Noname manuscript No.
(will be inserted by the editor)

Learning to Rank with (a Lot of) Word Features

Bing Bai · Jason Weston · David Grangier · Ronan Collobert · Kunihiko Sadamasa · Yanjun Qi · Olivier Chapelle · Kilian Weinberger

Abstract In this article we present Supervised Semantic Indexing (SSI) which defines a class of nonlinear (quadratic) 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 from a supervised signal directly on the ranking task of interest, which we argue is the reason for our superior results. As the query and target texts are modeled separately, our approach is easily generalized to different retrieval tasks, such as cross-language retrieval or online advertising placement. Dealing with models on all pairs of words features is computationally challenging. We propose several improvements to our basic model for addressing this issue, including low rank (but diagonal preserving) representations, correlated feature hashing (CFH) and sparsification. We provide an empirical study of all these methods on retrieval tasks based on Wikipedia documents as well as an Internet advertisement task. We obtain state-of-the-art performance while providing realistically scalable methods.

Keywords semantic indexing · feature hashing · learning to rank · cross language retrieval · content matching

1 Introduction

Ranking text documents given a text-based query is one of the key tasks in information retrieval. A typical solution is to: (i) embed the problem in a feature space, e.g. model queries and target documents using a vector representation; and then (ii) choose (or

B. Bai, J.Weston. D. Grangier, R. Collobert, Y. Qi, K. Sadamasa
NEC Labs America, Princeton, NJ, USA.
Tel.: +1 609-951-2687
Fax: +1 609-951-2482
E-mail: bbai@nec-labs.com, jasewestonagmail.com, dgrangier@nec-labs.com, collober@nec-labs.com, yanjun@nec-labs.com, kunihiko@nec-labs.com

O. Chapelle, K. Weinberger
Yahoo! Research, Santa Clara, CA
E-mail: chap@yahoo-inc.com, kilian@yahoo-inc.com

learn) a similarity metric that operates in this vector space. Ranking is then performed by sorting the documents based on their similarity score with the query.

For example, a classical vector space model, see e.g. [1], uses weighted word counts (e.g. via tf-idf) as the feature space, and the cosine similarity for ranking. 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. That is, words are considered to be independent, which is clearly a false assumption.

Latent Semantic Indexing [15], and related methods such as pLSA and LDA [26, 5], 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. Supervised LDA (sLDA) [4] has been proposed where a set of auxiliary labels are trained on jointly with the unsupervised task. However, the supervised task is not a task of learning to rank because the supervised signal is at the document level and is query independent.

In this article we propose Supervised Semantic Indexing (SSI) which defines 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. This signal is defined at the (query, documents) level and can either be point-wise — for instance the relevance of the document to the query — or pairwise — a given document is better than another for a given query. In this work, we focus on pairwise preferences. 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 [29]. 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 [21]. Moreover, our approach can model queries and documents separately, which can accommodate for differing word distributions between documents and queries. This is essential in the case of cross-language retrieval [23] where queries and documents are in different languages. This can also be applied to other applications as well, for instance matching questions and answers [41].

Learning to rank as a supervised task is not a new subject, however most methods and models have typically relied on optimizing over only a few hand-constructed features, e.g. based on existing vector space models such as tf-idf, the title, URL, PageRank and other information, see e.g. [29, 8]. Our work is orthogonal to those works, as it presents a way of learning a model for query and target texts by considering features generated by all pairs of words between the two texts[1]. The difficulty here is that such feature spaces are very large and we present several models that deal with memory, speed and capacity control issues. In particular we propose constraints on our model that are diagonal preserving but otherwise low rank, a direct sparsity constraint on the model, and a technique of hashing features (sharing weights) based on their correlation, called correlated feature hashing (CFH). In fact, all of our proposed methods can be used in conjunction with other features and methods explored in previous work for further gains.

We show experimentally on retrieval tasks derived from Wikipedia that our method strongly outperforms word-feature based models such as tf-idf vector space models, LSI

[1] This work is an expanded version of a poster paper [2] with further algorithmic proposals, applications and experiments.

and other baselines on document-document, query-document tasks and cross-language retrieval tasks. Finally, we give results on an Internet advertising task using proprietary data from an online advertising company.

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

2 Supervised Semantic Indexing

The goal of our work is to learn a similarity function $f(q, d)$ between a query q and a document d, which will be used in ranking. Without loss of generality, we take the ranking framework of "pairwise preference" [25]. That is, 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 function $f(q, d)$ such that $f(q, d^+) > f(q, d^-)$, expressing that d^+ should be ranked higher than d^-.

This section is organized as follows. In Subsection 2.1 we present the basic model of SSI, which is essentially a perceptron on quadratic word features. To the best of our knowledge, we are the first to consider such features using this model. For scalability and model capacity control, we propose in Subsection 2.2 several improvements to this model. Namely, in 2.2.1 we propose a novel asymmetric low rank approximation, which enables separate modelling of query and documents, yielding improved scalability and testing performance. In 2.2.2 we study a sparsified version of the basic model. In addition, we propose a novel feature hashing scheme in 2.2.3 to reduce feature numbers, in an way more meaningful than "Hash Kernels" [37]. Subsection 2.3 discusses the training of all models. Subsection 2.4 exposes several applications of SSI, which will be further investigated in experiments (section 4).

2.1 Basic Model

Let us denote the set of documents in the corpus as $\{d_t\}_{t=1}^{\ell} \subset \mathbb{R}^{\mathcal{D}}$ and a query text as $q \in \mathbb{R}^{\mathcal{D}}$, where $\mathcal{D}$ is the dictionary size[2], and the j^{th} dimension of a vector indicates the frequency of occurrence of the j^{th} word, e.g. using the tf-idf weighting and then normalizing to unit length [1].

The set of models we propose are all special cases of the following type of model:

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

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 learned from 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. We do not use stemming since our model can already match words with common stems (if it is useful for the task). Note that negative correlations via negative values in the weight matrix W can also be encoded.

[2] In fact in our resulting methods there is no need to restrict that both q and d have the same dimensionality $\mathcal{D}$ but we will make this assumption for simplicity of exposition.

Expressed in another way, given the pair q, d we are constructing the joint feature map:

$$\Phi_{((i-1)\mathcal{D}+j)}(q,d) = (qd^\top)_{ij} \tag{2}$$

where $\Phi_s(\cdot)$ is the s^{th} dimension in our feature space, and choosing the set of models:

$$f(q,d) \ = w \cdot \Phi(q,d). \tag{3}$$

Note that a model taking pairs of words as features is essential here, a simple approach concatenating (q,d) into a single vector and using $f(q,d) = w \cdot [q,d]$ is not a viable option as it would result in the same document ordering for any query.

We could train any standard method such as a ranking perceptron or a ranking SVM using our choice of features. However, without further modifications, this basic approach has a number of problems in terms of speed, storage space and capacity as we will now discuss.

Efficiency of a dense W matrix We analyze both memory and speed considerations. Firstly, this method so far assumes that W fits in memory (unless sparsity is somehow enforced). If the dictionary size $\mathcal{D} = 30000$, then this requires 3.4Gb of RAM (assuming floats), and if the dictionary size is 2.5 Million (as it will be in our experiments in Section 4) this amounts to 14.5 Terabytes. The vectors q and d are sparse so the speed of computation of a single query-document pair involves mn computations $q_i W_{ij} d_j$, where q and d have m and n non-zero terms, respectively. We have found this is reasonable for training, but may be an issue at test time[3]. 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. The capacity of this model is also obviously rather large. As every pair of words between query and target is modeled separately it means that any pair not seen during the training phase will not have its weight trained. Regularizing the weights so that unseen pairs have $W_{ij} = 0$ is thus essential (this is discussed in Section 2.3). However, this is still not ideal and clearly a huge number of training examples will be necessary to train so many weights, most of which are not used for any given training pair (q,d).

Overall, a dense matrix W is challenging in terms of memory footprint, computation time and controlling its capacity for good generalization. In the next section we describe ways of improving over this basic approach.

2.2 Improved Models

We now describe several special cases by constraining the form (1) that lead to proposals for improved models in terms of accuracy and efficiency.

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

2.2.1 Low rank (but diagonal preserving) W matrices

An efficient scheme is to constrain W in the following way:

$$W = U^\top V + I. \tag{4}$$

Here, U and V are $N \times \mathcal{D}$ matrices. This induces a N-dimensional "latent concept" space in a similar way to LSI. This similarity is the main inspiration to name this class of models "Supervised Semantic Indexing". However, in this paper we actually refer to any method that performs supervised learning to rank using word-word relationships based on their semantics related to the task, whether low rank or not.

