Learning Vector Representations for Similarity Measures

Wen-tau Yih
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
scottyih@microsoft.com

Christopher Meek
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
meek@microsoft.com

Abstract

Conventional vector-based similarity measures consider each term separately. In methods such as *cosine* or overlap, only identical terms occurring in both term vectors are matched and contribute to the final similarity score. Non-identical but semantically related terms, such as "car" and "automobile", are completely ignored. To address this problem, we propose a novel approach that learns a new vector construction from the original term vectors. The weight of each element in the output vector is a linear combination of the term-weighting scores of related terms. Depending on the configuration, our method can learn extended term vectors using the same vocabulary, as well as "concept" vectors with reduced dimensionality. In both settings, it outperforms existing methods significantly in the task of measuring document similarity, reflected in various metrics consistently.

1 Introduction

Measuring text similarity has many useful applications and has been studied extensively in both NLP and IR communities. For example, mixtures of corpus and knowledge based methods have been invented for judging word similarity (Lin, 1998; Mihalcea et al., 2006; Hughes and Ramage, 2007; Agirre et al., 2009). Large-scale approaches that leverage the Web-size corpus are proposed for extending entity sets (Pennacchiotti and Pantel, 2009; Vyas and Pantel,). For the Web search scenario, measuring the similarity between short text segments has been applied to tasks such as query suggestions (Jones et al., 2006; Sahami and Heilman, 2006). Judging the degree of similarity between documents is also fundamental to classical IR problems such as document retrieval (Manning et al., 2008). In all these applications, the vector-based similarity method is the most widely used. Term vectors are first constructed to represent the original text objects, where each term is associated with a real-valued weight indicating its importance. A preselected function operating on these vectors, such as cosine or Jaccard, is used to output the final similarity score. Such approach has not only been shown effective, but also enjoys several advantages in practice. For instance, text objects can be stored in term vectors individually. Fast similarity search can be done using pruned inverse index or random projection methods (Ravichandran et al., 2005).

While the quality of a similarity measure depends on both the term selection strategy and the term-weighting function, the latter clearly receives more attention, as numerous term-weighting schemes have been invented. In contrast, the selection of active terms (i.e., terms that have non-zero weights) is determined by the raw text. A term typically has zero weight if it does not occur in the original text, such as a document or some context extracted from a large corpus. Given that efficient similarity functions match only identical terms, non-identical but semantically related terms are ignored in such computation. As an illustrative example, suppose the two compared term-vectors are: {*buy*:0.3, *pre-owned*: 0.5, *car*: 0.4} and {*purchase*:0.4, *used*:0.3, *automobile*:0.2}. Even though both vectors represent very similar concepts, their similarity score will

be 0, no matter whether it is computed using cosine, overlap or Jaccard. Changing term-weighting functions will not solve this problem, unless new elements with non-zero weights are created.

Traditional strategy for handling this issue is to map the terms to some concept elements. Words can be clustered first by their semantic relatedness to form vectors with categorical elements (Broder et al., 2007). The original term vectors can also be projected to some concept space via techniques like latent semantic analysis (LSA) (Deerwester et al., 1990). However, information may be lost after the mapping process. More importantly, such approaches are often trained indirectly and difficult to fine tune to best fit the similarity measuring tasks for the target domain.

In this paper, we propose a novel discriminative approach that learns the new vector representation from the original term vectors. Our method can be viewed as an instantiation of the Siamease neural network architecture (Bromley et al., 1993). The first layer corresponds to the original term vector and the middle layer is the new vector representation. Given pairs of raw term vectors and their labels (e.g., similar or not), the model is trained by optimizing the cosine scores of the output vectors. Our approach can be viewed as a natural generalization of many existing frameworks. For example, when the vocabulary remains the same, the elements in the output vector can be previously inactive terms. This is equivalent to learning a Mahalanobis distance matrix, where the sparsity can be controlled empirically or by incorporating linguistic constraints. When generating the concept vectors, the model functional form is exactly the same as LSA based on SVD term-document vector decomposition. In other words, our approach learns the latent space matrix *discriminatively*. Finally, our method is also complementary to a recently proposed term-weighting framework (Yih, 2009). When replacing the input layer with the term-level features instead of individual term weights, the model parameters of the term-weighting function can be jointly learned. As we demonstrate in both settings, our method outperforms existing methods significantly in the task of measuring document similarity and its superior performance is reflected consistently in various evaluation metrics.

