Pattern Recognition Letters 80 (2016) 150–156

Contents lists available at ScienceDirect

Pattern Recognition Letters

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

Representation learning for very short texts using weighted word embedding aggregation☆

Cedric De Boom*, Steven Van Canneyt, Thomas Demeester, Bart Dhoedt

Department of Information Technology, Ghent University – iMinds, Technologiepark 15, 9052 Zwijnaarde, Belgium

ARTICLE INFO

Article history:
Received 10 February 2016
Available online 28 June 2016

Keywords:
Information storage and retrieval
Natural language processing
Artificial intelligence
Word embeddings
Representation learning

ABSTRACT

Short text messages such as tweets are very noisy and sparse in their use of vocabulary. Traditional textual representations, such as tf-idf, have difficulty grasping the semantic meaning of such texts, which is important in applications such as event detection, opinion mining, news recommendation, etc. We constructed a method based on semantic word embeddings and frequency information to arrive at low-dimensional representations for short texts designed to capture semantic similarity. For this purpose we designed a weight-based model and a learning procedure based on a novel median-based loss function. This paper discusses the details of our model and the optimization methods, together with the experimental results on both Wikipedia and Twitter data. We find that our method outperforms the baseline approaches in the experiments, and that it generalizes well on different word embeddings without retraining. Our method is therefore capable of retaining most of the semantic information in the text, and is applicable out-of-the-box.

1. Introduction

Short pieces of texts reach us every day through the use of social media such as Twitter, newspaper headlines, and texting. Especially on social media, millions of such short texts are sent every day, and it quickly becomes a daunting task to find similar messages among them, which is at the core of applications such as event detection [6], news recommendation [11], etc.

In this paper we address the issue of finding an effective vector representation for a very short text fragment. By effective we mean that the representation should grasp most of the semantic information in that fragment. For this we use semantic word embeddings to represent individual words, and we learn how to weigh every word in the text through the use of tf-idf (term frequency – inverse document frequency) information to arrive at an overall representation of the fragment.

These representations will be evaluated through a semantic similarity task. It is therefore important to point out that textual similarity can be achieved on different levels. At the most strict level, the similarity measure between two texts is often defined as being (near) paraphrases. In a more relaxed setting one is interested in topic- and subject-related texts. For example, if a sentence is about the release of a new Star Wars episode and another about Darth Vader, they will be dissimilar in the most strict sense, although they share the same underlying subject. In this paper we focus on the broader concept of topic-based semantic similarity, as this is often applicable in the already mentioned use cases of event detection and recommendation.

Our main contributions are threefold. First, we construct a technique to calculate effective text representations by weighing word embeddings, for both fixed- and variable-length texts. Second, we devise a novel median-based loss function to be used in the context of minibatch learning to mitigate the negative effect of outliers. Finally we create a dataset of semantically related and non-related pairs of text from both Wikipedia and Twitter, on which the proposed techniques are evaluated.

We will show that our technique outperforms most of the baselines in a semantic similarity task. We will also demonstrate that our technique is independent of the word embeddings being used, so that the technique is directly applicable and thus does not require additional model training when used in different contexts, in contrast to most state-of-the art techniques.

In the next section, we start with a summary of the related work, and our own methodology will be devised in Section 3. Next we explain how data is collected in Section 4, after which we discuss our experimental results in Section 5.

☆ This paper has been recommended for acceptance by Jie Zou.

* Corresponding author. Tel.: +32 9 331 49 42.
E-mail address: cedric.deboom@ugent.be (C. De Boom).

http://dx.doi.org/10.1016/j.patrec.2016.06.012
0167-8655/

2. Related work

In this work we use so-called word embeddings as a basic building block to construct text representations. Such an embedding is a distributed vector representation of a single word in a fixed-dimensional semantic space, as opposed to term tf-idf vectors, in which a word is represented by a one-hot vector [1,19]. A word's term frequency (tf) is the number of times the word occurs in the considered document, and a word's document frequency (df) is the number of documents in the considered corpus that contain that word. Its (smoothed) inverse document frequency (idf) is defined as:

$$\text{idf} \triangleq \log \frac{N}{1+\text{df}}, \tag{1}$$

in which N is the number of documents in the corpus [19]. A tf-idf-based similarity measure is based on exact word overlap. As texts become smaller in length, however, the probability of having words in common decreases. Furthermore, these measures ignore synonyms and any semantic relatedness between different words, and are prone to negative effects of homonyms.

Instead of relying on exact word overlap, one can incorporate semantic information into the similarity process. Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA) are two examples, in which every word is projected into a semantic (topic) space [2,7]. At test time, inference is performed to obtain a semantic vector for a particular sentence. Both training and inference of standard LSI and LDA, however, are computationally expensive on large vocabularies.

