Time-Aware Rank Aggregation for Microblog Search

Shangsong Liang† s.liang@uva.nl
Zhaochun Ren† z.ren@uva.nl
Wouter Weerkamp‡ wouter@904labs.com
Edgar Meij§ emeij@yahoo-inc.com
Maarten de Rijke† derijke@uva.nl
†University of Amsterdam, Amsterdam, The Netherlands
‡904Labs, Amsterdam, The Netherlands
§Yahoo Labs, Barcelona, Spain

ABSTRACT

We tackle the problem of searching microblog posts and frame it as a rank aggregation problem where we merge result lists generated by separate rankers so as to produce a final ranking to be returned to the user. We propose a rank aggregation method, TimeRA, that is able to infer the rank scores of documents via latent factor modeling. It is time-aware and rewards posts that are published in or near a burst of posts that are ranked highly in many of the lists being aggregated. Our experimental results show that it significantly outperforms state-of-the-art rank aggregation and time-sensitive microblog search algorithms.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

Keywords

Rank aggregation; data fusion; ad hoc retrieval; microblog search

1. INTRODUCTION

The task of searching microblog posts has been studied widely. Many effective algorithms have been proposed, including, for example, those based on language modeling [9], query expansion [21], and time-sensitive features [3]. Different microblog search algorithms have different merits; hence, combining these alternative algorithms into a single model may boost retrieval performance. In this paper we follow this idea and tackle the challenge of microblog search by following a *rank aggregation* (also called *data fusion*) approach. Our input consists of a number of ranked lists generated by different microblog search algorithms and the output is an aggregated and, hopefully, better list.

Rank aggregation approaches can inherit the merits of the individual search algorithms whose outputs are being aggregated. Also, they may boost recall [31] and because they aggregate the results of multiple strategies, they mitigate risks associated with opting for a single strategy [31]. In rank aggregation, documents in the fused list are ranked in deceasing order of their fusion scores. The fusion score of a document is usually a function of the rank scores from the individual input lists, such as the weighted sum. Previous work on rank aggregation often assumes—either implicitly or explicitly—that the rank score of a document is set to zero for an input list if that document does not appear in the list. We challenge this assumption; our intuition is that documents that are similar to a document d that does occur in an input list, should get similar rank scores. Let us call a document that does not appear in list L, but does appear in other lists we aim to fuse, a *missing document* for L (see Figure 1). We propose a rank aggregation algorithm where we apply a latent factor model to predict the rank scores of missing documents. In our rank aggregation algorithm, we define a list-document rank score matrix, factorize this matrix, and utilize the factorized list-specific and document-specific matrices to assign scores to missing documents.

State-of-the-art rank aggregation methods often work on the assumption that only documents that are ranked highly in many of the result lists being aggregated are likely to be relevant [1, 4, 5, 11, 13, 22]. As a result, a relevant document might be ranked low in an aggregated list if it appears in only few of the input result lists and is ranked low within these. The characteristics of microblog environments suggest a different perspective. In such environments people tend to talk about a topic within specific short time intervals [10, 20, 23]. E.g., people talked about the "2014 Eastern Synchronized Skating Sectional Championship" mainly between January 30 and February 1 2014, which is when the championship took place. Posts generated before or after the event are less likely to talk about the championship and, hence, less likely to be relevant to a query about the championship.

Inspired by these characteristics of microblog search scenarios, we propose a rank aggregation algorithm, TimeRA, that is time-aware. TimeRA detects windows (intervals) of timestamps of posts ranked high in many of the lists to be fused. These windows give rise to bursts of posts, and subsequently TimeRA rewards posts that are published in the vicinity of a burst that contains highly scored posts. Specifically, if a post d_i and posts $d_1, \ldots, d_k$ were created within the same narrow time window, and the posts $d_1, \ldots, d_k$ are ranked highly in many of the lists being merged, then post d_i will be "rewarded," even if, in the extreme case, it appears in only one list where it is ranked low. Fig. 1 illustrates this: post d_2 is ranked low in list L_1 but will be rewarded as it was published in a time window in which a large number of posts are published that are ranked high

CIKM'14, November 3–7, 2014, Shanghai, China.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2598-1/14/11 ...$15.00
http://dx.doi.org/10.1145/2661829.2661905.

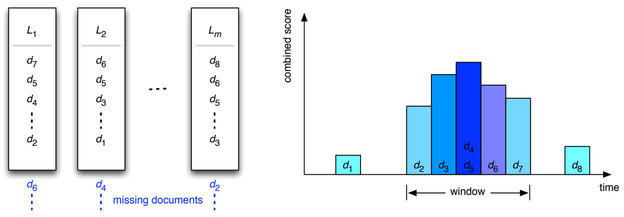


Figure 1: Rewarding posts that are published in the same narrow time frame as a large number of (supposedly) relevant posts. The vertical axis indicates the summed scores of posts with the same timestamps. There are m lists in response to a query. Post d_2 only occurs in list L_1 and is ranked low; d_8 also occurs in a single list, L_m, but is ranked very high. Surrounding d_2 there are many documents that are ranked high in many lists; therefore, d_2 should be "rewarded," while d_8 should not. Documents marked in blue are "missing" documents; e.g., d_6 is a missing document for L_1, as it is not observed in L_1 during the aggregation process.

in many lists. But d_8, while ranked high in L_m, receives no such bonus as it was published outside the time window.

Our results show that TimeRA is able to effectively aggregate result lists, subject to real-time search requirements, running almost as fast as standard fusion methods such as CombSUM [5], and outperform both time-sensitive microblog search methods and competing fusion algorithms.

Our contributions can be summarized as follows:

- We propose a rank aggregation-based algorithm for microblog search, TimeRA, that outperforms traditional and state-of-the-art unsupervised and supervised rank aggregation methods as well as the best single result list that it aggregates.
- We examine the relative contributions of the main ingredients of TimeRA (burst detection and score inference) and find that burst detection is effective in helping rank aggregation and that inferring scores for missing documents during aggregation boosts performance as well.
- We provide a detailed analysis of the performance of TimeRA and offer a number of examples where we observe the effect hypothesized in Fig. 1, i.e., of posts in bursts having their rank score boosted.
- We show that the effect of TimeRA on ranking microblog posts is different from the effect of existing time-sensitive microblog search models.
- And, finally, we study the efficiency of TimeRA and show that it meets real-time search requirements.

2. RELATED WORK

Three major types of research are closely related to our work: microblog search, rank aggregation and latent factor modeling.

2.1 Microblog search

Searching posts that are short and creatively spelled in microblogging platforms, like Twitter, is challenging. Previous work on microblog search has focused on employing traditional IR techniques, i.e., learning to rank [6], document expansion [19], pseudo-relevance feedback retrieval [21] to improve the performance. Some previous work explores time information to boost performance. For instance, Li and Croft [12] propose a time-sensitive language model to explore time and relevance of queries; Massoudi et al. [19] integrate temporal information into a dynamic expansion microblog retrieval model. And Dakka et al. [3] consider the publication time of documents in conjunction with the topic similarity to derive a time-sensitive ranking. More recently, Miyanishi et al. [21] integrate temporal and lexical evidence into their query expansion method for microblog search.

We propose to develop a rank aggregation-based approach to microblog search. To the best of our knowledge, this is the first attempt to tackle the problem of microblog search as a rank aggregation problem.

2.2 Rank aggregation

Rank aggregation approaches attempt to inherit the merits of different search algorithms, and to combine their result lists in order to produce a new and hopefully better ranking [14, 16, 26]. In recent years, rank aggregation has featured in many applications, including federated search [28], vertical search [2], and more recently, search result diversification [14, 15]. Many rank aggregation methods have been proposed, including CombSUM and CombMNZ [26], semantic fusion [16], cluster-based fusion [7] and λ-Merge [27]. CombSUM and CombMNZ are the oldest but fast and effective rank aggregation methods. The outranking approach only utilizes the rank of documents in the result lists during aggregation. Both semantic fusion and cluster-based fusion use the assumption that content-similar documents should be clustered to boost the performance of aggregation. Finally, λ-Merge tries to learn a model from labeled data for aggregation.

Previous rank aggregation algorithms ignore the temporal information of the documents to be merged. We propose a new rank aggregation where we fully utilize the temporal information and reward posts that are published in the vicinity of a burst to be ranked higher in the fused list. To the best of our knowledge, this is the first attempt to integrate temporal information into rank aggregation.

2.3 Latent factor modeling

Latent factor models are often used in collaborative filtering (CF) and recommender systems [8, 25]. Matrix factorization methods form a group of well-known latent factor models, that include, for instance, singular value decomposition [8], probabilistic matrix factorization [25] and social regularization [18].These methods first model users with latent interests and the products with latent features by matrix factorization, and then try to predict the rating of products for the given users with the observations of the existing users' rating data [8, 18, 25]. Motivated by latent factor models, rather than simply setting no rank scores for missing documents as in previous rank aggregation methods [5, 7, 27, 31], our proposed rank aggregation algorithm applies matrix factorization to model both result lists (called "users" in CF) and documents ("products" in CF) as a mixture of latent factors ("interests" in CF), such that the rank scores of missing documents ("the rating of unobserved products" in CF) in a result list for aggregation can be inferred. Reasons of why a latent topic model can infer scores can be found in [8, 18, 25].

