Learning Term-weighting Functions for Similarity Measures

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

Abstract

Measuring the similarity between two texts is a fundamental problem in many NLP and IR applications. Among the existing approaches, the cosine measure of the term vectors representing the original texts has been widely used, where the score of each term is often determined by a TFIDF formula. Despite its simplicity, the quality of such cosine similarity measure is usually domain dependent and decided by the choice of the term-weighting function. In this paper, we propose a novel framework that learns the term-weighting function. Given the labeled pairs of texts as training data, the learning procedure tunes the model parameters by minimizing the specified loss function of the similarity score. Compared to traditional TFIDF term-weighting schemes, our approach shows a significant improvement on tasks such as judging the quality of query suggestions and filtering irrelevant ads for online advertising.

1 Introduction

Measuring the semantic similarity between two texts is an important problem that has many useful applications in both NLP and IR communities. For example, Lin (1998) defined a similarity measure for automatic thesaurus creation from a corpus. Mihalcea et al. (2006) developed several corpus-based and knowledge-based word similarity measures and applied them to a paraphrase recognition task. In the domain of web search, different methods of measuring similarity between short text segments have recently been proposed for solving problems like query suggestion and alternation (Jones et al., 2006; Sahami and Heilman, 2006; Metzler et al., 2007; Yih and Meek, 2007).

Among these similarity measures proposed in various applications, the vector-based methods are arguably the most widely used. In this approach, the text being compared with is first represented by a term vector, where each term is associated with a weight that indicates its importance. The similarity function could be cosine (i.e., the inner product of two normalized unit term vectors, or equivalently a linear kernel), or other kernel functions such as the Gaussian kernel.

There are essentially two main factors that decide the quality of a vector-based similarity measure. One is the *vector operation* that takes as input the term vectors and computes the final similarity score (e.g., cosine). The other is how these term vectors are constructed, including the term selection process and how the weights are determined. For instance, a TFIDF scheme for measuring document similarity may follow the bag-of-words strategy to include all the words in the document when constructing the term vectors. The weight of each term is simply the product of its term frequency (i.e., the number of occurrences in the document) and inverse document frequency (i.e., the number of documents in a collection that contain this term).

Despite its simplicity and reasonable performance, such approach suffers from several weaknesses. For instance, the similarity measure is not domain-dependent and cannot be easily adjusted to better fit the final objective, such as being a metric value used for clustering or providing better ranking results. Researchers often need to experiment with variants of TFIDF formulas and different term selection strategies (e.g., removing stopwords or stemming) to achieve acceptable performance (Manning et al., 2008). In addition, when more information is available, such as the position of a term in the document or whether a term is part of an anchor text, incorporating it in the similarity measure in a principled manner may not be easy.

In this paper, we propose a general ***t***erm-***w***eighting l***ea***rning framewor***k***, TWEAK, that learns the term-weighting function for the vector-based similarity measures. Instead of using a fixed formula to decide the weight of each term, TWEAK uses a parametric function of features of each term, where the model parameters are learned from labeled data. Although the weight of each term conceptually represents its importance with respect to the document, tuning the model parameters to optimize for such objectives may not be the best strategy due to two reasons. While the label of whether a pair of texts is similar is not difficult to collect from human annotators[1], the label of whether a term in a document is important is often very ambiguous and hard to decide. Even if such annotation issue can be resolved, aligning the term weights with the *true* importance of each term may not necessarily lead to our real objective – deriving a better similarity measure for the target application. Therefore, our learning framework, TWEAK, assumes that we are given only the labels of the pairs of texts being compared, such as whether the two texts are considered similar by human subjects.

TWEAK is flexible in choosing various loss functions that are close to the true objectives, while still maintaining the simplicity of the vector-based similarity measures. For example, a system that implements the TFIDF cosine measure can easily replace the original term-weighting scores with the ones output by TWEAK without changing other portions of the algorithm. TWEAK is also novel compared to other existing learning methods for similarity measures. For instance, we do not learn the scores of all the terms in the vocabulary directly, which is one of the methods proposed by Bilenko and Mooney (2003). Because the vocabulary size is typically large in the text domain (e.g., all possible words in English), learning directly the term-weighting scores may suffer from the data sparsity issue and cannot generalize well in practice. Instead, we focus on learning the model parameters for features that each term may have, which results in a much smaller feature space. TWEAK also differs from the model combination approach proposed by Yih and Meek (2007), where the output scores of different similarity measures are combined via a learned linear function. In contrast, TWEAK effectively learns a new similarity measure by tuning the term-weighting function and can potentially be complementary to the model combination approach.

