Short Text Similarity with Word Embeddings

Tom Kenter
tom.kenter@uva.nl

Maarten de Rijke
derijke@uva.nl

University of Amsterdam, Amsterdam, The Netherlands

ABSTRACT

Determining semantic similarity between texts is important in many tasks in information retrieval such as search, query suggestion, automatic summarization and image finding. Many approaches have been suggested, based on lexical matching, handcrafted patterns, syntactic parse trees, external sources of structured semantic knowledge and distributional semantics. However, lexical features, like string matching, do not capture semantic similarity beyond a trivial level. Furthermore, handcrafted patterns and external sources of structured semantic knowledge cannot be assumed to be available in all circumstances and for all domains. Lastly, approaches depending on parse trees are restricted to syntactically well-formed texts, typically of one sentence in length.

We investigate whether determining short text similarity is possible using only semantic features—where by *semantic* we mean, pertaining to a representation of meaning—rather than relying on similarity in lexical or syntactic representations. We use word embeddings, vector representations of terms, computed from unlabelled data, that represent terms in a semantic space in which proximity of vectors can be interpreted as semantic similarity.

We propose to go from word-level to text-level semantics by combining insights from methods based on external sources of semantic knowledge with word embeddings. A novel feature of our approach is that an arbitrary number of word embedding sets can be incorporated. We derive multiple types of meta-features from the comparison of the word vectors for short text pairs, and from the vector means of their respective word embeddings. The features representing labelled short text pairs are used to train a supervised learning algorithm. We use the trained model at testing time to predict the semantic similarity of new, unlabelled pairs of short texts.

We show on a publicly available evaluation set commonly used for the task of semantic similarity that our method outperforms baseline methods that work under the same conditions.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: H.3.3 Information Search and Retrieval; I.2 Artificial Intelligence [**I.2.7 Natural Language Processing**]: Text analysis

CIKM'15, October 19–23, 2015, Melbourne, Australia.

ACM 978-1-4503-3794-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2806416.2806475 .

Keywords

Short Text Similarity; Word Embeddings

1. INTRODUCTION

Determining semantic similarity between two texts is to find out if two pieces of text mean the same thing. Being able to do so successfully is beneficial in many settings in information retrieval like search [26], query suggestion [30], automatic summarization [3] and image finding [12].

Many approaches have been proposed for semantic matching that use lexical matching and linguistic analysis, next to semantic features. Methods for lexical matching aim to determine whether the words in two short texts look alike, e.g., in terms of edit distance [31], lexical overlap [23] or largest common substring [21]. While this might work for trivial cases, it is arguably not robust as it allows for simple mistakes. For example, the *US* would be closer to the *UK* this way, than it would be to the *States*. Features based on linguistic analysis, like dependency parses or syntactic trees, are often used for short text similarity [19, 39]. Linguistic tools such as parsers are commonly available these days for many languages, though the quality might vary between languages. However, not all texts are necessarily parseable (e.g., tweets) and high-quality parses might be expensive to compute at run time. More importantly still, relying on parse trees limits an approach to single sentences, while the work presented here, even though it is evaluated on sentences, incorporates no theoretical constraint restricting it to (syntactically well-formed) sentences.

For semantic features, many approaches use external sources of structured semantic knowledge such as Wikipedia [6] or WordNet [6, 17, 18, 31, 37]. Wikipedia is structured around entities and as such is primarily of avail in settings where a focus on rather well-known persons and organisations can be assumed, such as, e.g., news articles. Such an assumption cannot always be made however. A drawback of using dictionaries or WordNet is that high-quality resources like these are not available for all languages, and proper names, domain-specific technical terms and slang tend to be underrepresented [2].

In the present work we aim to make as few assumptions as possible. We aim for a generic model, that requires no prior knowledge of natural language (such as parse trees) and no external resources of structured semantic information.

Recent developments in distributional semantics, in particular neural network-based approaches like [32, 34] only require a large amount of unlabelled text data. This data is used to create a, so-called, semantic space. Terms are represented in this semantic space as vectors that are called *word embeddings*. The geometric properties of this space prove to be semantically and syntactically

meaningful [13, 32–34], that is, words that are semantically or syntactically similar tend to be close in the semantic space.

A challenge for applying word embeddings to the task of determining semantic similarity of short texts is going from word-level semantics to short-text-level semantics. This problem has been studied extensively over the past few years [4, 25, 39].

In this work we propose to go from word-level to short-text-level semantics by combining insights from methods based on external sources of semantic knowledge with word embeddings. In particular, we perform semantic matching between words in two short texts and use the matched terms to create a saliency-weighted semantic network. A novel feature of our approach is that an arbitrary number of word embedding sets can be incorporated, regardless of the corpus used for training, the underlying algorithm, its parameter settings or the dimensionality of the word vectors. We derive multiple types of meta-features from the comparison of the word vectors for short text pairs and from the vector means of their respective word embeddings, that have not been used before for the task of short text similarity matching.

We show on a publicly available test collection that our generic method, that does not rely on external sources of structural semantic knowledge, outperforms baseline methods that work under the same conditions and outperforms all methods, to our knowledge, that do use external knowledge bases and that have been evaluated on this dataset.

The rest of the paper is structured as follows. In Section 2 we describe relevant literature. We present our method for short text similarity in Section 3. The experiments and results are detailed in Section 4 and Section 5. In Section 6 we conclude.

2. RELATED WORK

In this section we discuss previous work related to the different aspects of our method.

Distributional semantics.

Distributional semantic approaches are based on the intuition that words appearing in similar contexts tend to have similar meanings. The Latent Semantic Analysis algorithm (LSA) [14] incorporates this intuition by building a word-document co-occurrence matrix and performing singular value decomposition (SVD) on it to get a lower-dimensional representation. Words are represented as vectors in this lower dimensional space. The distance between these word vectors (measured, e.g., with the cosine function) can be used as a proxy for semantic similarity. The full co-occurrence matrix, however, can become quite substantial for a large corpus, in which case the SVD becomes memory-intensive and computationally expensive.

Word vectors—also referred to as word embeddings—have recently seen a surge of interest as new ways of computing them efficiently have become available. In [32, 33] an algorithm called word2vec is proposed. There are two architectures to word2vec, continuous bag-of-words (CBOW) and Skip-gram. Both are a variation on a neural network language model [9, 13], but rather than predicting a word conditioned on its predecessor, as in a traditional bi-gram language model, a word is predicted from its surrounding words (CBOW) or multiple surrounding words are predicted from one input word (Skip-gram). To avoid computing a full softmax over the entire vocabulary, hierarchical softmax can be applied on a Huffman tree representation of the vocabulary, which saves calculations, at the potential loss of some accuracy. An additional strategy to get better embeddings is negative sampling, where, instead of only using the words observed next to one another in the training data as positive examples, random words are sampled from the corpus and presented to the network as negative examples.