To the best of our knowledge, this is the first attempt to predict rank scores for missing documents in rank aggregation.

3. PRELIMINARIES

We detail the task that we address and recall two standard rank aggregation algorithms, CombSUM and CombMNZ.

3.1 Problem formulation

The task we address is this: given an index of microblog posts, a query, and a set of ranked lists of posts with timestamps produced in response to the query, aggregate the lists into a final ranked list of posts to be returned in response to the query. Hence, the rank aggregation problem consists of finding a ranking function f_X such

Table 1: Main notation used in TimeRA.

Notation	Gloss
$\mathbf{R}$	$m \times \lvert\mathcal{C}_\mathcal{L}\rvert$ list-post rank score matrix
A	number of latent factors
$\mathbf{S}$	matrix used for inferring latent topics for $\mathcal{L}$
$\mathbf{V}$	matrix used for inferring latent topics for $\mathcal{C}_\mathcal{L}$
$\mathbf{S}_i$	a column vector in S used for aspects of L_i
$\mathbf{V}_j$	a column vector in V used for aspects of d_j
R_{ij}	rank score of d_j in list L_i
$\Re_{ij}$	inferred rank score of post $d_j \in \mathcal{C}_\mathcal{L} \setminus L_i$
$d_k L_i d_j$	d_k ranks higher than d_j in L_i
t_i	timestamp
d_{t_i}	post with timestamp t_i
T	number of different timestamps of posts in $\mathcal{C}_\mathcal{L}$
b	a burst of posts
$\mathcal{B}$	set of all bursts in $\mathcal{C}_\mathcal{L}$ (given query q)
I_{ij}	indicate whether $d_j \in L_i$
σ_b	timestamp standard deviation of a burst
$r(d_j\|d_k)$	"reward" score for d_j if d_j is within or near the burst b to which d_k belongs
$w(d_j\|L_i)$	rank punishment weighting function for $d_j \in L_i$
β	relative weight of burst information in TimeRA

that:

$$\mathcal{L} = \{L_1, L_2, \ldots, L_m\}, q, \mathcal{C} \xrightarrow{f_X} L_f,$$

where $\mathcal{L}$ is a set of lists to be fused, L_i is the i-th result list, $m = |\mathcal{L}|$ is the number of input result lists, $\mathcal{C}$ is the corpus of microblog posts, q is a query, L_f is the final fused list.

3.2 Standard rank aggregation

Before we move on to the details of our rank aggregation method, we set the stage regarding rank aggregation by briefly recalling two standard fusion methods: CombSUM and CombMNZ. Let R_{ij} denote the rank score of document d_j in list L_i based on the rank of d_j in the list L_i; in the literature on rank aggregation [5, 7, 11, 31], one often finds $R_{ij} = 0$ if $d_j \notin L_i$ (d_j still in the combined set of posts $\mathcal{C}_\mathcal{L} := \bigcup_{i=1}^{m} L_i$). In both CombSUM and CombMNZ, R_{ij} is often defined as:

$$R_{ij} = \begin{cases} \frac{(1+|L_i|)-\text{rank}(d_j|L_i)}{|L_i|} & d_j \in L_i \\ 0 & d_j \notin L_i, \end{cases} \tag{1}$$

where $|L_i|$ is the length of L_i and $\text{rank}(d_j|L_i) \in \{1, \ldots, |L_i|\}$ is the rank of d_j in L_i. The well-known CombSUM fusion method [5, 31], for instance, scores d_j by the sum of its rank scores in the lists:

$$f_{\text{CombSUM}}(d_j|q) := \textstyle\sum_{i=1}^{m} R_{ij},$$

while CombMNZ [5, 31] rewards d_j that ranks high in many lists:

$$f_{\text{CombMNZ}}(d_j|q) := |\{L_i : d_j \in L_i\}| \cdot f_{\text{CombSUM}}(d_j|q),$$

where $|\{L_i : d_j \in L_i\}|$ is the number of lists in which d_j appears.

4. TIME-AWARE RANK AGGREGATION

We detail our time-aware rank aggregation algorithm in this section; §4.1 describes the way we detect bursts, §4.2 details our aggregation method, TimeRA, and §4.3 contains a brief complexity analysis of TimeRA.

To be able to present rank aggregation methods in a uniform way, we first introduce and summarize our notation in Table 1. Although introduced some time ago, CombSUM and CombMNZ are still effective and very efficient rank aggregation methods. More recent fusion methods have been proposed but most of them may be inappropriate for searching posts in microblogging scenarios that are dynamic and need rapid updates. For instance, cluster-based rank aggregation [7] and λ-Merge [27] have been shown to be more effective than CombSUM and CombMNZ in several cases. However, a drawback of cluster-based fusion is that it has to access the content of documents to compute inter-document similarities so as to generate a set of clusters and determine probabilities based on the clusters. A drawback of λ-Merge is that it also has to access the content to extract features and it often overfits [27].

4.1 Bursts detection

Our proposed TimeRA method utilizes burst information. To detect bursts, we proceed as follows. Let t be a timestamp. Let $d_t \in \mathcal{C}_\mathcal{L}$ denote a post with timestamp t. Here, $\mathcal{C}_\mathcal{L}$ is the set of posts appearing in the result lists. We regard posts published during the same hour as having the same timestamp. Although it is possible to define "the same timestamp" in many different ways, we found that this is a suitable level of granularity for the effectiveness of rank aggregation.Before we detect bursts in posts $\mathcal{C}_\mathcal{L}$, we need to define H_t, the *burst-time score at time t*. Let R_{kt} (obtained by Eq. 1) be the rank score of d_t given L_k. Then:

$$H_t = \frac{\sum_{d_t \in \mathcal{C}_t} \sum_{k=1}^{m} R_{kt}}{\sum_{t'=1}^{T} \sum_{d_{t'} \in \mathcal{C}_{t'}} \sum_{k=1}^{m} R_{kt'}} - \frac{1}{T}, \tag{2}$$

where T is the total number of different timestamps belonging to posts in $\mathcal{C}_\mathcal{L}$, $\mathcal{C}_t$ is a set of posts having the same timestamp t, $\sum_{d_t \in \mathcal{C}_t} \sum_{k=1}^{m} R_{kt}$ is the sum of the rank scores of posts having the same timestamp t, and $\sum_{t'=1}^{T} \sum_{d_{t'} \in \mathcal{C}_{t'}} \sum_{k=1}^{m} R_{kt'}$ is the sum of the rank scores of all posts in $\mathcal{C}_\mathcal{L}$. According to Eq. 2, the burst-time score $H_t > 0$ if it is above the average score (i.e., $1/T$), otherwise $H_t \leq 0$.

Using Eq. 2, we compute a burst-time score H_t for each time point $t \in \{t_1, t_2, \ldots, t_T\}$. In this manner we generate a *burst-time score sequence* $\vec{H} = \{H_{t_1}, H_{t_2}, \ldots, H_{t_T}\}$. Following [24], a segment $\vec{H}[t_i : t_j] = \{H_{t_i}, H_{t_{i+1}}, \ldots, H_{t_j}\}$, where $1 \leq i \leq j \leq T$, is a *maximal segment* in $\vec{H}$ if:

i. All proper subsequences of $\vec{H}[t_i : t_j]$ have a lower score.[1]

ii. No proper super-segments of $\vec{H}[t_i : t_j]$ in $\vec{H}$ satisfy item i.

As an example, consider the input sequence $\vec{H} = \{2, -2, 4, 3, -3, -4, -1, -3, 5, -1, 3, -2\}$. The maximal segments in this sequence are $\{2\}$, $\{4, 3\}$ and $\{5, -1, 3\}$. The segment $\{2, -2, 4, 3\}$ is not maximal, since it has a nonempty zero-scoring prefix $\{2, -2\}$ appending to the left of $\{4, 3\}$; $\{5\}$ is not a maximal segment, since $\{5, -1, 3\}$ has a total higher score of 7.

Each maximal segment $\vec{H}[t_i : t_j]$ gives rise to a *burst* of posts $b[t_i : t_j]$ with start timestamp t_i and end timestamp t_j: it contains any post $d \in \mathcal{C}_\mathcal{L}$ whose timestamp is between t_i and t_j. We write $\mathcal{B}$ to denote the *set of all bursts*, i.e., $\mathcal{B} = \bigcup b[t_i : t_j]$. We let b be short for $b[t_i : t_j]$ in the following.