[1]As argued in (Sheng et al., 2008), low-cost labels may nowadays be provided by outsourcing systems such as Amazon's Mechanical Turk or online ESP games.

As will be demonstrated in our experiments, in applications such as judging the relevance of different query suggestions and determining whether a paid-search ad is related to the user query, TWEAK can incorporate various kinds of term–document information and learn a term-weighting function that significantly outperforms the traditional TFIDF scheme in several evaluation metrics, when using the same vector operation (i.e., cosine) and the same set of terms.

We organize the rest of the paper as follows. Sec. 2 first gives a high-level view of our term-weighting learning framework. We then formally define our model and present the loss functions that can be optimized for in Sec. 3. Experiments on target applications are presented in Sec. 4. Finally, we compare our approach with some related work in Sec. 5 and conclude the paper in Sec. 6.

2 Problem Statement

To simplify the description, assume that the texts we are comparing are two documents. A general architecture of vector-based similarity measures can be formally described as follows. Given two documents D_p and D_q, a similarity function maps them to a real-valued number, where a higher value indicates these two documents are semantically more related, considered by the measure.

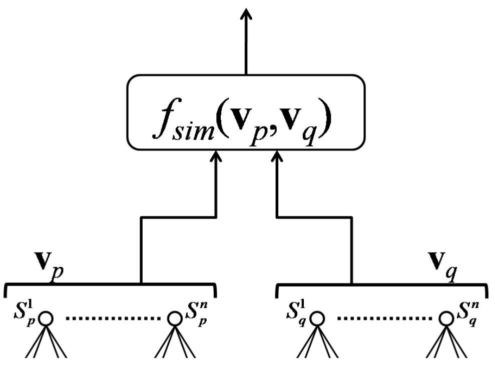
Suppose a pre-defined vocabulary set $\mathcal{V} = \{t_1, t_2, \cdots, t_n\}$ consists of all possible terms (e.g., tokens, words) that may occur in the documents. Each document D_p is represented by a term vector of length n: $\mathbf{v}_p = (s_p^1, s_p^2, \cdots, s_p^n)$, where $s_p^i \in \mathbf{R}$ is the weight of term t_i, and is determined by the term-weighting function tw that depends on the term and the document (i.e., $s_p^i \equiv tw(t_i, D_p)$). The similarity between documents D_p and D_q is then computed by a vector operation function $f_{sim} : (\mathbf{v}_p, \mathbf{v}_q) \to \mathbf{R}$, illustrated in Fig. 1.

Determining the specific functions f_{sim} and tw effectively decides the final similarity measure. For example, the functions that construct the traditional TFIDF cosine similarity can be:

$$f_{sim}(\mathbf{v}_p, \mathbf{v}_q) \equiv \frac{\mathbf{v}_p \cdot \mathbf{v}_q}{||\mathbf{v}_p|| \cdot ||\mathbf{v}_q||} \quad (1)$$

$$tw(t_i, D_p) \equiv tf(t_i, D_p) \cdot \log\left(\frac{N}{df(t_i)}\right) \quad (2)$$

Figure 1: A general architecture of vector-based similarity measures



where N is the size of the document collection for deriving document frequencies, tf and df are the functions computing the term frequency and document frequency, respectively.

In contrast, TWEAK also takes a specified vector function f_{sim} but assumes a parametric term-weighting function $tw_{\mathbf{w}}$. Given the training data, it learns the model parameters $\mathbf{w}$ that optimize for the designated loss function.

3 Model

As a specific instantiation of our learning framework, the term-weighting function used in this paper is a linear combination of features extracted from the input term and document. In particular, the weight of term t_i with respect to document D_p is

$$s_p^i = tw_{\mathbf{w}}(t_i, D_p) \equiv \sum_j w_j \phi_j(t_i, D_p), \qquad (3)$$

where ϕ_j is the j-th feature function and w_j is the corresponding model parameter.

