Supervised Semantic Indexing

Bing Bai, Jason Weston, Ronan Collobert, and David Grangier

NEC Labs America,
4 Independence Way, Princeton, USA 08540
{bbai,jasonw,collober,dgrangier}@nec-labs.com

Abstract. We present a class of models that are discriminatively trained to directly map from the word content in a query-document or document-document pair to a ranking score. Like Latent Semantic Indexing (LSI), our models take account of correlations between words (synonymy, polysemy). However, unlike LSI our models are trained with a supervised signal directly on the task of interest, which we argue is the reason for our superior results. We provide an empirical study on Wikipedia documents, using the links to define document-document or query-document pairs, where we obtain state-of-the-art performance using our method.

Key words: supervised, semantic indexing, document ranking

1 Introduction

Ranking text documents given a text-based query is one of the key tasks in information retrieval. A typical solution is to embed the problem in a feature space endowed with a similarity metric of choice. For example, classical vector space models use weighted word counts and the cosine similarity. In this case, the model is chosen by hand and no machine learning is involved. This type of model often performs remarkably well, but suffers from the fact that only exact matches of words between query and target texts contribute to the similarity score.

Latent Semantic Indexing [4], and related methods such as pLSA and LDA [10, 2], are *unsupervised* methods that choose a low dimensional feature representation of "latent concepts" where words are no longer independent. They are trained with reconstruction objectives, either based on mean squared error (LSI) or likelihood (pLSA, LDA). These models, being unsupervised, are still agnostic to the particular task of interest.

In this article we define a class of models that can be trained on a *supervised* signal (i.e., labeled data) to provide a ranking of a database of documents given a query. For example, if one has click-through data yielding query-target relationships, one can use this to train these models to perform well on this task. Or, if one is interested in finding documents related to a given *query document*, one can use known hyperlinks to learn a model that performs well on this task. Although click-through data has been used to learn models before, these models have typically relied on existing vector space models, and have only optimized a

few features [13]. This work is orthogonal to those works, and can in fact be used in conjunction with those methods for further gains. We show experimentally on Wikipedia that our method strongly outperforms tf-idf vector space models and LSI on both document-document and query-document tasks.

The rest of this article is as follows. In Section 2 we describe our method, Section 3 briefly discusses prior work , Section 4 describes an experimental study of our method, and Section 5 concludes with a discussion.

2 Supervised Semantic Indexing

Let us denote the set of documents in the corpus as and a query text as $q \in \mathbb{R}^{\mathcal{D}}$, where $\mathcal{D}$ is the dictionary size, and the j^{th} dimension of a vector indicates the frequency of occurrence of the j^{th} word, i.e. using the tf-idf weighting and then normalizing to unit length.

The set of models we propose are simply of the following type:

$$f(q,d) \;=\; q^\top W d \;=\; \sum_{i,j=1}^{\mathcal{D}} q_i W_{ij} d_j \tag{1}$$

where $f(q,d)$ is the score between a query q and a given document d, and $W \in \mathbb{R}^{\mathcal{D}\times\mathcal{D}}$ is the weight matrix, which will be learnt with a supervised signal. This model can capture synonymy and polysemy as it looks at all possible cross terms, and can be tuned directly for the task of interest. Note that we do not use stemming since our model can already match words with common stems.

We now discuss two main issues: (i) how to train it, and (ii) how to control training and test time efficiency. The latter will lead us to several algorithmic proposals that are still of the form (1) but with constraints on the form of W.

2.1 Training

Suppose we are given a set of tuples $\mathcal{R}$ (labeled data), where each tuple contains a query q, a relevant document d^+ and an irrelevant (or lower ranked) document d^-. We would like to choose W such that $q^\top W d^+ > q^\top W d^-$, that is d^+ should be ranked higher than d^-. To do this we employ the margin ranking loss [9] which has already been used in several IR methods before [13, 3, 6], and minimize:

$$\sum_{(q,d^+,d^-)\in\mathcal{R}} \max(0, 1 - q^\top W d^+ + q^\top W d^-). \tag{2}$$

We train this using stochastic gradient descent (see, e.g. [3]): iteratively, pick a random tuple and make a gradient step for that tuple. We use the (fixed) learning rate which minimizes the training error. Our method thus far is essentially a margin ranking perceptron with a particular choice of features, and hence related to a ranking SVM [13], except we have a highly scalable optimizer. However, we note that such an optimizer cannot be easily applied for probabilistic methods such as pLSA because of their normalization constraints. Recent methods like LDA [2] also suffer from scalability issues.