4.2 The aggregation method

Next we detail our time-aware rank aggregation method, TimeRA. TimeRA utilizes matrix factorization techniques. The input of the matrix factorization in TimeRA is an $m \times n$ matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$ which we call a *list-document* matrix; here, again, m is the number of result lists and n is the number of posts to be fused, i.e., $n = |\mathcal{C}_\mathcal{L}|$. $\mathbf{R}$ is initialized by Eq. 1, viz., its elements R_{ij} are defined by Eq. 1. The output of the matrix factorization consists of two new matrices $\mathbf{S} \in \mathbb{R}^{A \times m}$ and $\mathbf{V} \in \mathbb{R}^{A \times n}$, obtained by

[1]The burst-time score of a sequence is the sum of the burst-time scores of the elements in the sequence.

```
Algorithm 1: TimeRA: Time-Aware rank aggregation.
Input : A query $q$, a number of ranked lists of posts to be fused, $\mathcal{L}=\{L_1, L_2, \ldots, L_m\}$, the combined set of posts
        $\mathcal{C}_{\mathcal{L}} := \bigcup_{i=1}^{m} L_i$, learning rate $\delta$.
Output: A final rank aggregation list of posts $L_f$.
1  $L_f = \varnothing$, initialize $\mathbf{R}$ by Eq. 1, Initialize $\mathbf{S}$ and $\mathbf{V}$ with random values, let $m = |\mathcal{L}|$ and $t = 0$;
2  for $j = 1, 2, \cdots, n$ do
3      $f_{\mathrm{TimeRA}}(d_j|q) = 0$;
4  Generate a set of bursts $\mathcal{B}$;
5  repeat
6      for $i = 1, 2, \cdots, m$ do
7          $\mathbf{S}_i^{(t+1)} = \mathbf{S}_i^{(t)} - \delta \frac{\partial C_2}{\partial \mathbf{S}_i^{(t)}}$ based on Eq. 15;
8      for $j = 1, 2, \cdots, n$ do
9          $\mathbf{V}_j^{(t+1)} = \mathbf{V}_j^{(t)} - \delta \frac{\partial C_2}{\partial \mathbf{V}_j^{(t)}}$ based on Eq. 15;
10     $t = t + 1$;
11 until $C_2$ given by Eq. 12 converges;
12 $\mathbf{S} = \mathbf{S}^{(t)}, \mathbf{V} = \mathbf{V}^{(t)}$;
13 for $j = 1, 2, \cdots, n$ do
14     Obtain $f_{\mathrm{TimeRA}}(d_j|q)$ score for $d_j$ via $\mathbf{S}$ and $\mathbf{V}$ by Eq. 14;
15 Construct final fused list $L_f$ via decreased scores of $f_{\mathrm{TimeRA}}(d|q)$.
```

factorizing $\mathbf{R}$, which we call the *factor-list* matrix and the *factor-post* matrix, respectively. Here, A is the number of latent factors. After obtaining $\mathbf{S}$ and $\mathbf{V}$, we can infer the normalized scores of missing posts, based on which we arrive at the aggregation scores of all the posts.

We present TimeRA in Algorithm 1. We first provide a high-level walkthrough. To begin, it sets matrices $\mathbf{S} \in \mathbb{R}^{A\times m}$, $\mathbf{V} \in \mathbb{R}^{A\times n}$ with random values and $f_{\mathrm{TimeRA}}(d|q)$ with the value 0 (lines 1–3 in Algorithm 1). Let $\mathbf{S}_i$ be the i-th column of $\mathbf{S}$, meant to get the latent factors of the list L_i after matrix factorization, and let $\mathbf{V}_j$ be the j-th column of $\mathbf{V}$, meant to get latent factors of post d_j after matrix factorization. After detecting bursts (line 4), TimeRA utilizes burst information and tries to get the final $\mathbf{S}$ and $\mathbf{V}$ by performing a gradient descent process on a cost function C_2 (lines 5–12). To this end, aggregation scores $f_{\mathrm{TimeRA}}(d|q)$ of $d \in \mathcal{C}_{\mathcal{L}}$ can be obtained by $\mathbf{S}$ and $\mathbf{V}$ (lines 13–14). The defined cost function plays an important role: (1) it tries to keep the original rank scores of posts that rank high in many of the lists to be fused, (2) it rewards posts that rank low in a few lists but in the vicinity of bursts, and (3) it gets the latent factors of both the lists and the posts such that missing posts' rank scores can be inferred.

Our matrix factorization approach in TimeRA seeks to approximate the list-document matrix $\mathbf{R}$ by a multiplication of A latent factors,

$$\mathbf{R} \approx \mathbf{S}^\top \mathbf{V} \tag{3}$$

To obtain $\mathbf{S}$ and $\mathbf{V}$, traditionally, the Singular Value Decomposition (SVD) method [8] is utilized to approximate the list-document rank matrix $\mathbf{R}$ by minimizing

$$\mathbf{S}, \mathbf{V} = \arg\min_{\mathbf{S},\mathbf{V}} \frac{1}{2} ||\mathbf{R} - \mathbf{S}^\top \mathbf{V}||_F^2, \tag{4}$$

where $||\cdot||_F^2$ denotes the Frobenius norm. Due to the definition that $R_{ij} = 0$ if d_j is a missing document, i.e., $d_j \in \mathcal{C}_{\mathcal{L}} \setminus L_i$, we only need to factorize the observed scores in $\mathbf{R}$ so that (4) changes to

$$\mathbf{S}, \mathbf{V} = \arg\min_{\mathbf{S},\mathbf{V}} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij} (R_{ij} - \mathbf{S}_i^\top \mathbf{V}_j)^2, \tag{5}$$

where I_{ij} is an indicator function that is equal to 1 if d_j appears in L_i and equal to 0 otherwise, $\mathbf{S}_i \in \mathbb{R}^{A\times 1}$ is a column vector in $\mathbf{S}$ that serves to get latent factors of L_i, and, similarly, $\mathbf{V}_j \in \mathbb{R}^{A\times 1}$ is a column vector in $\mathbf{V}$ that serves to get latent factors of d_j. As $R_{ij} \in [0, 1]$ according to Eq. 1, (5) can be rewritten as

$$\mathbf{S}, \mathbf{V} = \arg\min_{\mathbf{S},\mathbf{V}} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij} (R_{ij} - g(\mathbf{S}_i^\top \mathbf{V}_j))^2, \tag{6}$$

where $g(x)$ is the logistic function defined by $g(x) = 1/(1 + \exp(-x))$, which makes it possible to bound the range of $\mathbf{S}_i^\top \mathbf{V}_j$ within the same range $[0, 1]$ by R_{ij}.

In order to avoid overfitting, two regularization terms are added to Eq. 6:

$$\begin{aligned}\mathbf{S}, \mathbf{V} = \arg\min_{\mathbf{S},\mathbf{V}} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij} (R_{ij} - g(\mathbf{S}_i^\top \mathbf{V}_j))^2 \\ + \frac{\lambda_1}{2} ||\mathbf{S}||_F^2 + \frac{\lambda_2}{2} ||\mathbf{V}||_F^2,\end{aligned} \tag{7}$$

where $\lambda_1, \lambda_2 > 0$. For a probabilistic interpretation with Gaussian observation noise for Eq. 7 we refer to Salakhutdinov and Mnih [25]. To reduce the model's complexity we set $\lambda_1 = \lambda_2$ in all experiments below [8, 18, 25].

The cost function defined by Eq. 7 punishes posts equally when they shift from the original rank scores R_{ij}. However, a post ranked higher should be punished more than posts ranked lower if they shift from the original rank scores: higher ranked posts are more likely to be relevant so that keeping their original rank scores will be of more value. Thus, we modify Eq. 7 to obtain the following cost function C_1:

$$\begin{aligned}\mathbf{S}, \mathbf{V} = \arg\min_{\mathbf{S},\mathbf{V}} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij} \cdot w(d_j|L_i) (R_{ij} - g(\mathbf{S}_i^\top \mathbf{V}_j))^2 \\ + \frac{\lambda_1}{2} ||\mathbf{S}||_F^2 + \frac{\lambda_2}{2} ||\mathbf{V}||_F^2,\end{aligned} \tag{8}$$

where $w(d_j|L_i)$ is a rank punishment weighting function defined for d_j given L_i. Specifically, we define $w(d_j|L_i)$ as

$$w(d_j|L_i) = \begin{cases} \frac{1}{2^{\mathrm{rank}(d_j|L_i)-1}} & d_j \in L_i \\ 0 & d_j \notin L_i. \end{cases} \tag{9}$$

Our next step is to bring in burst information. After detecting a set of bursts, we introduce the following item into the cost function C_1 to boost the relevance of posts by burst information, such that

$$\begin{aligned}\mathbf{S}, \mathbf{V} = \arg\min_{\mathbf{S},\mathbf{V}} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{b\in\mathcal{B}} \frac{1}{|\{d_k \in b : d_k L_i d_j\}|} \\ \cdot \sum_{\substack{d_k \in b \\ d_k L_i d_j}} I_{ij} I_{ik} r(d_j|d_k) w(d_k|L_i) (R_{ik} - g(\mathbf{S}_i^\top \mathbf{V}_j))^2,\end{aligned} \tag{10}$$

where $\mathcal{B}$ is a set of detected bursts, b is a burst in $\mathcal{B}$, $d_k L_i d_j$ means d_k is ranked higher than d_j in list L_i, $|\{d_k \in b : d_k L_i d_j\}|$ is the total number of posts in the burst b that are ranked higher than the post d_j in L_i, and $r(d_j|d_k)$ is the "reward" score for d_j from d_k if d_j is within or near the burst b to which d_k belongs. In Eq. 10, $d_k L_i d_j$ indicates that only the posts ranked higher than the post d_j to be rewarded are capable of boosting the relevance of d_j.

