Learning for Efficient Supervised Query Expansion via Two-stage Feature Selection

Zhiwei Zhang[1]*, Qifan Wang[2], Luo Si[13], Jianfeng Gao[4]
[1]Dept of CS, Purdue University, IN, USA [2]Google Inc, Mountain View, USA
[3]Alibaba Group Inc, USA [4]Microsoft Research, Redmond, WA, USA
{zhan1187,lsi}@purdue.edu, wqfcr618@gmail.com, jfgao@microsoft.com

ABSTRACT

Query expansion (QE) is a well known technique to improve retrieval effectiveness, which expands original queries with extra terms that are predicted to be relevant. A recent trend in the literature is Supervised Query Expansion (SQE), where supervised learning is introduced to better select expansion terms. However, an important but neglected issue for SQE is its efficiency, as applying SQE in retrieval can be much more time-consuming than applying Unsupervised Query Expansion (UQE) algorithms. In this paper, we point out that the cost of SQE mainly comes from term feature extraction, and propose a Two-stage Feature Selection framework (TFS) to address this problem. The first stage is adaptive expansion decision, which determines if a query is suitable for SQE or not. For unsuitable queries, SQE is skipped and no term features are extracted at all, which reduces the most time cost. For those suitable queries, the second stage is cost constrained feature selection, which chooses a subset of effective yet inexpensive features for supervised learning. Extensive experiments on four corpora (including three academic and one industry corpus) show that our TFS framework can substantially reduce the time cost for SQE, while maintaining its effectiveness.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Search and Retrieval

Keywords

Query Expansion, Supervised Learning, Efficiency

1. INTRODUCTION

Queries provided by users can sometimes be ambiguous and inaccurate in an information retrieval system, which may generate unsatisfactory results. Query expansion (QE) is a well known technique to address this issue, which expands the original queries with some extra terms that are predicted to be relevant [26]. It is hoped that these expanded terms can capture user's true intent that is missed in original query, thus improving the final retrieval effectiveness. In the past decades, various applications [26, 6] have proved its value.

Unsupervised QE (UQE) algorithms used to be the mainstream in the QE literature. Many famous algorithms, such as relevance model (RM) [21] and thesaurus based methods [29], have been widely applied. However, recent studies [5, 22] showed that a large portion of expansion terms selected by UQE algorithms are noisy or even harmful, which limits their performance. *Supervised Query Expansion* (SQE) is proposed to overcome this disadvantage by leveraging the power of supervised learning. Most of existing SQE algorithms [5, 22, 13, 25, 2] follow a classical machine learning pipeline: (1) utilize UQE to select initial candidate terms; (2) features of candidate terms are extracted; (3) pre-trained classifiers or rankers are utilized to select the best terms for expansion. Significant effectiveness improvement has been reported over their unsupervised counterparts, and SQE has become the new state-of-the-art.

Besides effectiveness, efficiency is another important issue in QE-involved retrieval [35]. As we will show later, UQE algorithms are usually very efficient to apply. Therefore, when UQE is adopted in retrieval, the major inefficiency comes from the second retrieval, which retrieves the entire corpus for the expanded queries. This issue is traditionally handled by indexing or documents optimization [35, 3, 30]. But recently Diaz [11] showed that simply reranking the retrieval results of original queries can already provide nearly identical performance with very low time costs, particularly for precision-oriented metrics.

However, the efficiency issue of SQE algorithms imposes new challenge beyond the UQE case. Compared with UQE algorithms, SQE requires extra time to apply supervised learning, which can incur significant time cost. Moreover, this issue is unique to SQE, and cannot be addressed by previous QE efficiency methods such as indexing optimization or reranking. Unfortunately, although important, this issue has been largely neglected in the literature.

The above observations motivate us to propose new research to address this SQE efficiency problem. In this paper, we point out that the major time cost of applying SQE algorithms comes from term feature extraction. Indeed leveraging extensive features can enhance the effectiveness of supervised learning, so that better expansion terms can be selected. However, it also inevitably decreases the efficiency. Aiming at this point, we propose a Two-stage Fea-

*Part of this work was done while the first author interned at Microsoft.

SIGIR '16, July 17-21, 2016, Pisa, Italy

DOI: http://dx.doi.org/10.1145/2911451.2911539

ture Selection framework (TFS) to balance the two conflicting goals. The first stage is Adaptive Expansion Decision (AED), which predicts whether a query is suitable for SQE or not. For unsuitable queries, SQE is skipped with no features being extracted, so that the time cost is reduced most. For suitable queries, the second stage conducts Cost Constrained Feature Selection (CCFS), which chooses a subset of effective yet inexpensive features for supervised learning. We then instantiate TFS for a RankSVM based SQE algorithm. Extensive experiments on four corpora (including three academic and one industry corpus) show that our TFS framework can substantially reduce the time cost of SQE algorithm, meanwhile maintaining its effectiveness.

The rest of the paper is organized as follows: Sec. 2 introduces the preliminaries of our work, including problem analysis and literature review; Sec. 3 presents the Two-stage Feature Selection framework and its instantiation; Sec. 4 gives all experiments, and in Sec. 5 we conclude this paper.

2. PRELIMINARIES

In this section, we will thoroughly analyze the SQE efficiency problem. Meanwhile we will review the literature, and point out the difference between our work and previous works. The discussions below are presented in three subsections, each covering one specific aspect.

2.1 QE Algorithm Analysis

First we will review some basics about query epansion.

QE Formulation. Suppose we have a user query q with n terms $q = \{t_i^q | i = 1:n\}$. Suppose m expansion terms are selected by a QE algorithm, denoted as $\{t_i^e | i = 1:m\}$. Then the expanded query q^e is their union, i.e. $q^e = \{t^q\} \cup \{t^e\}$. Each term $t \in q^e$ is weighted by the interpolated probability

$$P(t|q^e) = (1-\lambda)P(t|q) + \lambda P_{QE}(t) \quad (1)$$

where $P(t|q)$ is the probability of term t occurring in q (i.e. $P(t|q) = \frac{\text{frequency of term } t \text{ in } q}{\text{query length } |q|}$), $P_{QE}(t)$ is the term probability given by QE algorithm, and λ is the interpolation coefficient to be tuned. As can be seen, the key question here is how to select good expansion terms.

UQE versus SQE. Unsupervised QE (UQE) algorithms used to be the mainstream in QE literature. For example, some well known UQE algorithms include relevance model (RM) [21], positional relevance model [24], and mixture model [36]. UQE algorithms are very popular because on one hand their formulations are in general simple, and on the other hand their empirical performance is quite reasonable. However, recent works [5, 22] observed that a large portion of the expansion terms from UQE can be noisy or even harmful, which limits their performance.

SQE tackles this problem by introducing supervised learning to predict the quality of candidate expansion terms. Cao et al. [5] proposed perhaps the first SQE research, where they designed a set of term features and applied SVM for term classification (either good or bad). Later Lee et al. [22] claimed that ranking oriented term selection outperforms classification oriented methods. Gao et al. [13, 12] applied SQE to web search, where search log is utilized for candidate term generation. Some other extensions include QE robustness [25], query reformulation [2], etc. A common pipeline of SQE training and testing [5, 13] is summarized in Alg. 1. Notice here we only concern test-time efficiency, rather than the training-time efficiency.

Algorithm 1 SQE Training and Testing Pipeline

▷ Training SQE model $\mathcal{H}$
1: For training query q, record its retrieval accuracy r^q (e.g. ERR@20). Select M candidate terms $\{t_i^c | \text{i=1:M}\}$ via UQE.
2: Each time, a single candidate term t^c is appended to q, i.e. $q^c = q \cup t^c$; record its retrieval accuracy r_c^q; then $\Delta r_c^q = r_c^q - r^q$ is the label for t^c.
3: Extract term features $\mathcal{F}_t$ for all t^c, and train a classifier (based on if $\Delta r_c^q > 0$ or $\Delta r_c^q \leq 0$) or train a ranker (based on the ranking order of Δr_c^q), denoted as $\mathcal{H}$.

▷ Testing (i.e. applying $\mathcal{H}$ in QE retrieval)
1: For testing query q, use UQE to select M candidate terms.
2: Extract $\mathcal{F}_t$ for all candidate terms.
3: Apply $\mathcal{H}$ to get top m terms for expansion.

2.2 QE Efficiency Analysis

Now we will analyze the efficiency issue when QE is applied in retrieval.

