip-gram derived models. The result of Basic Approach I using two sets of self-trained embeddings on our framework is better than the result of the second baseline, which uses one single CNN. We ascribe the reason to be that we combine more classifiers that use different features (e.g., from different embeddings). It is similar to the ensemble method in machine learning, thus improving the overall performance. The result of the Basic Approach II is very similar to the result of the Basic Approach I.

This indicates part of our concern that simply shifting all the embeddings in the vector space by the same distance is insufficient to boost performance. Results for the Advanced Model I is the highest, outperforming baselines, the basic approaches and also Advanced Model II. Results for Advanced Model II, the modified CBOW, are lower than the results of Advanced Model I, the modified skip-gram. Compared to CBOW, skip-gram model trains over more data since each word in the corpus can be a training tuple. Thus the skip-gram model favors small datasets. Since both of our labeled datasets are small, it is perhaps unsurprising that the results of Advanced Model I are better.

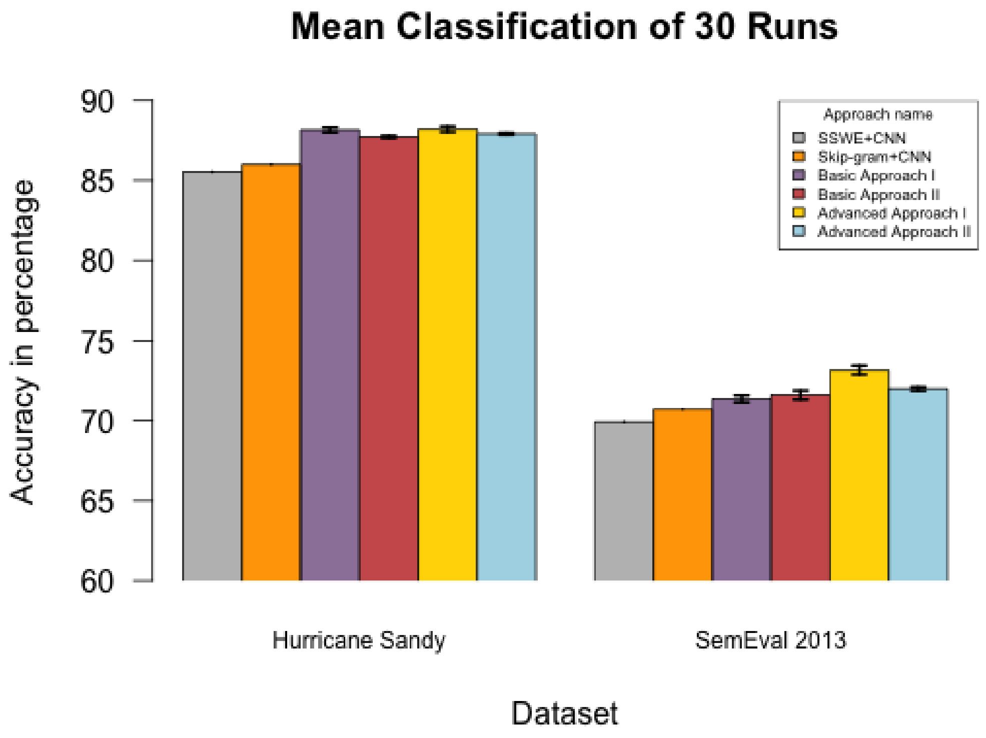


Figure 8: Mean classification accuracy of thirty additional runs for the proposed models in bar chart, with standard errors, compared to the two existing baseline approaches.

There is stochasticity in the proposed approaches. For example, all embeddings are initialized with random values. To reduce the uncertainty in our measured results, we performed thirty additional runs and present the mean performance in Figure 8 along with standard errors on the proposed models. Compared to SemEval 2013 dataset, results on the Hurricane Sandy dataset tend to have tighter error bars. For the Hurricane Sandy dataset, the mean Basic Approach I accuracy was more than 2 percentage points higher than the results of baseline's in Table 4. For both datasets, the advanced model remains the best performer, achieving a mean relative improvement of 3.1% (Hurricane Sandy) to 4.6% (SemEval 2013) over the SSWE+CNN baseline. This suggests our proposed approaches to generate class-specific word embeddings combined with the parallel CNN framework can improve the performance on text classification tasks.