If d_j, the post to be rewarded, is generated at (almost) the same time as the highly relevant post d_k, it will be rewarded more than posts that are further away in time from d_k, which is measured by $r(d_j|d_k)$ in Eq. 10. Accordingly, we define $r(d_j|d_k)$ based on a normal distribution as

$$r(d_j|d_k) = \exp\left\{-\frac{(t_{d_j} - t_{d_k})^2}{2\sigma_b^2}\right\}, \tag{11}$$

where t_{d_j} and t_{d_k} are the timestamps of post d_j and d_k, respectively, and σ_b is the standard deviation of timestamps in burst b:

$$\sigma_b^2 = \frac{\sum_{i=1}^{n_b}\{i - \frac{n_b+1}{2}\}^2}{n_b} = \frac{n_b^2 - 1}{12},$$

where n_b is the number of different timestamps of posts in b. According to Eq. 11, the more likely d_j is generated at the same time with d_k, the larger the score $r(d_j|d_k)$ is, resulting in a bigger reward for d_j from d_k.

We are almost ready now to define the cost function C_2 that we use in TimeRA. We integrate the original rank score of the posts in the result lists (Eq. 8) with rewards for posts generated in the vicinity of bursts (Eq. 10). To this end, we substitute Eq. 10 into Eq. 8. Our cost function C_2 for TimeRA is defined as

$$\begin{aligned}
&\mathbf{S}, \mathbf{V} = \\
&\arg\min_{\mathbf{S},\mathbf{V}} \frac{1-\beta}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij} w(d_j|L_i)(R_{ij} - g(\mathbf{S}_i^\top \mathbf{V}_j))^2 \\
&+\frac{\beta}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{b\in\mathcal{B}} \frac{1}{|\{d_k \in b : d_k L_i d_j\}|} \sum_{\substack{d_k \in b\\ d_k L_i d_j}} I_{ij} I_{ik} \qquad (12)\\
&\qquad r(d_j|d_k) w(d_k|L_i)(R_{ik} - g(\mathbf{S}_i^\top \mathbf{V}_j))^2 \\
&\qquad + \frac{\lambda_1}{2}||\mathbf{S}||_F^2 + \frac{\lambda_2}{2}||\mathbf{V}||_F^2,
\end{aligned}$$

where $\beta \in [0, 1]$ is a free parameter that governs the linear mixture.

In words, in response to a query q, TimeRA uses a two-component mixture model to score $d \in \mathcal{C}_\mathcal{L}$. The first component (the first term of Eq. 12) tries to maintain the original rank scores R_{ij}. The second component (the second term of Eq. 12), which uses bursts for posts, "rewards" d if it is strongly associated with the posts in the bursts. Clearly, if $\beta = 0$ in Eq. 12, TimeRA will only try to maintain the original rank scores, i.e., Eq. 8.

A local minimum of the objective function given by Eq. 12 can be found by performing gradient descent in both $\mathbf{S}_i$ and $\mathbf{V}_j$. The same can apply to Eqs. 4, 5, 6, 7, and 8. The algorithm for obtaining $\mathbf{S}$ and $\mathbf{V}$ is straightforward: we first randomly initialize $\mathbf{S}$ and $\mathbf{V}$, then iteratively update these two matrices based on their gradients until the value of the cost (objective) function converges. The derivation of these equations are included in Appendix A.

After optimizing Eq. 12, posts $d_j \in L_i$ they will end up with a score $g(\mathbf{S}_i^\top \mathbf{V}_j) = R_{ij}$ + "some reward." Unlike previous work that assigns 0 to missing documents $d_j \in \mathcal{C}_\mathcal{L} \setminus L_i$ [5, 7, 27, 31], we infer a rank score $\mathfrak{R}_{L_i d_j}$ for missing posts d_j as:

$$\mathfrak{R}_{ij} = \begin{cases} g(\mathbf{S}_i^\top \mathbf{V}_j) & \text{if } g(\mathbf{S}_i^\top \mathbf{V}_j) \le \min(R_{id}) \\ \min(R_{id}) & \text{if } g(\mathbf{S}_i^\top \mathbf{V}_j) > \min(R_{id}) \end{cases}, \tag{13}$$

where $\min(R_{id})$ is the minimum rank score of the lowest ranked post that appears in L_i as computed by Eq. 1. As shown in Eq. 13, if the inferred rank score $g(\mathbf{S}_i^\top \mathbf{V}_j)$ is smaller than the minimum rank score, we maintain the inferred score for that post. However, if the inferred rank score is greater than the minimum rank score, we give up the inferred score and let $\mathfrak{R}_{ij} = \min(R_{id})$, as $d_j \notin L_i$ means that it should at least be ranked lower than any post $d \in L_i$.

The final rank aggregation score for $d_j \in \mathcal{C}_\mathcal{L}$ is obtained by

$$f_{\text{TimeRA}}(d_j|q) = \sum_{\substack{i=1\\ d_j \in L_i}}^{m} g(\mathbf{S}_i^\top \mathbf{V}_j) + \sum_{\substack{i=1\\ d_j \in \mathcal{C}_\mathcal{L} \setminus L_i}}^{m} \mathfrak{R}_{ij}. \tag{14}$$

The final rank aggregation score for a post consists of two parts, i.e., scores for lists L_i it appears in (the first term in Eq. 14) and scores for lists it does not appear in (the second term in Eq. 14), as inferred by Eq. 13. Finally, the final fused list L_f produced in response to q is obtained by ranking posts in decreasing order of $f_{\text{TimeRA}}(d_j|q)$. We call the model defined by Eq. 14, *TimeRA*, and the variant that ignores inferred rank information $\mathfrak{R}_{ij}$ in Eq. 14 is called *TimeRA-Infer* ("TimeRA minus Infer").

4.3 Analysis of time-aware rank aggregation

The main computation of TimeRA is detecting bursts and its gradients against matrices $\mathbf{S}$ and $\mathbf{V}$. Our burst detection has computational complexity $O(n)$ because the detection algorithm runs in linear time [24], and the gradients of C_2 have computational complexity of $O(e \times m \times n \times A)$. In practice, the number of result lists to be fused, m, the number of posts to be aggregated, n, the number of latent factors, A, and the number of epochs, e, are all quite small. Therefore, the rank aggregation procedure can run in real-time, as we will see in the experiments presented in §6.6.

In the limiting case, TimeRA reverts back to CombSUM. That is, if we set $\beta = 0$ (ignoring burst information), set the weight function $w(d|L_i) = 1$ for all documents d and lists L_i, and, moreover, set $\lambda_1 = \lambda_2 = 0$, and do not try to infer the rank score of posts, then $g(\mathbf{S}_i^\top \mathbf{V}_j) = R_{ij}$ after performing gradient descent in Eq. 12. Our experimental results below show that the performance of TimeRA is almost the same as CombSUM when $\beta = 0$. Note that $\lambda_1, \lambda_2 \ne 0$ in the experiments.

5. EXPERIMENTAL SETUP

In this section, we list our research questions, describe the data set, specify our baselines, detail the metrics as well as our training and optimization setup, and describe the experiments.

5.1 Research questions

The research questions guiding the remainder of the paper are:

RQ1 Does TimeRA outperform traditional and state-of-the-art unsupervised or supervised rank aggregation methods and the best single result list? (§6.1)

RQ2 What are the relative contributions of the main ingredients of TimeRA (burst detection and score inference)? (§6.2)

RQ3 What is the effect of using burst information in TimeRA (i.e., what is the impact of the parameter β in Eq. 12)? (§6.3)

RQ4 What is the effect of the number of lists to be aggregated in TimeRA? (§6.4)

RQ5 Can we observe the hypothesized effect sketched in Fig. 1, i.e., posts in bursts being rewarded? (§6.5)

RQ6 Does TimeRA meet real-time search requirements? (§6.6)

RQ7 Does TimeRA beat existing time-sensitive microblog search models? (§6.7)

5.2 Data set

In order to answer our research questions we use the Tweets 2011 corpus [17, 29] provided by the TREC 2011 and 2012 Microblog tracks. The collection is comprised of around 16 million tweets collected over a period of 2 weeks (January 23, 2011 until February 8, 2011). Different types of tweets are present in this data set, including replies and retweets, each with their own timestamp.

The task at the TREC 2011 and 2012 Microblog tracks was: given a query with a timestamp, return relevant and interesting tweets in reverse chronological order. In response to a query, a run was required to retrieve 30 posts. In total, for the 2011 track 49 test topics were created and 2965 tweets were deemed relevant; some topics have just two relevant tweets, while others have more than 100 relevant tweets.

A total of 59 groups participated in the TREC 2011 Microblog track, with each team submitting at most four runs, which resulted in 184 runs[2] [17]. The official evaluation metric was precision at 30 (p@30) [17, 29]. The p@30 scores of these 184 runs varied dramatically, with the best run achieving a p@30 score of 0.4551 and the worst runs achieving scores below 0.10. Details about the implementation of each run can be found in [17, 29]. We also run experiments on the TREC 2012 Microblog track runs, and find that they yield the same overall results and trends. Due to space limitations, we only report results for the TREC 2011 edition runs below.