2 Previous Work

In this section, we review previous work that is closely related to our proposed approach. Other related work will be surveyed in Sec. 5.

2.1 Latent Semantic Analysis

The earliest work of projecting a term vector into a concept space is Latent Semantic Analysis (LSA) (Deerwester et al., 1990). Suppose each term vector represents a document in the corpus. LSA first forms an $n \times d$ term-document matrix $\mathbf{C}$, where n is the number of terms in the vocabulary and d is the number of document in the corpus. LSA performs a singular value decomposition (SVD) on $\mathbf{C}$ of the form:

$$\mathbf{C} = \mathbf{USV}^T$$

where the diagonal elements of $\mathbf{S}$ are the ordered singular values. SVD can be used to generate a low-rank matrix approximation of $\mathbf{C}$ by retaining only the l biggest singular values in $\mathbf{S}$. More importantly, a $n \times l$ matrix can be form accordingly that maps a term vector to a concept vector of l elements.

2.2 Distance Metric Learning

Measuring the similarity between two vectors can be viewed equivalent to measuring their distance. For instance, it can be easily shown that the cosine score has a bijection mapping to the Euclidean distance of the normalized vectors. Learning similarity measures is thus closely related to the research of distance metric learning. One class of distance metric is Mahalanobis distance, which generalizes the standard squared Euclidean distance by modeling the relations of elements in different dimensions by a positive semi-definite matrix $\mathbf{A}$. Given two vectors $\mathbf{x}$ and $\mathbf{y}$, their squared Mahalanobis distance is:

$$d_{\mathbf{A}} = (\mathbf{x} - \mathbf{y})^T \mathbf{A} (\mathbf{x} - \mathbf{y})$$

Learning the Mahalanobis matrix $\mathbf{A}$ has been a main research focus recently (e.g., (Weinberger and Saul, 2009; Davis et al., 2007)). However, even the fastest approach requires $O(n^2)$ computation time, where n is the dimensionality of the input vectors, and thus impractical for high dimensional problems often encountered in the text domain. To tackle this issue,

methods of learning a low-rank Mahalanobis matrix have been proposed. In this paper, we compared our approach to the high dimension low-rank (HDLR) metric learning and the identity plus low-rank (HDILR) algorithms presented in (Davis and Dhillon, 2008). Both approaches starts from a baseline low-rank matrix and can be treated as learning the Mahalanobis matrix in the projected space.

2.3 Term-weighting Learning Framework

Observing that the quality of a similarity measure depends heavily on its term-weighting scheme Yih (2009) proposes a learning framework, TWEAK, to learn the weighting function from the data. The main idea of TWEAK is to use a parametric function over various term-level features, such as *term frequency* or *term position*, to determine the term weights. For example, a simple linear term-weighting function for term t_i from document d can be defined as:

$$tw(t_i, d) = \sum_j \lambda_j \phi_j(t_i, d) \quad (1)$$

where λ_j is the learned model parameter for feature function ϕ_j.

Although the learning target here is the term-weighting function, it is unnatural and difficult to ask human annotators to give the "true" term weights as labels. Instead, the training procedure in this framework is to tune the models to produce higher cosine scores for similar document pairs. While TWEAK has been shown to outperform other fixed term-weighting mechanisms (e.g., TFIDF), its performance is still limited by the choice of *active* terms and is not able to capture the relatedness of different terms.

3 Approach

In this section, we first describe the network designs for computing the new vectors and the corresponding similarity score, followed by some notes on model training.

3.1 Network Design

Our network consists of two layers. The first layer corresponds to the input term vector, where each node represents a term in the original vocabulary. Each of these nodes can take m different term-level features. When a pre-determined term weight like TFIDF is preferred, m can be set to 1 and the term weight can be treated as the single feature. The second layer is the learned new vector representation that captures relations among terms. Similarly, each node corresponds to an element in the new space. The similarity score of two documents is decided by the cosine score of the two corresponding second layer vectors. How the links between first layer and second layer nodes are constructed depends on configurations. In this paper, we explored two options: *extending term vectors* and *learning concept vectors*.