An alternative way of getting word embeddings, called GloVe, is proposed in [34]. Rather than being based on language models it is based on global matrix factorisation. As such, it is closer to LSA, only a word-word co-occurrence matrix is used. GloVe avoids the large computational cost of, e.g., LSA by not building the full co-occurrence matrix, but training directly on the non-zero elements in it. As a cost function, the model uses a weighted least squares variant. The weighting function has two parameters, an exponent and a maximum cut-off value that influence the performance.

As both algorithms produce high-quality word embeddings and their implementations are publicly available, we use them in our experiments. In Section 5 we report on the results and analyse the effect of different parameter settings for both methods.

Text-level semantics without external semantic knowledge.

Word embeddings, as described above, provide a way of comparing terms to one another semantically. It is not evident, however, how longer pieces of text should be represented with them. Several approaches have been proposed to go from word-level semantics to phrase-, sentence-, or even document-level semantics.

Le and Mikolov [25] propose a variation on the word2vec algorithm for calculating paragraph vectors, by adding an explicit paragraph feature to the input of the neural network. A convolutional neural network, built on top of word2vec word embeddings, is employed for modelling sentences in [20]. Other corpus-based methods have been proposed, such as [21], in which both semantic and string distance features are employed, and [38] in which a vector space model is used. All four methods, in line with the work presented here, do not rely on external sources of structured semantic knowledge, nor on natural language resources. As such, these methods are natural baselines for our experiments in Section 4. It is problematic to reproduce the work presented in [25], however, as the original source code was not released by the authors and it is not clear, algorithmically, how the second step – the inference for new, unseen texts – should be carried out. Therefore, we omit this method as a baseline.

Many methods rely on natural language resources such as parsers. Socher et al. [39] propose recursive auto-encoders for the task of semantic textual similarity. This method relies on full parse trees for every sentence it processes. Annesi et al. [4] apply a kernel method on dependency parse tree features. Another strong method is presented in [22] where features from dependency parser are used to train a supervised method. The latter method, to our knowledge, yields the highest performance on the MSR Paraphrase Corpus [15, 35], an evaluation set commonly used for textual similarity experiments, and the one we use in our experiments in Section 5.

Sentence representations based on word2vec word embeddings are also the focus in [24], where a convolutional neural network is trained on top of word2vec word embeddings. However, the method is only evaluated on sentence classification tasks (not on semantic similarity).

Text-level semantics with external knowledge.

A large body of research has been directed at using sources of structured semantic knowledge like Wikipedia and WordNet for semantic text similarity tasks. In [17, 18, 28] methods very similar to one another are proposed, using pairings of words and Wordnet-based measures for semantic similarity. Our method of aligning words as described in Section 3 draws on this work. The key difference between these approaches and ours, apart from the fact

that WordNet is used, is that parsing/POS tagging is carried out [18, 28], as the WordNet-based measures are limited to comparing words having the same POS tag. Furthermore, no full-scale machine learning step is involved. All methods present one overall score, based on a threshold which is calculated through a simple regression step [18, 28] or set manually [17].

Corpus methods are combined with WordNet-based measures in [27, 31]. In [31] an IDF-weighted alignment approach, based both on WordNet-based and corpus-based similarities, is proposed. Texts are parsed and only similarities within identical part-of-speech categories are considered. Finally, a single score is calculated as an average over the maximum similarities. In [27] a WordNet similarity measure is combined with word order scores. In neither approaches any machine learning step is applied.

SemEval STS.

Recently, the SemEval 2012 Semantic Text Similarity (STS) task [1] and SemEval 2013 STS task [2] (part of *SEM'13) were organised. A full description of the work of all participating teams (over 30 in both years) is beyond the scope of this section. We discuss the approaches of the best-scoring teams.

The best-scoring teams in 2012 both calculate a large number of features based on a wide variety of methods. Additionally, handcrafted rules are applied that deal with currency values, negation, compounds, number overlap [37] and with literal matching [6]. The main difference with our approach, apart from the handcrafted rules, is in the features extracted, and in particular the number of additional resources required (WordNet, a dependency parser, NER tools, lemmatizer, POS tagger, stop word list [37], and WordNet, Wikipedia, Wiktionary, POS tagger, SMT system for three language pairs [6]).

In 2013, we see similar approaches where the best teams extract features from sentence pairs and use regression models (SVRs) to predict a similarity score. The features in [19] are based on LSA, WordNet and additional lists of related words and stopwords. In [29] features are calculated from aggregated similarity measures based on named entity recognition with WordNet and Levenshtein distance, higher order word co-occurrence similarity, the RelEx system, dependency trees and reused features of SemEval 2012 participants. Additionally, handcrafted features like lists of aliases (e.g., *USA* and *United States*) are used. A parallel between our work and both these approaches is the use of word alignment.

Finally, in [5] a method similar to the one we propose here is presented, for a related, but different task of detecting semantic similarity between texts of different lengths. Next to WordNet-based features, a word alignment method is used based on word embeddings, analogous to what we propose. A crucial difference with our approach is that only a single feature is derived from this score, rather than several bins. Moreover, only a single set of word embeddings is used, while we show in our experiments in §5 that it is beneficial to use multiple sets.

Meta-level features.

As described below in Sections 3.1 and 3.2 below, we use bin-based features to capture the characteristics of the differences between vectors and the distribution of word embeddings. This is similar to, e.g., [11] where meta-level features are proposed, in a text classification setting using the kNN algorithm, to exploit the distribution of the nearest neighbour similarities and the within-class cohesion.

Input : List of sentence pairs $((s_{1,1}, s_{1,2}), (s_{2,1}, s_{2,2}), \ldots, (s_{n,1}, s_{n,2}))$
Input : List of associated labels $L = [l_1, l_2, l_3, \ldots, l_n]$
Required: Sets of word embeddings $[WE_1, WE_2, ..., WE_m]$
Required: Multiple feature extractors $[fe_1, fe_2, \ldots, fe_l]$
Output : A trained prediction model M

1 F = empty feature matrix;
2 **for** $i \leftarrow 1$ **to** n **do**
3 $\quad \vec{f}$ = <>;
4 $\quad$ **for** $j \leftarrow 1$ **to** m **do**
5 $\quad\quad$ **for** $k \leftarrow 1$ **to** l **do**
6 $\quad\quad\quad \vec{f} = \texttt{concat}(\vec{f}, fe_k((s_{i,1}, s_{i,2}), WE_j))$;
7 $\quad\quad$ **end**
8 $\quad$ **end**
9 $\quad F[i] \leftarrow \vec{f}$;
10 **end**
11 $M = trainModel(F, L)$;

Algorithm 1: Pseudocode of the feature generation step of our method for semantic similarity of short texts

3. SHORT TEXT SIMILARITY WITH SEMANTICS ONLY

To calculate semantic similarity between two short texts we use a supervised machine learning approach. Algorithm 1 shows the pseudocode of the training phase. The training data for the supervised step consists of sentence pairs and associated labels that represent the semantic similarity between the two sentences. Multiple sets of word embeddings can be leveraged, possibly derived from different corpora, with different (hyper)parameter settings or with different algorithms. Every sentence pair in the training data is represented by a set of features. A list of functions that generate features from a set of word embeddings and a sentence pair is required. We detail the different kinds of features below in Section 3.1 and Section 3.2.