5.3 Baselines and evaluation

We compare TimeRA to 5 aggregation baselines: 2 traditional unsupervised methods, i.e., CombSUM, CombMNZ, 2 start-of-the-art cluster-based fusion methods, ClustFuseCombSUM and ClustFuseCombMNZ [7], and a start-of-the-art supervised method, λ-Merge [27], in which we integrate temporal features. As TimeRA utilizes temporal information, we also compare TimeRA to 4 state-of-the-art time-sensitive microblog search algorithms: time-based language model (TBLM) [12], textual quality factor model with temporal query expansion (LM-T(qe)) [19], direct time-sensitive BM25 retrieval model (DIRECT-BM25 (mean)) [3] and temporal tweet selection feedback method (TSF+QDRM) [21]. To build the index of the dataset that some of our baselines require, we apply Porter stemming, tokenization, and stopword removal (using INQUERY lists) to posts using the Lemur toolkit.[3] Features used in λ-Merge, including time-sensitive features, and the settings of λ-Merge are briefly described in Appendix B.

For performance evaluation we use the official TREC Microblog 2011 metric, p@30. We also report on p@5, p@10, p@15 and MAP scores. MAP scores are of special interest to us: we hypothesize that TimeRA has both a precision and recall-enhancing effect and we use MAP to measure this. Statistical significance of observed differences between two runs' performances is tested using a two-tailed paired t-test and is denoted using $^\blacktriangle$ (or $^\blacktriangledown$) for significant differences for $\alpha = .01$, or $^\vartriangle$ (and $^\triangledown$) for $\alpha = .05$.

A single free parameter in Eq. 12, β ($\in \{0, 0.1, \ldots, 1\}$), is incorporated in TimeRA, which is set using leave-one-out cross validation performed over the entire set of 49 queries. The performance of MAP is optimized in the learning phase. In other words, the performance for a query is attained using a value of β that maximizes MAP performance over all other queries. The same optimization strategy is used for one of our baselines, cluster-based fusion. Other baselines do not incorporate free parameters. Following [18], we set the parameters $\lambda_1 = \lambda_2 = 0.001$.

5.4 Experiments

We report on 7 main experiments in this paper aimed at understanding (1) the performance of TimeRA in general via sampling lists and fusing them; (2) the contribution of the main ingredients in TimeRA; (3) the performance of TimeRA with increasing numbers of runs to be fused; (4) query level performance; (5) TimeRA's efficiency; (6) the effect of inferring rank scores of posts by TimeRA; (7) the performance of TimeRA against temporal retrieval models.

[2]Available from http://trec.nist.gov.

[3]http://www.lemurproject.org

Table 2: Sampled runs used in some of the experiments.

Class	Sampled runs	Performance
Class 1	clarity1, waterlooa3, FASILKOM02, isiFDL, DFReeKLIM30, PRISrun1	$0.40 \leq$ p@30
Class 2	KAUSTRerank, ciirRun1, gut, dutirMixFb, normal, UDMicroIDF	$0.30 \leq$ p@30 < 0.40
Class 3	WESTfilext, LThresh, qRefLThresh, run3a, Nestor, uogTrLqea	$0.20 \leq$ p@30 < 0.30
Class 4	FASILKOM02, waterlooa3, gut, UDMicroIDF, run3a, qRefLThresh	$0.20 \leq$ p@30

To understand the overall performance of TimeRA, we sample $\sim$10% from the ranked lists produced by participants in the TREC 2011 Microblog track based on the lists' p@30 distribution: 18 out of the runs submitted to the TREC 2011 Microblog track, 6 with p@30 scores between 0.20 and 0.30 (Class 3), 6 between 0.30 and 0.40 (Class 2), and 6 over 0.40 (Class 1). We also randomly choose two runs from each class to construct Class 4; see Table 2. The runs in Class 1 are the 6 best runs in the TREC 2011 Microblog track. In every class, we use run1, run2, run3, run4, run5 and run6 to refer to the runs in descending order of p@30.

Next, to understand the contributions of the two main ingredients of TimeRA, viz., burst detection and inferring scores, we make comparisons among TimeRA, TimeRA-Infer and CombSUM. We also gradually increase the parameter β in Eq. 12 from 0.0 to 1.0 to see if burst information is helpful to boost fusion performance.

To understand the effect of the number of lists being merged, we randomly choose $k = 2, 4, 6, \ldots, 36$ lists from the 184 lists and aggregate them. We repeat the experiments 20 times and report the average results and standard deviation. In order to understand the query-level performance of TimeRA, we provide a detailed comparison of its performance against the baseline methods. To determine whether TimeRA can respond to a given query in (near) real time, we again randomly fuse $k = 2, 6, 12, 18, 30$ lists for all 49 test queries and report the average time required. Finally, we compare TimeRA against state-of-the-art time-sensitive retrieval models that utilize time/burst information.

6. RESULTS AND ANALYSIS

§6.1 and §6.2 show the results of fusing the sample lists, the contributions of burst detection and score inference in TimeRA, respectively; §6.3 analyzes the effect of using burst information; §6.4 shows the effect of the number of lists on the overall performance; §6.5 provides a topic-level analysis; §6.6 examines the run times. Finally, §6.7 compares TimeRA against time-sensitive models.

6.1 Fusing the sample lists

The performance of TimeRA and the 5 baselines is presented in Table 3, with numbers based on the $\sim$10% sample mentioned in §5.4. The performance of all the fusion methods is better than that of the best performing result list that is used in the merging process (run1) for all classes and on almost all metrics. Many of these improvements are statistically significant. More importantly, when fusing the top 6 result lists (Class 1), all of the p@30 scores generated by any rank aggregation method are higher than that of the best run in TREC 2011 Microblog track (0.4551), especially for TimeRA, which achieves 0.5531. These findings attest to the merits of using rank aggregation methods for microblog search.

It is also worth noting in Table 3 that in almost all cases, the cluster-based method does not beat the standard fusion method that it integrates, and the performance differences between the two are usually not significant. The reason behind this may be that it is challenging to do clustering in a microblog environment, with limited amounts of text and very creative language usage.

The performance of TimeRA is better than that of the baseline methods, and all of the differences are substantial and statistically significant. The performance of λ-Merge is almost the same as that of CombSUM, CombMNZ and the cluster-based methods when fusing the lists in Class 2, but in the other classes the performance tends to be a bit below that of the other methods, on all metrics. This may be due to overfitting.

Interestingly, the higher the quality of the result lists that are being aggregated, the bigger the improvements that can be observed in Table 3. For instance, the p@30 scores after aggregation are highest in Class 1 followed by those in Class 2 and Class3, and the quality of Class 1 is best followed by Class 2 and Class 3, respectively. The p@30 aggregation scores in Class 4 are almost the same as those in Class 2, as some of the lists' scores in Class 4 are better than those in Class 2.

6.2 Contributions of the main ingredients

Next, we compare the relative contributions of the main ingredients of TimeRA against the second best baseline, viz., CombSUM: burst detection and score inference. The effect of burst detection in TimeRA can be seen through comparisons between TimeRA-Infer and CombSUM; the effect of score inference can be seen through comparisons between TimeRA and TimeRA-Infer in Fig. 2.

Interestingly, in Fig. 2 there are large gaps in performance between TimeRA-Infer and CombSUM in terms of all of metrics; all improvements are statistically significant. This illustrates that burst detection makes an important contribution to the performance of rank aggregation. When we compare TimeRA and TimeRA-Infer in Fig. 2, we see that the performance of TimeRA-Infer in terms of p@5 is almost the same as that of TimeRA, while in terms of p@10 and p@30, TimeRA has some small advantages over TimeRA-Infer—some of these improvements are statistically significant. This observation confirms that inferring scores for posts during aggregation can boost performance as well. It also shows, however, that enhancing the performance of p@k becomes easier for larger values of k. This is because the cost of boosting the performance (i.e., changing the original rank score to be higher) is smaller when the posts are ranked lower. TimeRA is unable to beat TimeRA-Infer in terms of p@5 (.6939 for both), but TimeRA does boost the p@30 performance (.5531$^{\vartriangle}$ for TimeRA vs .5405 for TimeRA-Infer).

As burst information is such an important contributor to the performance of TimeRA, we analyze it further in §6.3.

6.3 The use of burst information

Next we examine the effect of using different amounts of burst information or cluster information in our time-aware aggregation or cluster-based methods, respectively. What is the impact of the free parameter β in Eq. 12 and in the cluster-based methods? Fig. 3 depicts the MAP performance curves for all rank aggregation methods when fusing the lists in Class 1, Class 2, Class 3 and Class 4, respectively. For $\beta = 0$, TimeRA almost amounts to CombSUM, while the cluster-based methods are the same as the standard fusion methods they incorporate, e.g., ClustFuseCombSUM has no difference with CombSUM in this case; more weight is put on burst information and cluster information with higher values of β in TimeRA and the cluster-based methods, respectively. For $0 < \beta < 1$, both the CombSUM scores of posts and burst information are utilized for aggregating lists in TimeRA.

In each of our four classes of runs, when aggregating lists, the MAP scores of TimeRA where burst information is used ($\beta > 0$) are always higher than that of any other fusion method. In Class 1 and Class 4 the gain increases gradually as the weight of burst information increases. These findings attest to the merits of using burst information to boost the performance in fusing ranked lists for microblog search. Putting more weight on cluster information in the cluster-based methods hurts performance in many cases.