Moreover, to better measure the performance of the class-specific word embeddings we trained, we compare word embeddings trained from the advanced approach I with the embeddings trained from skip-gram model, one of our baselines in the first dataset Hurricane Sandy. Most tuples in the test set that mention "hurricane","hurricane Sandy" and "Frankenstorm" are recognized correctly in both advanced approaches and baseline models. For example, "Frankenstorm was actually the name of the creator. This hurricane should properly be called Frankenstorm's monster.", "Praying for everyone in the path of Hurricane Sandy." and "This hurricane blowing me now." To better verify the effectiveness of the proposed advanced approach I, we look at some tuples from test set that are classified correctly (True Positive) in advanced approach I but classified incorrectly (False Negative) in the second baseline using skip-gram model as shown in Examples a, b and c in Table 5. We found that after adding class information in the advanced approach I, tuples such as Example a, b and c can be recognized as "hurricane-related" even without obvious words or hashtag indicators.

Example a	My power was out for like 5 days when Irene hit.
Example b	Stranger danger! My power's out too. Be safe #drinkingtillifallasleep
Example c	People on the other side of the country won't see specifically how #lbi is doing. It's all grouped into the east coast.

Table 5: Three tuples extracted from test set of Hurricane Sandy dataset

4.4. Parameter Sensitivity

In order to evaluate how changes to the parameterization of the proposed approaches affect its performance on classification tasks, we conducted experiments on the binary text classification tasks.

4.4.1. Word Embedding Dimensionality

For this test, to focus on the performance of parameterization, we need to remove the randomness introduced during the training procedure. We first fix the seed parameter in the initialization of the word embedding vectors so that the experiments could be repeated with identical starting points; then we use only a single thread to eliminate randomness introduced by operating system thread scheduling. To further reduce the randomness in word vector initialization, during the experiments of trials with different word vector dimensionality, we initialize the maximum dimension n of word vector so that each word vector with different dimensionality s will be initialized by selecting the top s numbers from the initialized word vector of size n. The result of these steps was a process that repeatedly assigned exactly the same random values regardless of the size of the word vector representation.

Figure 9 shows the effects of performance when increasing the number of dimensions in our three proposed models in classification tasks of two datasets. During all the experiments, we have fixed the window size to be 5. All the experiments are performed starting from word vector of dimension 5 to word vector of dimension 165 with an interval of 10. So altogether there are 17 points on each plot. We use standard Local Polynomial Regression Fitting method (loess) in R to fit a polynomial surface for each plot to show the trend of performance as dimensionality increases. Figure 9a, 9c and 9e examine the effects of varying the dimensionality in the Hurricane Sandy dataset. As shown in Figures 9a and 9e, the optimal dimensionality for Basic Approach I and the Advanced Approach is obtained near 65 dimensions, while Figure 9c shows a steady trend. Figures 9b, 9d and 9f examine the effects of varying the dimensionality in the SemEval 2013 dataset. The experimental results of Advanced model II in the Hurricane Sandy dataset in Figure 9g become steady around 87.69% when the number of dimensions reaches 73. For the SemEval dataset, the Advanced Model II achieves highest performance at around 55 dimensions as shown in Figure 9h. The performance is quite consistent between both Hurricane Sandy dataset and SemEval 2013 dataset in the sense that the Advanced model I in both datasets shows a better performance at a lower dimensionality (around 75 for Hurricane Sandy dataset and around 45 for SemEval 2013 dataset) compared to the other proposed approaches.