However, its differences from LSI should be highlighted:

- Most importantly it is trained from a supervised signal using preference relations (ranking constraints).
- Further, U and V differ so it does not assume the query and target document should be embedded to the latent space in the same way. This can be suitable when the query text distribution is very different to the document text distribution, such as the typical case of brief queries and longer documents. The extreme case of cross language retrieval with queries and target texts in different languages is naturally modeled in this setup.
- Finally, 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. This is important because the diagonal of the W matrix gives the specificity of picking out when a word co-ocurrs in both documents (indeed, setting $W = I$ is equivalent to cosine similarity using tf-idf, see below). The matrix I is full rank and therefore cannot be approximated with the low rank model $U^\top V$, so our model combines both. Note that the weights of U and V are learnt so one does not need a weighting parameter multiplied by I.

However, the efficiency and memory footprint are as favorable as LSI. Typically, one caches the N-dimensional representation for each document to use at query time. Moreover, the low rank induces a form of capacity control (we have far fewer parameters to learn) which we observe leads to improved generalization ability.

We also highlight several other regularization variants, which are further possible ways of constraining W:

- $W = I$: 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).
- $W = D$, where D is a diagonal matrix: one learns a re-weighting of tf-idf using labeled data (still no synonymy or polysemy). This is similar to a method proposed in [21].
- $W = U^\top U + I$: we constrain the model to be symmetric; the query and target document are treated in the same way.
- $W = U^\top V + D$: where D is a diagonal matrix, to be learnt. Here, we can learn a re-weighting of tf-idf at the same time as a latent concept space.

2.2.2 Sparse W matrices

If W was itself a sparse matrix, then computation of $f(\cdot)$ would be considerably faster, not to mention the memory savings. If the query has m non-zero terms, and any given

row of W has p non-zero terms, then there are at most mp terms in $v = q^\top W$ (compared to $m\mathcal{D}$ terms for a dense W, and m terms for a classical vector space cosine).

The simplest way to do this is to apply a regularizer such as minimizing the L_1-norm of W (see, e.g. [24]):

$$\min_W \gamma||W||_1 + \sum_{q \in \mathcal{Q}} L(W, q, \mathbf{d}, Y(q, \mathbf{d}))$$

where $L(\cdot)$ is a function that computes the loss of the current model given a query $q \in \mathcal{Q}$ and a labeling $Y(q, \mathbf{d})$ of the relevance of the documents $\mathbf{d}$ with respect to q. However, in general any sparsity promoting regularizer or feature selection technique could be used.

2.2.3 Correlated Feature Hashing

Another way to both lower the capacity of our model and decrease its storage requirements is to share weights among features.

Hash Kernels (Random Hashing of Words) In [37] the authors proposed a general technique called "Hash Kernels" where they approximate the feature representation $\Phi(x)$ with:

$$\bar{\Phi}_j(x) = \sum_{i \in \mathcal{W}: h(i) = j} \Phi_i(x)$$

where $h : \mathcal{W} \to \{1, \ldots, \mathcal{H}\}$ is a hash function that reduces an the feature space down to $\mathcal{H}$ dimensions, while maintaining sparsity, where $\mathcal{W}$ is the set of initial feature indices. The software Vowpal Wabbit[4] implements this idea (as a regression task) for joint feature spaces on pairs of objects, e.g. documents. In this case, the hash function used for a pair of words (s, t) is $h(s, t) = \text{mod}(sP + t, \mathcal{H})$ where P is a large prime. This yields

$$\bar{\Phi}_j(q, d) = \sum_{(s,t) \in \{1,\ldots,\mathcal{D}\}^2 : h(s,t) = j} \Phi_{s,t}(q, d). \tag{5}$$

where $\Phi_{s,t}(\cdot)$ indexes the feature on the word pair (s, t), e.g. $\Phi_{s,t}(\cdot) = \Phi_{((s-1)\mathcal{D}+t)}(\cdot)$. This technique is equivalent to *sharing* weights, i.e. constraining $W_{st} = W_{kl}$ when $h(s, t) = h(k, l)$. In this case, the sharing is done pseudo-randomly, and collisions in the hash table generally results in sharing weights between term pairs that share no meaning in common.

Correlated Feature Hashing We thus suggest a technique to share weights (or equivalently hash features) so that collisions actually happens for terms with close meaning. For that purpose, we first sort the words in our dictionary in frequency order, so that $i = 1$ is the most frequent, and $i = \mathcal{D}$ is the least frequent. For each word $i = 1, \ldots, \mathcal{D}$, we calculate its DICE coefficient [38] with respect to each word $j = 1, \ldots, \mathcal{F}$ among the top $\mathcal{F}$ most frequent words:

$$\text{DICE}(i, j) = \frac{2 \cdot \text{cooccur}(i, j)}{\text{occur}(i) + \text{occur}(j)}$$

[4] `http://hunch.net/~vw/`

where cooccur(i, j) counts the number of co-occurences for i and j at the document or sentence level, and occur(i) is the total number of occurences of word i. Note that these scores can be calculated from a large corpus of completely unlabeled documents. For each i, we sort the $\mathcal{F}$ scores (largest first) so that $S_p(i) \in \{1, \ldots, \mathcal{F}\}$ correspond to the index of the p^{th} largest DICE score $DICE(i, S_p(i))$. We can then use the Hash Kernel approximation $\bar{\Phi}(\cdot)$ given in equation (5) relying on the "hashing" function:

$$h(i, j) = (S_1(i) - 1)\mathcal{F} + S_1(j)$$

This strategy is equivalent to pre-processing our documents and replacing all the words indexed by i with $S_1(i)$. Note that we have reduced our feature space from $\mathcal{D}^2$ features to $\mathcal{H} = \mathcal{F}^2$ features. This reduction can be important as shown in our experiments, see Section 4: e.g. for our Wikipedia experiments, we have $\mathcal{F} = 30,000$ as opposed to $\mathcal{D} = 2.5$ Million. Typical examples of the top k matches to a word using the DICE score are given in Table 7.

Moreover, we can also combine correlated feature hashing with the low rank W matrix constraint described in Section 2.2.1. In that case U and V becomes $\mathcal{F} \times N$ dimensional matrices instead of $\mathcal{D} \times N$ matrices instead because the set of features is no longer the entire dictionary, but the first $\mathcal{F}$ words.

Correlated Feature Hashing by Multiple Binning It is also suggested in [37] to hash a feature $\Phi_i(\cdot)$ so that it contributes to multiple features $\bar{\Phi}_j(\cdot)$ in the reduced feature space. This strategy theoretically lessens the consequence of collisions. In our case, we can construct multiple hash functions from the values $S_p(\cdot), \quad p = 1, \ldots, k$, i.e. the top k correlated words according to their DICE scores:

$$\bar{\Phi}_j(q, d) = \frac{1}{k} \sum_{\substack{p = 1, \ldots, k \\ (s,t) \in \{1, \ldots, \mathcal{D}\}^2 : h_p(s,t) = j}} \Phi_{s,t}(q, d) \tag{6}$$

where

$$h_p(s, t) = (S_p(s) - 1)\mathcal{F} + S_p(t). \tag{7}$$

Pseudocode for constructing the feature map $\bar{\Phi}(q, d)$ is given in algorithm 1.

Algorithm 1 Construction of the correlated feature hashing (6)

Initialize the $\mathcal{F}$ dimensional vectors a and b to 0.
for $p = 1, \ldots, k$ **do**
  for all $i = 1, \ldots, \mathcal{D}$ such that $q_i > 0$ or $d_i > 0$ **do**
    $j \leftarrow S_p(i)$
    $a_j \leftarrow a_j + q_i$
    $b_j \leftarrow b_j + d_i$
  end for
end for
$\forall i, j \in [1 \ldots \mathcal{F}]$, $\bar{\Phi}_{((i-1)\mathcal{F}+j)}(q, d) \leftarrow \frac{1}{k}(ab^\top)_{ij}$

Equation (6) defines the reduced feature space as the mean of k feature maps which are built using hashing functions using the $p = 1, \ldots, k$ most correlated words. Equation (7) defines the hash function for a pair of words i and j using the p^{th} most correlated words $S_p(i)$ and $S_p(j)$. That is, the new feature space consists of, for each word in

the original document, the top k most correlated words from the set of size $\mathcal{F}$ of the most frequently occurring words. Hence as before there are never more than $\mathcal{H} = \mathcal{F}^2$ possible features. Overall, this is in fact equivalent to pre-processing our documents and replacing all the words indexed by i with $S_1(i), \ldots, S_k(i)$, with appropriate weights.