3.1.1 Extending Term Vectors (VL-Ext)

In this configuration, the output vector space remains the same, but the new weight of term t_k is not only decided by its original term-weighting function over the term-level features, but also depends on other related terms. In other words, previously inactive terms can now have non-zero weights. More formally, let C_k be the set of terms that are considered related to t_k, then the new term-weighting function tw' is a weighted combination of the original term weights of the related terms:

$$tw'(t_k) = \sum_{t_i \in C_k} \alpha_{ik} tw(t_i) \quad (2)$$

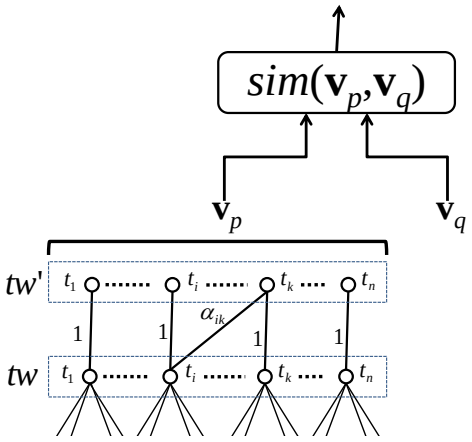
where α_{ik} is the coefficient between terms t_i and t_k. For simplicity, we assume that $t_k \in C_k$ and $\alpha_{kk} = 1$ for all k. Fig. 1 shows this design.

As the vocabulary size for text problems is typically large, naively treating each term related to all other terms creates a quadratic number of edges, and is thus impractical. Alternatively, the model sparsity is controlled by choosing the sets of related terms (i.e., C_k). This can be done by directly pairing terms belonging to two similar text objects presented in the training data, or acquired via linguistic resources such as WordNet. Notice that adding a link simply provides the opportunity for the training procedure to capture the relation between the linked terms. Except for increasing computation time, the negative effect of adding "wrong" links is limited.

3.1.2 Learning Concept Vectors (VL-CV)

In this setting, the new vectors belong to some latent *concept* space. Each node in the second layer

Figure 1: Learning extended term vectors. This configuration adds a second-layer of nodes using the same vocabulary set. The new term weights tw' are linear combinations of weights of some related terms (e.g., t_i = "car" and t_k = "automobile"). The model parameters to decide the original term weights can be learned jointly.



is a *concept* term that does not occur in the original vocabulary. As we do not explicitly define the meaning of each concept term, its weight will be a linear combination of the weights of all the terms in the original term vector. In other words, these two layers of nodes form a complete bipartite graph as shown in Fig. 2. The weight of a concept node c_k is thus defined as:

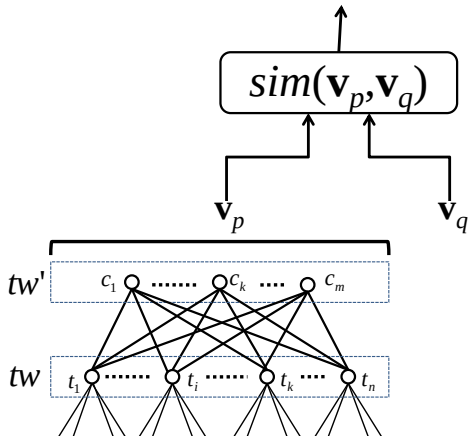
$$tw'(c_k) = \sum_{t_i \in V} \alpha_{ik} tw(t_i) \tag{3}$$

When using the linear weighted sum as defined in Eq. 3, the model functional form is exactly the same as the low-rank matrix derived by LSA. The main difference is that the coefficients α_{ik} are tuned using labeled data, which help fit the target domain better. Also note that the underlying term-weighting function (i.e., $tw(t_i)$) does not need to be pre-computed, but can instead be learned concurrently.

3.2 Model Derivation

Although training the model parameters can be done using standard back-propagation, computing the overall gradient and applied standard optimization algorithm such as L-BFGS is in fact straightforward. Below we first show how to compute the complete similarity score using concise matrix notations, and then discuss the loss function and training setting.

Figure 2: Learning concept vectors. The second layer consists of a small number of concept nodes that do not appear in the original vocabulary. The weight of each concept node is a linear combination of all the original term weights.



3.2.1 Vector Computation

Suppose the number of term-level features is m and size of the original vocabulary $\mathcal{V}$ is n. Each text object (e.g., a document) can be represented by a large sparse $m \times n$ matrix $\mathbf{\Phi}$, where column k is the feature vector of term t_k:

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1(t_1) & \cdots & \phi_1(t_n) \\ \phi_2(t_1) & \cdots & \phi_2(t_n) \\ \vdots & \cdots & \vdots \\ \phi_m(t_1) & \cdots & \phi_m(t_n) \end{bmatrix}_{m \times n}$$

To alleviate the out-of-vocabulary issue, the model parameters on the first layer are shared. The model weights for the term-level features can be represented by a size m column vector $\mathbf{\Lambda}$:

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_m \end{bmatrix}^T$$

With linear term-weighting functions, the corresponding term vector $\mathbf{F}$ is thus $\mathbf{\Phi}^T\mathbf{\Lambda}$, which is the values of the first-layer nodes in our network.