4.4.2. Number of Most Similar Words to the Polarity Words

We have conducted grid search for hyper-parameters word embedding dimension and the number of words which have the highest similarity scores to the polarity word. We denote the number of words which have the highest similarity scores to the polarity word as n. We generated word embedding dimension candidates from the list [30,50,65,80]. We generated the candidates of n from the list [50,70,90,100,150,200]. We exhaustively generated a grid of parameter values specified by the two lists above. Altogether we have 24 (4*6) different combinations of word embedding dimension and n. Other than the Basic Approach I which does not utilize the class vector, we apply all 24 combinations on the Basic Approach II, the Advanced Approach I and the Advanced Approach II on the two datasets, Hurricane Sandy and Semeval. We performed grid search for the two hyper-parameters, word embedding dimension and n, the number of most similar words to the polarity words. We found that our approach is fairly robust to the choice of n. We found that there are no obvious trending or conclusion can be made as to which n and dimension is the best fit. In general, different approach in different dataset prefers a different combination of hyper-parameters. In all cases, $n = 100$ seems to be a good setting (except for Advanced Approach I in Semeval dataset, peak value is obtained at $n = 150$). But the accuracy value does change with different word embedding dimension settings.

We performed additional experiments to use the polarity word itself directly as the class vector representation. We found that all the accuracy values of Basic Approach II, Advanced Approach I and Advanced Approach II dropped slightly. We think that if n is larger than one, the class representation might have a greater chance to incorporate the words that can most accurately represent the classs semantic meaning.

4.5. Discussion

In this section we first compare and contrast the computational costs of our approach and then consider some observations that could lead to opportunities for future research.

4.5.1. Computational Complexity

We introduced in this paper simple but effective models to tackle the light polysemy problem. Existing context-based word embedding algorithms utilize more complicated algorithms to search and generate all possible embeddings per word regardless of the future application task. For example, Huang et al. adopted K-means to cluster all the contexts in which the target word appears. By solving the light polysemy problem instead, we avoid the complex computation in this step by taking advantage of the linear compositionality property. The linear compositionality property in our approaches require only vector addition. Similarly, Huang et al. pre-defined k as the number of embeddings (clusters) per word; the time complexity of k-means is $O(nkdi)$, where n is the number of d-dimensional vectors (in our case the number of contexts a word has in the corpus), k is the number of clusters and i the number of iterations needed until convergence [41]. Huang et al. need to perform k-means on every word in the vocabulary. Thus the computational complexity for Huang et al.'s approach is $O(|V|nkdi)$, where $|V|$ is the vocabulary size of the corpus. When the corpus is large, Huang et al.'s approach is hard to scale. In contrast, we do not need to iteratively re-compute the centroids across the whole corpus for every single word. To generate a word embedding per class, the computation for our work only needs the addition operation, which takes linear time $\mathrm{O}(d)$. For space complexity, vectors representing the contexts and the centroids in Huang et al.'s approach need to be stored. Specifically, the storage required is $\mathrm{O}(|V|(n+k)d)$. Our approaches need $\mathrm{O}(bd)$ since we only store the vector representation of class, where b is the number of the classes in the corpus.

4.5.2. Future Research

To choose "good" and "bad" as the polarity words is risky. We found in the Twitter dataset that people describe positive and negative emotion using lexicons with great variety, such as "Gas by my house hit $3.99!! I'am going to Chapel Hill on Sat!", "Twitition Mcfly come back to Argentina but this time we want to come to mar del plata!!!" and "Never start working on your dreams and goals tomorrow......tomorrow never comes....if it means anything to U, ACT NOW! #getafterit"[3]. These three tweets have no lexicon that are associated with "good" or "bad". Thus how to choose or generate polarity words to produce a vector representation of the class is still an open question.

Another observation is that summation is best suited for elementary words such as "water". When an elementary word is added to a complex-

[3]These three tweets are extracted from SemEval 2013 training data.

meaning word, such as "massacre", we found the meaning of the elementary word is often overwhelmed by the complex-meaning word. This problem is best demonstrated by finding the most n similar words from the vocabulary using cosine similarity. When we add $vector(water)$ to $vector(massacre)$, the top ranked words are "massacres", "killings" and "murders". The semantic of word "water" seems to have disappeared, which introduces another research problem.