QE in Retrieval. The retrieval process with QE can be described as follows. Let $\mathcal{C}$ denote the target corpus upon which we will run and evaluate retrieval; denote $\mathcal{S}$ as the resource from which expansion terms are extracted. In traditional pseudo relevance feedback (PRF) scenario [6], $\mathcal{S} = \mathcal{C}$. In more general scenario, $\mathcal{S}$ is not necessarily the same as $\mathcal{C}$. For example, in web search, $\mathcal{C}$ (e.g. Clueweb09) might be too low-quality to be used for QE [1]; instead some other resources of higher quality can be used as $\mathcal{S}$ (e.g. search log [10] or Wikipedia [1]). Assuming a retrieval algorithm (e.g. BM25 or KL divergence) is utilized, then a typical process of QE in retrieval is summarized in Table 1:

Table 1: QE in retrieval with and without reranking.

(1) First Retrieval: search original query q on resource $\mathcal{S}$;
(2) Applying QE Model: select expansion terms $\{t^e\}$ from $\mathcal{S}$.
(3) Second Retrieval:
(Full) retrieve corpus $\mathcal{C}$ for expanded query q^e.
——————OR——————
(Reranking)
(A) If $\mathcal{C} \neq \mathcal{S}$, retrieve $\mathcal{C}$ for q, denote the results as L; If $\mathcal{C} = \mathcal{S}$, then let the results of first retrieval as L;
(B) Rerank L for expanded query q^e.

In the above table we list two possible implementations of second retrieval. The full second retrieval is more traditional, in which the entire target corpus $\mathcal{C}$ is retrieved for expanded query q^e. This, however, is painfully time-consuming, particularly on large scale corpus. Recently Diaz [11] suggested to rerank the retrieval results of original query q as the results for q^e. Diaz pointed out that this reranking implementation can provide nearly identical performance as the full second retrieval, particularly for precision-oriented evaluation metrics. Our preliminary experiments also verified this statement. Therefore throughout this paper, we will utilize reranking as the default implementation for second retrieval. Notice in the (A) step, we present the different implementation details regarding both PRF scenario ($\mathcal{C} = \mathcal{S}$) and non-PRF scenario ($\mathcal{C} \neq \mathcal{S}$).

Existing QE Efficiency Studies. Despite the usefulness of reranking, the majority of existing works on QE efficiency still focused on how to speed up the full second retrieval. As far as we know, all of these works addressed the problem by optimizing underlying data structures such as indexing or document representation. Billerbeck et al. [3] proposed to use compact document summaries to reduce retrieval time. Lavrenko et al. [20] pre-calculated pairwise document similarities to reduce the amount of calculation

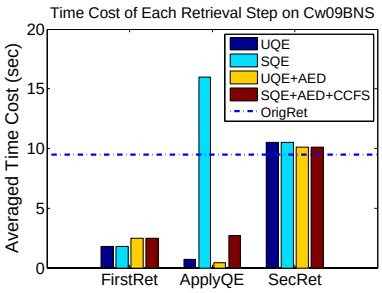

Figure 1: Comparison of the time cost of each retrieval step. AED and CCFS are the two-stages in our TFS framework, the target corpus $\mathcal{C}$ is Cw09BNS, resource $\mathcal{S}$ is Wikipedia, the number of expansion terms is 20, and the averaged time costs per query are reported by running experiments using a single-thread program on a single PC. The blue line is the averaged retrieval time cost for original query.

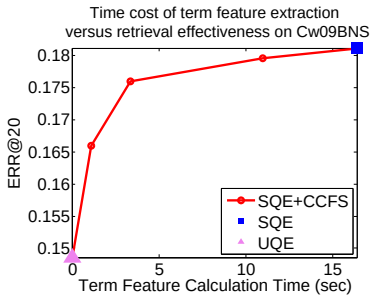

Figure 2: With more time spent on term feature extraction, more term features will be obtained, and higher retrieval accuracy can be achieved.

when searching expanded queries. Wu et al. [35] utilized a special index structure named impact-sorted indexing that improves the scoring procedures in retrieval. Theobald et al. [30] proposed the idea of merging inverted list of different terms in an incremental on-demand manner so that document scan can be delayed as much as possible. Unfortunately, our goal now is not the second retrieval, which is handled by reranking as [11] does. Nor can the inefficiency challenge of SQE be handled by the above data-level approaches.

2.3 SQE Efficiency Analysis

Now we will show why the efficiency issue of SQE is a unique and challenging problem beyond the UQE case.

Step-wise Time Cost Analysis. First let's see how UQE and SQE differs in the time cost spent on each retrieval step. On Clueweb09-B corpus, we conduct UQE (RM [21]) and SQE (applying RankSVM [19] based on Cao et al's work [5]) for QE with 20 expansion terms. As comparison, we apply our Two-stage Feature Selection framework (TFS) to SQE. Notice that although UQE does not involve term feature extraction, we can still apply adaptive expansion decision to UQE. In Figure 1, we show the time cost of each retrieval step with respect to Table 1. More experiment details can be found in Sec. 4. Here we mainly discuss the observations that motivate our research.

We can observe that, indeed applying SQE model can be much more time-consuming than applying UQE model, which supports our previous statement and validates our motivation. Notice here, step "FirstRet" and "ApplyQE" are involved in expansion term selection, while "SecRet" includes only retrieving $\mathcal{C}$ for original query q and reranking for expanded query q^e. Also notice the reranking in second retrieval only incurs very low time cost.

Feature Extraction in SQE. It is then natural to ask, which part of SQE incurs the major time cost, and why?

We argue that term feature extraction is the major inefficient part in SQE models. Recall the testing phase in Alg. 1, there are three steps to apply SQE model. The first step is essentially UQE, which is in general very efficient. The third step, which applies learned SQE model $\mathcal{H}$ for term classification or ranking, is also efficient in practice. For example, many SQE works [5, 22, 13, 2] adopted linear model, which is extremely fast yet effective. Therefore, the second step of term feature extraction constitutes the majority of inefficiency.

But why would we spend much time on term feature extraction? This is because predicting the quality of terms is very challenging, so we want plenty of powerful features to enhance the learning algorithm [5, 13, 22]. In Figure 2, we show the retrieval accuracy (ERR@20) when different time cost is spent on term feature extraction. The purple triangle is UQE which does not extract any term feature (i.e. time cost equals zero); the blue rectangle is the full SQE model with all available term features (defined later). In the middle is our proposed cost constrained feature selection method. Clearly, with more time spent, more term features will be obtained, and the retrieval accuracy is higher as well. However, this inevitably degrades the efficiency.

We can further observe that, with more term features, although the retrieval accuracy of SQE is usually higher, the marginal gain becomes smaller. This motivates us to find a subset of features such that their total cost is low, meanwhile the effectiveness is reasonably maintained. This coincides with the idea of feature selection [15], although most of such methods only concern the number of selected features while ignoring their difference in time cost, which can be suboptimal.

3. TWO-STAGE FEATURE SELECTION FOR EFFICIENT SQE RETRIEVAL

Based on the above analysis, in this section we will present the proposed Two-stage Feature Selection (TFS) framework. Below we will first present TFS as a general framework, then instantiate it for an example SQE algorithm.

3.1 A General Framework

Assume the initial full set of term features are $\mathcal{F}_t$, where subscript t indicates the features are for terms. As analyzed earlier, when retrieval effectiveness is the only goal, $\mathcal{F}_t$ tends to become abundant and inefficient. Therefore, the goal of our TFS framework is to select a subset of term features $\mathcal{F}_t^*$ from $\mathcal{F}_t$, so that the effectiveness and efficiency can be optimally balanced. As mentioned earlier, the TFS framework includes the following two stages:

- Adaptive Expansion Decision (AED), which predicts whether a query is suitable for SQE or not. For unsuitable queries, SQE is skipped with no term features being extracted, which reduces the time cost most. To this end, AED builds a classifier $\mathcal{V}_{AED}$ with pre-defined query feature $\mathcal{F}_q$, so that $\mathcal{V}_{AED}(\mathcal{F}_q) < 0$ for unsuitable queries and $\mathcal{V}_{AED}(\mathcal{F}_q) > 0$ for suitable ones.
- Cost Constrained Feature Selection (CCFS), which selects a subset of effective yet inexpensive term features for SQE to apply. For those SQE-suitable queries that pass the first stage, this second stage can further reduce the time cost

Table 2: SQE in Retrieval with TFS.