At training time, we range over all sentences (Algorithm 1, line 2), all sets of word embeddings (line 4) and feature extraction functions (line 5) to compile a feature vector per sentence pair (line 6). The feature vectors are stored in a matrix (line 9). We train a supervised learning method from the features and the labels of the training examples (line 11). As the labels in the evaluation set that we use are binary, we build a classifier. At testing time, features are generated for the sentence pairs in the test set in a similar fashion as in the training phase, and a final prediction is made with the classifier trained in the training step.

A convenient property of our method of computing semantic textual similarity for short texts, and one which we leverage in our experiments as detailed in Section 4, is that different sets of word embeddings can be combined, regardless of the dimensionality of the word vectors, the parameters that were used at construction time, or the algorithms that were used to generate them.

As we are interested in the performance of different feature types, we carry out experiments per feature type and with various combinations. See Section 4 for further details on the experimental setup.

In the next section we describe the various types of features we derive from the word embeddings. In Section 4.2 we detail how we obtain the word embeddings themselves.

3.1 From word-level semantics to short-text-level semantics

The meaning of longer pieces of text (containing multiple terms) can be captured by taking the mean of the individual term vec-

tors.[1] This approach is taken, next to other approaches, in, e.g., [5, 20, 40]. It works surprisingly well, and we use several features based on vector means, described below. Means, or sums, however, are rather poor ways of describing the distribution of word embeddings across a semantic space. It would be desirable to capture more properties of the two texts, especially with respect to the terms that do or do not match. We will first turn to our algorithm for constructing saliency-weighted semantic networks, which aims to capture this intuition. After that, in Section 3.2, we discuss features based on the mean vectors.

3.1.1 *Saliency-weighted semantic network*

In Figure 1 the word embeddings of two short texts are represented as dots in a two-dimensional space. As can be observed from the picture, the two texts have terms that are close to each other (at the top and bottom in the figure), while the ones at the far left and right have no counterpart in their immediate vicinity. Regardless of this discrepancy, the means of the two are close to one another. The fact that both texts have a term unlike any term in the other text is not well represented by the means. A classifier, however, can benefit from more elaborate information about the distribution of word embeddings across the semantic space.

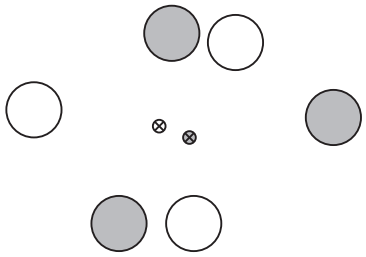

Figure 1: Hypothetical example — two-dimensional representation of the word embeddings for two short texts (each consisting of three terms), represented as transparent and opaque dots respectively. The corresponding means of the two sets of embeddings are depicted as $\otimes$.

We want a way of taking into account the distribution of terms in one short text in the semantic space compared to distribution of terms in another text. Of course, not all terms are equally important. Common terms (like determiners) do not contribute as much to the meaning of a text as less frequent words do. Inverted document frequency (idf) is often used to implement this notion. Idf is usually combined with term frequency, e.g., in the BM25 algorithm. As BM25 has proven to be surprisingly successful [36] we derive our idf weighting scheme from it. Our function for calculating semantic text similarity (sts) is:

$$f_{sts}(s_l, s_s) = \sum_{w \in s_l} \text{IDF}(w) \cdot \frac{sem(w, s_s) \cdot (k_1 + 1)}{sem(w, s_s) + k_1 \cdot (1 - b + b \cdot \frac{|s_s|}{\text{avgsl}})} \quad (1)$$

Here, s_l is the longest text of the two, s_s is the shortest and avgsl is the average sentence length in the training corpus.

The semantic similarity of term w with respect to short text s is represented by $sem(w, s)$:

$$sem(w, s) = \max_{w' \in s} f_{sem}(w, w'). \quad (2)$$

The function f_{sem} returns the semantic similarity between two terms. As terms are represented as vectors in our case, a natural choice for f_{sem}, which we use in our experiments in Section 4, is the cosine similarity between the two vectors.

As is apparent from (1), we always take the longest short text of the two as a reference when calculating f_{sts}. We do so for two reasons. Firstly, we want f_{sts} to be symmetrical. Calculating the semantic similarity between two short texts should yield the same score regardless of their order. Secondly, the reason why the longest of the two short texts is summed over is that we do not want terms to be overlooked. Suppose we have two texts, where one consists of a subset of terms contained in the other. If the shortest text would be taken as a reference this would lead to a perfect score. However, if we take the longest text as a reference, the incongruity between the texts does have its bearing on the score, as desired.

We should note that, although (1) bears a superficial resemblance to the BM25 formula, it in fact models something completely different. We borrow the b and k_1 parameters that have a smoothing effect, together with the length normalisation: the average sentence length, $avgsl$, in our case. The key difference, however, lies in the introduction of semantic similarity term in the formula. Where a tf*idf weighting scheme relies on literal matches between the query and documents it matches, we are, in the present setting, interested in particular in semantic matches. By using (2) for calculating semantic similarity, the maximum similarity of terms in s_s is taken into account for every term in s_l.

One interpretation of f_{sts} is that it allows for non-literal, semantic matching. As noted above, in a tf*idf weighting scheme, terms only contribute to the score if they match perfectly. In f_{sts} all terms contribute, with the semantically most related ones contributing most.

An alternative interpretation of f_{sts} with (2) is as a word alignment method. As a $\max$ is being computed in (2) over all words in a sentence, semantically close words are aligned to one another. In this way f_{sts} bears similarity to other alignment approaches such as [5, 19, 29].

Yet another alternative view is that f_{sts} applies saliency weighting to a semantic network. If we interpret the dots in Figure 1 as vertices of a graph, the $\max$ in (2) draws edges between the vertices and weights them according to (1). As a result, a mismatch between two terms such as the far left and right ones in Figure 1 is of little consequence if both have a low IDF score (e.g., they are function words). If they are salient however, the mismatch has a larger impact.

As can be seen from (1) the f_{sts} score is a sum over $|s_l|$ terms. However, rather than giving the overall score to the final learning algorithm, we want to capture more information about the way the score is composed. Therefore, we make bins of its summands and normalise by the number of summands, so the value for every bin represents the percentage of summands in (1) between the minimum and maximum values for that bin.