As for the vector operation function f_{sim}, we use the same cosine function (Eq. 1). Notice that we choose these functional forms for their simplicity and good empirical performance shown in preliminary experiments. However, other smooth functions can certainly be used.

The choice of loss function for training model parameters depends on the true objective in the target application. In this work, we consider two different learning settings: learning directly the *similarity metric* and learning the *preference ordering*, and compare several loss functions experimentally.

3.1 Learning Similarity Metric

In this setting, we assume that the learning algorithm is given a set of document pairs. Each of them is associated with a label that indicates whether these two documents are similar (e.g., a binary label where 1 means similar and 0 otherwise) or the degree of similarity (e.g., a real-valued label ranges from 0 to 1), considered by the human subjects. A training set of m examples can be denoted as $\{(y_1, (D_{p_1}, D_{q_1})), (y_2, (D_{p_2}, D_{q_2})), \cdots, (y_m, (D_{p_m}, D_{q_m}))\}$, where y_k is the label and (D_{p_k}, D_{q_k}) is the pair of documents to compare. Following the vector construction described in Eq. 3, let $\mathbf{v}_{p_1}, \mathbf{v}_{q_1}, \cdots, \mathbf{v}_{p_m}, \mathbf{v}_{q_m}$ be the corresponding term vectors of these documents.

We consider two commonly used loss functions, *sum-of-squares error* and *log loss*[2]:

$$L_{sse}(\mathbf{w}) = \frac{1}{2}\sum_k^m (y_k - f_{sim}(\mathbf{v}_{p_k}, \mathbf{v}_{q_k}))^2 \qquad (4)$$

$$\begin{aligned} L_{log}(\mathbf{w}) = \sum_k^m & -y_k \log(f_{sim}(\mathbf{v}_{p_k}, \mathbf{v}_{q_k})) \\ & -(1-y_k)\log(1 - f_{sim}(\mathbf{v}_{p_k}, \mathbf{v}_{q_k})) \end{aligned} \qquad (5)$$

Eq. 4 and Eq. 5 can further be regularized by adding $\frac{\alpha}{2}||\mathbf{w}||^2$ in the loss function, which may improve the performance empirically and also constrain the range of the final term-weighting scores. Learning the model parameters for minimizing these loss functions can be done using standard gradient-based optimization methods. We choose the L-BFGS (Nocedal and Wright, 2006) method in our experiments for its guarantee to find a local minimum and fast convergence. The derivation of gradients is fairly straightforward, which we skip here.

Notice that other loss functions can also be used in this framework. Interested readers can refer to, say, (Bishop, 1995), for other loss functions and their theoretical justifications.

3.2 Learning Preference Ordering

In many applications where the similarity measure is applied, the goal is to obtain a *ranked* list of the candidate elements. For example, in the task of

[2]Although in theory the cosine function may return a negative value and make the log-loss uncomputable, this can be easily avoided in practice by selecting appropriate initial model parameters and by constraining the term-weighting scores to be non-negative.

filtering irrelevant ads, a good similarity measure is expected to rank appropriate ads higher than the irrelevant ones. A desired trade-off of false-positive (mistakenly filtered good ads) and false-negative (unfiltered bad ads) can be achieved by selecting a decision threshold. The exact value of the similarity measure, in this case, is not crucial. For these applications, it is more important if the model parameters can better predict the *pairwise preference*. Learning preference ordering is also motivated by the observation that preference annotations are generally more reliable than categorical similarity labels (Carterette et al., 2008) and has been advocated recently by researchers (e.g., Burges et al. (2005)).

In the setting of learning preference ordering, we assume that each training example consists of *two* pairs of documents, associated with a label indicating which pair of documents is considered more preferable. A training set of m examples can be formally denoted as $\{(y_1, (x_{a_1}, x_{b_1})), (y_2, (x_{a_2}, x_{b_2})), \cdots, (y_m, (x_{a_m}, x_{b_m}))\}$, where $x_{a_k} = (D_{p_{a_k}}, D_{q_{a_k}})$ and $x_{b_k} = (D_{p_{b_k}}, D_{q_{b_k}})$ are two pairs of documents and $y_k \in \{0, 1\}$ indicates the pairwise order preference, where 1 means x_{a_k} should be ranked higher than x_{b_k} and 0 otherwise.