6.4 Effect of the number of lists being merged

We explore the effect of varying the number of lists to be merged on the performance of TimeRA. Fig. 4 shows the fusion results of randomly sampling $k \in \{2, 4, 6, \dots, 36\}$ lists from the 184 lists. For each k, we repeat the experiment 20 times and report on the average scores. We use CombSUM as a representative example for comparisons with TimeRA; the results of other baseline methods are worse or qualitatively similar to those of CombSUM.

From Fig. 4 we can see that TimeRA performs better than CombSUM over all performance evaluation metrics no matter how many lists are fused. For both precision metrics (p@5 and p@30) we find that as long as the number of lists $\leq$ 10, the performance of both TimeRA and CombSUM gradually increases as the number of lists to be merged increases. The increases level off when the number of lists exceeds 12. For MAP we find that performance keeps increasing until we fuse 26 lists; then, the performance increase levels off.

Interestingly, in Fig. 4 the improvement of TimeRA over CombSUM on p@5 becomes smaller when more lists are merged. For example, the increase in p@5 of TimeRA over CombSUM is .1063 (.5861 for TimeRA vs .4798 for CombSUM) when two lists are fused. The performance increase, however, drops to only .0281 (.6712 for TimeRA vs .6431 for CombSUM) for 36 lists. Looking at the other metrics, which take a larger part of the merged result into account (p@30 and especially MAP), the gap remains.

6.5 Query-level analysis

We take a closer look at per test query improvements of TimeRA over other runs. For brevity, we only consider CombSUM as a representative and we only consider runs in Class 1. The results of TimeRA against CombSUM for other classes of runs and for other baseline methods are qualitatively similar. Fig. 5 shows per query performance differences in terms of AP, p@5, p@10 and p@30, respectively, between TimeRA and CombSUM. TimeRA displays both a precision and recall enhancing effect (with increases in precision oriented metrics as well as in MAP). As the metric at hand considers a larger chunk of the result list, there are more instances where TimeRA outperforms CombSUM. This is due mainly to topics that are discussed only in very specific time intervals. Examples include queries MB010 (Egyptian protesters attack museum), MB011 (Kubica crash) and MB015 (William and Kate fax save-the-date) etc. For such queries we found evidence of the intuition depicted in Fig. 1: posts that are ranked low in a small number lists but that TimeRA pushes up because they are central to a burst. E.g., in response to query MB010, post #30354903104749568 is ranked near the bottom in just two lists (at ranks 26 and 27 in the runs clarity1 and DFReekLIM30, respectively). Many posts for the query were generated around the same time interval (Jan. 26–29) and are ranked high in many lists; post #30354903104749568 was also published around this time and ranked 6th in the merged list because of this.

Queries for which TimeRA cannot beat CombSUM tend to be quite general and unrelated to any specific time windows. Examples include queries MB023 (Amtrak train service), MB027 (reduce energy consumption) and MB029 (global warming and weather) etc. For a very small number of queries, TimeRA's performance is slightly worse than that of CombSUM. One reason that we observed for this phenomenon is that not all posts that are ranked low in a small number of lists but central to a burst need to be rewarded. An example here is query MB024 (Super Bowl, seats).

Table 3: Retrieval performance on the ∼10% sample lists. Boldface marks the best result per metric; a statistically significant difference between TimeRA and the best baseline method is marked in the upper right hand corner of the TimeRA score. A significant difference with run1 for each method is marked in the upper left hand corner using the same symbols. None of the differences between the cluster-based method and the standard method it incorporates are statistically significant.

	Class 1					Class 2				
	MAP	p@5	p@10	p@15	p@30	MAP	p@5	p@10	p@15	p@30
run1	.2210	.5918	.5673	.5347	.4551	.1457	.4612	.4143	.3714	.3571
run2	.2690	.5959	.5796	.5442	.4537	.1886	.4776	.4347	.3878	.3463
run3	.2318	.5755	.5367	.5034	.4401	.1525	.4041	.4143	.3878	.3408
run4	.2058	.5714	.5367	.4939	.4211	.1376	.3959	.3939	.3796	.3218
run5	.2575	.5673	.4980	.4721	.4211	.1688	.3878	.3633	.3605	.3136
run6	.2098	.5469	.5102	.4694	.4095	.1820	.4122	.3796	.3619	.3027
CombSUM	▲.3404	▲.6245	.5816	.5524	▲.4966	▲.2625	▲.5306	▲.4531	▲.4286	▲.3735
ClustFuseCombSUM	▲.3398	▲.6240	.5802△	.5503	▲.4899	▲.2612	▲.5287	△.4500	▲.4213	▲.3686
CombMNZ	▲.3385	▲.6245	.5755	.5524	▲.5020	▲.2581	▲.5347	△.4592	▲.4354	▲.3789
ClustFuseCombMNZ	▲.3355	▲.6231	.5748	.5502△	▲.4987	▲.2560	▲.5330	▲.4523	▲.4311	▲.3731
λ-Merge	▲.3245	▲.6213	.5734	.5456	△.4833	▲.2573	▲.5634	▲.4952	▲.4463	▲.3901
TimeRA	▲**.3834**▲	▲**.6939**▲	▲**.6510**▲	▲**.6259**▲	▲**.5531**▲	▲**.3037**▲	▲**.6327**▲	▲**.5551**▲	▲**.5211**▲	▲**.4320**▲

	Class 3					Class 4				
	MAP	p@5	p@10	p@15	p@30	MAP	p@5	p@10	p@15	p@30
run1	.1661	.4041	.3408	.2898	.2122	.2058	.5714	.5367	.4939	.4211
run2	.0997	.3429	.3000	.2653	.2095	.2098	.5469	.5102	.4694	.4095
run3	.1636	.3959	.3122	.2571	.2041	.1376	.3959	.3939	.3796	.3218
run4	.0753	.3265	.2735	.2585	.2034	.1820	.4122	.3796	.3619	.3027
run5	.0571	.2980	.2551	.2408	.2020	.1636	.3959	.3122	.2571	.2041
run6	.0994	.3510	.2735	.2408	.2016	.0753	.3265	.2735	.2585	.2034
CombSUM	▲.2150	▲.4857	▲.4327	▲.3837	▲.2952	▲.2795	▲.6122	▲.5327	▲.4721	▽.3918
ClustFuseCombSUM	▲.2142	▲.4804	▲.4267	▲.3806	▲.2882	△.2741	▲.6088	△.5297	.4674	▼.3865
CombMNZ	▲.2187	▲.4898	▲.4327	▲.3932	▲.2973	▲.2794	▲.6000	▲.5449	▲.4830	.4048
ClustFuseCombMNZ	▲.2151	▲.4872	▲.4265	▲.3886	▲.2894	▲.2744	▲.5975	▲.5407	△.4782	.3958
λ-Merge	▲.2125	▲.5057	▲.4373	▲.3827	▲.2933	▲.2753	▲.6046	▲.5387	▲.4918	.4047
TimeRA	▲**.2491**▲	▲**.5347**▲	▲**.4694**▲	▲**.4122**▲	▲**.3293**▲	▲**.3210**▲	▲**.6735**▲	▲**.5918**▲	▲**.5429**▲	▲**.4347**▲

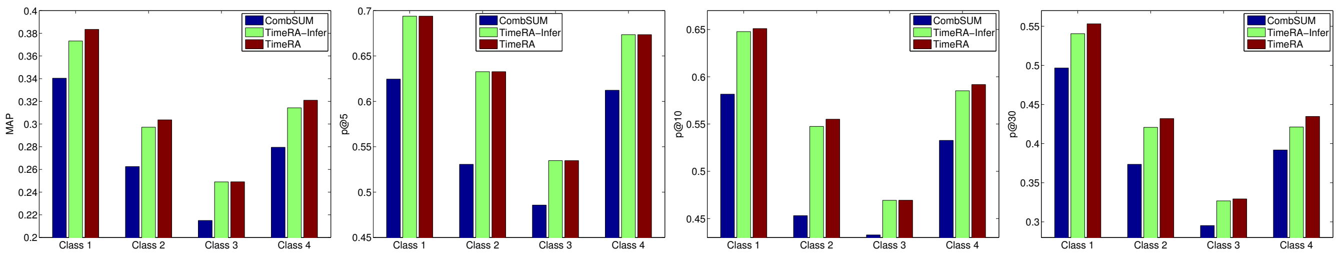


Figure 2: Retrieval performance of CombSUM, TimeRA-Infer (without using score inference for posts) and TimeRA in terms of MAP, p@5, p@10, and p@30 when fusing the runs in the 4 classes. Note that figures should be viewed in color.