(1) First retrieval:
(1.1) Search original query q on resource $\mathcal{S}$.
(1.2) AED: Extract query feature $\mathcal{F}_q$. If $\mathcal{V}_{AED}(\mathcal{F}_q) \leq 0$, directly go to step (3.1); otherwise continue.
(2) Apply QE model:
(2.1) Apply UQE to generate candidate expansion terms.
(2.2) CCFS: extract term features $\mathcal{F}_t^*$ for candidate terms.
(2.3) Select the best terms to form expanded query q^e.
(3) Second retrieval (Reranking):
(3.1) If $\mathcal{V}_{AED}(\mathcal{F}_q) \leq 0$: if target corpus is the resource (i.e. $\mathcal{C} = \mathcal{S}$), then return the retrieval results in step (1.1); if $\mathcal{C} \neq \mathcal{S}$, retrieve $\mathcal{C}$ for original query q. Exit.
(3.2) If $\mathcal{V}_{AED}(\mathcal{F}_q) > 0$: if $\mathcal{C} = \mathcal{S}$, then rerank the retrieval results in step (1.1) for q^e; if $\mathcal{C} \neq \mathcal{S}$, retrieve $\mathcal{C}$ for original query q, then rerank the results for q^e. Exit.

to some extent. To this end, CCFS builds a feature selector $\mathcal{V}_{CCFS}$, which requires the time cost u_f of each term feature f, and a pre-defined overall time cost upper bound U. In this way, there is $\mathcal{F}_t^* = \mathcal{V}_{CCFS}(\mathcal{F}_t)$ and $\sum_{f \in \mathcal{F}_t^*} u_f \leq U$.

Accordingly, the complete retrieval process is shown in Table 2, which is self-evident to interpret. Below we will give more details about the two stages.

3.1.1 Adaptive expansion decision (AED)

Training. The training process of AED classifier $\mathcal{V}_{AED}$ is as follows. For training query q, we first retrieve corpus $\mathcal{C}$ and record its retrieval accuracy as r_q (e.g. ERR@20 value). Then we apply the SQE model $\mathcal{H}$ from Alg. 1 to get the expanded query q^e. Retrieve $\mathcal{C}$ for q^e by following the procedures in Table 2, and denote the retrieval accuracy as $r_q^{\mathcal{H}}$. Then if $r_q^{\mathcal{H}} > r_q$, we assign label $+1$ to query q, which means SQE can help improve the retrieval effectiveness for q; otherwise we assign -1 to q. Finally, we extract query features $\mathcal{F}_q$ for q, and adopt some supervised learning algorithm (e.g. SVM) to get the classifier $\mathcal{V}_{AED}$.

Discussion. The idea of AED stems from query performance prediction. It is known that query expansion may hurt the retrieval effectiveness for some queries [25, 7]. Therefore, accurately predicting those queries and avoid applying expensive SQE model to them can substantially improve the efficiency. This idea of adaptive expansion has also been applied in [18, 27, 23, 8], although their works mainly focused on retrieval effectiveness and did not report the efficiency advantage that this method might bring.

3.1.2 Cost constrained feature selection (CCFS)

Despite the existence of AED, queries that pass AED still face the problem of expensive term feature extraction in SQE. Now we will explain how cost constrained feature selection is designed for those SQE-suitable queries.

Algorithm Design. As mentioned, CCFS aims to select a subset of term features $\mathcal{F}_t^*$ from the complete feature set $\mathcal{F}_t$, so that the overall time cost will not exceed a pre-defined upper bound U, i.e. $\mathcal{F}_t^* = \mathcal{V}_{CCFS}(\mathcal{F}_t)$ with $\sum_{f \in \mathcal{F}_t^*} u_f \leq U$.

Our CCFS algorithm is formulated as follows. Since the SQE model $\mathcal{H}$ is used to predict the quality of expansion terms, we assume $X \in \mathbb{R}^{N*K}$ as the feature matrix for N candidate terms, where each row is a K-dimensional feature vector for each term. Denote $Y \in \mathbb{R}^{N*1}$ as the corresponding labels for terms in X, which is calculated as the Δr_c^q in Alg. 1. Assume the SQE model $\mathcal{H}$ is learned via a loss function $L_{\mathcal{H}}(X, Y|\theta)$, where θ is the model parameter. We introduce feature selector $d \in \{0,1\}^K$, where the ith element

Algorithm 2 Cost Constrained Feature Selection

Input Learning algorithm $\mathcal{H}$, algorithm parameter θ, data $X \in \mathbb{R}^{N*K}$, label $Y \in \mathbb{R}^{N*1}$, algorithm loss function $L_{\mathcal{H}}(X, Y|\theta)$, feature cost vector $u \in \mathbb{R}^{K*1}$, final cost upper bound U, cost decrease ΔU.

Output Feature selector $d^* \in \{0,1\}^K$ satisfying $u^T d^* \leq U$, and the learned parameter θ^*.

1: $t \leftarrow 0, d^* = \mathbf{1}_{K*1}, U^0 = u^T \mathbf{1}_{K*1}$
2: **do**
3: $X \leftarrow Xd^*$, learn $\theta^* = \arg\min_\theta L_{\mathcal{H}}(X, Y|\theta)$.
4: Learn $d^* = \arg\min_d L_{\mathcal{H}}(Xd, Y|\theta^*)$, $s.t.$ $d \in \{0,1\}^K$, $u^T d \leq U^i$.
5: $U^{t+1} \leftarrow U^t - \Delta U, t \leftarrow t+1$.
6: **while** $U^i > U$

$d_i = 1$ means the ith feature is selected. With fixed d, those unselected features in X will become invalid, which is equivalent to $X \leftarrow Xd$. Together θ and d form a revised learning objective as follows:

$$d, \theta = \arg\min L_{\mathcal{H}}(Xd, Y|\theta), \quad s.t. \quad u^T d \leq U. \tag{2}$$

The optimization process is shown in Alg. 2, where a coordinate descent strategy is adopted to iteratively optimize w.r.t θ and d. During the iteration, we gradually decrease the cost upper bound (i.e. ΔU), so that the feature selection process can be smooth. In extreme case where $U \geq U^0$, Alg. 2 will produce the same results as vanilla $\mathcal{H}$ where no selection occurs (i.e. $d = \mathbf{1}_{K*1}$).

Discussion. Feature selection is a hot research topic in machine learning [15], and has been successfully adopted in information retrieval [13, 14, 33]. Popular methods include L1 based regularization [28, 13], cascading structure [31], feature number constraint [34], etc. Theoretically, any feature selection method can reduce time cost. But in practice we find better effectiveness can be achieved if the time cost of each feature can be explicitly considered. Unfortunately, most of previous research did not model such feature cost difference, which can be suboptimal. Wang et al. [33] proposed a greedy search algorithm for time cost aware feature selection in learning to rank. We also compare this method in our experiments, which shows our formulation outperforms this greedy design. CCFS can also be applied to the AED if necessary, which is straightforward. But due to space limitation, we simplify our experiments by only applying CCFS to term features.

3.2 Instantiation for RankSVM based SQE

So far we have elaborated all the details of TFS as a general framework. Now we will instantiate it with respect to a representative SQE algorithm, to show the implementation details. For SQE, we adopt a RankSVM based SQE method. Linear model has been widely applied in SQE literature [2, 13, 12]. Moreover, a ranking perspective has been proven to be very effective [22, 12] for SQE. Therefore, we believe RankSVM will make a representative example. Also Lavrenko's Relevance Model (RM) is utilized as the UQE model to generate candidate expansion terms, which is arguably one of the most successful UQE algorithms.

3.2.1 AED

The AED stage is simple to implement. Here we utilize SVM with Gaussian kernel for $\mathcal{V}_{AED}$ training. The adopted query features $\mathcal{F}_q$ are listed in Table 3, which are all well known query performance prediction features in the literature [17, 23, 16, 37].