Hashing n-grams One can also use these techniques to incorporate n-gram features into the model without requiring a huge feature representation that would have no way of fitting in memory. We simply use the DICE coefficient between an n-gram i and the first $\mathcal{F}$ words $j = 1, \ldots, \mathcal{F}$, and proceed as before. In fact, our feature space size does not increase at all, and we are free to use any value of n. Typical examples of the top k matches for a 2-gram using the DICE score are given in Table 8.

2.3 Training Methods

We now discuss how to train the models we have described in the previous section.

2.3.1 Learning the Basic Model

Given the similarity function Eq. 1, we would like to choose W such that $q^\top W d^+ > q^\top W d^-$, expressing that d^+ should be ranked higher than d^-.

For that purpose, we employ the margin ranking loss [25] which has already been used in several IR methods before [29,8,21], and minimize:

$$\sum_{(q,d^+,d^-)\in\mathcal{R}} \max(0, 1 - q^\top W d^+ + q^\top W d^-). \tag{8}$$

This optimization problem is solved through stochastic gradient descent, (see, e.g. [8]): iteratively, one picks a random tuple and makes a gradient step for that tuple:

$$W \leftarrow W + \lambda(q(d^+)^\top - q(d^-)^\top), \quad \text{if } \; 1 - q^\top W d^+ + q^\top W d^- > 0$$

Obviously, one should exploit the sparsity of q and d when calculating these updates. To train our model, we choose the (fixed) learning rate λ which minimizes the training error. We also suggest to initialize the training with $W = I$ as this initializes the model to the same solution as a cosine similarity score. This introduces a prior expressing that the weight matrix should be close to I, considering term correlation only when it is necessary to increase the score of a relevant document, or conversely decreasing the score of a non-relevant document. Termination is then performed by viewing when the error is no longer improving, using a validation set (i.e. using early stopping).

Our method thus far is a margin ranking perceptron [12] with a particular choice of features (2). The margin perceptron has been shown to be equivalent to SVM [13], although possessing a different training algorithm (SVM typically uses quadratic programming, while the margin perceptron uses stochastic gradient descent). The problems we study (see section 4) include millions of training examples, making classical SVM training intractable. Stochastic training is highly scalable and is easy to implement for our model. The regularizing effect of the large margin term $w^T w$ in SVMs can be achieved by weight decay or early stopping in stochastic learning for margin perceptrons [13]. In this work, we rely on early stopping: early stopping monitors the

performance on held-out validation data during training and stops learning when validation data stops improving. This strategy hence regularizes the model by expressing preference for weights close to the initial weights [10,13]. Empirically we did not experience severe overfitting problems in most of our experiments, as we worked with large-sized training sets, more details can be found in section 4.5.

However, we note that such optimization cannot be easily applied to probabilistic methods such as pLSA because of their normalization constraints. Recent methods like LDA [5] also suffer from scalability issues.

Researchers have also explored optimizing various alternative loss functions other than the ranking loss including optimizing normalized discounted cumulative gain (NDCG) and mean average precision (MAP) [8,7,9,44]. In fact, one could use those optimization strategies to train our models instead of optimizing the ranking loss as well.

2.3.2 Learning a Low Rank W Matrix

When the W matrix is constrained, e.g. $W = U^\top V + I$, training is done in a similar way as before. It this case, the gradient steps to optimize the parameters U and V are:

$$U \leftarrow U + \lambda V(d^+ - d^-)q^\top, \qquad \text{if} \quad 1 - f(q, d^+) + f(q, d^-) > 0$$
$$V \leftarrow V + \lambda U q(d^+ - d^-)^\top, \qquad \text{if} \quad 1 - f(q, d^+) + f(q, d^-) > 0.$$

Note this is no longer a convex optimization problem. In our experiments we initialized the matrices U and V randomly using a normal distribution with mean zero and standard deviation one.

2.3.3 Learning a Sparse W Matrix

Directly employing an L_1-norm regularizer is known to be difficult to optimize online (see, e.g. [31] for an explanation and ways to deal with this problem). The use of the L_1 regularizer is also hampered by the excessive cost of its validation on large scale problems, such as the ones addressed in Section 4. In this work, therefore, we enforce sparsity through feature selection. Potentially we could appeal to any feature selection method (see [24] for a review). Here, we used a simple, intuitive "projection" method that can be seen as a generalization of the Recursive Feature Elimination (RFE) feature selection algorithm (see [24], Chapter 5). Algorithm 2 gives the pseudo-code of this technique.

Algorithm 2 Sparse Model Learning

1. Train the model with a dense matrix W as before.
2. For each row i of W, find the k active elements with the smallest values of $|W_{ij}|$. Constrain these elements to equal zero, i.e. make them inactive (the reason to choose the row here is that if the query has m non-zero terms, and any given row of W has p non-zero terms, then the method has at most mp terms in $v = q^\top W$ – compared to the m terms in a classical vector space model – as we mentioned before.).
3. Train the model with the constrained W matrix.
4. If W contains more than p non-zero terms in each row go back to 2.

This algorithm is motivated by the removal of the smallest weights being linked to the smallest change in the objective function [24] and has been shown to work well in several experimental setups. Overall, we found this scheme to be simple and efficient, while yielding good results.

2.3.4 Learning with Feature Hashing

Feature hashing simply corresponds to a different choice of feature map, depending on the hashing technique chosen. Hence the training techniques described above can be used for training with feature hashing.

2.4 Applications

2.4.1 Standard Retrieval

We consider two standard retrieval models: returning relevant documents given a keyword-based query, and finding related documents with respect to a given query document, which we call the query-document and document-document tasks.

Our models naturally can be trained to solve these tasks. We note here that so far our models have only included features based on the bag-of-words model, but there is nothing stopping us adding other kinds of features as well. Typical choices include: features based on the title, body, words in bold font, the popularity of a page, its PageRank, the URL, and so on, see e.g. [1]. However, for clarity and simplicity, our experiments use a setup where the raw words are used.

2.4.2 Cross Language Retrieval

Cross Language Retrieval [23] is the task of retrieving documents in a target language E given a query in a different source language F[5]. Our model

$$f(q_F, d_E) = q_F^\top W d_E$$

can naturally deal with this task without the need for machine translation because it directly learns the correspondence between the two languages using labeled data in the form of tuples $\mathcal{R}$. The use of a non-symmetric low rank model like (4) naturally suits this task (in this case adding the identity does not make sense):

$$f(q_F, d_E) = q_F^\top (U_F^\top V_E) d_E \tag{9}$$

Given additional supervised data for a standard monolingual retrieval problem, which will typically be more abundant than for the cross-lingual problem, it is also possible to regularize the solution we find by training on this task at the same time (multi-tasking) using the model:

$$f(q_E, d_E) = q_E^\top (V_E^\top V_E) d_E. \tag{10}$$

[5] For example, Google provides such a service at `http://translate.google.com/translate_s`.

Alternatively, it is also possible to use the model $f(q_E, d_E) = q_E^\top (U_E^\top V_E) d_E$, where only V_E is shared between tasks. Multi-tasking is achieved by adding the objective functions together. For stochastic gradient descent training, this amounts to performing a gradient step for one of the objectives followed by a gradient step for the other. Training (10) will act as a regularizer when training the model (9) because the two functions share the parameters V_E, so both tasks aim to find a good latent space for target documents in language E. Similarly, one can also regularize U_F.

2.4.3 Content Matching

Our models can also be applied to identify two differing types of text as a matching pair, for example a sequence of text (which could be a web page or an email or a chat log) with a targeted advertisement. In this case, click-through data can provide supervision. Here, again for simplicity, we assume both text and advert are represented as words. In practice, however, other types of engineered features could be added for optimal performance.

3 Prior Work

A tf-idf vector space model and LSI [15] are two main baselines we will compare to. We already mentioned pLSA [26] and LDA [5] both have scalability problems and are not reported to generally outperform LSA and TF-IDF [18]. Moreover in the introduction we discussed how sLDA[4] provides supervision at the *document* level (via a class label or regression value) and is not a task of learning to rank, whereas here we study supervision at the (query,documents) level. [42] uses cooccurences of multiple type objects to build better latent concept vectors. However, it still falls into the "unsupervised" category.

In [21] the authors learned the weights of an orthogonal vector space model on Wikipedia links, improving over the OKAPI method. Joachims et al.[29] trained a SVM with hand-designed features based on the title, body, search engines rankings and the URL. Burges et al.[8] proposed a neural network method using a similar set of features (569 in total). In contrast we limited ourselves to body text (not using title, URL, etc.) and train on at most $\mathcal{D}^2 = 900$ million features.