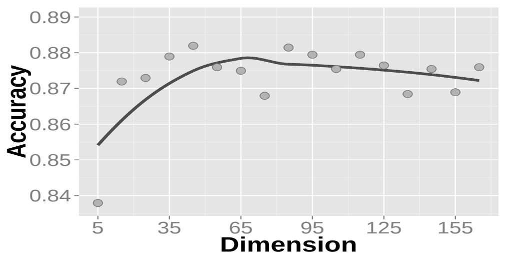
This work should also be extensible into a multi-class text classification setting. Note that it would become non-trivial to decide manually the polarity words for each class in a multi-class text classification scenario. One possible solution could be a topic-model-based approach to automatically define the polarity words for each class. Then word embeddings could be trained to represent the top words. The number of the top words would also be a hyper-parameter.

5. Conclusion

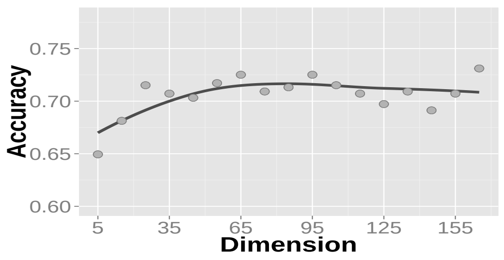
In this paper, we used the linear compositionality property to improve the learning of class-specific word embeddings for a text classification task. We explored four models to learn class-specific word embeddings. We devised a classification framework to take multiple sets of class-specific word embeddings as input. We tested our methods on two Twitter datasets. Our results showed that for text classification tasks with clear polarity words, our proposed approaches can increase performance.

Acknowledgments

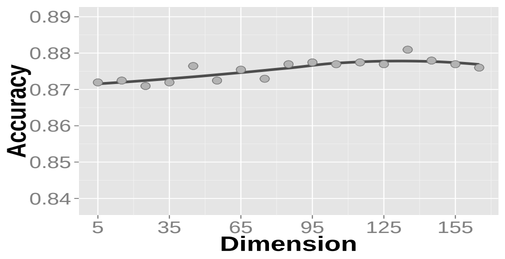
This material is based in part upon work supported by the National Science Foundation under Grant No. CMMI-1541177.