2.2 Controlling Efficiency (and Capacity)

Efficiency of a dense W matrix We consider both memory and speed considerations. Firstly, this method so far assumes that W fits in memory. For example, if the dictionary size $\mathcal{D} = 30000$, then this requires 3.4Gb of RAM (assuming floats).

The vectors q and d are sparse so the speed of computation of a single query-document pair involves mn computations $q_j W_{ij} d_i$, where q and d have m and n nonzero terms, respectively. We have found this is reasonable for training, but may be an issue at test time[1]. Alternatively, one can compute $v = q^\top W$ once, and then compute vd for each document. This is the same speed as a classical vector space model where the query contains $\mathcal{D}$ terms, assuming W is dense.

Sparse W matrices If W was itself a sparse matrix, then computation of $f(\cdot)$ would be considerably faster. If the query has m nonzero terms, and any given column of W has p nonzero terms, then the method is at most mp times slower than a classical vector space model. We can enforce W to be sparse using standard *feature selection* algorithms; we hence generalize the "Recursive Feature Elimination" algorithm (see [8], Chapter 5) yielding a simple, intuitive method:

1. First, we train the model with a dense matrix W as before.
2. For each column i of W find the k active elements with the smallest values of $|W_{ij}|$. Constrain these elements to equal zero (make them inactive).
3. Train the model with the constrained W matrix.
4. If W contains more than p nonzero terms in each column go back to 2.

This scheme is simple, efficient, and yielded good results (cf. Section 4).

Low rank W matrices An alternative efficient scheme is to constrain W in the following way:

$$W = U^\top V + I. \tag{3}$$

This induces a low dimensional "latent concept" space in a similar way to LSI. However, it differs in several ways: most importantly it is trained with a supervised signal. Further, U and V differ so it does not assume query and target document should be embedded in the same way, and the addition of the identity term means this model automatically learns the tradeoff between using the low dimensional space and a classical vector space model. In terms of efficiency however it is the same: its speed depends on the dimensionality of U and V.

We also highlight several variants:

- $\mathbf{W = I}$: if q and d are normalized tf-idf vectors this is equivalent to using the standard cosine similarity with no learning (and no synonymy or polysemy).
- $\mathbf{W = D}$, where D is a diagonal matrix: one learns a re-weighting of tf-idf using labeled data. This is similar to a method proposed in [6].
- $\mathbf{W = U^\top U + I}$: we constrain the model to be symmetric; the query and target document are treated in the same way.

[1] Of course, any method can be sped up by applying it to only a subset of pre-filtered documents, filtering using some faster method.

Table 1. Results and standard errors for Wikipedia document-document ranking.

Algorithm	Parameters	Rank-Loss	MAP	P@10
TFIDF	0	1.62%	0.329±0.011	0.162±0.007
Query Expansion	2	1.62%	0.330±0.011	0.162±0.006
αLSI + $(1-\alpha)$TFIDF	$200\mathcal{D}$+1	1.28%	0.346±0.010	0.170±0.007
SSI: $W = D$	$\mathcal{D}$	1.41%	0.355±0.011	0.177±0.007
SSI: $W = U^\top U + I$	$200\mathcal{D}$	0.41%	0.506±0.012	0.225±0.007
SSI: $W = U^\top V + I$	$400\mathcal{D}$	0.30%	0.517±0.012	0.229±0.007
SSI: W unconstrained	$\mathcal{D}^2$	0.41%	0.477±0.012	0.212±0.007
SSI: sparse W	$1000\mathcal{D}$	0.41%	0.461±0.012	0.213±0.007
SSI: sparse W	$100\mathcal{D}$	0.40%	0.462±0.011	0.209±0.007
SSI: sparse W	$10\mathcal{D}$	0.53%	0.425±0.011	0.197±0.007

Table 2. Results and standard errors for Wikipedia query-document ranking.