Although LSI and LDA have been used with success in the past, Skip-gram models have been shown to outperform them in various tasks [18,20]. In Skip-gram, part of Google's word2vec toolkit[1], distributed word embeddings are learned through a neural network architecture to predict its surrounding words in a fixed window.

Once the word embeddings are obtained, we have to combine them into a useful sentence representation. One possibility is to use a multilayer perceptron (MLP) with the whole sentence as an input, or a 1D convolutional neural network [4,9,10,26]. Such an approach, however, requires either an input of fixed length or aggregation operations – such as dynamic k-max pooling [12] – to arrive at a sentence representation that has the same dimensionality for every input. Recurrent neural networks (RNNs) and variants can overcome the problem of fixed dimensionality or aggregation, since one can feed word after word in the system and in the end arrive at a text representation [22–24]. The recently introduced Skip-thought vectors, heavily inspired on Skip-gram, combine the learning of word embeddings with the learning of a useful sentence representation using an RNN encoder and decoder [15]. RNN-based methods present a lot of advantages over MLPs and convolutional networks, but still retraining is required when using different types of embeddings.

Paragraph2vec is another method, inspired by the Skip-gram algorithm, to derive sentence vectors [17]. The technique requires the user to train vectors for frequently occurring word groups. The method, however, is not usable in a streaming or on-the-fly fashion, since it requires retraining for unseen word groups at test time.

Aggregating word embeddings through a mean, max, min... function is still one of the most easy and widely used techniques to derive sentence embeddings, often in combination with an MLP or convolutional network [4,21,25,27]. On one hand, the word order is lost, which can be important in e.g. paraphrase identification. On the other hand, the methods are simple, out-of-the-box and do not require a fixed length input.

[1] Available at code.google.com/archive/p/word2vec.

Related to the concepts of semantic similarity and weighted embedding aggregation, there is extensive literature. Kusner et al. [16] calculate a similarity metric between documents based on the travel distance of word embeddings from one document to another one. We on the other hand will derive vectors for the documents themselves. Kenter and de Rijke [14] learn semantic features for every sentence in the dataset based on a saliency weighted network for which the BM25 algorithm is used. However, the features are being learned for every sentence prior to test time, and therefore not applicable in a real-time streaming context. Finally, Kang et al. [13] calculate a cosine similarity matrix between the words of two sentences that are sorted based on their idf value, which they use as a feature vector for an MLP. Their approach is similar to our work in the sense that the authors use idf information to rescale term contribution. Their primary goal, however, is calculating semantic similarity instead of learning a sentence representation. In fact, the authors totally discard the original word embeddings and only use the calculated cosine similarity features.

3. Methodology

The core principle of our methodology is to assign a weight to each word in a short text. These weights are determined based on the idf value of the individual words in that text. The idea is that important words – i.e. words that are needed to determine most of the text's semantics – usually have higher idf values than less important words, such as articles and auxiliaries... Indeed, the latter appear more frequently in various different texts, while words with a high idf value mostly occur in similar contexts. The final goal is to combine the weighted words into a semantically effective, single text representation.

To achieve this goal, we will model the problem of finding a suitable text representation as a semantic similarity task between couples of short texts. In order to classify such couples of text fragments into either semantically related pairs or non-related pairs, the vector representations of these two texts are directly compared. In this paper we use a simple threshold function on the distance between the two text representations, as we want related pairs to lie close to each other in their representation space, and non-related pairs to lie far apart:

$$g(t_1, t_2) = \begin{cases} \text{pair} & \text{if } d(t_1, t_2) \leq \theta \\ \text{non-pair} & \text{if } d(t_1, t_2) > \theta \end{cases}. \tag{2}$$

In this expression t_1 and t_2 are two short text vector representations of dimensionality ν, $d : (x, y) \in \mathbb{R}^{2\nu} \rightarrow \mathbb{R}^+$ is a vector distance function of choice (e.g. cosine distance, euclidean distance...), θ is a threshold, and $g(\cdot)$ is the binary prediction of semantic relatedness.

3.1. Basic architecture

As mentioned before, we will assign a weight to each word in a text according to that word's idf value. To learn these weights, we devise a model that is visualized in Fig. 1. In the learning scheme, we use related and non-related couples of text as input data. First, the words in every text are sorted from high to low idf values. Original word order is therefore discarded, as is the case in usual standard aggregation operations. After that, every embedding vector for each of the sorted words is multiplied with a weight that can be learned. Finally, the weighted vectors are averaged to arrive at a single text representation.

In more detail, consider a dataset $\mathcal{D}$ consisting of couples of short texts. An arbitrary couple is denoted by C, and the two texts of C by C^α and C^β. We indicate the vector representation of word j in text C^α by C^α_j. All word vectors have the same dimensionality ν. Each text C^α also has an associated length $n(C^\alpha)$, i.e. the number of words in C^α. For now, in this section, we assume that