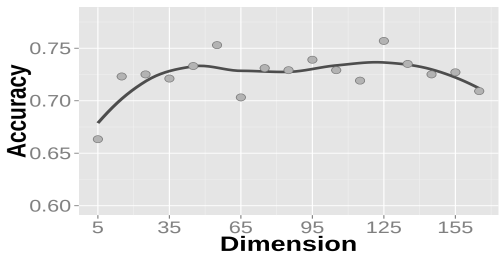
(a) Basic approach I: Hurricane Sandy



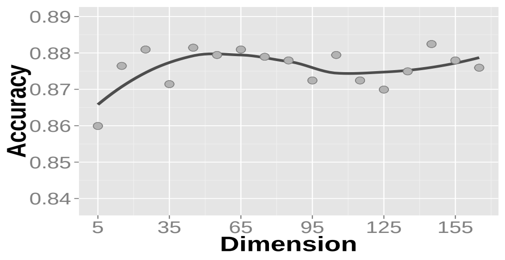
(b) Basic approach I: SemEval 2013



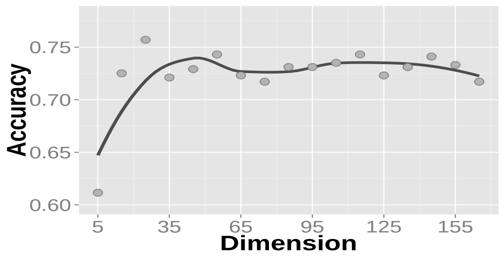
(c) Basic approach II: Hurricane Sandy



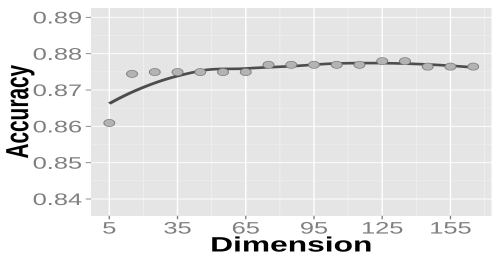
(d) Basic approach II: SemEval 2013



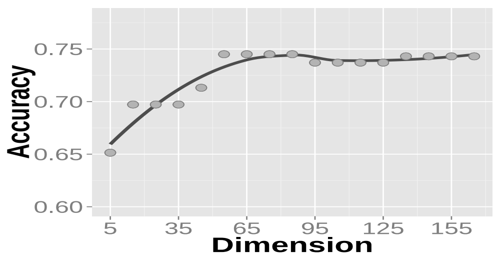
(e) Advanced approach I: Hurricane Sandy



(f) Advanced approach I: SemEval 2013



(g) Advanced approach II: Hurricane Sandy



(h) Advanced approach II: SemEval 2013

Figure 9: Parameter Sensitivity Study of Word Embedding Dimensionality

[1] A. Nematzadeh, S. C. Meylan, T. L. Griffiths, Evaluating vector-space models of word representation, or, the unreasonable effectiveness of counting words near other words, in: Proceedings of the 39th Annual Meeting of the Cognitive Science Society.

[2] Z. S. Harris, Distributional structure, Word 10 (1954) 146–162.

[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in Neural Information Processing Systems, pp. 3111–3119.

[4] Q. Liu, Z.-H. Ling, H. Jiang, Y. Hu, Part-of-speech relevance weights for learning word embeddings, arXiv preprint arXiv:1603.07695 (2016).

[5] S. K. Sienčnik, Adapting word2vec to named entity recognition, in: Proceedings of the 20th Nordic Conference of Computational Linguistics, Vilnius, Lithuania, 109, Linköping University Electronic Press, pp. 239–243.

[6] O. Levy, Y. Goldberg, Dependency-based word embeddings., in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, volume 2, pp. 302–308.

[7] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, B. Qin, Learning sentiment-specific word embedding for Twitter sentiment classification., in: Proceeding of the 52nd Annual Meeting of the Association for Computational Linguistics.

[8] Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, A neural probabilistic language model, Journal of Machine Learning Research 3 (2003) 1137–1155.

[9] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of tricks for efficient text classification, in: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, volume 2, pp. 427–431.

[10] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, Deep contextualized word representations, in: Proceedings of the 2018 Conference of the North American Chapter of the

Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), volume 1, pp. 2227–2237.

[11] X. Zheng, J. Feng, Y. Chen, H. Peng, W. Zhang, Learning context-specific word/character embeddings, AAAI Conference on Artificial Intelligence (2017).

[12] F. Tian, H. Dai, J. Bian, B. Gao, R. Zhang, E. Chen, T.-Y. Liu, A probabilistic model for learning multi-prototype word embeddings, in: Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pp. 151–160.

[13] A. Olteanu, C. Castillo, F. Diaz, S. Vieweg, Crisislex: A lexicon for collecting and filtering microblogged communications in crises, in: Proceedings of the International AAAI Conference on Weblogs and Social Media.

[14] E. H. Huang, R. Socher, C. D. Manning, A. Y. Ng, Improving word representations via global context and multiple word prototypes, in: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1, Association for Computational Linguistics, pp. 873–882.

[15] A. Neelakantan, J. Shankar, A. Passos, A. McCallum, Efficient non-parametric estimation of multiple embeddings per word in vector space, arXiv preprint arXiv:1504.06654 (2015).

[16] M. Yu, M. Dredze, Improving lexical embeddings with semantic knowledge, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), volume 2, pp. 545–550.