Query Expansion, often referred to as blind relevance feedback, is another way to deal with synonyms, but requires manual tuning and does not always yield a consistent improvement [45].

The authors of [22] used a related model to the ones we describe, but for the task of image retrieval, and [20] also used a related (regression-based) method for advert placement. They both use the idea of using the cross product space of features in the perceptron algorithm as in equation (2) which is implemented in related software to these two publications, PAMIR[6] and Vowpal Wabbit[7]. The task of document retrieval, and the use of sparse or low rank matrices, or of using correlated feature hashing (cf. Section 2.2), is not studied. Earlier work [3] models the probability of a query given a document based on "translation probabilities", which are the conditional probabilities of a query word given a document word. Although this algorithm shares the same

[6] `http://www.idiap.ch/pamir/`

[7] `http://hunch.net/~vw/`

intuition with our work, its lack of problem size control could lead to intractability for a large vocabulary, due to the exploding number of query-document word pairs.

Several authors [36,30] have proposed interesting nonlinear versions of (unsupervised) LSI using neural networks and showed they outperform LSI or pLSA. However, in the case of [36] we note their method is rather slow, and a dictionary size of only 2000 was used. Finally, [39] proposes a "supervised" LSI for classification. This has a similar sounding title to ours, but is quite different because it is based on applying LSI to document classification rather than improving ranking via known preference relations. The authors of [17] proposed "Explicit Semantic Analysis" which represents the meaning of texts in a high-dimensional space of concepts by building a feature space derived from the human-organized knowledge from an encyclopedia, e.g. Wikipedia. In the new space, cosine similarity is applied. SSI could be applied to such feature representations so that they are not agnostic to a particular supervised task as well.

Another related area of research is in distance metric learning [43,28,19]. Methods like Large Margin Nearest Neighbor LMNN [43] also learn a model similar to the basic model (2.1) with the full matrix W (but not with our improvements to this model). LMNN further constrain W to be a positive semidefinite matrix. This makes the training computational cost considerable, e.g. even after significant optimization of the learning algorithm it still takes 3.5 hours to train on 60,000 examples and 169 features on a handwritten digit recognition problem. This would hence not be scalable for large scale text ranking experiments. Nevertheless, Chechik et al. compared LMNN [43], LEGO [28] and MCML [19] to a stochastic gradient method with a full matrix W (the basic model (2.1)) on a small image ranking task and report in fact that the stochastic method provides both improved results and efficiency[8].

Methods for cross-language retrieval range from first applying machine translation and then a conventional retrieval method such as LSI [23], a direct method of applying LSI for this task called CL-LSI [16] or using Kernel Canonical Correlation Analysis, KCCA [40]. While the latter is a strongly performing method, it also suffers from scalability problems and requires translated text pairs for training.

4 Experimental Study

Learning a model of term correlations over a large vocabulary is a considerable challenge that requires a large amount of training data. Standard retrieval datasets like TREC[9] or LETOR [32] contain only a few hundred training queries, and are hence too small for that purpose. Moreover, some datasets only provide few pre-processed features like page-rank, or BM25, and not the actual words. Click-through from web search engines could provide valuable supervision. However, such data is not publicly available, and hence experiments on such data are not reproducible.

We hence conducted most experiments on Wikipedia and used links within Wikipedia to build a large scale ranking task. Thanks to its abundant, high-quality labeling and structuring, Wikipedia has been exploited in a number of applications such as disambiguation [6,14], text categorization [34,27], relationship extraction [35,11], and searching [33] etc. Specifically, Wikipedia link structures were also used in [34,33,35].

[8] Oral presentation at the (Snowbird) Machine Learning Workshop, see `http://snowbird.djvuzone.org/abstracts/119.pdf`

[9] `http://trec.nist.gov/`

We considered several tasks: document-document retrieval described in Section 4.1, query-document retrieval described in Section 4.2, and cross-language document-document retrieval described in Section 4.3. In Section 4.4 we also give results on an Internet advertising task using proprietary data from an online advertising company.

In these experiments we compare our approach, Supervised Semantic Indexing (SSI), to the following methods: tf-idf with cosine similarity (TFIDF), Query Expansion (QE), LSI[10] and αLSI + $(1-\alpha)$ TFIDF. Moreover SSI with an "unconstrained W" is just a margin ranking perceptron with a particular choice of feature map, and SSI using hash kernels is the approach of [37] employing a ranking loss. For LSI we report the best value of α and embedding dimension (50, 100, 200, 500, 750 or 1000), optimized on the training set ranking loss. We then report the low rank version of SSI using the same choice of dimension. Query Expansion involves applying TFIDF and then adding the mean vector $\beta \sum_{i=1}^{\mathcal{E}} d_{r_i}$ of the top $\mathcal{E}$ retrieved documents multiplied by a weighting β to the query, and applying TFIDF again. We report the error rate where β and $\mathcal{E}$ are optimized using the training set ranking loss.

For each method, we measure the ranking loss (the percentage of tuples in $\mathcal{R}$ that are incorrectly ordered), precision $P(n)$ at position $n = 10$ (P@10) and the mean average precision (MAP), as well as their standard errors. For computational reasons, MAP and P@10 were measured by averaging over a fixed set of 1000 test queries, where for each query the linked test set documents plus random subsets of 10,000 documents were used as the database, rather than the whole testing set. The ranking loss is measured using 100,000 testing tuples (i.e. 100,000 queries, and for each query one random positive and one random negative target document were selected).

4.1 Document-Document Retrieval

We considered a set of 1,828,645 English Wikipedia documents as a database, and split the 24,667,286 links[11] randomly into two portions, 70% for training and 30% for testing. We then considered the following task: given a query document q, rank the other documents such that if q links to d then d should be highly ranked.

Limited Dictionary Size In our first experiments, we used only the top 30,000 most frequent words. This allowed us to compare all methods with the proposed approach, Supervised Semantic Indexing (SSI), using a completely unconstrained W matrix as in equation (1). LSI is also feasible to compute in this setting. We compare several variants of our approach, as detailed in Section 2.2.

Results on the test set are given in Table 1. All the variants of our method SSI strongly outperform the existing techniques TFIDF, LSI and QE. SSI with unconstrained W performs worse than the low rank counterparts – probably because it has too much capacity given the training set size. Non-symmetric low-rank SSI $W = U^\top V + I$ slightly outperforms its symmetric counterpart $W = U^\top U + I$ (see also Table 4 for other choices of N). SSI with sparse W degrades in performance with increased sparsity (from 1000 non-zero elements per column to 10 non-zero elements) but still outperforms our baselines. Diagonal SSI $W = D$ is only a learned re-weighting

[10] We use the SVDLIBC software `http://tedlab.mit.edu/~dr/svdlibc/` and the cosine distance in the latent concept space.

[11] We removed links to calendar years as they provide little information while being very frequent.

Table 1 Empirical results for document-document ranking on Wikipedia (limited dictionary size of $\mathcal{D}$ =30,000 words).

Algorithm	Parameters	Rank-Loss	MAP	P@10
TFIDF	0	1.62%	0.329±0.010	0.163±0.006
QE	2	1.62%	0.330±0.010	0.163±0.006
LSI	$1000\mathcal{D}$	4.79%	0.158±0.006	0.098±0.005
αLSI + $(1-\alpha)$TFIDF	$200\mathcal{D}$+1	1.28%	0.346±0.011	0.170±0.007
SSI: $W = D$	$\mathcal{D}$	1.41%	0.355±0.009	0.177±0.007
SSI: W unconstrained	$\mathcal{D}^2$	0.41%	0.477±0.011	0.212±0.007
SSI: sparse W	$1000\mathcal{D}$	0.41%	0.461±0.010	0.213±0.007
SSI: sparse W	$100\mathcal{D}$	0.40%	0.462±0.010	0.209±0.007
SSI: sparse W	$10\mathcal{D}$	0.53%	0.425±0.011	0.197±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±0.011**	**0.229±0.007**

Table 2 Empirical results for document-document ranking on Wikipedia (unlimited dictionary size, all $\mathcal{D} = 2.5M$ words). Results for random hashing (i.e., hash kernels [37]) and correlated feature hashing (CFH) on all words are included.