Table 3: Query Features $\mathcal{F}_q$

Description	Formulation
Qry length	$\|q\|$
Qry Entropy	$\sum_{t\in q} -P(t\|D_F)log_2 p(t\|D_F)$
Qry Clarity 1	$\sum_{t\in q} P(t\|q)log\frac{P(t\|q)}{P(t\|C)}$
Qry Clarity 2	$\sum_{t\in q} -P(t\|D_F)log\frac{P(t\|D_F)}{P(t\|C)}$
Feedback Radius	$\frac{1}{\|D_F\|}\sum_{d\in D_F}\sum_{t\in d} P(t\|d)log\frac{P(t\|d)}{p(t\|\bar{d})}$, $P(t\|\bar{d}) = \frac{1}{\|D_F\|}\sum_{d\in D_F} P(t\|d)$
Qry IDF Var	$\text{var}(\text{idf}(t\in q))$, $idf(t) = \frac{log_2(\|C\|+0.5)/\|D_t\|}{log_2(\|C\|+1)}$
Max IDF	$\max idf(t\in q)$
AvICTF	$\frac{1}{\|q\|}log_2\prod_{t\in q}\frac{N}{N_t}$
Qry Collection Similarity	$\sum_{t\in q}(1+logN_t)log(1+\frac{\|C\|}{\|D_t\|})$
Max of QCS	$\max_{t\in q}(1+logN_t)log(1+\frac{\|C\|}{\|D_t\|})$
Qry Variability (QVar)	$\sum_{t\in q}\sqrt{\frac{1}{\|D_t\|}\sum_{d\in D_t}(w_{d,t}-\bar{w}_t)^2}$, $\bar{w}_t = \frac{1}{\|D_t\|}\sum_{d\in D_t} w_{d,t}$, $w_{d,t} = 1+logN_{d,t} * log(1+\frac{\|C\|}{\|D_t\|})$
Max of QVar	$\max_{t\in q}\sqrt{\frac{1}{\|D_t\|}\sum_{d\in D_t}(w_{d,t}-\bar{w}_t)^2}$
Similar Qry Click Score	$\frac{1}{\|Q^{sim}\|}\sum_{q'\in Q^{sim}} RankScore(q', click\_doc)$
Similar Qry Click Rank	$\frac{1}{\|Q^{sim}\|}\sum_{q'\in Q^{sim}} Rank(q', click\_doc)$

Here q is the given query, t represents term, D_F is the pseudo relevant documents obtained from first retrieval of unexpanded query, C is the entire corpus, $|C|$ is the total document number in corpus, d represents document, N is the total term number in corpus, N_t is the number of term t in corpus, D_t is the set of documents containing t, $N_{d,t}$ is the number of t in doc d. The last two features are based on search log for industry dataset, which calculate the average clicked doc score/ranks of similar queries (Q^{sim}) in search log (see Sec. 4 for more details).

Table 4: Term Features $\mathcal{F}_t$

Description	Formulation
UQEScore	log score of UQE model
Term Doc Num	$log\|D_e\|$
Term Prob in Corpus	$log(N_e/N)$
Term Distribution	$log\frac{\sum_{d\in D_F} N_{d,e}}{\sum_{d\in D_F}\sum_{t\in d} N_{d,t}}$
Co-occurrence with single query term	$log\frac{1}{\|q\|}\sum_{t\in q}\frac{\sum_{d\in D_F} Win(t,e\|d)}{\sum_{d\in D_F}\sum_{t\in d} N_{d,t}}$
Co-occurrence with pair of query terms	$log\frac{1}{\|t_{i,j}\in q\|}\sum_{\|t_{i,j}\in q\|}\frac{\sum_{d\in D_F} Win(t_{i,j},e\|d)}{\sum_{d\in D_F}\sum_{t\in d} N_{d,t}}$
Term Proximity	$log\frac{\sum_{t\in q} Win(t,e)dist(t,e\|D_F)}{\sum_{t\in q} Win(t,e)}$
Document Number of t,e together	$log(\sum_{d\in D_F} I(t,e\|d)+0.5)$
Probability in similar queries in search log	$log\frac{1}{\|Q^{sim}\|}\sum_{q\in Q} P(e\|q)$
Probability in docs that similar queries clicked	$log\frac{1}{\|D_{click}\|}\sum_{d\in D_{click}} P(e\|d)$

Here e is the target term. D_F means the working set of documents. $Win(t,e|d)$ is the co-occurrence times that term t and e appear within distance 10 in d. $Win(t_{i,j},e|d)$ is the co-occurrence times that e appear within distance 10 with both t_i, t_j. $dist(t,e|D_F)$ is the minimal terms between t and e in doc set D_F. The last two rows are search log based term features for industry dataset, which calculate the probability of e in similar queries and their clicked documents. These two features are essentially one-step random walk features in a more general context [13]. In [5], the doc working set D_F has two choices, one is the pseudo relevant docs and the other is the entire corpus. The latter, however, is prohibitively expensive in large dataset. Therefore, we relax D_F as follows: assume the number of pseudo relevant docs is K_1, and the number of final evaluation is K_2 (actually fixed as 1000 in this paper); D_F is set as the top $\{0.5K_1, K_1, 2K_1, 2.5K_2, 5K_2\}$ docs from first retrieval.

3.2.2 CCFS

Applying Alg. 2 in practice requires a deeper insight into the SQE model itself. Nonetheless, as we show below, Alg. 2 can generate elegant solutions that are easy to implement.

Objective. For training queries q, let $x_i^q \in \mathbb{R}^{|\mathcal{F}_t|}$ be the feature vector for the ith candidate term of q. Here the term features $\mathcal{F}_t$ are adopted from [5, 13], and are listed in Table 4. Following the notations of u, U, d in Alg. 2, the objective of cost constrained RankSVM is as follows:

$$\begin{aligned} &w, d = \arg\min \frac{1}{2}w^T w + \frac{G}{|\mathcal{P}|}\sum_{q,i,j\in\mathcal{P}} \gamma_{q,i,j}\xi_{q,i,j} \\ &s.t. \forall(q,i,j)\in\mathcal{P}: w^T(x_i^q \circ d) > w^T(x_j^q \circ d) + 1 - \xi_{q,i,j}, \quad \xi_{q,i,j} \geq 0, \\ &\quad d\in\{0,1\}^{|\mathcal{F}_t|}, \quad u^T d \leq U \end{aligned} \tag{3}$$

Here $\mathcal{P}$ is the set of pairwise candidate terms (q,i,j) where $\Delta r_i^q > \Delta r_j^q$; $\circ$ is element-wise multiplication; and we add $\gamma_{q,i,j} = |\Delta r_i^q - \Delta r_j^q|$ as loss weight that emphasizes large relevance difference, which works well in practice. Notice for RankSVM, there's no need to add offset.

Eq. 3 is how Eq. 2 looks like when RankSVM objective [19] is introduced. The first three lines (if without d) of Eq. 3 constitute vanilla RankSVM (with slight modification of $\gamma_{q,i,j}$), while the feature selector d and the last line of constraints formulate the cost constrained version of RankSVM. The outcome d indicates what features are selected under cost upper bound U, and the w is the resulted RankSVM model based on the selected features.

Optimization. We solve Eq. 3 by converting it into the dual form via Lagrange multiplier [4], which gives:

$$\begin{aligned} &\min_{\alpha_k, d} \sum_{k=1}^K \alpha_k - \frac{1}{2}\big(\sum_{k=1}^K \alpha_k(\delta x_k \circ d)\big)^T\big(\sum_{k=1}^K \alpha_k(\delta x_k \circ d)\big) \\ &\text{s.t.} \quad 0 \leq \alpha_k \leq \frac{G\gamma_k}{K}, \quad d\in\{0,1\}^{|\mathcal{F}_t|}, \quad u^T d \leq U \end{aligned} \tag{4}$$

where $\forall(q,i,j)$ in Eq. 3 is re-indexed by $k = 1:K$ with each k representing one (q,i,j) triplet; $y_k = 1$ if $\Delta r_i^q > \Delta r_j^q$, otherwise $y_k = -1$; $\delta x_k = x_i^q - x_j^q$; $\gamma_k = \gamma_{q,i,j}$. $\alpha = [\alpha_1, ..., \alpha_K]$ is the dual parameter to be learned, and we can get w as

$$w = \sum_{k=1}^K \alpha_k(\delta x_k \circ d) \tag{5}$$

Based on Eq. 4, now the CCFS problem can be easily optimized by iteratively solving step 3 and step 4 in Alg. 2.

(Step 3) Fix d and optimize w. When d is fixed, we can absorb d into X as $X \leftarrow X \circ d$. Under this circumstance, Eq. 3 and Eq. 4 become the standard RankSVM training problem without cost constraint, for which we can utilize existing algorithms for optimization. Considering the potential large scale of pairwise term comparison, here we adopt cutting plane method [19] for efficient optimization.

(Step 4) Fix w and optimize d. With fixed w, now we aim to find the optimal d. Notice that from step 3, we have $w = \sum_{k=1}^K \alpha_k \delta x_k$ (recall X is updated to absorb previous d in step 3). In this way, Eq. 4 can be reformulated as follows:

$$\begin{aligned} d^* &= \arg\max\big(\sum_{k=1}^K \alpha_k(\delta x_k \circ d)\big)^T\big(\sum_{k=1}^K \alpha_k(\delta x_k \circ d)\big) \\ &= \arg\max\Big(\big(\sum_{k=1}^K \alpha_k \delta x_k\big) \circ \big(\sum_{k=1}^K \alpha_k \delta x_k\big)\Big)^T d \\ &= \arg\max_d (w \circ w)^T d \\ &s.t. \quad d\in\{0,1\}^{|\mathcal{F}_t|}, u^T d \leq U \end{aligned} \tag{6}$$

This is a standard linear integer programming problem, for which we utilize Gurobi[1] for efficient optimization.