Suppose the second layer contains l nodes. All the coefficients α_{ik} of the links between the nodes in the first and second layers form an $n \times l$ matrix $\mathbf{A} = [\alpha_{ik}]_{n \times l}$. Notice that in the setting of learning extended vectors (VL-Ext), $l = n$ and the diagonal

elements of $\mathbf{A}$ are all 1's. The matrix is also very sparse – most entries are 0's unless terms t_i and t_k are considered related. In the setting of learning concept vectors (VL-CV), l is much smaller than n and $\mathbf{A}$ is a dense matrix.

The new vector representation $\mathbf{G}$ is therefore:

$$\mathbf{G} = \mathbf{A}^T\mathbf{F} = \mathbf{A}^T\mathbf{\Phi}^T\mathbf{\Lambda} = \mathbf{\Phi}^{gT}\mathbf{\Lambda},$$

where

$$\mathbf{\Phi}^g = (\mathbf{A}^T\mathbf{\Phi}^T)^T = \mathbf{\Phi}\mathbf{A}$$

3.2.2 Model Training

The two model configurations, VL-Ext and VL-CV, differ only on the number of nodes and the sparseness of the links between layers. Training these models can be done using the same procedure. As in most applications, similarity scores are used to select the closet text objects given the query. Whether the similarity measure can yield better ordering is more important than the absolute scores. Therefore, we choose a pairwise learning setting by considering a pair of similarity scores (i.e., from two pairs of vector pairs) and try to tune the model so that more similar objects can have a higher score.

We use a loss function that is very close to the one proposed in (Dekel et al., 2004) for label ranking. Suppose each pair of such training examples consists one similar objects and one dissimilar objects (according to the annotation). Let Δ_i be the difference of the similarity scores of these two document pairs. The loss function L, which can be shown to upper bound the pairwise accuracy (i.e., the 0-1 loss of the pairwise predictions), is:

$$L(\mathbf{\Lambda}; \mathbf{A}) = \sum_{i=1} \log(1 + \exp(-\Delta_i)) \qquad (4)$$

In addition, Eq. 4 can be regularized by adding $\frac{\alpha}{2}||\mathbf{\Lambda}||^2$ and $\frac{\beta}{2}||\mathbf{A}||^2$ in the loss function.

Optimizing the model parameters can be done using gradient based methods, such as stochastic gradient decent or L-BFGS, as long as the similarity function is differentiable. In this work, we use the cosine function in all experiments. The gradient derivation is left in the appendix for interested readers.

4 Experiments

We compare our vector learning method with other approaches in the task of measuring document similarity. We start from introducing the data set we used and then describe the methods we compared, followed by the experimental results evaluated using various metrics.

4.1 Data

The dataset we used is the Reuters-21578 document collection, which consists of 21,578 news articles (Lewis, 1997). Among them, 10,107 documents are annotated with some *topic* labels from a list of 135 categories. As the category labels are not mutually exclusive, a document can be assigned to more than one category. We followed the standard setting to split the document collection into the training and testing subsets (Lewis, 1992), which contain 7,310 and 2,848 documents, respectively. Because the task here is to judge the document similarity rather than text classification, we created all possible pairs of documents in the training and testing document collections separately. If a pair of documents have the same topic annotation, then we treated these two documents as similar documents and assigned a label 1 to this document pair. Otherwise, the label is 0. As a result, we created 26,714,396 and 4,054,129 document pairs from the training and testing collections, respectively. We down sampled the training data to include 11,741,051 pairs of documents with balanced class distribution. 5% of them were used as development set and the rest for model training. When evaluating the models, however, all the testing document pairs were used.

4.2 Evaluation Metrics

The right choice of the evaluation metric depends on the exact application of the similarity measures. Here we discuss two possible scenarios: *classification* and *retrieval*. In the classification scenario, each example is a pair of two documents, T_p and T_q, and the system needs to answer whether these two documents are similar. Such binary prediction is often made by comparing the similarity score with a pre-selected decision threshold θ and a positive prediction is made when the similarity score $f_{sim}(T_p, T_q) > \theta$. As θ varies, the recall/precision

and the false-positive/false-negative rates of such classifiers on the testing set change as well. To have summary statistics to capture such trade-off, we report the AUC and Max-F1 scores for this setting, where the former is the area under the ROC curve and the latter finds the point on the recall-precision curve that achieves the maximum F_1 score.