Algorithm	Params	Rank Loss	MAP	P@10
TFIDF	0	0.842%	0.432±0.012	0.193±0.007
QE	2	0.842%	0.432±0.012	0.193±0.007
αLSI_{30k} + $(1-\alpha)$TFIDF	$200 \times 30k + 1$	0.721%	0.433±0.012	0.193±0.007
SSI: $W = (U^\top U)_{2.5M} + I$	$50\mathcal{D}$	0.200%	0.503±0.012	0.220±0.007
SSI: $W = (U^\top V)_{100k} + I$	$100 \times 100k$	0.178%	0.536±0.012	0.233±0.008
SSI: $W = (U^\top V)_{60k} + I$	$100 \times 60k$	0.172%	0.541±0.012	0.232±0.008
SSI: $W = (U^\top V)_{30k} + I$	$200 \times 30k$	0.158%	0.547±0.012	0.239±0.008
SSI: Hash Kernels [37]	1M	2.98%	0.239±0.009	0.127±0.005
SSI: Hash Kernels	3M	1.75%	0.301±0.01	0.152±0.006
SSI: Hash Kernels	6M	1.37%	0.335±0.01	0.164±0.007
SSI: Hash Kernels + αI	1M+1	0.525%	0.466±0.011	0.207±0.007
SSI: Hash Kernels + αI	3M+1	0.370%	0.474±0.012	0.211±0.007
SSI: Hash Kernels + αI	6M+1	0.347%	0.485±0.011	0.215±0.007
SSI: CFH (2-grams)	$300 \times 30k$	0.149%	0.559±0.012	0.249±0.007
SSI: CFH (1-grams)	$300 \times 30k$	**0.119%**	**0.614±0.012**	**0.263±0.008**

Table 3 Empirical results for document-document ranking in two train/test setups: partitioning into train+test sets of links, or into train+test sets of documents with no cross-links (limited dictionary size of 30,000 words). The two setups yield similar results.

Algorithm	Testing Setup	Rank Loss	MAP	P@10
SSI: $W = U^\top V + I$	Partitioned links	0.407%	0.506±0.012	0.225±0.007
SSI: $W = U^\top V + I$	Partitioned docs+links	0.401%	0.503±0.010	0.225±0.006

of word weights, but still slightly outperforms TFIDF. In terms of our baselines, LSI is slightly better than TFIDF but QE in this case does not improve much over TFIDF, perhaps because of the difficulty of this task, i.e. there may too often many irrelevant documents in the top $\mathcal{E}$ documents initially retrieved for QE to help.

Unlimited Dictionary Size In our second experiment we no longer constrained methods to a fixed dictionary size, so all 2.5 million words are used. Due to being unable to compute LSI for the full dictionary size, we used the LSI computed in the previous

Table 4 Adjusting the latent concept dimension N. Results reported on the document-document ranking task on a limited dictionary size of 30,000 words.

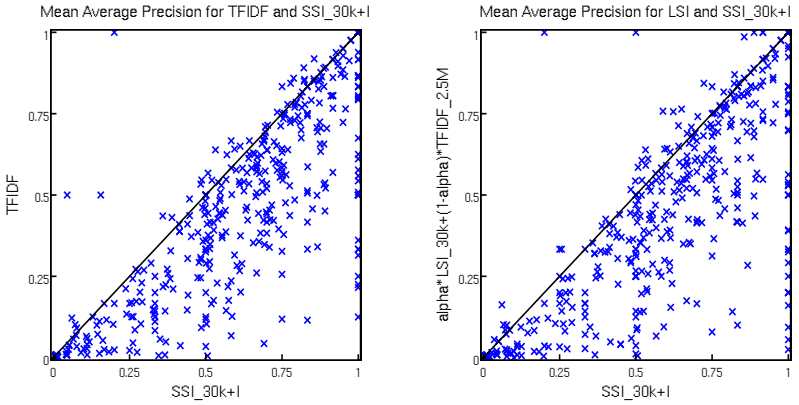
Algorithm	Dimension N	Rank Loss	MAP	P@10
SSI: $W = (U^\top U) + I$	100	0.407%	0.506±0.012	0.225±0.007
SSI: $W = (U^\top V) + I$	100	0.387%	0.514±0.012	0.225±0.007
SSI: $W = (U^\top U) + I$	200	0.380%	0.510±0.012	0.226±0.008
SSI: $W = (U^\top V) + I$	200	0.302%	0.517±0.012	0.229±0.007
SSI: $W = (U^\top U) + I$	500	0.374%	0.495±0.012	0.221±0.007
SSI: $W = (U^\top V) + I$	500	0.310%	0.514±0.012	0.228±0.007

experiment on 30000 words and combined it with TFIDF using the entire dictionary. In this setting we compared our baselines with the low rank SSI method $W = (U^\top V)_n + I$, where n means that we constrained the rows of U and V for infrequent words (i.e. all words apart from the most frequent n) to equal zero. The reason for this constraint is that it can stop the method overfitting: if a word is used in one document only then its embedding can take on any value independent of its content. Infrequent words are still used in the diagonal of the matrix (via the $+I$ term). The results, given in Table 2, show that using this constraint outperforms an unconstrained choice of $n = 2.5M$. Figure 1 shows scatter plots where SSI outperforms the baselines TFIDF and LSI in terms of average precision.

Overall, compared to the baselines the same trends are observed as in the limited dictionary case, indicating that the restriction in the previous experiment did not bias the results in favor of any one algorithm. Note also that as a page has on average just over 3 test set links to other pages, the maximum P@10 one can achieve in this case is 0.31, while our best model reaches 0.263 for this measure.

Hash Kernels and Correlated Feature Hashing On the full dictionary size experiments in Table 2 we also compare Hash Kernels [37] with our Correlated Feature Hashing method described in Section 2.2.3. For Hash Kernels we tried several sizes of hash table $\mathcal{H}$ (1M, 3M and 6M), we also tried adding a diagonal to the matrix learned in a similar way as is done for LSI. We note that if the hash table is big enough this method is equivalent to SSI with an unconstrained W, however for the hash sizes we tried Hash Kernels did not perform as well. For correlated feature hashing, we simply used the SSI model $W = (U^\top V)_{30k} + I$ from the 7^{th} row in the table to model *the most frequent 30,000 words* and trained a second model using equation (6) with $k = 5$ to model all other words, and combined the two models with a mixing factor (which was also learned). The result "SSI: CFH (1-grams)" is the best performing method we have found. Doing the same trick but with 2-grams instead also improved over Low Rank SSI, but not by as much. Combining both 1-grams and 2-grams, however did not improve over 1-grams alone.

Training and Testing Splits In some cases, one might be worried that our experimental setup has split training and testing data only by partitioning the links, but not the documents, hence performance of our model when new unseen documents are added to the database might be in question. We therefore also tested an experimental setup where the test set of documents is completely separate from the training set of documents, by completely removing all training set links between training and testing documents. In fact, this does not alter the performance significantly, as shown in Table



(a) (b)

Fig. 1 Scatter plots of Average Precision for 500 documents: (a) SSI_{30k}+$\text{I}_{2.5M}$ vs. $\text{TFIDF}_{2.5M}$, (b) SSI_{30k}+$\text{I}_{2.5M}$ vs. the best combination of LSI_{30k} and $\text{TFIDF}_{2.5M}$.

3. This outlines that our model can accommodate a growing corpus without frequent re-training.

Importance of the Latent Concept Dimension In the above experiments we simply chose the dimension N of the low rank matrices to be the same as the best latent concept dimension for LSI. However, we also tried some experiments varying N and found that the error rates are fairly invariant to this parameter. For example, using a limited dictionary size of 30,000 words we achieve a ranking loss 0.39%, 0.30% or 0.31% for N=100, 200, 500 using a $W = U^\top V + I$ type model. More detailed results are given in Table 4.

Importance of the Identity matrix for Low Rank representations The addition of the identity term in our model $W = U^\top V + I$ allows this model to automatically learn the tradeoff between using the low dimensional space and a classical vector space model. The diagonal elements count when there are exact matches (co-ocurrences) of words between the documents. The off-diagonal (approximated with a low rank representation) captures topics and synonyms. Using only $W = I$ yields the inferior TFIDF model. Using only $W = U^\top V$ also does not work as well as $W = U^\top V + I$. Indeed, we obtain a mean average precision of 0.42 with the former, and 0.51 with the latter. Similar results can be seen with the error rate of LSI with or without adding the $(1 - \alpha)$TFIDF term, however for LSI this modification seems rather *ad-hoc* rather than being a natural constraint on the general form of W as in our method.

Ignoring the Diagonal On the other hand, for some tasks it is not possible to use the identity matrix at all, e.g. for cross-language retrieval. Out of curiosity, we thus also tested our method SSI training a dense matrix W where the diagonal is constrained to be zero[12], so only synonyms can be used. This obtained a test ranking loss of 0.69%

[12] Note that the model $W = U^\top V$ with the identity achieved a ranking loss of 0.56%, however this model can represent at least some of the diagonal.