3.1.2 *Unweighted semantic network*

To convey as much information as possible to the final classifier we also construct an unweighted semantic network. For a short text pair (s_1, s_2), we compute the cosine similarities in the semantic space between all terms in short text s_1 and all terms in s_2. This gives us a matrix of similarities between the terms in s_1 and s_2. From this matrix we compute two sets of features.

Firstly, we take all similarities and bin them. If we think of the word embeddings as nodes in a graph this would correspond to an fully connected, unweighted, bipartite graph. In Figure 1 this would be represented by connecting every opaque dot to every transparent dot.

[1] This is sometimes referred to as vector BOW approach

Secondly, the maximum similarity for every word is computed, and bins are made of these maximum values. In this way, small distances between words (such as the top and bottom ones in Figure 1) end up in the same bin, while outliers (the ones at the far left and right) end up in a separate bin.

3.2 Text level features

3.2.1 *Distance between vectors means*

As noted previously, a standard way of combining word embeddings to capture the meaning of longer pieces of text is to take the mean of the individual term vectors. This aggregation over terms gives us one vector per sentence. We calculate both the cosine similarity and the Euclidean distance between the vectors for every sentence pair in the test set.

3.2.2 *Bins of dimensions*

The cosine similarity between two vectors can be interpreted as an aggregation over the differences per dimension. As such, it does not capture all information about the similarities or differences between the two vectors. For example, taking the cosine similarity between two vectors that are highly similar in many dimensions and quite different in few, could lead to the same result as taking the cosine similarity between two vectors that differ slightly in all dimensions. Intuitively though, these are two different situations. In order to capture this intuition, we make bins of the number of dimensions in the mean vector of s_1 and the mean vector of s_2 that match within certain limits. See §4.3 for the exact values.

4. EXPERIMENTAL SETUP

The primary focus of our experiments is to determine how our method, which relies solely on semantic features, compares to other methods that work under the same conditions, and to methods that do rely on external sources of structured semantic knowledge, linguistic tools and handcrafted rules. To do so, we perform experiments on the MSR Paraphrase Corpus [15, 35], the evaluation set most commonly used for this purpose.

As described in the previous section, we compute features from word embeddings, which are obtained from large amounts of unlabelled data. A practical feature of word embeddings is that word vectors computed on a large corpus can be made available, without the necessity of disclosing the entire training corpus as well (which can be problematic due to copyright issues). In our experiments we compute features from four publicly available sets of word embeddings (see Section 4.2.1 and Section 4.2.2 for more details). As the word vectors are not trained by ourselves, we refer to them as Out-of-the-Box (OoB).

We distinguish between two feature sets. The *saliency-weighted semantic network* features capture information about the similarity of the distribution of word vectors in the two sentences (Section 3.1.1). The *unweighted* features are calculated from the unweighted semantic network (Section 3.1.2) and the means of the word embeddings of both sentences (Section 3.2). Our hypothesis is that saliency-weighted semantic networks can add valuable features for a classifier that learns to predict semantic similarities between short texts. To verify this hypothesis we perform experiments without the features based on the saliency-weighted semantic networks, and with the saliency-weighted semantic network features added.

Additionally, as there are many parameters that have an impact on the word embeddings we use to construct features, we want to investigate which settings lead to word embeddings best suited for our approach. As it is not possible to do this with out-of-the-box vectors, we construct our own vectors from a publicly available text corpus (see 4.2.3 for details). As we use the features derived from these additional sets of word embeddings supplementary to the OoB features, we refer to them as *auxiliary*.

For the experiments with features derived from the OoB vector sets, the only hyper-parameters of our model are the regularisation parameters of the learning algorithm. We choose their optimal setting by cross validating on the training data with folds of 10 examples. The experiments including the auxiliary vectors are aimed at demonstrating the potential of our method and the effect of the parameter settings. Therefore, we show the best results obtained across all settings and discuss the individual parameter settings in detail.

4.1 Learning algorithm

As discussed in Section 3 we use the features described above to represent sentence pairs in the training material and we train a supervised learning algorithm on these features. As the MSR Paraphrase Corpus is annotated with binary labels (see Section 4.6.1) we use a classifier for prediction. In particular, we use Support Vector Classifier (SVC) with a Radial Basis Function (rbf) kernel because the feature space is not necessarily linear.

4.2 Word embeddings

For ease of comparison with other approaches using word embeddings, we use four sets of vectors that are publicly available (two word2vec sets and two GloVe sets) which we refer to as Out-of-the-Box sets (OoB). Additionally we train our own word vectors, both with word2vec and GloVe, and perform runs with different settings for both algorithms (the auxiliary vectors).

Once word embeddings have been trained on a corpus, there is no way to fold terms that were not observed during training into the semantic space. One way of dealing with these out-of-vocabulary (OOV) words when calculating the features described below is to simply ignore them. However, it is possible that important semantic information is present specifically in these new words. For example, names of persons or organisations occurring in a test set, which are likely to be semantically relevant, might have been absent from the training data. Therefore, following, e.g., [24], we map OOV words to random vectors, while remembering which OOV word maps to which random vector.

The intuition behind this simple scheme is the following. If two texts are being compared in which two different OOV names appear, this incongruity would go unnoticed if the OOV terms would be ignored. Likewise, if the same OOV term is observed in two texts being compared, this contributes to the similarity score between the two (while it would be silently ignored otherwise).

4.2.1 *OoB: Word2vec*

Mikolov et al. [33] experiment with several settings of the word2vec algorithm to produce the highest quality word embeddings. The resulting vectors have been made publicly available.[2] The vectors are 300-dimensional and were trained on a corpus of about 100 billion words.

Baroni et al. [7] compare word2vec word embeddings to traditional distributional semantics approaches. The best performing vectors were released by the authors.[3] The vectors are 400-dimensional, a 5-word context window was used, with 10 negative samples and subsampling.

[2]See https://code.google.com/p/word2vec/

[3]See http://clic.cimec.unitn.it/composes/semantic-vectors.html

4.2.2 OoB: GloVe

In [34] an algorithm is proposed for deriving word embeddings optimised especially for word analogy and similarity tasks. As the GloVe algorithm differs from the word2vec algorithm, it is interesting to see if and how GloVe word embeddings behave differently from word2vec vectors when applied to the task of short text similarity. In our experiments we use two sets of publicly available GloVe vectors. Both sets are 300-dimensional. The word vectors of the first set were trained on a very large corpus of 840 billion tokens while the other set was trained on a 42 billion token corpus.[4]

4.2.3 Auxiliary word embeddings

As we are interested in the utility of the vectors and what settings work best for the task of predicting short text similarity, we calculate auxiliary word embeddings both with the word2vec algorithm and with GloVe. We train word embeddings on a publicly available data set released by INEX.[5] The corpus contains 1.2 billion tokens.