During iteration, the upper bound U is meant to be a tunable parameter for the users. In practice, we can set U as a portion of the overall time cost, i.e. $U = \lambda * \sum_{f\in\mathcal{F}_t} u_f$,

[1] http://www.gurobi.com/

where λ take values such as $\{1, \frac{1}{2}, \frac{1}{4}, ...\}$. ΔU controls the number of iterations. In our implementation, we set $\Delta U = \frac{\sum_{f \in \mathcal{F}_t} u_f - U}{\#Iter}$, where $\#Iter$ is the desired iteration number (e.g. $\#Iter = 5$ or 10).

4. EXPERIMENTS

So far we have elaborated all the details of the proposed TFS framework. Below we will present extensive experiments to verify its validity.

4.1 Datasets

We adopt four corpora for experiments, including three academic and one industry corpus.

Robust04. This dataset includes about 0.5 million high quality documents. 250 queries (301-450 and 601-700) provided by TREC'04 robust track [32] are utilized for the experiments. MAP is the primary evaluation metric. Notice here by primary evaluation metric, we mean the metric that is used to rank TREC competition teams.

Cw09BNS. Clueweb09 category B is used, which includes 50 million web pages. We utilize University of Waterloo's spam scores [9][2] to remove those with spam scores lower than 70, which leaves 29 million web pages. 150 queries (51 to 200 from TREC'10/11/12 web track) are examined. ERR@20 is the primary evaluation metric. We denote this dataset as Cw09BNS, as NS stands for no spam.

Cw12BNSLD. Clueweb12 category B is used, which also includes 50 million web pages. Since category B contains very few relevant documents that are labeled by TREC, we add all the labeled relevant documents into this dataset. Again University of Waterloo's spam scores [9][3] are applied to remove those spam web pages (with the same threshold 70), which leaves about 15 million documents. 50 queries from TREC'13 web track are utilized, with ERR@20 being the primary evaluation metric. We denote this dataset as Cw12BNSLD, as LD means labeled documents are added.

Industry. This is a large scale web page corpus collected from a major search engine company (i.e. Bing). We incorporate an industry corpus to diversify our experiment settings. For example, the availability of industry search log provides a new resource for query expansion, as well as the search log related features in Table 3 and Table 4. Specifically, this corpus includes about 50 million web pages and 2000 queries. NDCG@20 is the primary evaluation metric as in previous research on a similar industry corpus [13].

4.2 Settings

Now we present all the detailed experiment settings.

Corpus Preprocessing. We utilize Indri[4], one of the most popular academic search systems, to index all our corpora in the form of inverted index [38]. Krovetz stemmer is applied for stemming, and standard InQuery stopwords are removed. Except stopwords removal, we do not conduct any further pruning that might reduce document lengths.

Code & Hardware. In accordance with Indri index, all our algorithms and experiments are implemented in C++ using Lemur/Indri API. The code is compiled by GCC4.7.3 with -O3 option. The code runs in a single thread on a single lab Linux server, which is equipped with a AMD 64bit 2.0GHz quad-core CPU, 12G memory and a 3TB disk that contains all the indexed corpora.

[2]https://plg.uwaterloo.ca/~gvcormac/clueweb09spam/

[3]http://www.mansci.uwaterloo.ca/ msmucker/cw12spam/

[4]http://www.lemurproject.org/indri.php

Table 5: Examples of Search Log Records

Query	Clicked URL	Score	Rank
bloomberg news	http://www.bloomberg.com/news/	201	1
firefox com	http://www.mozilla.org/ en-US/firefox/fx/	81	2
gibson althea	http://en.wikipedia.org/ wiki/Althea_Gibson	145	3

Smaller ranks and higher scores represent a better match between queries and clicked URLs. Notice these search log queries should not be misinterpreted as the 2000 queries for QE test. They actually serve as $\mathcal{S}$ to find the relevant web pages for those 2000 queries.

QE scenarios. As mentioned in Sec. 2, we can get different QE scenarios, depending on the resource $\mathcal{S}$ upon which expansion terms are extracted. For Robust04, we apply the traditional pseudo relevance feedback (PRF) for query expansion, where resource $\mathcal{S}$ is identical to the target corpus $\mathcal{C}$. Top 20 documents retrieved for q are considered relevant, from which expansion terms are extracted.

This PRF scenario, however, did not work well on the other corpora, which include web pages of relatively low quality [9]. We find that, on Cw09BNS and Cw12BNSLD, even after filtering spams, the traditional PRF still does not work well, which is also reported in [1]. Therefore, we try the strategy of $\mathcal{S} \neq \mathcal{C}$.

For Cw09BNS and Cw12BNSLD, we follow the suggestion of Bendersky et al. in [1] to use Wikipedia as $\mathcal{S}$. Top 5 ranked Wikipedia documents of original query q are considered relevant, upon which QE is applied. On Industry corpus, we follow the idea of Gao et al. in [13] to use search log as $\mathcal{S}$. The search log is also acquired from the same search engine company, which includes one million distinct click records. Each record contains four elements: user issued query, clicked URL, the score and the rank of the clicked URL returned by the search engine. A snapshot of the search log records is shown in Table 5. For each of the 2000 queries to be experimented, we first find the top 20 similar queries from the log; then the corresponding clicked web pages are considered relevant, from which expansion terms are extracted. This is actually a one-step random walk in search log [13].

Models & Parameters. Following [5], we utilize KL divergence (KLD) as the basic retrieval model for all the experiments below. The Dirichlet coefficient is set as 1500. The UQE and SQE algorithm are the same as explained in Sec. 3.2, i.e. relevance model for UQE and RankSVM for SQE. For both of them, we empirically set the number of expansion terms as $m = 20$. Other values of m will be examined in Sec. 4.8. For SQE, the number of candidate terms are empirically set as $M = 100$. Selected expansion terms are added to the original query by probability interpolation, as introduced in Eq. 1. The interpolation coefficient λ is tuned over a finite set of values $\{0, 0.1, ..., 0.9, 1\}$ on the training/validation set.

Evaluation Metrics. Both retrieval effectiveness and efficiency will be evaluated. For effectiveness, MAP and Prec@20 are used for Robust04, and ERR@20 and NDCG@20 are utilized for the other three web page corpora. For efficiency, we report the retrieval time costs per query, which is averaged on each query set.

Training/Validation/Testing. For all the query sets, based on the order of their query IDs, we select the first 40% queries for training all the models (e.g. SQE and TFS), the

Table 6: Retrieval Performance on Robust04

	MAP($\star$)	$\ominus$	Prec @20	$\ominus$	Time (sec)	%
OrigRet	0.268	-	0.345	-	0.13	-
UQE	0.319	0	0.369	0	0.61	0
UQE$^{\alpha}$	0.321	0.002	0.373	0.004	0.78	+27.9%
SQE	0.327	0	0.381	0	4.73	0
SQE$^{\alpha}$	0.329	0.002	0.383	0.002	4.65	-1.7%
SQE$^{\beta(\frac{1}{4})}$	0.325	-0.002	0.378	-0.003	1.66	-64.9%$\diamond$
SQE$^{\alpha\beta(\frac{1}{4})}$	0.327	0	0.380	-0.001	1.76	-62.8%$\diamond$

Table 7: Retrieval Performance on Cw09BNS

	ERR @20($\star$)	$\ominus$	NDCG @20	$\ominus$	Time (sec)	%
OrigRet	0.129	-	0.169	-	9.5	-
UQE	0.149	0	0.190	0	11.3	0
UQE$^{\alpha}$	0.159	0.01$\uparrow$	0.194	0.004	11.67	+3.3%
SQE	0.181	0	0.197	0	27.9	0
SQE$^{\alpha}$	0.187	0.006	0.191	-0.006	20.7	-25.8%$\diamond$
SQE$^{\beta(\frac{1}{4})}$	0.176	-0.005	0.186	-0.011$\downarrow$	15.6	-44.1%$\diamond$
SQE$^{\alpha\beta(\frac{1}{4})}$	0.186	0.005	0.191	-0.006	13.8	-49.5%$\diamond$