We use a loss function that is very similar to the one proposed by Dekel et al. (2004) for label ranking. Let Δ_k be the difference of the similarity scores of these two document pairs. Namely,

$$\Delta_k = f_{sim}(\mathbf{v}_{p_{a_k}}, \mathbf{v}_{q_{a_k}}) - f_{sim}(\mathbf{v}_{p_{b_k}}, \mathbf{v}_{q_{b_k}})$$

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:

$$L(\mathbf{w}) = \sum_{k=1}^{m} \log(1+\exp(-y_k \cdot \Delta_k - (1-y_k) \cdot (-\Delta_k))) \quad (6)$$

Similarly, Eq. 6 can be regularized by adding $\frac{\alpha}{2}||\mathbf{w}||^2$ in the loss function.

4 Experiments

We demonstrate how to apply our term-weighting learning framework, TWEAK, to measuring similarity for short text segments and to judging the relevance of an ad landing page given an query. In addition, we compare experimentally the performance of using different training settings, loss functions and features against the traditional TFIDF term-weighting scheme.

4.1 Similarity for Short Text Segments

Judging the similarity between two short text segments is a crucial problem for many search and online advertising applications. For instance, *query reformulation* or *query substitution* needs to measure the similarity between two queries. A product keyword recommendation system needs to determine whether the given product name and the suggested keyword is related.

Because the length of the text segment is typically short, ranging from a single word to a dozen words, naively applying methods based on word overlapping such as the Jaccard coefficient leads to poor results (Sahami and Heilman, 2006; Yih and Meek, 2007). To overcome this difficulty, Sahami and Heilman (2006) proposes a Web-kernel function, which first expands the short text segment by issuing it to a search engine as the query, and then collectes the snippets of the top results to construct a pseudo-document. TFIDF term vectors of the pseudo-documents are used to represent the original short text segments and the cosine score of these two vectors is used as the similarity measure.

In this section, we apply TWEAK to this problem by replacing the TFIDF term-weighting scheme with the learned term-weighting function, when constructing the vectors from the pseudo-documents. Our target application is *query suggestion* – automatically presenting queries that are related to the one issued by the user. In particular, we would like to use our similarity measure as a filter to determine whether queries suggested by various algorithms and heuristics are indeed closely related to the target query.

4.1.1 Task & Data

Our query suggestion dataset has been previously used in (Metzler et al., 2007; Yih and Meek, 2007) and is collected in the following way. From the search logs of a commercial search engine, a random sample of 363 thousand queries from the top 1 million most frequent queries in late 2005 were first taken as the query and suggestion candidates. Among them, 122 queries were chosen randomly as our target queries; each of them had up to 100 queries used as suggestions, generated by various query suggestion mechanisms.

Given these pairs of query and suggestions, human annotators judged the level of similarity using a 4-point scale – *Excellent*, *Good*, *Fair* and *Bad*, where Excellent and Good suggestions are considered clearly related to the query intent, while the other two categories mean the suggestions are either too general or totally unrelated. In the end, 4,852 query/suggestion pairs that had effective annotations were collected. The distribution of the four labels is: Excellent - 5%, Good - 12%, Fair - 44% and Bad - 39%.

For the simplicity of both presentation and implementation, query/suggestion pairs labeled as Excellent or Good are treated as positive examples and the rest as negative ones. Notice that TWEAK is not restricted in using only binary labels. For instance, the pairwise preference learning setting only needs to know which pair of objects being compared is more preferred. The model and algorithm do not have to change regardless of whether the label reflects the degree of similarity (e.g, the original 4-scale labels) or binary categories. For the metric learning setting, an ordinal regression approach (e.g, (Herbrich et al., 2000)) can be applied for multi-category labels.

We used the same query expansion method as described in (Sahami and Heilman, 2006). Each query/suggestion was first issued to a commercial search engine. The result page with up to 200 snippets (i.e., titles and summaries) was used as the pseudo-document to create the term vector that represents the original query/suggestion. As described earlier in Eq. 3, the weight of each term is a linear function of a set of predefined features, which are described next.