The word2vec algorithm has several parameters: the architecture (CBOW or Skip-gram), word sampling threshold, whether or not to apply hierarchical softmax and the number of negative examples. Preliminary experiments indicated that a sampling threshold of 10^{-5} is most robust across settings. We use the default window width of 5 and vector dimensionality of 300.

For the auxiliary GloVe vectors we use the same dimensionality of 300. We set the number of training iterations to 100 as is suggested in [34] for training vectors of dimension 300 and up. There are two parameters in particular to experiment with: (1) the exponent of the weighting function used in the cost function, which we set to any of $[.1, .5, .75, .9]$, and (2) the cut-off in the weighting function, which we set to any of $[10, 50, 100, 500, 1000]$.

Preprocessing of the corpus consists of tokenization with NLTK sentence splitter and token splitter [10] with additional removal of non-ascii quotes and non-word characters. No stemming or stopping is carried out. All text is lowercased.

4.3 Parameter settings

As discussed in Section 3.1 and Section 3.2 a binning approach is used for most features. We use three bins in most cases, where one bin is meant to capture highly similar values, one bin is for the medium values and the third bin is for very dissimilar values. The values were obtained by examining the raw features for the training material. For features calculated from the saliency-weighted semantic network, the values are 0–.15, .15–.4, .4–∞. For the unweighted semantic network features the values are -1–.45, .45–.8, .8–∞ (the same values are used for when all similarities are taken into account, as when only the maximum similarities are considered). For the bins of dimensions, preliminary experiments showed that a four-bin approach, with two bins for similar and highly similar values worked slightly better than a three-bin approach. We use values $-\infty$–.001, .001–.01, .01–.02, .02–∞.

As an extensive tuning of the parameters k_1 and b is beyond the scope of this paper we use the default settings of $k_1 = 1.2$ and $b = 0.75$ when computing f_{sts} in our experiments. The IDF values were calculated from the INEX data set described above.

4.4 Feature sets

All feature sets are calculated per set of word embeddings. Hence, for 3 saliency-weighted semantic network bins, 2×3 unweighted semantic network bins, 2 distances and 4 dimensional bins, we have 15 features per set of word embeddings, and 60 features in total per sentence pair, when, e.g., the 4 OoB sets are used.

4.5 Baselines

As discussed in §2 the systems for detecting short text similarity as described in [20, 21, 38] are natural baselines to our method as they work under the same conditions, i.e., no external sources of structured semantical knowledge are used and no prior knowledge of natural language (such as parse trees) is required.

4.6 Evaluation

We first describe the evaluation set used in our experiments. Then we discuss the associated evaluation metrics.

4.6.1 Evaluation sets

We use the Microsoft Research Paraphrase Corpus data set [15, 35] in our primary experiments in Section 5 as it is commonly used for evaluation in short text similarity tasks [4, 17, 20, 21, 31, 38]. The set consists of sentence pairs judged for semantic similarity on a binary scale. The annotator guidelines allowed for an interpretation of semantic similarity that went beyond strict semantic identity, as using the latter notion would yield only trivial examples. The set consists of 5801 sentence pairs in total, divided in a training set of 4076 and a test set of 1725 examples.

Other evaluation sets.

Li et al. [27] present a data set comprising 65 pairs of dictionary glosses extracted from two sources. The set is used for evaluation in, e.g., [18, 21]. We omit this set in our experiment because of its limited size.

The task of semantic textual similarity was part of the SemEval 2012 and SemEval 2013 campaigns [1, 2]. The SemEval data is impractical for evaluation in a supervised learning setting with a substantial number of features, as the training data is limited (maximally 750 training examples per subset in the SemEval 2012 data) which leads to overfitting. Additionally, more recent work in semantic textual similarity is evaluated on the MSR Paraphrase corpus. For these two reasons, we evaluate our methods for calculating semantic text similarity on the latter.

4.6.2 Evaluation metrics

As the MSR Paraphrase Corpus has binary annotations, accuracy is the metric most often applied, together usually, with precision, recall and F_1 [4, 17, 20, 31].

5. RESULTS AND ANALYSIS

In this section we show the results of our experiments. We are interested in answering two questions. Firstly, we want to see if our method performs better than the baseline methods that work under the same conditions. Secondly, we want to know if a semantics-only approach, without access to external sources of knowledge, can yield results comparable to the state-of-the-art methods that do use external semantic knowledge bases and/or features based on computationally more involved processes as syntactic parsing.

In Table 1 results of our experiments are listed. For convenience, the results from the baseline methods, as reported in the literature, are displayed in the top three rows.[6] The rows marked 'unwghtd' use all features described above, but for the features based on the saliency-weighted semantic network. The rows marked 'un-

[4]Both sets of vectors can be downloaded from http://nlp.stanford.edu/projects/glove/

[5]The data consists of an English Wikipedia dump from November 2012. It was released as a test collection for the INEX 2013 tweet contextualisation track [8]

[6]No precision and recall numbers were reported in [20].

Table 1: Results on the MSR Paraphrase Corpus set. The rows marked 'unwghtd' display results for runs based on all features, but for the saliency-weighted semantic network features. The rows marked 'unwghtd + swsn' display results for runs that had features based on saliency-weighted semantic network added as well. Results marked† are significantly different from the best performing OoB run (two-tailed paired t-test, p-value $< .007$)

Baseline methods		Acc.	p	r	F_1
Convolutional NNs [20]		.699	–	–	.809
VSM [38]		.710	.710	.954	.814
Corpus-based PMI [21]		.726	.747	.891	.813
Our method	**Features**	**Acc.**	**p**	**r**	F_1
OoB	unwghtd	.746	.768	.882	.822
OoB	unwghtd + swsn	.751	.768	.896	.827
OoB + aux w2v	unwghtd	.754	.770	.897	.829
OoB + aux w2v	unwghtd + swsn	.757	.775	.894	.830
OoB + aux Glv	unwghtd	.756	.774	.894	.830
OoB + aux Glv	unwghtd + swsn	.758	.771	.907	.833
OoB + both aux	unwghtd	.762†	.780†	.893†	.833†
OoB + both aux	unwghtd + swsn	.766†	.781†	.906†	.839†

wghtd + swsn' use the both the 'unwghtd' features and the saliency-weighted semantic network features.

5.1 Using out-of-the-box vectors

For the rows marked 'OoB' only out-of-the-box word embeddings were used of the four sets described in Section 4.2.1 and Section 4.2.2. The settings for the regularisation parameters of the classifier are determined by cross validating on the training set. For the experiment with only unweighted features, the hyper-parameter settings are C $= 10^8$ and gamma $= 10^{-5}$. For the experiment including the saliency-weighted semantic network features we have C $= 10^6$ and gamma $= 10^{-4}$.