In the retrieval setting, we assume that given a query document, we need to return a set of documents that cover the same topic of the query document. Using each document in the testing collection, all other documents in the test set are ranked according to their similarity score compared to the query document. Documents with top similarity scores are "retrieved" as the result. For this scenario, we report two evaluation metrics: *precision at k* and the mean averaged precision (MAP). The former judges the precision when the system is allowed to return only k documents for each query document. The latter reports the averaged precision at $k \in \{1, 2, \cdots, m_q\}$, where m_q is the number of documents in the collection that are similar to the query document.

4.3 Results

We compare the two variations of our vector learning approach with existing methods that generate output vectors in the same space.

4.3.1 Similarity based on term vectors

For learning extended term-vectors (VL-Ext), where the vocabulary set remains the same, we compare it with the raw term vector using TFIDF weighting and TWEAK-learned weighting, plus the identity plus low-rank (HDILR) algorithm (Davis and Dhillon, 2008). Notice that only VL-Ext and HDILR are able to capture the relations between terms.

In this setting, the vocabulary set consists of all the terms (i.e., unigram tokens) that occur in the training document collection. When generating the raw term vectors, only the terms that occur in the document are consider active. For TFIDF, the exact term-weighting formula we used is

$$tf(t_i, d) \cdot \log(\frac{N}{df(t_i)}) \quad (5)$$

where $tf(t_i, d)$ is the term frequency of term t_i with respect to document d, $df(t_i)$ is the document frequency of t_i and N is the number of documents used for deriving the DF table. We derive the document frequency table using documents in the training corpus. For TWEAK, the term-weighting function is a simple linear function (Eq. 1). Table 1 lists all the features we used. The input raw term vectors for HDILR uses the TFIDF vectors. As suggested in (Davis and Dhillon, 2008), the baseline low-rank matrix for this method is derived using LSA. Finally, our approach for learning extended vectors (VL-Ext) uses the same term-level features in the first layer as used by TWEAK. The related term sets C_k are formed using the term vectors learned by TWEAK that represent the training documents in the following way. For each training document pair, non-identical terms belonging to two compared term vectors are all paired fist and ranked by the product of their normalized term weights. The top 30,000 term pairs with the highest cross-product scores are then used as sets of related terms. Conceptually, the process can be viewed as choosing the most influential terms given a set of term vectors. Of course, terms paired in this way may not necessarily be semantically related. As the coefficients (i.e., α_{ik}) will be tuned during the model training process, empirically this simple term pairing method works fine[1].

Feature	Remark
Bias	an always 1 feature
TF	term-frequency
DF	document-frequency
IsCap	whether the term is capitalized
Loc	the position of the first occurrence
Len	the length of the document
QF	query log frequency
POS	part-of-speech tags

Table 1: Term-level features used in the models. The POS tags are clasped to 9 tags. The QF feature is the number of times the term is seen as a query in a commercial search engine during an 18-month period. The logarithmic values of TF, DF, Len and QF are also used as features.

For TWEAK and VL-Ext, 40,000 random pairs

[1] In experiments not reported here, we tested using WordNet to select related terms based on (Lin, 1998), but had inferior results. We suspects that this is due to the limited coverage of terms in WordNet.

Method	AUC	Max-F_1	Prec@5	Prec@10	MAP
TFIDF	0.817	0.621	0.799	0.773	0.601
TWEAK	0.890	0.728	0.838	0.812	0.625
HDILR	0.826	0.632	0.810	0.784	0.607
VL-Ext	**0.968**	**0.846**	**0.873**	**0.852**	**0.749**

Table 2: The experimental results of our approach for learning extended vectors and other baselines. Across different evaluation metrics, our method outperforms others consistently.

Method	AUC	Max-F_1	Prec@5	Prec@10	MAP
LSA	0.947	0.806	0.772	0.762	0.703
VL-CV	**0.986**	**0.915**	0.817	0.811	0.779
HDLR	0.976	0.886	**0.841**	**0.834**	**0.795**

Table 3: The experimental results of our approach for learning concept vectors and other baselines. Our approach performs better than LSA and is comparable to HDLR.

of documents pairs are selected as training examples and the maximum number of training iterations when using L-BFGS is 200. For HDILR, we use all the training document pairs with up to 100 iterations. For all these methods, regularization parameters are selected based on the best model performance on the development set.