Table 8: Retrieval Performance on Cw12BNSLD

	ERR @20($\star$)	$\ominus$	NDCG @20	$\ominus$	Time (sec)	%
OrigRet	0.258	-	0.618	-	4.37	-
UQE	0.261	0	0.632	0	6.13	0
UQE$^{\alpha}$	0.262	0.001	0.626	-0.006	6.27	+2.3%
SQE	0.291	0	0.660	0	23.5	0
SQE$^{\alpha}$	0.292	0.001	0.664	0.004	18.1	-23%$\diamond$
SQE$^{\beta(\frac{1}{4})}$	0.287	-0.004	0.665	0.005	10.5	-55.3%$\diamond$
SQE$^{\alpha\beta(\frac{1}{4})}$	0.288	-0.003	0.665	0.005	9.1	-61.3%$\diamond$

Table 9: Retrieval Performance on Industry

	NDCG @20($\star$)	$\ominus$	ERR @20	$\ominus$	Time (sec)	%
OrigRet	0.372	-	0.253	-	11.5	-
UQE	0.387	0	0.260	0	12.1	0
UQE$^{\alpha}$	0.391	0.004	0.268	0.008	12.3	+1.7%
SQE	0.403	0	0.281	0	22.7	0
SQE$^{\alpha}$	0.408	0.005	0.285	0.004	19.1	-15.8%$\diamond$
SQE$^{\beta(\frac{1}{4})}$	0.4	-0.003	0.276	-0.005	15.1	-33.5%$\diamond$
SQE$^{\alpha\beta(\frac{1}{4})}$	0.406	0.003	0.279	-0.002	13.9	-38.8%$\diamond$

middle 10% queries for parameter validation, and the remaining 50% queries for testing evaluation. All experiments are repeated three times to report averaged time cost.

TFS Notation. As the two stages in TFS can be applied independently, we will utilize superscript α and β to indicate the case when AED and CCFS are applied alone, such as UQE$^{\alpha}$ and SQE$^{\beta}$. When the full set of TFS is applied for SQE, then we denote as SQE$^{\alpha\beta}$.

4.3 More on Time Cost

As mentioned, the time costs reported below are all obtained by running experiments using a single thread on a Linux server. The absolute value might appear larger than previous works (mainly on Cw09BNS and Cw12BNSLD), for which we feel necessary to give a full explanation.

Versus Previous Studies. The reason why previous studies such as [3, 35, 20] reported very low time costs of QE retrieval is mainly due to their selection and pre-processing upon the corpus. For example, (1) Lavrenko et al. [20] and Billerbeck et al. [3] utilized corpora that only contains $O(10^5 \sim 10^6)$ documents; in comparison our Clueweb09/12 and Industry corpora have $O(10^7)$ documents, which are at least 10 times larger. (2) Billerbeck [3] and Wu [35] reduced the document length into 20 $\sim$ 100 terms long, while the averaged document length for our corpora are 500 $\sim$ 800, which are again about 5 $\sim$ 40 times larger. The difference of corpus size and document length is the major reason why our reported time costs are larger than previous studies.

Versus Indri Official. With the above settings, our reported time costs are actually quite reasonable. As a proof, the Indri official website[5] reported an averaged time cost of 20 seconds per query (wall clock time) on Cw09B (50 million docs, with spam, one thread program on a 3.0GHz CPU, average query length is about 4), while ours is 9.5 seconds per query (29 million docs, no spam, 2.0GHz CPU, average query length is 2.4). After normalizing various factors[6], we can conclude that our time cost per query is very close to that reported by Indri official website. Although this is not an exact comparison, it indeed partially supports our claim.

[5]http://www.lemurproject.org/clueweb09/indri-howto.php

[6]We divide the averaged time cost per query by the number of documents (in millions) and the averaged query length, and multiply the CPU frequency (in GHz). The result can be seen as an atomic time cost for a single query term on a million documents using 1GHz CPU. In this way, the atomic time cost from official Indri website is 0.3 seconds, while ours is 0.27, which is very close.

Versus Engineering Strategy. The absolute value of time costs can be reduced by engineering strategies such as better hardware or distributed/parallel computing, which are widely adopted in commercial search engines like Bing and Google. However, such devices are usually very expensive to equip, and are not available to us. Moreover, accurately counting the time costs in distributed/parallel computing environment becomes difficult, because usually the computing resouces (e.g. CPU or memory) are automatically allocated and can vary as time passes. The advantage of us utilizing single thread program on single computer is that, the overall time costs directly reflects the amount of computation (thus the efficiency), and makes it easy to compare different retrieval methods.

4.4 Overall Performance

We first present the major results of retrieval performance on the four corpora, as shown in Table 6, 7, 8 and 9.

Comparison Methods. We conduct extensive comparison with the following retrieval configurations:

(1) Retrieval for original queries without QE (OrigRet);

(2$\sim$3) UQE and UQE+AED (UQE$^{\alpha}$);

(4$\sim$7) SQE, SQE+AED (SQE$^{\alpha}$), SQE+CCFS (SQE$^{\beta(\frac{1}{4})}$), and SQE+TFS (SQE$^{\alpha\beta(\frac{1}{4})}$). Here $\beta(\frac{1}{4})$ is an example parameter for upper bound U in CCFS, which means the upper bound U is a quarter of the overall time costs of all term features, i.e. $U = \frac{1}{4}\sum_{f\in\mathcal{F}_t} u_f$. Other choices of upper bounds will be examined in Sec. 4.5.

As mentioned earlier, by default reranking (top 1000 documents) is utilized in the second retrieval to report the time costs for the above retrieval methods.

Table Explanation. We adopt evaluation metrics regarding both effectiveness (e.g. ERR@20, NDCG@20, MAP) and efficiency (Time in seconds). Here ($\star$) indicates primary evaluation metric. We treat UQE and SQE as the baseline, so that we can show how AED, CCFS and TFS improve the efficiency respectively. We use column "$\ominus$" to represent the effectiveness difference between UQE/SQE and their speedup versions, and use % for the relative time cost reduction. For example, on Cw09BNS, SQE$^{\alpha}$ vs SQE has ERR@20 difference 0.187-0.181=0.006; their time cost change (%, in percentage) is (20.7-27.9)/27.9=-25.8%. $\uparrow$ and $\downarrow$ label the positive and negative effectiveness difference that are statistically significant, and $\diamond$ means the time cost reduction upon SQE is statistically significant (t-test, $p < 0.05$).

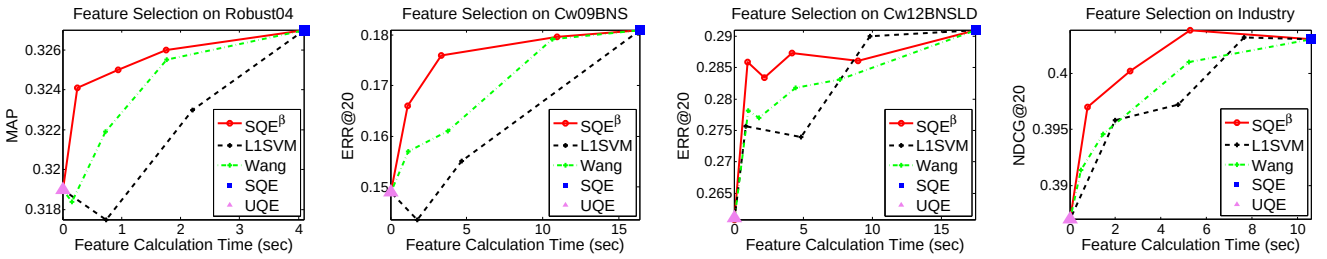

Figure 3: Comparison between different feature selection methods. The horizontal axis is the time cost for term feature extraction (excluding any retrieval time). Purple triangles at left end of curves are UQE method with no term features, and blue rectangles at right end are SQE algorithm with all available features. AED is not applied here.

Major Observations. From the tables we can draw the following two major observations.

(1) SQE is more effective but also less efficient than UQE and OrigRet. Compared with OrigRet and UQE, the retrieval effectiveness of SQE can be substantially higher. However, the time costs are also substantially larger. This verifies our motivation that the efficiency issue of SQE is an important research topic.

(2) Both AED and CCFS can substantially improve the efficiency of SQE, meanwhile only incurring insignificant effectiveness fluctuation. In the above tables, we progressively add each component to SQE, so that one can see how the efficiency is progressively improved. In general we can conclude that for SQE, CCFS achieves higher efficiency than AED, and their combination (i.e. our TFS framework) achieves the most efficiency improvement. For effectiveness, although both positive and negative changes are observed, most of them are statistically insignificant (t-test, p<0.05). I.e. most of the effectiveness changes are not labeled by ↑ or ↓. Therefore, we can conclude that our TFS framework can well maintain the effectiveness of SQE.