Table 1 shows that the result when using only publicly available, out-of-the-box word vectors (the rows marked OoB), surpass all the baselines.[7]

An important observation is that the best scoring approach on this data set using WordNet-based features, to our knowledge, reports an accuracy of .741 and an F_1 score of .824 [17]. As we can see, our generic approach with only publicly available vectors, without any tuning or optimisation, outperforms this method. This is an important finding, as it shows that for computing semantic similarity, the labour-intensive construction of a rich semantic knowledge source such as WordNet is not a necessity. As an aside, our finding supports a claim made in [5] in a different context of matching texts of different lengths, "that traditional knowledge-based features are cornered by novel corpus-based word meaning representations."

5.2 Using auxiliary vectors

To show the potential of our method we report the results when, next to the OoB vectors, the auxiliary vectors—generated from the embeddings trained on the INEX data as described in Section 4.2.3—are used. The bottom rows in Table 1 show the best results obtained with these vectors.

The results of the experiments with the auxiliary vectors are consistently better than the baselines and the results with only OoB vectors, both in terms of accuracy and F_1.

If we compare the results of our method that only uses semantic features, to the current state-of-the-art methods, which rely on linguistic analysis and handcrafted features, we observe that when optimal settings are used, our method can outperform the tree kernel approach described in [4]. This method uses features derived from dependency parses, and yields an accuracy of .753 on this data set (no precision, recall and F_1 score are reported). Furthermore, our top-performing run, the bottom row in Table 1, shows results comparable results presented in [39], based on dynamic pooling and unfolding recursive auto-encoders trained on parse trees—that has an accuracy of .768 and F_1 of .836. Our top-performing run does slightly better in terms of F_1 and slightly worse with respect to the accuracy. It is important to note, however, that the results in [39] are only achieved when, next to the general neural network-based method, several handcrafted features are added, which are designed especially for the evaluation set at hand (the features are primarily dealing with representation of numbers). Interestingly, our method performs better than the neural network-based approach in [39] when the latter is run without the test-set-specific handcrafted features, in which case it yields an accuracy of .726.

The best performance on the MSR Paraphrase Corpus, to our knowledge, is presented in [22]. Matrix decomposition is performed on a co-occurrence matrix, and saliency weighting is applied, where the saliency weight per word or n-gram is optimised on the training data. A dependency parser is used to generate the ngrams. The best performance is obtained by performing matrix decomposition on the training and test data combined. But even if only the training set is used, an accuracy of .786 and .846 F_1 is attained, when no additional hand-crafted features are used. This result is better than ours in terms of accuracy, while in terms of F_1 the scores are comparable.

The word2vec vectors and GloVe vectors, when added separately, yield better performance. This is particularly noteworthy as it shows that high-quality word embeddings can be produced for the present setting by both algorithms, even when the corpus used for training was substantially smaller than what is commonly used (namely ∼1B tokens, against 3B and 100B tokens for the OoB word2vec sets and 42B and 840B for the OoB GloVe vectors).

When both the auxiliary word2vec and GloVe vectors are added to the OoB vectors—the rows marked 'OoB + both aux'—we see another increase in performance specifically in terms of precision. These results are significantly different from the OoB run with both feature sets. The results in the 'OoB + both aux' rows also surpass the results in the 'OoB + aux w2v' and 'OoB + aux Glv' separately. This is particularly interesting, as it indicates that, while the performance of both vectors sets on their own is comparable, the two models capture different semantic information.

Finally, an overall observation from Table 1 is that adding the saliency-weighted semantic network features consistently yields better performance. We note that this goes against an observation in [17]: *"Since experiments with document specificity weightings (such as tf-idf) had shown that using these factors actually reduced performance no such weighting factor was used here."*

5.2.1 Parameter analysis

Space constraints prevent us from providing an extensive overview of results across all parameter settings. An important observation, however, is that although the settings matter, few consistent

[7]We cannot calculate statistical significance between our results and the baseline results as for the appropriate test—a matched-pairs t-test—we need to compare our output to the outputs of the baseline systems, which are not publicly available (only the aggregate results are).

patterns emerge. To illustrate, for the 'OoB + both aux' setting, the worst performance in terms of accuracy across parameters, with optimal regularisation parameters was .730, which is worse than the performance with only OoB vectors, but above baseline performance. For the auxiliary word2vec vectors, negative sampling seems to be beneficial in general, with a value of 10 being a robust choice. Both the choice of architecture (CBOW or Skip-gram) and applying hierarchical softmax or no seems to be of little consequence.

For the GloVe vectors different values for the exponent of the weighting function and the cut-off in the weighting function were used. The moderate values (.5, .75 for the exponent, and 50 or 100 for the maximum cut-off) yielded the best results.

Lastly, the regularisation parameters (C and gamma) of the classifier are hyper-parameters of our model. We use large values for C (in the range of 10^6–10^9 depending on the number of features) and small values for gamma (10^{-4}, 10^{-5}). Not surprisingly, the setting of these parameters in particular has substantial repercussions on the final performance. To illustrate again, the worst performance with optimal features but across regularisation parameters was .690, which is lower than the lowest baseline. The worst overall performance (worst features and worst regularisation) is roughly equal, at .685, which suggests that the harm is primarily caused by suboptimal regularisation.

These findings indicates that both algorithms for generating word embeddings, word2vec and GloVe, are robust across reasonable parameter settings. While it is important to find the optimal combination of all parameters, the settings for the word embeddings matter less than regularising the learning algorithm.

5.2.2 *Feature importance*

In addition to studying the effect of the different sets of word embeddings as discussed above, it is interesting to see how different sets of features affect the performance. To analyse the effect per feature set we perform an ablation study, where we leave out a set of features for all word embedding sets we calculate features from.[8]

Table 2: Ablation study results

Omitted feature set	Acc.	p	r	F_1
max unweighted sn bins	.739	.766	.874	.817
swsn bins	.741	.768	.874	.818
dimension bins	.746	.767	.886	.822
all unweighted sn bins	.747	.763	.898	.825
distances	.759	.778	.892	.831

In Table 2 the results are shown for leaving out different feature sets, sorted by accuracy. All other settings (the word2vec and GloVe parameters for the auxiliary vectors and the regularisation parameters for the classifier) are identical to the ones used for the top performing run in Table 1 (bottom row).

As can be seen from Table 2 leaving out the word alignment methods, weighted by saliency or not, has the most dramatic effect on performance. This indicates that aligning words is a successful strategy for determining semantic similarity between short texts.

An interesting observation is that leaving out the distance features has the least effect on performance. As noted above, these features measure the distance between the means of the word vectors in both sentences, and are a default method for going from word-level to sentence-level, applied, next to other methods, in e.g., [5, 20, 40]. Table 2 does suggest that binning the differences between dimensions of the mean vectors, as proposed in this paper, increases the gain obtained from them.