4.1.2 Features

Because the pseudo-documents are constructed using the search result snippets instead of regular web documents, special formatting or link information provided by HTML is not very meaningful. Therefore, we focused on using features that are available for plain-text documents, including:

- **Bias**: 1 for all examples.
- **TF**: We used $\log(tf + 1)$ as the *term frequency* feature, where tf is the number of times the term occurs in the original pseudo-document.
- **DF**: We used $\log(df + 1)$ as the *document frequency* feature, where df is the number of documents in our collection that contain this term.
- **QF**: The search engine query log reflects the distribution of the words/phrases in which people are interested (Goodman and Carvalho, 2005; Yih et al., 2006). We took a log file with the most frequent 7.5 million queries and used $\log(qf + 1)$ as feature, where qf is the query frequency.
- **Cap**: A capitalized word may indicate being part of a proper noun or being more important. When the term is capitalized in at least one occurrence in the pseudo-document, the value of this feature is 1; otherwise, it is 0.
- **Loc & Len**: The beginning of a regular document often contains a summary with important words. In the pseudo-documents created using search snippets, words that occur in the beginning come from the top results, which are potentially more relevant to the original query/suggestion. We created two specific features using this location information. Let loc be the word position of the target term and len be the total number of words of this pseudo-document. The logarithmic value $\log(loc + 1)$ and the ratio loc/len were both used as features. In order for the learning procedure to adjust the scaling, the logarithmic value of the document length, $\log(len + 1)$, was also used.

4.1.3 Results

We conducted the experiments using 10-fold cross-validation. The whole query/suggestion pairs were first split into 10 subsets of roughly equal sizes. Pairs with the same target query were put in the same subset. In each round, one subset was used for testing. 95% of the remaining data was used for training the model and 5% was used as the development set. We trained six models with different values of the regularization hyperparameter $\alpha \in \{0.003, 0.01, 0.03, 0.1, 0.3, 1\}$ and determined which model to use based on its performance on the development set, although the result actually did not vary a lot as α changed.

We compared three learning configurations – metric learning with sum-of-squares error (Metric$_{sse}$) and log loss (Metric$_{log}$) and the pairwise preference learning (Preference). The learned term-weighting functions were used to compare with the Web-kernel similarity function, which implemented the TFIDF term-weighting scheme using Eq. 2.

Table 1: The AUC scores, mean averaged precision and precision at 3 of similarity measures using different term-weighting functions. The numbers with the † sign are statistically significantly better compared to the Web-kernel method.

Method	AUC	MAP	Prec@3
Web-kernel	0.732	0.540	0.556
Metric$_{sse}$	0.775†	0.590	0.553
Metric$_{log}$	0.781†	0.585	0.545
Preference	**0.782**†	**0.597**†	**0.570**

We evaluated these models using three different evaluation metrics: the AUC score, precision at k and MAP (mean averaged precision). The area under the ROC curve (AUC) is typically used to judge the overall quality of a ranking function. It has been shown equivalent to the averaged accuracy of the pairwise preference predictions of all possible element pairs in the sequence, and can be calculated by the the following Wilcoxon-Mann-Whitney statistic (Cortes and Mohri, 2004):

$$A(f;\mathbf{x},\mathbf{y}) = \sum_{i,j:y_i>y_j} \mathbf{I}_{f(x_i)>f(x_j)} + \frac{1}{2}\mathbf{I}_{f(x_i)=f(x_j)},$$

where f is the similarity measure, $\mathbf{x}$ is the sequence of compared elements and $\mathbf{y}$ is the labels.

Another metric that is commonly used in a ranking scenario is *precision at* k, which computes the accuracy of the top-ranked k elements and ignores the rest. We used $k = 3$ in our task, which means that for each target query, we selected three suggestions with the highest similarity scores and computed the averaged accuracy.

One issue of precision at k is that it does not provide an overall quality measure of the ranking function. Therefore, we also present MAP (mean averaged precision), which is a single number that summarizes the performance of the ranking function by considering both precision and recall, and has been shown reliable in evaluating various information retrieval tasks (Manning et al., 2008). Suppose there are m relevant elements in a sequence, where $r_1, r_2, \cdots, r_m$ are their locations. The averaged precision is then:

$$AP = \frac{1}{m}\sum_{j=1}^{m} \mathrm{Prec}(r_j),$$

where $\mathrm{Prec}(r_j)$ is the precision at r_j. We computed the averaged precision values of the 10 test sets in our cross-validation setting and report their mean value.