Algorithm	Parameters	Rank-Loss	MAP	P@10
TFIDF	0	14.0%	0.083±0.007	0.035±0.003
αLSI + $(1-\alpha)$TFIDF	$200\mathcal{D}$+1	9.73%	0.089±0.007	0.037±0.003
SSI: $W = U^\top U + I$	$200\mathcal{D}$	3.10%	0.213±0.007	0.095±0.003
SSI: $W = U^\top V + I$	$400\mathcal{D}$	2.91%	0.229±0.007	0.100±0.003

3 Prior Work

A tf-idf vector space model and LSI [4] are the main baselines we will compare to. We already mentioned pLSA [10] and LDA [2]; here we briefly discuss other relevant methods. Query Expansion (QE) is another way to employ synonyms, but requires manual tuning and does not always yield a consistent improvement [14]. The authors of [6] learn the weights of an orthogonal vector space model on Wikipedia links, improving over the OKAPI method. Joachims et al.[13] trained a SVM with hand-designed features based on the title, body, search engines rankings and the URL. Burges et al.[3] proposed a neural network method using similar features (569 in total). In contrast we limited ourselves to body text (not using title, URL, etc.) and train on $\mathcal{D}^2 = 900$ million features. We note in [7] related models were used for image retrieval, and in [5] for advert placement.

4 Experimental Study

We argue that standard retrieval datasets [12, 11] are too small to train our model, whereas click-through from search engines is not publicly available. We hence used a set of 1,828,645 Wikipedia documents as a database, and split the 24,667,286 links randomly into 70% for training and 30% for testing. All methods use only the top 30,000 most frequent words. We considered the following task: given a query document q, rank the other documents such that if q links to d then d is highly ranked. We trained several variants of our approach, called Supervised Semantic Indexing (SSI), as described in Section 2.2. Results on the test set in comparison to tf-idf/cosine similarity (TFIDF), αLSI + $(1-\alpha)$ TFIDF and QE (Standard Rochio [1], optimizing β and $|D_r|$, fixing $\gamma = 0$) are given

in Table 1. For LSI we report the best value of α and dimensionality, optimized on the training set. We then report the low rank version of SSI using the same choice of dimension. In terms of ranking loss (the percentage of tuples in $\mathcal{R}$ that are correctly ordered), mean average precision[2] (MAP) and precision at position 10 (P@10), all our methods strongly outperform the existing techniques.

We also tested our approach in a query-document setup. We used the same setup as before but kept only 10 random words from query documents in an attempt to make it like a "keyword search". We obtained similar improvements to before, as shown in Table 2.

5 Discussion

We have described a versatile, powerful set of discriminatively trained models for document ranking. Many generalizations are possible: exploring the use of the same models for cross-language retrieval, adding more features into our model, or generalizing to nonlinear models. Future work will explore these avenues further.

References

1. R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*. Addison-Wesley Harlow, England, 1999.
2. D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.
3. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML 2005*, pages 89–96.
4. S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.
5. Sharad Goel, John Langford, and Alexander Strehl. Predictive indexing for fast search. In *Advances in Neural Information Processing Systems 21*. 2009.
6. David Grangier and Samy Bengio. Inferring document similarity from hyperlinks. In *CIKM '05*, pages 359–360, New York, NY, USA, 2005. ACM.
7. David Grangier and Samy Bengio. A discriminative kernel-based approach to rank images from text queries. *IEEE Trans. PAMI.*, 30(8):1371–1384, 2008.
8. I.M. Guyon, S.R. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature Extraction: Foundations and Applications*. Springer, August 2006.
9. Ralf Herbrich, Thore Graepel, and Klaus Obermayer. *Large margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000.
10. Thomas Hofmann. Probabilistic latent semantic indexing. In *SIGIR 1999*, pages 50–57. ACM Press, 1999.
11. http://research.microsoft.com/en-us/um/beijing/projects/letor/index.html.
12. http://trec.nist.gov/.
13. T. Joachims. Optimizing search engines using clickthrough data. In *ACM SIGKDD*, pages 133–142, 2002.
14. Liron Zighelnic and Oren Kurland. Query-drift prevention for robust query expansion. In *SIGIR 2008*, pages 825–826, New York, NY, USA, 2008. ACM.

[2] For computational reasons, MAP and P@10 were measured by averaging over 1000 queries, and the true links and random subsets of 10,000 documents were used as the database, rather than the whole testing set. Rank-Loss uses 100,000 tuples.