[8]Note that it is not possible to use the feature weights as a proxy for feature importance, as this only works for linear kernel functions, and we use a classifier with an RBF kernel.

5.3 Error analysis

To see whether our method of computing semantic textual similarity for short texts is biased we perform an error analysis concerning two important attributes of the test data: sentence length and lexical overlap.

5.3.1 *Performance across sentence length*

Figure 2 shows an overview of the results of our experiments divided by sentence length.

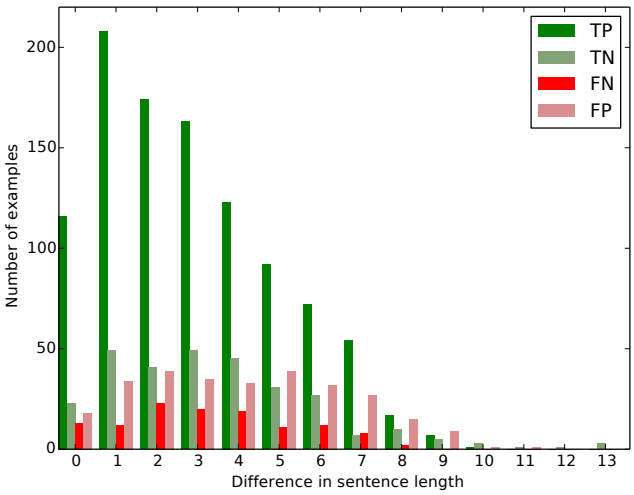

Figure 2: Results 'OoB + both aux – unwghtd + swsn' run divided by difference in sentence length (measured in words). TP: true positives, TN: True negatives, FP: false positives, FN: false negatives. (Best viewed in color.)

As expected, sentences that are alike in in terms of length are easier to perform well on, as reflected in the figure by the large bulge at the left side of the scale for the true positives. The hump for true negatives is less pronounced, which is easily explained by a lower frequency of negative examples in the test set.

One observation from Figure 2 is that the number of false negatives is rather constant in the left half of the figure, which means that it increases relatively with the difference in sentence length, while this is not the case for false positives. This means that the classifier has a tendency to predict semantic dissimilarity when the two input texts differ in length substantially.

Most importantly though, Figure 2 shows that the classifier always predicts the correct label in the majority of cases, regardless of the difference in sentence length.

5.3.2 *Performance across levels of lexical overlap*

As our saliency-weighted semantic network features perform semantic, rather than lexical, word pairings, it is interesting to see how our method performs across different levels of lexical (i.e., *literal*) overlap between the sentence pairs in our test collection.

In Figure 3 we show an overview of the results across different levels of lexical overlap of our best performing run on the MSR Paraphrase Corpus (OoB + both aux – unwghtd + swsn; see bottom row in Table 1).

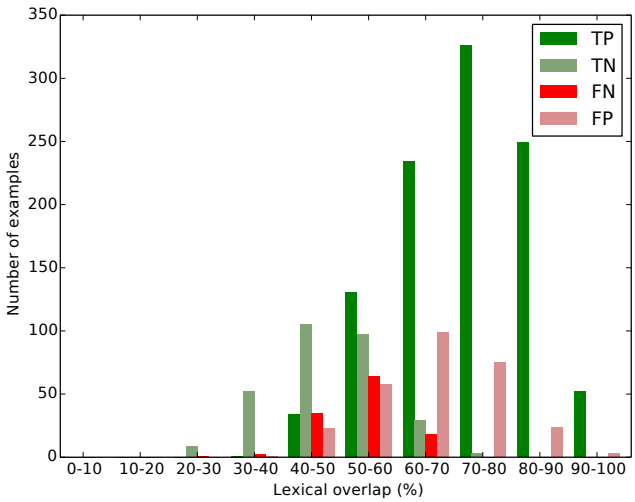

Figure 3: Results for the 'OoB + both aux – unwghtd + swsn' run, grouped by percentage of lexical overlap between test sentences in the MSR Paraphrase Corpus. (Best viewed in color.)

In Figure 3 we can clearly see four different distributions of results, where, e.g., the TP results peak at 70–80% and the TN results peak at 40–50%.

As is to be expected, the 'OoB + both aux – unwghtd + swsn' run is right most times at high levels of lexical overlap. If it is wrong it produced false positives, i.e., it predicts semantic similarity too often, which is easily explained by the high similarity between the sentences. An interesting glitch is the tiny FP bar at the far right of the figure (90%-100% overlap), which denotes 3 cases of high lexical overlap, where the annotators judged the test sentences not to be semantically similar, while our method for predicting short text semantic similarity did. Indeed, the differences are rather subtle, as illustrated by this example pair:

> *Air Canada, the largest airline in Canada and No. 11 in the world, has been under court protection from creditors since April 1*

and

> *The No. 11 airline in the world, Air Canada has been under court protection from creditors since April 1.*

There is a peak in the middle for the FN bars, at 50–60% overlap, where our method cannot make up for the relatively low level of lexical overlap, and mistakenly predicts semantic dissimilarity. Interestingly, though, the bars show that our algorithm, even for these difficult cases, still makes the correct prediction in the majority of cases. An additional noteworthy observation is that even when the majority of words in the sentences that make up the test pairs are different, at 40–50% overlap level, our algorithm still produces true positives, next to true negatives. This clearly shows the benefit of semantic matching over lexical matching. More importantly, it shows a meaningful distinction can be made by the algorithm, even for these non-trivial cases.

The final important observation from Figure 3 is that our method, the 'OoB + both aux – unwghtd + swsn' run, is right in the majority of cases across all levels of lexical overlap.

6. CONCLUSIONS AND FUTURE WORK

We have described a generic and flexible method for semantic matching of short texts, which leverages word embeddings of different dimensionality, obtained by different algorithms and from different sources. The method makes no use of external sources of structured semantic knowledge nor of linguistic tools, such as parsers. Instead it uses a word alignment method, and a saliency-weighted semantic graph, to go from word-level to text-level semantics. We compute features from the word alignment method and from the means of word embeddings, to train a final classifier that predicts a semantic similarity score.

We demonstrate on a large publicly available evaluation set that our generic, semantics-only method of computing semantic similarity between short texts outperforms all baseline approaches working under the same conditions, and that it exceeds all approaches using external sources of structured semantic knowledge that have been evaluated in this dataset, to our knowledge.

An important implication of our results is that distributional semantics has come to a level where it can be employed by itself in a generic approach for producing features that can be used to yield state-of-the-art performance on the short text similarity task, even if no manually tuned features are added that optimise for a specific test set or domain. Furthermore, the word embeddings, when employed as proposed above, substitute external semantic knowledge and make human "feature engineering" unnecessary. As our method does not depend on NLP tools, it can be applied to domains and languages for which these are sparse.