Table 2 shows the experimental results in various evaluation metrics. As TWEAK and HDILR generalize TFIDF in different ways, where the former incorporates information other than term frequency and document frequency, and the latter captures the relations between terms, they both outperform the raw TFIDF vector representations. In comparison, leveraging both term-level features and cross-term relations, VL-Ext achieves the best results of these methods.

4.3.2 Similarity based on concept vectors

For the case of learning concept vectors, we compare our approach with LSA and the high dimension low-rank (HDLR) metric learning algorithm (Davis and Dhillon, 2008)[2]. A term-document matrix is first constructed using all the documents in the training collection, where the entry is the corresponding TFIDF score. An SVD decomposition is performed on this matrix for generating low-rank approximation. Original TFIDF term vectors are mapped to vectors in a much smaller concept space in $\mathbf{R}^k$. We tested different values for k and found that on this dataset, LSA performs the best when $k = 5$. To have a fair comparison, we trained our model with the same number (i.e., $k = 5$) of concept nodes in the second layer and used only the TFIDF values as the term-level features. HDLR has the same initial setting as well. The initial model parameters of both our method and HDLR are set using the LSA projection matrix. Notice that all three methods here have exactly the same input and functional form as LSA, which provides us some insight on how much gain a discriminative training method could achieve.

As presented in the bottom part of the table, LSA achieves very decent performance on this dataset. Even with a small number of components, it achieves reasonably high AUC and Max-F_1 scores. Both VL-CV and HDLR perform better than LSA, across various evaluation metrics. However, it is not clear whether VL-CV and HDLR is better as the former has higher AUC and Max-F_1 scores and the latter has higher precision@k and MAP values.

5 Related Work

While it is not possible to provide a complete survey on all the work of text similarity measures and their applications, here we focus on contrasting our approach with different learning frameworks and other dimensionality reduction methods.

As discussed previously, our vector learning method can be viewed as a special instantiation of the Siamese architecture that learns jointly two neural networks with the same model weights (Bromley et al., 1993). There are several differences when comparing our approach with the original Siamese neural network framework. For example, one main difficulty in the text domain is the inherently large dimensionality. In order to handle this issue, we constrain the model sparsity by having limited number of links between nodes from different layers and by forcing some model parameters to be shared. The similarity scoring function is also different. Because the cosine score essentially normalizes the vectors, it is numerically more stable than directly computing functions like Euclidean distance.

[2]The algorithm published in (Davis and Dhillon, 2008) is incorrect. We revised it based on the original ITML algorithm (Davis et al., 2007).

Our vector representation learning method is also analogous to different approaches for learning latent representations. For example, Collobert and Weston (2008) propose a deep neural network framework for learning "word distribution", where each word is represented by a vector and the element values are determined using an unsupervised language model. Alternatively, Huang and Yates (2009) propose to learn an HMM model to provide additional features using the latent states. In both cases, the output representation is used as features for building classifiers for different tasks, such as named entity recognition and part-of-speech tagging. In contrast, the new vector representation output by our method is used directly for measuring text similarity.

Learning similarity measures is also very related to the work of distance metric learning (Schultz and Joachims, 2004; Davis et al., 2007; Weinberger and Saul, 2009), where most methods learn a Mahalanobis matrix. Because the computational complexity of these methods is at least $O(n^2)$, where n is the dimensionality. Directly applying them to the problems in the text domain is not practical. Alternatively, learning a low-rank matrix has been suggested (Davis and Dhillon, 2008; Wu et al., 2009). Our approach is equivalent to learning the Mahalanobis matrix, but differs in controlling the model sparsity. In addition, model weights on term-level features can be jointly learned in our method, which is not directly achievable in other metric learning approaches.

Finally, as for mapping term vectors to the latent concept representation, our approach relies on labeled pairs of documents to train the model in a discriminative fashion. It would be interesting to compare it with other generative methods, such as probabilistic latent semantic analysis (PLSA) (Hofmann,) and iterative scaling (Ando, 2000), as well as Latent Dirichlet Allocation (LDA) (Blei et al., 2003).