6.6 Run time comparisons

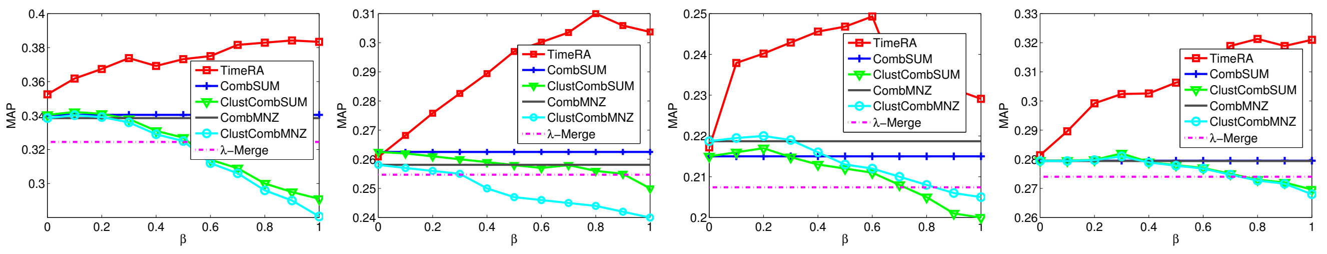
We now explore how fast TimeRA can merge result lists in response to a query. TimeRA is developed in C++ and the experiments are run on a 10.6.8 MacBook Pro computer with 4GB memory and a 2.3 GHz Intel core i5 processor. In Table 4, we randomly choose $k \in \{2, 6, 12, 18, 30\}$ lists from the 184 lists. For each k, we repeat the experiment 20 times and report the average run time per query (in seconds) that the fusion methods require. ClustSUM and ClustMNZ are short for ClustFuseCombSUM and ClustFuseCombMNZ, respectively in Table 4.

TimeRA does not run as fast as CombSUM or CombMNZ, but it manages to merge lists near real-time. TimeRA merges the lists within 0.1s when given 30 result lists and within 0.01s when fusing two lists. As the number of lists to be fused increases, the time spent on fusing is linear for CombSUM, CombMNZ, TimeRA, and λ-Merge; for ClustSUM and ClustMNZ, the time increases dramatically with larger numbers of lists. When fusing 30 lists, ClustMNZ needs to spend 235.91s, although it only spends 1.03s on fusing two lists. The times clocked for CombSUM and CombMNZ are similar, and likewise for those of ClustSUM and ClustMNZ.

6.7 Effect of fusing time-sensitive result lists

TimeRA uses temporal information in an essential way. How does it compare against retrieval models that explore temporal information? To answer this question, we conduct 4 additional experiments for generating 4 time-sensitive result lists, and explore the performance of the baseline fusion methods and TimeRA with each of which respectively fuses these 4 lists. These lists are generated by TBLM [12], LM-T(qe) [19], DIRECT-BM25 (mean) [3] and TSF+QDRM [21].

Table 5 shows the fusion result. Obviously, except ClustFuseCombMNZ and λ-Merge, fusion baselines and TimeRA outperform the best time-sensitive component run (TSF+QDRM) for all metrics and most of the improvements are statistically significant. This illustrates that exploring time information in data fusion has a different effect than utilizing time information in an individual ranking function, an effect that can lead to performance increases.



(a) Class 1 (b) Class 2 (c) Class 3 (d) Class 4

Figure 3: Effect of varying the value of β on the MAP performance of six aggregation methods, for lists in (a) Class 1, (b) Class 2, (c) Class 3 and (d) Class 4. More weight is put on burst information and on cluster information with higher values of β in TimeRA and the cluster-based methods, respectively. Note: the figures are not to the same scale.

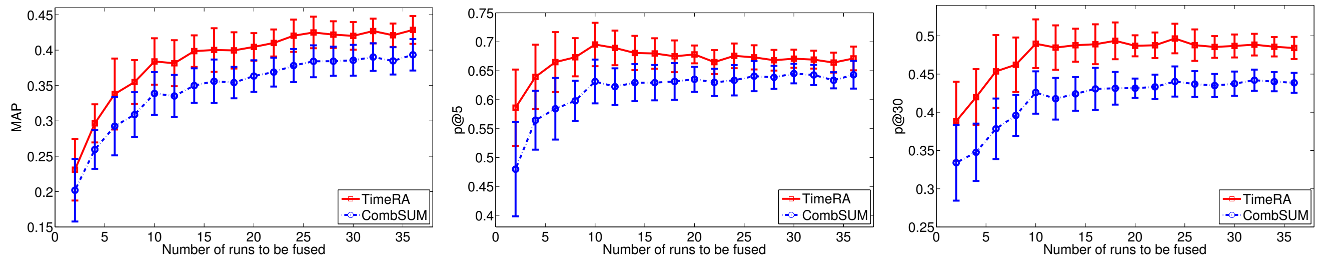


Figure 4: Effect on performance (in terms of MAP, p@5, and p@30) of the number of lists being merged. We plot averages and standard deviations. Note: the figures are not to the same scale.

Table 4: Time spent on fusing lists by different aggregation methods. Recorded in seconds with standard deviations (std).

	Number of lists				
	2	6	12	18	30
CombSUM	3.06e–4	5.76e–4	1.03e–3	1.98e–3	3.37e–3
std	1.13e–5	2.57e–5	6.93e–5	6.49e–5	7.50e–5
CombMNZ	3.06e–4	5.76e–4	1.03e–3	1.99e–3	3.38e–3
std	1.13e–5	2.57e–5	6.93e–5	6.52e–5	7.01e–5
TimeRA	4.77e–3	1.69e–2	2.05e–2	4.56e–2	9.60e–2
std	7.60e–5	7.41e–5	2.20e–4	3.53e–4	3.60e–4
λ-Merge	1.15	3.82	7.78	12.03	20.74
std	1.22e–1	5.82e–1	8.03e–1	1.09	1.18
ClustSUM	1.03	4.12	37.56	88.21	235.91
std	9.87e–2	4.24e–1	1.26	3.54	13.08
ClustMNZ	1.03	4.12	37.56	88.21	235.91
std	9.87e–2	4.24e–1	1.26	3.54	13.08

One reason behind this is that posts within intervals in which many relevant posts appear can only be confirmed to be relevant by gathering data from multiple lists, time-sensitive or not.

7. CONCLUSION

The special nature of microblog posts, e.g., their limited length and their creative language usage, raises challenges for searching them. However, this special nature also provides unique algorithmic opportunities. In this paper, we focus on utilizing time information to boost the performance of searching microblog posts. Specifically, we proposed a novel rank aggregation approach, TimeRA, that utilizes bursts and only rank information to aggregate result lists. TimeRA first detects bursts of posts across the lists utilizing original rank information of the posts, and then rewards posts that are ranked low in few lists but in the vicinity of a burst that contains higher ranked posts. It also infers the rank scores of missing posts by modeling lists and posts as a mixture of latent factors.

Our experimental results show that both utilizing burst information and score inference for rank aggregation can significantly enhance retrieval performance when compared against traditional and state-of-the-art, supervised and unsupervised rank aggregation approaches for microblog post search. Additional analyses show that TimeRA is a robust and efficient rank aggregation method that outperforms state-of-the-art temporal retrieval algorithms.

Table 5: Performance on 4 time-sensitive result lists. Boldface marks the better result per metric; a statistically significant difference between TimeRA and the best baseline is marked in the upper right hand corner of TimeRA score; a statistically significant difference with TSF+QDRM for each fusion method is marked in the upper left hand corner of the score.

	MAP	p@5	p@10	p@15	p@30
TSF+QDRM	.2834	.6220	.6856	.6279	.5368
DIRECT-BM25 (mean)	.2798	.6187	.6725	.6320	.5133
LM-T (qe)	.2346	.5836	.5648	.5178	.4471
TBLM	.2231	5742	5433	.5017	.4395
CombSUM	$^{\triangle}$.2962	$^{\triangle}$.6395	$^{\blacktriangle}$.6973	$^{\blacktriangle}$.6482	$^{\blacktriangle}$.5513
ClustFuseCombSUM	$^{\triangle}$.2951	$^{\triangle}$.6385	$^{\triangle}$.6927	$^{\blacktriangle}$.6407	$^{\blacktriangle}$.5476
CombMNZ	$^{\triangle}$.2948	$^{\triangle}$.6350	$^{\triangle}$.6918	$^{\triangle}$.6347	$^{\triangle}$.5420
ClustFuseCombMNZ	.2886	.6312	.6873	.6283	.5416
λ-Merge	.2847	.6275	.6876	.6279	.5383
TimeRA	$^{\blacktriangle}$**.3117**$^{\blacktriangle}$	$^{\blacktriangle}$**.6518**$^{\triangle}$	$^{\blacktriangle}$**.7023**$^{\triangle}$	$^{\blacktriangle}$**.6545**$^{\blacktriangle}$	$^{\blacktriangle}$**.5733**$^{\blacktriangle}$

As to future work, we plan to look into combining social information, such as user relationships into rank aggregation and further analyze our model in scenarios where the documents being searched are published in bursts.