As shown in Table 1, all three learned term-weighting functions lead to better similarity measures compared to the TFIDF scheme in terms of the AUC and MAP scores, where the preference order learning setting performs the best. However, for the precision at 3 metric, only the preference learning setting has a higher score than the TFIDF scheme, but the difference is not statistically significant[3]. This is somewhat understandable since the design of our loss function focuses on the overall quality instead of only the performance of the top ranked elements.

4.2 Query/Page Similarity

Measuring whether a page is relevant to a given query is the main problem in information retrieval and has been studied extensively. Instead of retrieving web pages that are relevant to the query according to the similarity measure, our goal is to implement a paid-search ad filter for commercial search engines. In this scenario, textual ads with bid keywords that match the query can enter the auction and have a chance to be shown on the search result page. However, as the advertisers may bid on keywords that are not related to their advertisements, it is important for the system to filter irrelevant ads to ensure that users only receive useful information. For this purpose, we measure the similarity between the query and the ad landing page (i.e., the page pointed by the ad) and remove the ad when the score of its landing page is below a pre-selected threshold[4].

Given a pair of query and ad landing page, while the *query* term vector is constructed using the same query expansion technique described in Sec. 4.1, the *page* term vector can be created directly from the web page since it is a regular document that contains enough content. As usual, our goal is to produce a better similarity measure by learning the term-weighting functions for these two types of vectors jointly.

[3]We conducted a paired-t test on the 10 individual scores from the cross-validation results of each learned term-weighting function versus the Web-kernel method. The results are considered statistically significant when the p-value is lower than 0.05.

[4]One may argue that the filter should measure the similarity between the query and ad-text. However, an ad will not provide useful information to the user if the final destination page is not relevant to the query, even if its ad-text looks appealing.

4.2.1 Data

We first collected a random sample of queries and paid-search ads shown on a commercial search engine during 2008, as well as the ad landing pages. Judged by several human annotators, each page was labeled as relevant or not compared to the issued query. After removing some pairs where the query intent was not clear or the landing page was no longer available, we managed to collect 13,341 query/page pairs with reliable labels. Among them, 8,309 were considered relevant and 5,032 were labeled irrelevant.

4.2.2 Features

In this experiment, we tested the effect of using different features and experimented with three feature sets: *TF&DF*, *Plain-text* and *HTML*. *TF&DF* contains only $\log(tf+1)$, $\log(df+1)$ and the bias feature. The goal of using this feature set is to test whether we can learn a better term-weighting function given the *same* amount of information as the TFIDF scheme has. The second feature set, *Plain-text*, consists of all the features described in Sec. 4.1.2. As mentioned earlier, this set of features can be used for regular text documents that do not have special formatting information. Finally, feature set *HTML* is composed of all the features used in *Plain-text* plus features extracted from some special properties of web documents, including:

- **Hypertext**: The anchor text in an HTML document usually provides important information. If there is at least one occurrence of the term that appears in some anchor text, the value of this feature is 1; otherwise, it is 0.
- **URL**: A web document has a uniquely useful property – the name of the document, which is its URL. If the term is a substring of the URL, then the value of this feature is 1; otherwise, it is 0.
- **Title**: The value of this feature is 1 when the term is part of the title; otherwise, it is 0.
- **Meta**: Besides Title, several meta tags used in the HTML header explicitly show the important words selected by the page author. Specifically, whether the term is part of a meta-keyword is used as a binary feature. Whether the term is in the meta-description segment is also used.

Table 2: The AUC scores, true-positive rates at false-positive rates 0.1 and 0.2 of the ad filter based on different term-weighting functions. The difference between any pair of numbers of the same evaluation metric is statistically significant.

Method	AUC	$\text{TPR}_{\text{fnr}=0.1}$	$\text{TPR}_{\text{fnr}=0.2}$
TFIDF	0.794	0.527	0.658
TF&DF	0.806	0.430	0.639
Plain-text	0.832	0.503	0.704
HTML	**0.855**	**0.568**	**0.750**

Because the term vector that represents the query is created from the pseudo-document (i.e., a collection of search snippets), the values of these HTML-specific features are all 0 for the query term vector. This set of features are only useful for deciding the weights of the terms in a page term vector.