It is interesting to see how other fields of research that deal with large corpora of unstructured text can benefit. For example, in automatically created probabilistic knowledge bases (e.g., [16]) triples are extracted from an input corpus and have a confidence score associated with them based on the number of sentences in the corpus describing the relation in the triple. Short text similarity can be used to improve this confidence score.

An evident limitation of calculating meta-features in the manner we propose, i.e., from averaged word vectors and word alignments, is that the order of words is not taken into account. This goes for any bag-of-words model, of course. The reason the difference between word-order-aware models and bag-of-word models is not apparent from the results on sentence similarity tasks over the years might be that the commonly used evaluation sets do not contain enough sentence pairs (if at all) in which word order is a crucial factor.

Acknowledgments.

We thank our anonymous reviewers for their valuable feedback, which helped to improve the paper. This research was supported by Amsterdam Data Science, the Dutch national program COMMIT, Elsevier, the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 312827 (VOX-Pol), the ESF Research Network Program ELIAS, the HPC Fund the Royal Dutch Academy of Sciences (KNAW) under the Elite Network Shifts project, the Netherlands eScience Center under project number 027.012.105, the Microsoft Research Ph.D. program, the Netherlands Institute for Sound and Vision, the Netherlands Organisation for Scientific Research (NWO) under project nrs 727.011.005, 612.001.116, HOR-11-10, 640.006.013, 612.066.930, CI-14-25, SH-322-15, the Yahoo! Faculty Research and Engagement Program, and Yandex.

7. REFERENCES

[1] E. Agirre, M. Diab, D. Cer, and A. Gonzalez-Agirre. Semeval-2012 task 6: A pilot on semantic textual similarity. In *SemEval 2012*, 2012.

[2] E. Agirre, D. Cer, M. Diab, A. Gonzalez-Agirre, and W. Guo. sem 2013 shared task: Semantic textual similarity, including a pilot on typed-similarity. In **SEM 2013*, 2013.

[3] R. M. Aliguliyev. A new sentence similarity measure and sentence based extractive technique for automatic text summarization. *Expert Systems with Applications*, 2009.

[4] P. Annesi, D. Croce, and R. Basili. Semantic compositionality in tree kernels. In *CIKM 2014*, 2014.

[5] C. Banea, D. Chen, R. Mihalcea, C. Cardie, and J. Wiebe. Simcompass: Using deep learning word embeddings to assess cross-level similarity. *SemEval 2014*, 2014.

[6] D. Bär, C. Biemann, I. Gurevych, and T. Zesch. Ukp: Computing semantic textual similarity by combining multiple content similarity measures. In *SemEval 2012*, 2012.

[7] M. Baroni, G. Dinu, and G. Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL 2014*, 2014.

[8] P. Bellot, G. Kazai, M. Preminger, M. Trappett, A. Doucet, M. Koolen, E. SanJuan, A. Trotman, S. Geva, A. Mishra, R. Schenkel, M. Sanderson, S. Gurajada, V. Moriceau, X. Tannier, F. Scholer, J. Kamps, J. Mothe, M. Theobald, and Q. Wang. Report on INEX 2013. *SIGIR Forum*, 47(2): 21–32, 2013.

[9] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning*. Springer, 2006.

[10] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python*. O'Reilly Media, Inc., 2009.

[11] S. Canuto, T. Salles, M. A. Gonçalves, L. Rocha, G. Ramos, L. Gonçalves, T. Rosa, and W. Martins. On efficient meta-level features for effective text classification. In *CIKM 2014*, 2014.

[12] T. A. Coelho, P. P. Calado, L. V. Souza, B. Ribeiro-Neto, and R. Muntz. Image retrieval using multiple evidence ranking. *IEEE Transactions on Knowledge and Data Engineering*, 16 (4):408–417, 2004.

[13] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML 2008*, 2008.

[14] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[15] B. Dolan, C. Quirk, and C. Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *COLING 2004*, 2004.

[16] X. L. Dong, K. Murphy, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD 2014*, 2014.

[17] S. Fernando and M. Stevenson. A semantic similarity approach to paraphrase detection. *CLUK 2008*, 2008.

[18] R. Ferreira, R. D. Lins, F. Freitas, S. J. Simske, and M. Riss. A new sentence similarity assessment measure based on a three-layer sentence representation. In *DocEng 2014*, 2014.

[19] L. Han, A. Kashyap, T. Finin, J. Mayfield, and J. Weese. UMBC EBIQUITY-CORE: Semantic textual similarity systems. In **SEM-2013*, 2013.

[20] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *NIPS 2014*, 2014.

[21] A. Islam and D. Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *TKDD*, 2008.

[22] Y. Ji and J. Eisenstein. Discriminative improvements to distributional sentence similarity. In *EMNLP*, 2013.

[23] V. Jijkoun and M. de Rijke. Recognizing textual entailment using lexical similarity. In *Proceedings Pascal 2005 Textual Entailment Challenge Workshop*, 2005.

[24] Y. Kim. Convolutional neural networks for sentence classification. In *EMNLP 2014*, 2014.

[25] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[26] H. Li and J. Xu. Semantic matching in search. *Foundations and Trends in Information Retrieval*, 7(5):343–469, 2014.

[27] Y. Li, D. McLean, Z. A. Bandar, J. D. O'shea, and K. Crockett. Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1138–1150, 2006.

[28] M. C. Lintean and V. Rus. Measuring semantic similarity in short texts through greedy pairing and word semantics. In *FLAIRS Conference*, 2012.

[29] E. Marsi, H. Moen, L. Bungum, G. Sizov, B. Gambäck, and A. Lynum. Ntnu-core: Combining strong features for semantic similarity. **SEM-2013*, 2013.

[30] D. Metzler, S. Dumais, and C. Meek. Similarity measures for short segments of text. In *ECIR 2007*, 2007.

[31] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, 2006.

[32] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[33] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS 2013*, 2013.

[34] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. *EMNLP 2014*, 2014.

[35] C. Quirk, C. Brockett, and W. B. Dolan. Monolingual machine translation for paraphrase generation. In *EMNLP 2004*, 2004.

[36] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[37] F. Šarić, G. Glavaš, M. Karan, J. Šnajder, and B. D. Bašić. Takelab: Systems for measuring semantic text similarity. In **SEM-2013*, 2012.

[38] P. Shrestha. Corpus-based methods for short text similarity. *Rencontre des Étudiants Chercheurs en Informatique pour le Traitement automatique des Langues*, 2011.

[39] R. Socher, E. H. Huang, J. Pennin, C. D. Manning, and A. Y. Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS 2011*, 2011.

[40] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS 2013*, 2013.