[17] M. Faruqui, J. Dodge, S. K. Jauhar, C. Dyer, E. H. Hovy, N. A. Smith, Retrofitting word vectors to semantic lexicons, CoRR abs/1411.4166 (2014).

[18] M. Yu, M. Gormley, M. Dredze, Factor-based compositional embedding models, in: NIPS Workshop on Learning Semantics.

[19] X. Chen, Z. Liu, M. Sun, A unified model for word sense representation and disambiguation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1025–1035.

[20] S. Kuang, B. D. Davison, Class-specific word embedding through linear compositionality, in: To appear in Proceedings of the IEEE International Conference on Big Data and Smart Computing (BigComp).

[21] Y. Kim, Convolutional neural networks for sentence classification, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), p. 17461751. ArXiv preprint arXiv:1408.5882.

[22] T. K. Landauer, P. W. Foltz, D. Laham, An introduction to latent semantic analysis, Discourse Processes 25 (1998) 259–284.

[23] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation, Journal of Machine Learning Research 3 (2003) 993–1022.

[24] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, Journal of Machine Learning Research 12 (2011) 2493–2537.

[25] W. Ling, C. Dyer, A. Black, I. Trancoso, Two/too simple adaptations of word2vec for syntax problems, in: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 1299–1304.

[26] Y. Chen, B. Perozzi, R. Al-Rfou, S. Skiena, The expressive power of word embeddings, in: ICML 2013 Workshop on Deep Learning for Audio, Speech, and Language Processing.

[27] A. Trask, P. Michalak, J. Liu, sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings, arXiv preprint arXiv:1511.06388 (2015).

[28] J. Guo, W. Che, H. Wang, T. Liu, Learning sense-specific word embeddings by exploiting bilingual resources, in: Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pp. 497–507.

[29] J. Su, S. Wu, B. Zhang, C. Wu, Y. Qin, D. Xiong, A neural generative autoencoder for bilingual word embeddings, Information Sciences 424 (2018) 287 – 300.

[30] M. Pelevina, N. Arefyev, C. Biemann, A. Panchenko, Making sense of word embeddings, arXiv preprint arXiv:1708.03390 (2017).

[31] D. Bollegala, Y. Yoshida, K. ichi Kawarabayashi, Using k-way co-occurrences for learning word embeddings, in: AAAI 2018 Conference on Artificial Intelligence.

[32] J. Pennington, R. Socher, C. D. Manning, GloVe: Global vectors for word representation, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1532–1543.

[33] T. Scheepers, E. Kanoulas, E. Gavves, Improving word embedding compositionality using lexicographic definitions, in: Proceedings of the 2018 World Wide Web Conference, WWW '18, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 2018, pp. 1083–1093.

[34] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, arXiv preprint arXiv:1607.04606 (2016).

[35] A. G. W. Ben Athiwaratkun, A. Anandkumar, Probabilistic fasttext for multi-sense word embeddings, in: Conference of the Association for Computational Linguistics (ACL) 2018.

[36] D. Reynolds, Gaussian mixture models, Encyclopedia of biometrics (2015) 827–832.

[37] H. Chen, B. Wei, Y. Liu, Y. Li, J. Yu, W. Zhu, Bilinear joint learning of word and entity embeddings for entity linking, Neurocomputing 294 (2018) 12 – 18.

[38] J. Mitchell, M. Lapata, Vector-based models of semantic composition., in: Proceeding of the Annual Meeting of the Association for Computational Linguistics, pp. 236–244.

[39] Q. Li, S. Shah, X. Liu, A. Nourbakhsh, Data sets: Word embeddings learned from tweets and general data, arXiv preprint arXiv:1708.03994 (2017).

[40] G. Attardi, DeepNL: a deep learning NLP pipeline, in: Proceedings of NAACL-HLT, pp. 109–115.

[41] J. A. Hartigan, M. A. Wong, Algorithm as 136: A k-means clustering algorithm, Journal of the Royal Statistical Society. Series C (Applied Statistics) 28 (1979) 100–108.