6 Conclusions

In this paper, we present a novel approach that learns new vector representations for similarity measures. Conceptually, the model can be viewed as a two-layer Siamease neural network, where the first layer is the original term vector with term-level features as input and the second layer captures the relations among different terms to form the output vectors. Our approach is a natural generalization of several existing frameworks. When vectors in a low-dimensional "concept" space are desired, our method can learn such representation discriminatively, having the same model functional form as in latent semantic analysis. When learning extended term vectors, the model parameters equivalently form a sparse Mahalanobis distance matrix. Finally, it also subsumes the recently proposed term-weighting learning framework, TWEAK, as the term-weighting function can be learned jointly. In both settings, we show experimentally that our method outperforms most other methods in the task of measuring document similarity, and is comparable to HDLR in learning concept vectors.

In the future, we would like to enhance our approach in various aspects. For instance, although we construct our model using linear term-weighting function and have linear weight combinations, it is very easy to use non-linear transformations, which may further improve the model performance. Studying different issues when learning low-dimensional concept vector representations, such as finding the optimal dimensionality automatically and making the learning procedure more efficient, is also on our agenda. Finally, we would like to apply our approach in various text similarity tasks, such as word similarity and entity recognition and discovery.

References

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Pasca, and Aitor Soroa. 2009. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of HLT-NAACL*, pages 19–27, June.

Rie Kubota Ando. 2000. Latent semantic space: iterative scaling improves precision of inter-document similarity measurement. In *SIGIR '00*, pages 216–223.

David M. Blei, Andrew Y. Ng, Michael I. Jordan, and John Lafferty. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Andrei Broder, Marcus Fontoura, Vanja Josifovski, and Lance Riedel. 2007. A semantic approach to contextual advertising. In *SIGIR '07*.

Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "Siamese" time delay neural network. *Interna-*

tional Journal Pattern Recognition and Artificial Intelligence, 7(4):669–688.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*, pages 160–167.

Jason V. Davis and Inderjit S. Dhillon. 2008. Structured metric learning for high dimensional problems. In *KDD*, pages 195–203.

Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. 2007. Information-theoretic metric learning. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 209–216, New York, NY, USA. ACM.

Scott Deerwester, Susan Dumais, George Furnas, Thomas Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Ofer Dekel, Christopher D. Manning, and Yoram Singer. 2004. Log-linear models for label ranking. In *Advances in Neural Information Processing Systems (NIPS 2003)*.

Thomas Hofmann. Probabilistic latent semantic indexing. In *SIGIR '99*.

Fei Huang and Alexander Yates. 2009. Distributional representations for handling sparsity in supervised sequence labeling. In *ACL*.

Thad Hughes and Daniel Ramage. 2007. Lexical semantic relatedness with random graph walks. In *Proceedings of EMNLP-CoNLL*, pages 581–589, June.

Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. 2006. Generating query substitutions. In *Proceedings of the 15th World Wide Web Conference*.

David Dolan Lewis. 1992. *Representation and Learning in Information Retrieval*. Ph.D. thesis, Computer Science Dept.; Univ. of Massachusetts; Amherst, MA 01003. Technical Report 91–93.

D. Lewis. 1997. Reuters 21578 data set. http://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection.

Dekang Lin. 1998. Automatic retrieval and clustering of similar words. In *Proc. of COLING-ACL 98*.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Pres.

Rada Mihalcea, Courtney Corley, and Carlo Strapparava. 2006. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of AAAI-2006*.

Marco Pennacchiotti and Patrick Pantel. 2009. Entity extraction via ensemble semantics. In *EMNLP '09*, pages 238–247.

Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *ACL '05*, pages 622–629.

Mehran Sahami and Timothy D. Heilman. 2006. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th World Wide Web Conference*.

Matthew Schultz and Thorsten Joachims. 2004. Learning a distance metric from relative comparisons. In *In NIPS*. MIT Press.

Vishnu Vyas and Patrick Pantel. Semi-automatic entity set refinement. In *NAACL '09*.

Kilian Q. Weinberger and Lawrence K. Saul. 2009. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244.

Lei Wu, Rong Jin, Steven Chu-Hong Hoi, Jianke Zhu, and Nenghai Yu. 2009. Learning bregman distance functions and its application for semi-supervised clustering. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2089–2097.

W. Yih. 2009. Learning term-weighting functions for similarity measures. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP-09)*.