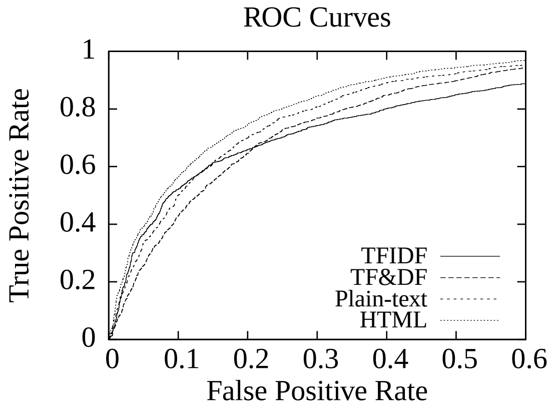
4.2.3 Results

We split our data into 10 subsets and conducted the experiments using the same 10-fold cross-validation setting described in Sec. 4.1.3, including how we used the development set to select the regularization hyper-parameter α. The pairs that have the same target query were again put in the same subsets. We used only the preference ordering learning setting for its good performance shown in the previous set of experiments. Models compared here were learned from the three different sets of features, as well as the same fixed TFIDF term-weighting formula (i.e., Eq. 2) used in Sec. 4.1. Table 2 reports the averaged results of the 10 testing sets in AUC, as well as the true-positive rates at two low false-positive rate points (FPR=0.1 and FPR=0.2). The difference between any pair of numbers of the same evaluation metric is statistically significant[5].

As we can see from the table, having more features does lead to a better term-weighting function. With all features (i.e., *HTML*), the model achieves the highest AUC score among all configurations. Features available in plain-text documents (i.e., *Plain-text*) other than term frequency and document frequency can still improve the performance significantly. When only the TF and DF features are available, the learned term-weighting function still outperforms the TFIDF scheme, al-

[5]We conduct paired-t tests as described in Sec. 4.1.3. All the p-values after Bonferroni correction are less than 0.01.

Figure 2: ROC Curves of the ad filters using different term-weighting functions



though the improvement gain is much smaller compared to the other two settings.

Notice that the behaviors of these models at different false-positive regions varies from the traditional TFIDF scheme. At a low false-positive point (e.g., FPR=10%), only the model that uses all features performs better than TFIDF. This phenomenon can be clearly observed from the ROC curves plotted in Fig. 2, where the models were trained using half of the data and applied to the other half to generate the similarity scores. If only the performance at a very low false-positive rate matters, TWEAK can still be easily adjusted by modifying the loss function using techniques such as training with utility (Domingos, 1999; Morik et al., 1999).

5 Related Work

Our term-weighting learning framework can be analogous to the "Siamese" architecture for learning jointly two neural networks that share the same set of model weights (Bromley et al., 1993). For instance, a term vector can be viewed as a very large single-layer neural network, where each term in the vocabulary is a node that takes as input the features and outputs the learned term-weighting score. Previous applications of this learning machine are typically problems in image processing or computer vision. For example, Chopra et al. (2005) designed an algorithm to learn a similarity metric for face verification, which is based on the difference between two vectors. In our earlier experiments (not reported in this paper) of using vector difference instead of cosine, we did not observe positive outcomes. We hypothesize that because the length of the term vector in our problem can be extremely large (i.e., the size of the vocabulary), a similarity measure based on vector difference can easily be affected by terms that do not occur in both documents, even when the co-occurred terms have very large weights.

Learning similarity measures for text has also been proposed by several researchers. For instance, Bilenko and Mooney (2003) applied SVMs to directly learn the weights of co-occurred words in two text records, which are then used for measuring similarity for duplicate detection. Although this approach worked moderately well in the database domain, it may not be suitable to handle general text similarity problems for two reasons. First, the vocabulary size is typically large, which results in a very high dimensional feature space for the learning problem. It is very likely that some rarely used and yet important terms occur in the testing documents but not in the training data. The weights of those terms may not be reliable or even be learned. Second, this learning approach can only learn the importance of the terms from the labels of whether two texts are considered similar, how to incorporate the basic information of these terms such as the position or query log frequency is not clear.