(limited dictionary size case), compare to 0.41% *with* the diagonal. indicating this could be a very good cross-language retrieval model, which we explore further in Section 4.3.

4.2 Query-Document Retrieval

We also tested our approach in a query-document setup. We used the same setup as before but we constructed queries by keeping only k random words from query documents in an attempt to mimic a "keyword search". First, using the same setup as in the previous section with a limited dictionary size of 30,000 words we present results for keyword queries of length $k = 5, 10$ and 20 in Table 5. SSI yields similar improvements as in the document-document retrieval case over the baselines. Here, we do not report full results for Query Expansion, however it did not give large improvements over TFIDF, e.g. for the $k = 10$ case we obtain 0.084 MAP and 0.0376 P@10 for QE at best. Results for $k = 10$ using an unconstrained dictionary are given in Table 6. Again, SSI yields similar improvements. Overall, non-symmetric SSI gives a slight but consistent improvement over symmetric SSI. Changing the embedding dimension N (capacity) did not appear to effect this, for example for $k = 10$ and $N = 100$ we obtain 3.11% / 0.215 / 0.097 for Rank Loss/MAP/P@10 using SSI $W = U^\top U + I$ and 2.93% / 0.235 / 0.102 using SSI $W = U^\top V + I$ (results in Table 5 are for $N = 200$). Finally, correlated feature hashing again improved over models without hashing.

4.3 Cross Language Document-Document Retrieval

We considered the same set of 1,828,645 English Wikipedia documents and a set of 846,582 Japanese Wikipedia documents, where 135,737 of the documents are known to be about the same concept as a corresponding English page (this information can be found in the Wiki markup provided in a Wikipedia dump.) For example, the page about "Microsoft" can be found in both English and Japanese, and they are cross-referenced. These pairs are referred to as "mates" in the literature, see e.g. [16].

We then consider a cross language retrieval task that is analogous to the task in Section 4.1: given a Japanese query document q_{Jap} with English mate q_{Eng}, rank the English documents so that a document d_{Eng} linked from q_{Eng} appear above the unlinked ones. For this task, the document q_{Eng} is removed and not considered available during training or testing. There are in total 8.1M such q_{Jap}-d_{Eng} links. The dataset is split into train and test as before.

The first type of baseline we considered is based on machine translation. We use a machine translation tool on the Japanese query, and then apply TFIDF or LSI. We consider three methods of machine translation: Google's API[13] or Fujitsu's ATLAS[14] is used to translate each query document, or we translate each word in the Japanese dictionary using ATLAS and then apply this word-based translation to a query. We also compared to CL-LSI [16] trained on all 90,000 Jap-Eng mates from the training set, with a feasible limited vocabulary of $30,000$ words in both language.

For SSI, we consider two cases: (i) apply the ATLAS machine translation tool to a document first, and then use SSI trained on the monolingual task of Section 4.1, which

[13] `http://code.google.com/p/google-api-translate-java/`

[14] `http://www.fujitsu.com/global/services/software/translation/atlas/`

Table 5 Empirical results for query-document ranking on Wikipedia where query has k keywords (this experiment uses a limited dictionary size of $\mathcal{D} = 30,000$ words). For each k we measure the ranking loss, MAP and P@10 metrics.

			$k = 5$	
Algorithm	Params	Rank Loss	MAP	P@10
TFIDF	0	21.6%	0.047±0.004	0.023±0.0007
αLSI + $(1-\alpha)$TFIDF	$200\mathcal{D}$+1	14.2%	0.049±0.004	0.023±0.0007
SSI: $W = U^\top U + I$	$200\mathcal{D}$	4.80%	0.161±0.007	0.079±0.003
SSI: $W = U^\top V + I$	$400\mathcal{D}$	**4.37%**	**0.166±0.007**	**0.083±0.003**

			$k = 10$	
Algorithm	Params	Rank Loss	MAP	P@10
TFIDF	0	14.0%	0.083±0.006	0.035±0.001
αLSI + $(1-\alpha)$TFIDF	$200\mathcal{D}$+1	9.73%	0.089±0.006	0.037±0.001
SSI: $W = U^\top U + I$	$200\mathcal{D}$	3.10%	0.2138±0.0009	0.095±0.004
SSI: $W = U^\top V + I$	$400\mathcal{D}$	**2.91%**	**0.229±0.009**	**0.100±0.004**

			$k = 20$	
Algorithm	Params	Rank Loss	MAP	P@10
TFIDF	0	9.14%	0.128±0.007	0.054±0.002
αLSI + $(1-\alpha)$TFIDF	$200\mathcal{D}$+1	6.36%	0.133±0.007	0.059±0.002
SSI: $W = U^\top U + I$	$200\mathcal{D}$	1.87%	0.287±0.01	0.126±0.005
SSI: $W = U^\top V + I$	$400\mathcal{D}$	**1.80%**	**0.302±0.01**	**0.130±0.005**

Table 6 Empirical results for query-document ranking for $k = 10$ keywords (unlimited dictionary size of $\mathcal{D} = 2.5$ million words).

Algorithm	Params	Rank	MAP	P@10
TFIDF	0	12.86%	0.128±0.008	0.035±0.003
αLSI + $(1-\alpha)$TFIDF	$200 \times 30k$+1	8.95%	0.133±0.008	0.051±0.003
SSI: $W = U^\top V + I$	$400 \times 30k$	3.02%	0.261±0.010	0.113±0.004
SSI: Correlated Feature Hashing	$500 \times 30k$	**2.02%**	**0.315±0.011**	**0.135±0.005**

we call SSI_{EngEng}, or (ii) train SSI directly with Japanese queries and English target documents using the model (9), which we call SSI_{JapEng}.

The results are given in Table 9 where the dictionary size was once again limited to 30,000 words in both languages. Results for an unrestricted dictionary size are given in Table 10.

TFIDF using the three translation methods give relatively similar results. Using LSI or CL-LSI slightly improves these results, depending on the metric. Machine translation followed by SSI_{EngEng} outperforms all these methods, however the direct SSI_{JapEng} which requires no machine translation tool at all, improves results even further. We conjecture this is because translation mistakes generate noisy features which SSI_{JapEng} circumvents.

However, we also consider combining SSI_{JapEng} with TFIDF or SSI_{EngEng} using a mixing parameter α and this provided further gains, at the expense of requiring a machine translation tool once more.

Regularizing using Monolingual Retrieval The reported experiments do not include the regularization approach based on multi-tasking introduced in Section 2.4.2, Equation (10). In fact, we did not find that this strategy helped much unless we reduced the amount of Japanese-English training data, while using all the English-English and

Table 7 Correlated Feature Hashing: some examples of words along with their top 5 matches (from the most frequent 30,000 words) by DICE coefficient generated from Wikipedia.

riemannian	manifolds, manifold, tensor, curvature, euclidean
crustacean	appendages, shrimp, crustaceans, crab, arthropods
gurkha	nepalese, rangoon, rifles, nepali, kathmandu
carotid	artery, arteries, aortic, plexus, sinus,
noam	chomsky, linguistics, linguist, syntactic, anarchists
daggers	dagger, swords, axes, knives, bows
batgirl	gotham, joker, luthor, batman, arkham
blasphemy	heresy, crucifixion, qur'an, punishable, gospels
daimlerchrysler	chrysler, daimler, benz, suv, jeep

Table 8 Correlated Feature Hashing: some examples of 2-grams along with their top 5 matching words (from the most frequent 30,000 words) as predicted by their DICE scores generated from Wikipedia. Note that find similar words to only *one* of the words of these 2-grams on its own e.g. "black" or "holes" or "star" or "trek" would clearly give very different results.

pearl harbor	battleship, destroyers, carriers, planes, torpedoes
star trek	starfleet, spock, klingon, voyager, starship
minor leagues	inning, hitter, rbi, pitchers, strikeouts
grateful dead	phish, allman, joplin, janis, hendrix
james brown	funk, funky, sly, aretha, motown
middle east	arab, egypt, asia, centuries, syria
black holes	hawking, spacetime, galaxies, cosmological, relativity

Japanese-Japanese data. For example taking only 1% (81,000 pairs) of the Japanese-English data gives a ranking loss of 2.43% with regularization, compared to 3.02% without.

Mate Finding Note that many cross-lingual experiments, e.g. [16], typically measure the performance of finding a "mate", the same document in another language, whereas our experiment tries to model a query-based retrieval task. However, we also performed an experiment in the mate-finding setting. In this case, SSI achieves a ranking test error of 0.53%, and CL-LSI achieves 0.81%.