Appendix A. Gradient Derivation

Let $\mathbf{F}_p = \mathbf{\Phi}_p^T \mathbf{\Lambda}$ and $\mathbf{F}_q = \mathbf{\Phi}_q^T \mathbf{\Lambda}$ be the term vectors of documents D_p and D_q, respectively. The gradient of $\mathbf{\Lambda}$ of the cosine score of these two term vectors can be derived as follows.

$$\begin{aligned}
\cos(D_p, D_q) &\equiv \frac{\mathbf{F}_p^T \mathbf{F}_q}{\|\mathbf{F}_p\| \|\mathbf{F}_q\|} \\
\nabla_{\mathbf{\Lambda}} \mathbf{F}_p^T \mathbf{F}_q &= (\nabla_{\mathbf{\Lambda}} \mathbf{\Phi}_p^T \mathbf{\Lambda}) \mathbf{F}_q + (\nabla_{\mathbf{\Lambda}} \mathbf{\Phi}_q^T \mathbf{\Lambda}) \mathbf{F}_p \\
&= \mathbf{\Phi}_p \mathbf{F}_q + \mathbf{\Phi}_q \mathbf{F}_p \\
\nabla_{\mathbf{\Lambda}} \frac{1}{\|\mathbf{F}_p\|} &= \nabla_{\mathbf{\Lambda}} (\mathbf{F}_p^T \mathbf{F}_p)^{-\frac{1}{2}} \\
&= -\frac{1}{2} (\mathbf{F}_p^T \mathbf{F}_p)^{-\frac{3}{2}} \nabla_{\mathbf{\Lambda}} (\mathbf{F}_p^T \mathbf{F}_p) \\
&= -(\mathbf{F}_p^T \mathbf{F}_p)^{-\frac{3}{2}} \mathbf{\Phi}_p \mathbf{F}_p \\
\nabla_{\mathbf{\Lambda}} \frac{1}{\|\mathbf{F}_q\|} &= -(\mathbf{F}_q^T \mathbf{F}_q)^{-\frac{3}{2}} \mathbf{\Phi}_q \mathbf{F}_q
\end{aligned}$$

Let A, B, C be $\mathbf{F}_p^T \mathbf{F}_q$, $1/\|\mathbf{F}_p\|$ and $1/\|\mathbf{F}_q\|$, respectively.

$$\nabla_{\mathbf{\Lambda}} \frac{\mathbf{F}_p^T \mathbf{F}_q}{\|\mathbf{F}_p\| \|\mathbf{F}_q\|} = -ABC^3 \mathbf{\Phi}_q \mathbf{F}_q - ACB^3 \mathbf{\Phi}_p \mathbf{F}_p$$

$$+BC(\mathbf{\Phi}_p\mathbf{F}_q + \mathbf{\Phi}_q\mathbf{F}_p)$$

When using the extended term-vectors $\mathbf{G}_p$ and $\mathbf{G}_q$, the gradient derivation for $\mathbf{\Lambda}$ is essentially the same. We just need to replace $\mathbf{\Phi}_p$ and $\mathbf{\Phi}_q$ with $\mathbf{\Phi}_p^g$ and $\mathbf{\Phi}_q^g$, respectively.

Let g_k be the k-th component of the term-vector G (i.e., the extended term-weighting score of t_k), and let f_i be the i-th component of the original term-weighting F. Therefore, we have

$$g_k = \sum_i \alpha_{ik} f_i + f_k$$

For each term t_k,

$$\begin{aligned}\frac{\partial}{\partial \alpha_{ik}} g_{p_k} \cdot g_{q_k} &= \frac{\partial}{\partial \alpha_{ik}}[(\sum_i \alpha_{ik} f_{p_i} + f_{p_k}) \\ &\qquad (\sum_i \alpha_{ik} f_{q_i} + f_{q_k})] \\ &= f_{p_i} g_{q_k} + f_{q_i} g_{p_k}\end{aligned}$$

Similarly,

$$\begin{aligned}\frac{\partial}{\partial \alpha_{ik}} g_k^2 &= 2g_k \cdot \frac{\partial}{\partial \alpha_{ik}}(\sum \alpha_{ik} f_i + f_k) \\ &= 2g_k f_i\end{aligned}$$

We can construct three sparse matrices A', B', C'. For each α_{ik},

$$\begin{aligned}A'_{ik} &= f_{p_i} g_{q_k} + f_{q_i} g_{p_k} \\ B'_{ik} &= f_{p_i} g_{q_k} \\ C'_{ik} &= f_{q_i} g_{p_k}\end{aligned}$$

The final gradient matrix is:

$$\nabla_{\mathbf{A}} \frac{\mathbf{G}_p^T \mathbf{G}_q}{\|\mathbf{G}_p\| \|\mathbf{G}_q\|} = -ABC^3C' + BCA' - ACB^3B'$$