An alternative learning approach is to combine multiple similarity measures with learned coefficients (Yih and Meek, 2007), or to apply the technique of *kernel alignment* (Cristianini et al., 2002) to combining a set of kernel functions for tuning a more appropriate kernel based on labeled data. This type of approaches can be viewed as constructing an ensemble of different existing similarity measures without modifying the term weighting function, and may not generate mathematically equivalent similarity functions as derived by TWEAK. Although learning in this approach is usually very fast due to the model form and the small number of parameters to learn, its improvement is limited by the quality of the individual similarity measures. In spite of the fundamental difference between our approach and this combination method, it is worth noticing that these two approaches are in fact complementary to each other. Having a newly learned term-weighting function effectively provides a new similarity measure and therefore can be combined with other measures.

6 Conclusions

In this paper, we presented a novel term-weighting learning framework, TWEAK, for improving similarity measures based on term vectors. Given the labels of text pairs for training, our method learns the model parameters to calculate the score of each term, optimizing the desired loss function that is suitable for the target application. As we demonstrated in the experiments, TWEAK with different features and training settings significantly outperforms the traditional TFIDF term-weighting scheme.

TWEAK also enjoys several advantages compared to existing methods. From an engineering perspective, adopting the new term-weighting scores produced by our model is straightforward. If a similarity measure has been implemented, the algorithm need not be changed – only the term vectors need to be updated. From the learning perspective, additional information regarding each term with respect to the document can now be incorporated easily via feature functions. Weights (i.e., model parameters) of these features are learned in a principled way instead of being adjusted manually. Finally, TWEAK is potentially complementary to other methods for improving the similarity measure, such as model combination of various types of similarity measures (Yih and Meek, 2007) or different term vector construction methods such as Latent Semantic Analysis (Deerwester et al., 1990).

In the future, we plan to explore more vector operations other than the inner-product (i.e., cosine) as well as different functional forms of the term-weighting function (e.g. log-linear instead of linear). Designing new loss functions to better fit the true objectives in various target applications and studying the quality of a similarity measure based on both term-weighting learning and model combination are also on our agenda. In terms of applications, we would like to apply TWEAK in other problems such as paraphrase recognition and near-duplicate detection.

Acknowledgments

The author thanks the anonymous reviewers for their valuable comments and is grateful to Asela Gunawardana, Chris Meek, John Platt and Misha Bilenko for many useful discussions.

References

Mikhail Bilenko and Raymond J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of KDD-2003*, pages 39–48.

Christopher M. Bishop. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press.

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

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine learning (ICML-05)*, pages 89–96.

Ben Carterette, Paul N. Bennett, David Maxwell Chickering, and Susan Dumais. 2008. Here or there: Preference judgments for relevance. In *Proceedings of the 30th European Conference on Information Retrieval (ECIR 2008)*.

Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of CVPR-2005*, pages 539–546.

Corinna Cortes and Mehryar Mohri. 2004. AUC optimization vs. error rate minimization. In *Advances in Neural Information Processing Systems (NIPS 2003)*.

Nello Cristianini, John Shawe-Taylor, Andre Elisseeff, and Jaz Kandola. 2002. On kernel-target alignment. In *Advances in Neural Information Processing Systems 14*, pages 367–373. MIT Press.

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)*.

Pedro Domingos. 1999. MetaCost: A general method for making classifiers cost-sensitive. In *Proceedings of KDD-1999*, pages 155–164.

Joshua Goodman and Vitor R. Carvalho. 2005. Implicit queries for email. In *Proceedings of the 2nd conference on Email and Anti-Spam (CEAS-2005)*.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. 2000. Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers*, pages 115–132.

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

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.

Donald Metzler, Susan Dumais, and Christopher Meek. 2007. Similarity measures for short segments of text. In *Proceedings of the 29th European Conference on Information Retrieval (ECIR 2007)*.

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

Katharina Morik, Peter Brockhausen, and Thorsten Joachims. 1999. Combining statistical learning with a knowledge-based approach – a case study in intensive care monitoring. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-1999)*, pages 268–277.

Jorge Nocedal and Stephen Wright. 2006. *Numerical Optimization*. Springer, 2nd edition.

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*.

Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. 2008. Get another label? Improving data quality and data mining using multiple, noisy labelers. In *Proceedings of KDD-2008*, pages 614–622.

Wen-tau Yih and Christopher Meek. 2007. Improving similarity measures for short segments of text. In *Proceedings of AAAI-2007*, pages 1489–1494.

Wen-tau Yih, Joshua Goodman, and Vitor Carvalho. 2006. Finding advertising keywords on web pages. In *Proceedings of the 15th World Wide Web Conference*.