4.4 Content Matching

We present results on matching adverts to web pages, a problem closely related to document retrieval. We obtained proprietary data from an online advertising company of the form of pairs of web pages and adverts that were clicked while displayed on that page. We only considered clicks in position 1 and discarded the sessions in which none of the ads was clicked. This is a way to circumvent the well known position bias problem — the fact that links appearing in lower positions are less likely to be clicked even if they are relevant. Indeed, by construction, every negative example comes from a slate of adverts in which there was a click in a lower position; it is thus likely that the user examined that negative example but deemed it irrelevant (as opposed to the user not clicking because he did not even look at the advert).

We consider these (webpage,clicked-on-ad) pairs as positive examples (q, d^+), and any other randomly chosen ad is considered as a negative example d^- for that query page. 1.9M pairs were used for training and 100,000 pairs for testing. The web pages

Table 9 Cross-lingual Japanese document-English document ranking (limited dictionary size of 30,000 words). Algorithms which use machine translations of the query are denoted with the subscript $_{EngEng}$; unless specified ATLAS document-based translation was used.

Algorithm	Rank Loss	MAP	P@10
TFIDF_{EngEng}(Google translated queries)	4.78%	0.319±0.009	0.259±0.008
TFIDF_{EngEng}(ATLAS word-based only)	8.27%	0.115±0.005	0.103±0.005
TFIDF_{EngEng}(ATLAS translated queries)	4.83%	0.290±0.008	0.243±0.008
LSI_{EngEng}(ATLAS translated queries)	7.54%	0.169±0.007	0.150±0.007
$\alpha\text{LSI}_{EngEng}+(1-\alpha)\text{TFIDF}_{EngEng}$	3.71%	0.300±0.008	0.253±0.008
CL-LSI_{JapEng}	9.29%	0.190±0.007	0.161±0.007
$\alpha\text{CL-LSI}_{JapEng}+(1-\alpha)\text{TFIDF}_{EngEng}$	3.31%	0.275±0.009	0.212±0.008
SSI_{EngEng}	1.72%	0.399±0.009	0.325±0.009
SSI_{JapEng}	0.96%	0.438±0.009	0.351±0.009
$\alpha\text{SSI}_{JapEng}+(1-\alpha)\text{TFIDF}_{EngEng}$	0.75%	0.493±0.009	0.377±0.009
$\alpha\text{SSI}_{JapEng}+(1-\alpha)\text{SSI}_{EngEng}$	**0.63%**	**0.524±0.009**	**0.386±0.009**

Table 10 Cross-lingual Japanese document-English document ranking (unlimited dictionary size).

Algorithm	Rank Loss	MAP	P@10
TFIDF_{EngEng}(Google translated queries)	4.16%	0.346±0.009	0.311±0.009
TFIDF_{EngEng}(ATLAS translated queries)	4.21%	0.338±0.009	0.280±0.008
$\alpha\text{LSI}_{EngEng}+(1-\alpha)\text{TFIDF}_{EngEng}$	3.24%	0.365±0.009	0.309±0.009
SSI_{EngEng}	1.53%	0.443±0.009	0.367±0.009
$\alpha\text{SSI}_{JapEng}+(1-\alpha)\text{TFIDF}_{EngEng}$	0.78%	0.499±0.009	0.380±0.009
$\alpha\text{SSI}_{JapEng}+(1-\alpha)\text{SSI}_{EngEng}$	**0.60%**	**0.527±0.009**	**0.391±0.009**

contained 87 features (words) on average, while the ads contained 19 features on average. The two classes (clicks and no-clicks) are roughly balanced. From the way we construct the dataset, this means than when a user clicks on an advert, he/she clicks about half of the time on the one in the first position.

We compared TFIDF, Hash Kernels and Low Rank SSI on this task. The results are given in Table 11. In this case TFIDF performs very poorly, often the positive (page, ad) pairs share very few, if any, features, and even if they do this does not appear to be very discriminative. Hash Kernels and Low Rank SSI appear to perform rather similarly, both strongly outperforming TFIDF. The rank loss on this dataset is two orders of magnitude higher than on the Wikipedia experiments described in the previous sections. This is probably due to a combination of two factors: first, the positive and negative classes are balanced, whereas there was only a few positive documents in the Wikipedia experiments; and second, clicks data are much more noisy.

We might add, however, that at test time, Low Rank SSI has a considerable advantage over Hash Kernels in terms of speed. As the vectors Uq and Vd can be cached for each page and ad, a matching operation only requires N multiplications (a dot product in the "embedding" space). However, for hash kernels $|q||d|$ hashing operations and multiplications have to be performed, where $|\cdot|$ means the number of non-zero elements. For values such as $|q| = 100, |d| = 100$ and $N = 100$ that would mean Hash Kernels would be around 100 times slower than Low Rank SSI at test time, and this difference gets larger if more features are used.

Table 11 Content Matching experiments on proprietary data of web-page/advertisement pairs.

Algorithm	Parameters	Rank Loss
TFIDF	0	45.60%
SSI: Hash Kernels [37]	1M	26.15%
SSI: Hash Kernels	10M	25.56%
SSI: $W = (U^\top V)_{10k} + I$	$50 \times 10k = 0.5M$	25.83%
SSI: $W = (U^\top V)_{20k} + I$	$50 \times 20k = 1M$	26.68%
SSI: $W = (U^\top V)_{30k} + I$	$50 \times 30k = 1.5M$	26.98%

4.5 Notes on Overfitting and Regularization

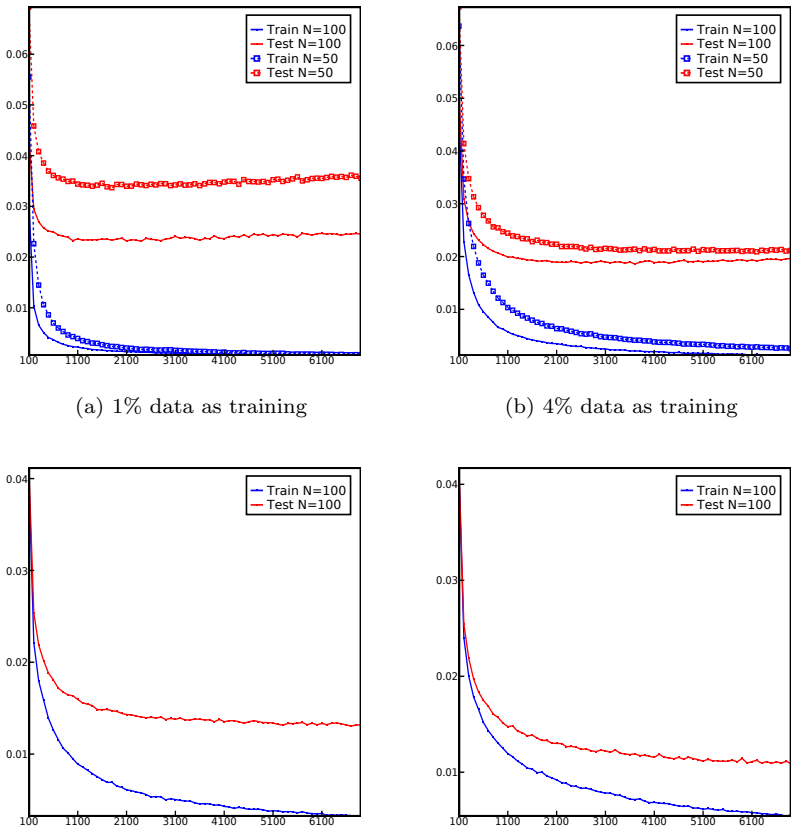
As stated earlier, our low rank model provides two mechanisms for capacity control: the selection of the latent dimension N and early stopping. Reducing the latent dimension N limits the rank of the matrix W and hence reduces the capacity of the model. Early stopping stops training when the performance over a validation set stops increasing. It hence stops learning before the training loss is minimized, expressing preference for weights close to the random initialization [10].

Figure 2 illustrates the effect of these two regularization strategies over cross-language experiments with different training set sizes. It appears that overfitting occurs when training data size is very small (1% and 4%). The gap between training and testing error reduces with increased training data size, as expected. However, especially for larger training set sizes, one can see that we did not experience a divergence between train and test error as the number of iterations of online learning increased.

In Figure 2(a) and 2(b), we also display the training curve for models with different capacity ($N = 50$ and $N = 100$). One can see that in both cases, the model with $N = 100$ yields the best performance. One can further note the advantage of early stopping: eg. in the first figure, it lasts longer for $N = 50$ than for $N = 100$ before the test error starts increasing. Early stopping can stop training in such cases and hence yield better generalization performance. For larger training sets, one can notice that early stopping is less important as the generalization error does not increase. However, it is still usefull to stop training when the generalization performance stalls as it simply reduces the training computational cost.