We also notice that for UQE, UQE$^\alpha$ has slightly increased time costs. There are two reasons for this phenomenon. First, for UQE there is no expensive term feature extraction, so that AED only skips the generation of UQE expansion terms and the reranking process in second retrieval. Since these two steps are already very fast, the reduced time cost is not substantial. Second, AED will result in some extra time costs for query feature extraction as well as the application of AED classifier. Therefore, the overall time costs of UQE$^\alpha$ will be slightly higher than UQE. But notice, the absolute value of such increase is very low (at most 0.37 seconds). Furthermore, as we will show in Sec. 4.9, the efficiency improvement of AED can be very substantial, if the full second retrieval is applied instead of reranking, which verifies the usefulness of AED.

Below, we will present more experiments to thoroughly investigate AED and CCFS. As CCFS will also be utilized in AED experiments, we will first analyze CCFS for sake of clear presentation.

4.5 Cost Constrained Feature Selection

In the above we have verified the validity of feature selection in speeding up SQE. Now we will investigate how our CCFS algorithm in Alg. 2 performs in this task.

Comparison Methods. The following two algorithms are compared with our CCFS algorithm.

L1-RankSVM. L1 regularization is a very popular feature selection method. When feature selection occurs, we replace the L2 regularizer in vanilla RankSVM (Eq. 3) with L1 regularizer $||w||_1$. By adjusting the coefficient G, $||w||_1$ will function to different extent, thus resulting in various combinations of features. Notice this method is unaware of the difference in the time costs of extracting each feature. L1General library[7] [28] is utilized for optimization.

Wang's method [33]. This is a greedy algorithm for cost aware feature selection, proposed by Wang et al. in [33] for learning to rank. In this algorithm, each feature is assigned a profit score, which is the ratio between feature weight and time cost. Features are sorted by profit scores; then top features are selected until the time cost upper bound is reached, which makes it a greedy selection. Different from L1-RankSVM, this is a cost aware feature selection method.

For both Alg. 2 and Wang's method, we use the same cost upper bounds as $U = \lambda \sum_{f \in \mathcal{F}_t} u_f$ as explained earlier, where we adjust λ to different values such as $\{1, \frac{1}{2}, \frac{1}{4}, ...\}$ to get all the nodes along the curves in Figure 3. For L1-RankSVM, each node represents a different G value, which is tuned on training set so that different time costs are obtained.

Overall Results. In Figure 3 we illustrate the curves of the three methods on all corpora. These curves represent the retrieval effectiveness when various feature extraction time is spent (excluding any retrieval time). The purple triangles at the left end of curves represent UQE algorithm, i.e. no term feature is extracted. The blue rectangles at the right end of curves represent SQE algorithm with all available term features. In the middle, various feature selection methods show different effectiveness-cost tradeoff. We can clearly observe that more features can produce higher retrieval accuracy, but this inevitably takes more time thus decreasing the efficiency.

CCFS performs best, particularly at low time cost. Comparing the three feature selection methods, we can see that our CCFS algorithm outperforms the others, especially when the feature cost is low. L1-RankSVM penalizes the number of selected features. That means, each feature is treated equally, ignoring the cost difference among different features. Since expensive features can be included, to reach a certain time cost, usually L1-RankSVM will over-penalizes the number of selected features, which deteriorates the retrieval effectiveness. Wang's method greedily selects features based on their profit scores, i.e. the ratio between feature weight and cost. Here the feature weights are the ones derived when all features are available. However, this selection process is suboptimal, because for a single feature, its weight will become different when fewer other features are available. Therefore, the profit score may not accurately reflect the importance of individual features, particularly when

[7] http://www.cs.ubc.ca/~schmidtm/Software/L1General.html

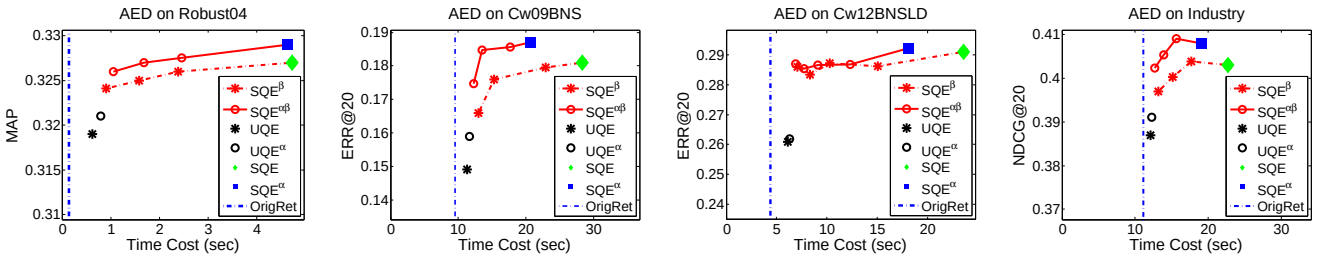

Figure 4: Experiments for AED. Red curves correspond to the SQE$^\beta$ curves in Figure 3. The UQE nodes (with zero term feature cost) are shown separately for better illustration. The horizontal axis is the overall retrieval time. The blue vertical line is the time cost of OrigRet, which is plotted as reference.

few features exist (i.e. time cost is low). In comparison, our CCFS algorithm iteratively updates learning objective, and decreases the cost upper bound smoothly. Therefore, CCFS performs better, particularly when time cost is low.

4.6 Adaptive Expansion Decision

Now we will examine how AED affects the effectiveness and efficiency of UQE and SQE retrieval. We show the performance in Figure 4, where UQE, SQE and SQE$^\beta$ algorithms are all examined before and after applying AED. For SQE$^\beta$, the CCFS curves from Figure 3 are utilized.

For both SQE and SQE$^\beta$, their performance is left-shifted after applying AED. Moreover, the SQE$^\beta$ curves shrink after AED. This means, the process of term feature extraction and second retrieval (reranking) are skipped for some of the queries (i.e. SQE-unsuitable), which makes the averaged time costs over all queries become smaller.

For UQE, the time costs of UQE$^\alpha$ is slightly higher. This has been explained in Sec. 4.4, which are due to the fast process of reranking and UQE expansion term generation, as well as the existence of AED overhead cost.

The extent of efficiency improvement of AED depends on the number of skipped queries. In Table 10 we give the detailed number of skipped queries on each corpus for SQE.

From the perspective of effectiveness, we can observe that on all corpora, for all of UQE, SQE and SQE$^\beta$, their effectiveness after applying AED is improved or at least maintained than before applying AED. This is particularly helpful in achieving a good balance between effectiveness and efficiency.

Table 10: Number of skipped queries in AED for SQE.

Dataset	#Test Query	#Skipped Query	Percentage
Robust04	125	8	6.4%
Cw09BNS	75	32	42.67%
Cw12BNSLD	25	8	32%
Industry	1000	361	36.1%

4.7 More on Step-wise Time Cost

In Figure 1 we have shown the step-wise time costs on Cw09BNS for UQE, SQE and their speedup improvements. There for CCFS we adopt the same upper bound as in Table 7 (i.e. $U = \frac{1}{4}\sum_{f\in\mathcal{F}_t} u_f$). This is a non Pseudo Relevance Feedback (PRF) scenario where $\mathcal{S}$ is Wikipedia and $\mathcal{C}$ is Cw09BNS. In this case, the time cost of second retrieval will be large because in second retrieval we need to firstly search query q on $\mathcal{C}$ then apply the reranking.

Now we further show the step-wise time costs for PRF scenario on Robust04 in Figure 5(a). In this case, $\mathcal{S} = \mathcal{C} =$ Robust04, so we only need to retrieve q once in first retrieval, and the second retrieval only needs to rerank the results of first retrieval. In this case, the cost of second retrieval will be much smaller than in non PRF scenario.

4.8 The Effect of Number of Expansion Terms

Now let's see how different number of expansion terms m affects the retrieval effectiveness and efficiency. In Figure 5(b), we show the effectiveness-cost curves when $m = \{10, 20, 30\}$ for SQE$^{\alpha\beta}$ on Industry corpus. We can see the effectiveness of $m = 20, 30$ is similar, while that of $m = 10$ is quite degraded. The time cost gap between OrigRet and SQE$^{\alpha\beta}$ curves includes the cost of first retrieval (i.e. searching $\mathcal{S}$), applying AED, extracting term features, etc. Also notice the overall time cost is not obviously affected as more expansion terms are utilized. This phenomenon is mainly due to the application of reranking. Otherwise if a full second retrieval is applied, the time cost of second retrieval will be (approximately) linear with the number of m, which can be very huge in practice (see Sec. 4.9).