Acknowledgements. This research was supported by the China Scholarship Council, the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreements nr 288024 (LiMoSINe) and nr 312827 (VOX-Pol), the Netherlands Organisation for Scientific Research (NWO) under project nrs 727.011.005, 612.001.116, HOR-11-10, 640.-006.013, the Center for Creation, Content and Technology (CCCT), the Dutch national program COMMIT, the ESF Research Network Program ELIAS, the Elite Network Shifts project funded by the Royal Dutch Academy of Sciences (KNAW), the Netherlands eScience Center under project

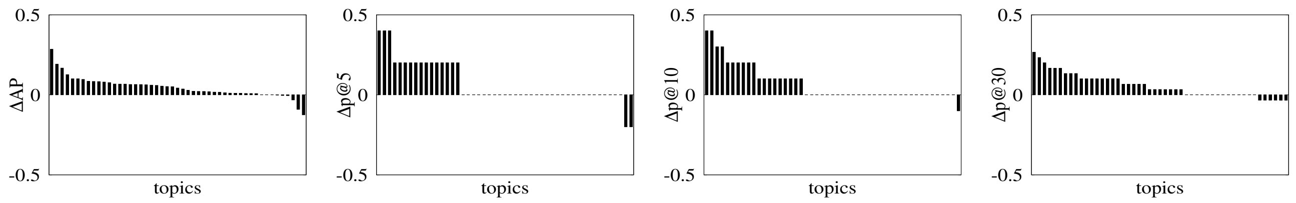


Figure 5: Per query performance differences, TimeRA vs. CombSUM. The figures shown are for the runs in Class 1, for MAP, p@5, p@10 and p@30 (left to right). A bar extending above the center indicates that TimeRA outperforms CombSUM, and vice versa.

number 027.012.105, the Yahoo! Faculty Research and Engagement Program, the Microsoft Research PhD program, and the HPC Fund.

8. REFERENCES

[1] J. A. Aslam and M. Montague. Models for metasearch. In *SIGIR*, pages 276–284, 2001.

[2] H. Bota, K. Zhou, J. M. Jose, and M. Lalmas. Composite retrieval of heterogeneous web search. In *WWW*, 2014.

[3] W. Dakka, L. Gravano, and P. Ipeirotis. Answering general time-sensitive queries. *IEEE Trans. Knowledge and Data Engin.*, 24(2): 220–235, 2012.

[4] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622, 2001.

[5] E. A. Fox and J. A. Shaw. Combination of multiple searches. In *TREC-2*, 1994.

[6] Z. Han, X. Li, M. Yang, H. Qi, S. Li, and T. Zhao. Hit at trec 2012 microblog track. In *TREC '12 Working Notes*, 2012.

[7] A. K. Kozorovitsky and O. Kurland. Cluster-based fusion of retrieved lists. In *SIGIR*, pages 893–902, 2011.

[8] M. Kurucz, A. A. Benczúr, and K. Csalogány. Methods for large scale SVD with missing values. In *KDD Cup and Workshop*, 2007.

[9] J. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *SIGIR*, 2001.

[10] T. Lappas, B. Arai, M. Platakis, D. Kotsakos, and D. Gunopulos. On burstiness-aware search for document sequences. In *SIGKDD*, pages 477–486, 2009.

[11] J. H. Lee. Combining multiple evidence from different properties of weighting schemes. In *SIGIR*, pages 180–188, 1995.

[12] X. Li and W. B. Croft. Time-based language models. In *CIKM*, pages 469–475, 2003.

[13] S. Liang, M. de Rijke, and M. Tsagkias. Late data fusion for microblog search. In *ECIR'13*, 2013.

[14] S. Liang, Z. Ren, and M. de Rijke. Fusion helps diversification. In *SIGIR*, 2014.

[15] S. Liang, Z. Ren, and M. de Rijke. Personalized search result diversification via structured learning. In *SIGKDD '14*, 2014.

[16] S. Liang, Z. Ren, and M. Rijke. The impact of semantic document expansion on cluster-based fusion for microblog search. In *ECIR*, pages 493–499, 2014.

[17] J. Lin, C. Macdonald, I. Ounis, and I. Soboroff. Overview of the TREC 2011 Microblog track. In *TREC 2011*. NIST, 2011.

[18] H. Ma, D. Zhou, and C. Liu. Recommender systems with social regularization. In *WSDM*, 2011.

[19] K. Massoudi, M. Tsagkias, M. de Rijke, and W. Weerkamp. Incorporating query expansion and quality indicators in searching microblog posts. In *ECIR*, pages 362–367, 2011.

[20] M. Mathioudakis, N. Bansal, and N. Koudas. Identifying, attributing and describing spatial bursts. In *VLDB*, pages 1091–1102, 2010.

[21] T. Miyanishi, K. Seki, and K. Uehara. Improving pseudo-relevance feedback via tweet selection. In *CIKM*, pages 439–448, 2013.

[22] M. Montague and J. A. Aslam. Condorcet fusion for improved retrieval. In *CIKM*, pages 538–548, 2002.

[23] M.-H. Peetz, E. Meij, M. de Rijke, and W. Weerkamp. Adaptive temporal query modeling. In *ECIR '12*, 2012.

[24] W. L. Ruzzo and M. Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *Int. Conf. Int. Syst. Molec. Biol.*, pages 234–241, 1999.

[25] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, 2008.

[26] J. A. Shaw and E. A. Fox. Combination of multiple searches. In *TREC 1992*, pages 243–252. NIST, 1993.

[27] D. Sheldon, M. Shokouhi, M. Szummer, and N. Craswell. LambdaMerge: merging the results of query reformulations. In *WSDM*, pages 795–804, 2011.

[28] M. Shokouhi and L. Si. Federated search. *Foundations and Trends in Information Retrieval*, 5(1):1–102, 2011.

[29] I. Soboroff, I. Ounis, and J. Lin. Overview of the TREC 2012 Microblog track (notebook version). In *TREC 2012*. NIST, 2012.

[30] W. Weerkamp and M. de Rijke. Credibility-inspired ranking for blog post retrieval. *Information Retrieval Journal*, 15(3–4):243–277, 2012.

[31] S. Wu. *Data fusion in information retrieval*, volume 13 of *Adaptation, Learning and Optimization*. Springer, 2012.

APPENDIX

A. DERIVATION OF THE MODELS

The following is the derivation of Eq. 12. We leave out the derivations of Eq. 4, 5, 6, 7 and 8 as they may be obtained in a similar way.

$$
\begin{aligned}
\frac{\partial C_2}{\partial \mathbf{S}_i} =& (1-\beta)\sum_{j=1}^{n} I_{ij} w(d_j|L_i)(g(\mathbf{S}_i^\top \mathbf{V}_j) - R_{ij}) g'(\mathbf{S}_i^\top \mathbf{V}_j)\mathbf{V}_j \\
&+ \sum_{j=1}^{n}\sum_{b\in\mathcal{B}} \frac{\beta}{|\{d_k : d_k L_i d_j |b\}|} \sum_{\substack{d_k\in b\\ d_k L_i d_j}} I_{ij} I_{ik} r(d_j|d_k) \\
&\cdot w(d_k|L_i)(g(\mathbf{S}_i^\top \mathbf{V}_j) - R_{ik}) g'(\mathbf{S}_i^\top \mathbf{V}_j)\mathbf{V}_j + \lambda_1 \mathbf{S}_i^\top \\
=& (1-\beta)\sum_{i=1}^{m} I_{ij} w(d_j|L_i)(g(\mathbf{S}_i^\top \mathbf{V}_j) - R_{ij}) g'(\mathbf{S}_i^\top \mathbf{V}_j)\mathbf{S}_i^\top \\
&+ \sum_{i=1}^{m}\sum_{b\in\mathcal{B}} \frac{\beta}{|\{d_k : d_k L_i d_j |b\}|} \sum_{\substack{d_k\in b\\ d_k L_i d_j}} I_{ij} I_{ik} r(d_j|d_k) \\
&\cdot w(d_k|L_i)(g(\mathbf{S}_i^\top \mathbf{V}_j) - R_{ik}) g'(\mathbf{S}_i^\top \mathbf{V}_j)\mathbf{S}_i^\top + \lambda_2 \mathbf{V}_j, \qquad (15)
\end{aligned}
$$

where $g'(x) = \exp(x)/(1+\exp(x))^2$ is the derivative of the logistic function $g(x) = 1/(1+\exp(-x))$.

B. λ-MERGE AND ITS FEATURES

Originally λ-Merge is an algorithm that fuses the result lists of query reformulations. The method we call λ-Merge is the same as the original one except that we consider the weights of query reformulations in the original λ-Merge as those of the result lists in response to the same query in our experiments. In total, we use 13 features, extracted at three levels: time-sensitive level, query-document level, document level; all features are represented by either binary or real numbers. At time-sensitive level, we obtain the retrieval scores from our 4 time-sensitive retrieval baselines, i.e., TBLM [12], LM-T(qe) [19], DIRECT-BM25 (mean) [3] and TSF+QDRM [21], as features. At the query-document level, we use 3 features: *Rank*, *Rankers* and *IsTop-N*. *Rank* is defined in Eq. 1. *Rankers* is the number of ranked lists in which the post appears over the total number of lists to be merged. *IsTop-N* is a binary feature to indicate if this document is within the top-N results in the list L_i, where $N \in \{5, 10, 15, 20\}$. At the document level, we extract features capable of indicating the quality of a post [17, 29]. The post features include *Link*, *Hashtag*, *Retweet*; the values of these three features are set to 1 if the document contains links, hashtags, and retweets. The document-level features also consist of content quality indicators of a post (*Density*, *Capitalization* and *Length*) [30].