5 Conclusion

We have described a versatile, powerful set of discriminatively trained models for document ranking. Many "learning to rank" papers have focused on the problem of selecting the objective to optimize (given a fixed class of functions) and typically use a relatively small number of hand-engineered features as input. This work is orthogonal to those works as it studies models with large feature sets generated by all pairs of words between the query and target texts. The challenge here is that such feature spaces are very large, and we thus presented several models that deal with the memory, speed and capacity control issues. In particular we proposed low rank and sparse learning and a technique of hashing correlated features. In fact, all of our proposed methods can be used in conjunction with other features and objective functions explored in previous work for further gains.



(a) 1% data as training

(b) 4% data as training

(c) 10% data as training

(d) 30% data as training

Fig. 2 Cross-language retrieval task: training with different training data sizes. In each figure, the (lower) blue curve is training error, and the (higher) red curve is testing error. Note that the total data size is 8.1M q_{Jap}-d_{Eng} pairs, so 1% data is about 80,000 pairs.

Our empirical study covered query and document retrieval, cross-language retrieval and ad-placement. Our main conclusions were: (i) we found that the low rank model outperforms the full rank margin ranking perceptron with the same features as well as its sparsified version. We also outperform classical methods such as TFIDF, LSI or query expansion. Finally, it is also better than or comparable to "Hash Kernel", another new supervised technique, in terms of accuracy, while having advantages in terms of efficiency; and (ii) Using Correlated feature hashing improves results even further. Both the low rank idea from (i) and correlated feature hashing (ii) prove to be effective ways to reduce the feature space size.

Many generalizations of our work are possible: adding more features into our models as we just mentioned, generalizing to other kinds of nonlinear models, and exploring the use of the same models for other tasks such as question answering. In general, web search and other standard retrieval tasks currently often depend on entering query keywords which are likely to be contained in the target document, rather than the user

directly describing what they want to find. Our models are capable of learning to rank using either the former or the latter.

References

1. R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*. Addison-Wesley Harlow, England, 1999.
2. B. Bai, J. Weston, R. Collobert, and D. Grangier. Supervised semantic indexing. In *European Conference on Information Retrieval*, 2009.
3. Adam Berger and John Lafferty. Information retrieval as statistical translation. In *ACM SIGIR' 99*, pages 222–229, 1999.
4. D. M. Blei and J. D. McAuliffe. Supervised topic models. In *In Advances in Neural Information Processing Systems (NIPS)*, 2007.
5. D. M. Blei, A. Ng, and M. I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.
6. R. Bunescu and M. Pasca. Using encyclopedic knowledge for named entity disambiguation. In *In EACL*, pages 9–16, 2006.
7. C. Burges, R. Ragno, and Q.V. Le. Learning to Rank with Nonsmooth Cost Functions. In *Advances in Neural Information Processing Systems: Proceedings of the 2006 Conference*. MIT Press, 2007.
8. 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, New York, NY, USA, 2005. ACM Press.
9. Z. Cao, T. Qin, T.Y. Liu, M.F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM Press New York, NY, USA, 2007.
10. R. Caruana, S. Lawrence, and L. Giles. Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems, NIPS 13*, pages 402–408. 2000.
11. S. Chernov, T. Iofciu, W. Nejdl, and X. Zhou. Extracting semantic relationships between wikipedia categories. In *In 1st International Workshop: SemWiki2006 - From Wiki to Semantics (SemWiki 2006), co-located with the ESWC2006 in Budva*, 2006.
12. M. Collins and N. Duffy. New ranking algorithms for parsing and tagging: kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics Morristown, NJ, USA, 2001.
13. R. Collobert and S. Bengio. Links between perceptrons, mlps and svms. In *ICML 2004*, 2004.
14. S. Cucerzan. Large-scale named entity disambiguation based on wikipedia data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 708–716, Prague, June 2007. Association for Computational Linguistics.
15. 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.
16. S.T. Dumais, T.A. Letsche, M.L. Littman, and T.K. Landauer. Automatic cross-language retrieval using latent semantic indexing. In *AAAI Spring Symposium on Cross-Language Text and Speech Retrieval*, 1997.
17. E. Gabrilovich and S. Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *International Joint Conference on Artificial Intelligence*, 2007.
18. P. Gehler, A. Holub, and M. Welling. The rate adapting poisson (rap) model for information retrieval and object recognition. In *Proceedings of the 23rd International Conference on Machine Learning*. 2006.
19. A. Globerson and S. Roweis. Visualizing pairwise similarity via semidefinite programming. In *AISTATS*. 2007.
20. S. Goel, J. Langford, and A. Strehl. Predictive indexing for fast search. In *Advances in Neural Information Processing Systems 21*. 2009.
21. D. Grangier and S. Bengio. Inferring document similarity from hyperlinks. In *CIKM '05*, pages 359–360, New York, NY, USA, 2005. ACM.

22. D. Grangier and S. Bengio. A discriminative kernel-based approach to rank images from text queries. *IEEE Trans. PAMI.*, 30(8):1371–1384, 2008.
23. G. Grefenstette. *Cross-Language Information Retrieval*. Kluwer Academic Publishers Norwell, MA, USA, 1998.
24. I. M. Guyon, S. R. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature Extraction: Foundations and Applications*. Springer, August 2006.
25. R. Herbrich, T. Graepel, and K. Obermayer. *Large margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000.
26. T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR 1999*, pages 50–57. ACM Press, 1999.
27. J. Hu, L. Fang, Y. Cao, H. Zeng, H. Li, Q. Yang, and Z. Chen. Enhancing text clustering by leveraging wikipedia semantics. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 179–186, New York, NY, USA, 2008. ACM.
28. P. Jain, B. Kulis, I. S. Dhillon, and K. Grauman. Online metric learning and fast similarity search. In *Advances in Neural Information Processing Systems (NIPS)*. 2008.
29. T. Joachims. Optimizing search engines using clickthrough data. In *ACM SIGKDD*, pages 133–142, 2002.
30. M. Keller and S. Bengio. A Neural Network for Text Representation. In *International Conference on Artificial Neural Networks, ICANN*, 2005. IDIAP-RR 05-12.
31. J. Langford, L. Li, and T. Zhang. Sparse Online Learning via Truncated Gradient. In *Advances in Neural Information Processing Systems 21*. 2009.
32. T.Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, 2007.
33. D. N. Milne, I. H. Witten, and D. M. Nichols. A knowledge-based search engine powered by wikipedia. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 445–454, New York, NY, USA, 2007. ACM.
34. Z. Minier, Z. Bodo, and L. Csato. Wikipedia-based kernels for text categorization. In *In 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 157–164, 2007.
35. M. Ruiz-casado, E. Alfonseca, and P. Castells. Automatic extraction of semantic relationships for wordnet by means of pattern learning from wikipedia. In *In NLDB*, pages 67–79. Springer Verlag, 2005.
36. R. Salakhutdinov and G. Hinton. Semantic Hashing. *Proceedings of the SIGIR Workshop on Information Retrieval and Applications of Graphical Models, Amsterdam.*, 2007.
37. Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, A. Strehl, and V. Vishwanathan. Hash kernels. In *Twelfth International Conference on Artificial Intelligence and Statistics*, 2009.
38. F. Smadja, K. R. McKeown, and V. Hatzivassiloglou. Translating collocations for bilingual lexicons: a statistical approach. *Comput. Linguist.*, 22(1):1–38, 1996.
39. J. Sun, Z. Chen, H. Zeng, Y. Lu, C. Shi, and W. Ma. Supervised latent semantic indexing for document categorization. In *ICDM 2004*, pages 535–538, Washington, DC, USA, 2004. IEEE Computer Society.
40. A. Vinokourov, J. Shawe-Taylor, and N. Cristianini. Inferring a Semantic Representation of Text via Cross-Language Correlation Analysis. *NIPS*, pages 1497–1504, 2003.
41. E. M. Voorhees and H. T. Dang. Overview of the trec 2005 question answering track. In *In TREC 2005*, 2005.
42. X. Wang, J. Sun, Z. Chen, and C. Zhai. Latent semantic analysis for multiple-type interrelated data objects. In *SIGIR'06*, 2006.
43. K. Weinberger and L. Saul. Fast solvers and efficient implementations for distance metric learning. In *International Conference on Machine Learning*. 2008.
44. Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR*, pages 271–278, 2007.
45. L. Zighelnic and O. Kurland. Query-drift prevention for robust query expansion. In *SIGIR 2008*, pages 825–826, New York, NY, USA, 2008. ACM.