4.9 Reranking vs Full Second Retrieval

Finally, for the SQE retrieval process, we compare the two strategies for second retrieval: reranking vs full second retrieval. Although we have argued the validity of reranking and have utilized it throughout the above experiments, we still feel it necessary to present a formal comparison with full second retrieval due to the following two reasons. First, as reviewed in Sec. 2.2, we find most of existing QE efficiency works [3, 35] only focused on indexing or document optimization, while ignored the value of reranking. It is only very recent that Diaz [11] pointed this out. Here we'd like to add more proof to support reranking. Second, in the above experiments for UQE and UQE$^\alpha$, we observed that pure AED may not result in substantial speedup, and point out that the application of reranking is the major reason for that. By further showing the time costs of full second retrieval, we can justify the value of AED.

In Figure 5(c) and (d), we show the performance of reranking top 1000 documents for expanded queries q^e with 20 expansion terms. We can see that for both the case of UQE and SQE, reranking does not incur obvious effectiveness loss, yet results in substantial efficiency improvement. Particularly for the UQE case, the speedup becomes obvious when full second retrieval is utilized on Cw09BNS. These observations verify our adoption of reranking instead of full second retrieval, as well as the usefulness of AED on efficiency.

5. CONCLUSION & FUTURE WORK

Supervised query expansion (SQE) has recently become the state-of-the-art in the QE literature, which usually outperforms the unsupervised counterparts. To obtain good retrieval effectiveness, SQE usually extracts many term fea-

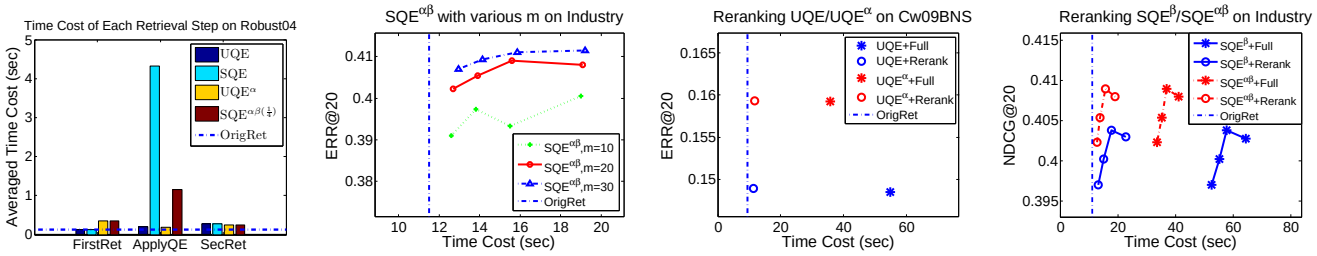

Figure 5: (a) Step-wise costs on Robust04, which is under PRF scenario. (b) Performance of SQE$^{\alpha\beta}$ on Industry with different number of expansion terms. (c) Reranking vs full second retrieval for UQE and UQE$^{\alpha}$ on Cw09BNS. (d) Reranking vs full second retrieval for SQE$^{\beta}$ and SQE$^{\alpha\beta}$ on Industry.

tures to predict the quality of expansion terms. However, this can seriously decrease its efficiency. This issue has not been studied before, nor can it be handled by previous data-level QE efficiency methods such as indexing or documents optimization. To address this problem, in this paper we propose a Two-stage Feature Selection (TFS) framework, which includes Adaptive Expansion Decision and Cost Constrained Feature Selection. Extensive experiments on four corpora show that the proposed TFS framework can significantly improve the efficiency of SQE algorithm, while maintaining its good effectiveness.

6. REFERENCES

[1] M. Bendersky, D. Fisher, and W. B. Croft. Umass at trec 2010 web track: Term dependence, spam filtering and quality bias. In *TREC*, 2010.

[2] M. Bendersky, D. Metzler, and W. B. Croft. Effective query formulation with multiple information sources. In *WSDM*, pages 443–452, 2012.

[3] B. Billerbeck and J. Zobel. Efficient query expansion with auxiliary data structures. In *Information System*, pages 573–584, 2006.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] G. Cao, J.-Y. Nie, J. Gao, and S. Robertson. Selecting good expansion terms for pseudo-relevance feedback. In *SIGIR*, pages 243–250, 2008.

[6] C. Carpineto and G. Romano. A survey of automatic query expansion in information retrieval. In *ACM Computing Surveys*, 2012.

[7] K. Collins-Thompson. Reducing the risk of query expansion via robust constrained optimization. In *CIKM*, 2009.

[8] K. Collins-Thompson and P. N. Bennett. Predicting query performance via classification. In *ECIR*, pages 140–152, 2010.

[9] G. V. Cormack, M. D. Smucker, and C. L. A. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *IR*, pages 441–465, 2011.

[10] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma. Probabilistic query expansion using query logs. In *WWW*, pages 325–332, 2002.

[11] F. Diaz. Condensed list relevance models. In *ICTIR*, 2015.

[12] J. Gao, S. Xie, X. He, and A. Ali. Learning lexicon models from search logs for query expansion. In *EMNLP*, pages 666–676, 2012.

[13] J. Gao, G. Xu, and J. Xu. Query expansion using path-constrained random walks. In *SIGIR*, pages 563–572, 2013.

[14] X. Geng, T.-Y. Liu, T. Qin, and H. Li. Feature selection for ranking. In *SIGIR*, pages 407–414, 2007.

[15] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *JMLR*, pages 1157–1182, 2003.

[16] C. Hauff, D. Hiemstra, and F. de Jong. A survey of pre-retrieval query performance predictors. In *CIKM*, pages 1419–1420, 2008.

[17] B. He and I. Ounis. Query performance prediction. In *Information Systems*, pages 585–594, 2006.

[18] B. He and I. Ounis. Combining fields for query expansion and adaptive query expansion. *IPM*, pages 1294–1307, 2007.

[19] T. Joachims. Training linear svms in linear time. In *KDD*, pages 217–226, 2006.

[20] V. Lavrenko and J. Allan. Real-time query expansion in relevance models. In *Tech Report, Univ Massachusetts Amherst*, 2006.

[21] V. Lavrenko and W. B. Croft. Relevance-based language models. In *SIGIR*, pages 120–127, 2001.

[22] C.-J. Lee, R.-C. Chen, S.-H. Kao, and P.-J. Cheng. A term dependency-based approach for query terms ranking. In *CIKM*, pages 1267–1276, 2009.

[23] Y. Lv and C. Zhai. Adaptive relevance feedback in information retrieval. In *CIKM*, pages 255–264, 2009.

[24] Y. Lv and C. Zhai. Positional relevance model for pseudo-relevance feedback. In *SIGIR*, pages 579–586, 2010.

[25] Y. Lv, C. Zhai, and W. Chen. A boosting approach to improving pseudo-relevance feedback. In *SIGIR*, pages 165–174, 2011.

[26] C. D. Manning, P. Raghavan, and H. Schütze. An introduction to information retrieval. In *Cambridge University Press*, 2009.

[27] S. C.-T. ownsend Y un Zhou W. Bruce Croft. A language modeling framework for selective query expansion. In *Univ Massachusetts Amherst Tech Report*, 2004.

[28] M. Schmidt. Graphical model structure learning with l1-regularization. *Univ British Columbia PhD Thesis*, 2010.

[29] H. Schütze and J. O. Pedersen. A coocurrence-based thesaurus and two applications to information retrieval. In *IPM*, 1997.

[30] M. Theobald, R. Schenkel, and G. Weikum. Efficient and self-tuning incremental query expansion for top-k query processing. In *SIGIR*, pages 242–249, 2005.

[31] P. Viola and M. J. Jones. Robust real-time face detection. *IJCV*, pages 137–154, 2004.

[32] E. M. Voorhees. Overview of the trec 2004 robust retrieval track. In *TREC*, pages 69–77, 2004.

[33] L. Wang, D. Metzler, and J. Lin. Ranking under temporal constraints. In *CIKM*, pages 79–88, 2010.

[34] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for svms. In *NIPS*, pages 668–674, 2000.

[35] H. Wu and H. Fang. An incremental approach to efficient pseudo-relevance feedback. In *SIGIR*, pages 553–562, 2013.

[36] C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *CIKM*, pages 403–410, 2001.

[37] Y. Zhao, F. Scholer, and Y. Tsegay. Effective pre-retrieval query performance prediction using similarity and variability evidence. In *ECIR*, pages 52–64, 2008.

[38] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*.