IRanker: Query-Specific Ranking of Reviewed Items

Moloud Shahbazi
UC Riverside
Email: mshah008@cs.ucr.edu

Matthew Wiley
UC Riverside
Email: mwile001@cs.ucr.edu

Vagelis Hristidis
UC Riverside
Email: vagelis@cs.ucr.edu

***Abstract*—Item (e.g., product) reviews are one of the most popular types of user-generated content in Web 2.0. Reviews have been effectively used in collaborative filtering to recommend products to users based on similar users, and also to compute a product's star rating. However, little work has studied how reviews can be used to perform query-specific ranking of items. In this paper, we present efficient top-k algorithms to rank items, by weighing each review's rating by its relevance to the user query. We propose a non-random access algorithm and perform a comprehensive evaluation of our method on multiple datasets. We show that our solution significantly outperforms the baseline approach in terms of query response time.**

I. Introduction

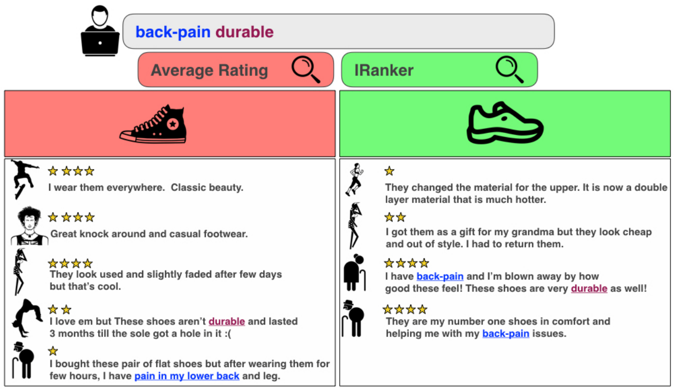
In the current era of Web 2.0, users generate huge numbers of reviews for products and services in various domains such as movies and physicians. A review typically consists of a text description and a star rating score. There has been much work on several aspects of Internet reviews such as extracting features from reviews [11], summarizing them [5], and detecting fake reviews [10].

A key challenge is how to leverage reviews to help users find the best items. Existing work has mainly focused on two directions. First, collaborative filtering studies how the reviews of similar users may be leveraged to recommend products to users [13]. Second, reviews – typically their average rating and number – are being used to rank products, along with other features like price, combined by learning to rank algorithms [3].

In this paper we argue for a more effective use of reviews in item ranking, by introducing a ranking scheme which takes into account the user's interest in particular aspects of the search result. Figure 1 shows an example of a user of an online footwear store who searches for "durable" shoes that are also suitable for "back-pain". If the search results are sorted by the commonly used average rating score, the left shoe is ranked higher, even though the right shoe has better ratings for the user's properties of interest. *IRanker improves the results' quality by considering the relevance of each review to the user query*, where a query can be expressed by a list of keywords or concepts.

Problem Statement: Our core problem is defined as follows: *given* (a) a set of reviews for each item, where each review consists of a set of concepts and a rating score, and (b) a query that consists of a set of concepts, *compute* the top ranked items for the query. In our solution, each review's rating is weighed by the similarity (relevance) of the review to the query.

Challenges and Algorithms Overview: A key challenge is that we cannot precompute a rating score for an item as it depends on the user query. Existing early termination algorithms that process list prefixes to compute the top-k results [4], [6] cannot be applied, because for an unseen review, we do not know its rating nor its similarity to the query.



★★★★
I wear them everywhere. Classic beauty.

★★★★
Great knock around and casual footwear.

★★★★
They look used and slightly faded after few days but that's cool.

★★
I love em but These shoes aren't durable and lasted 3 months till the sole got a hole in it :(

★
I bought these pair of flat shoes but after wearing them for few hours, I have pain in my lower back and leg.

★
They changed the material for the upper. It is now a double layer material that is much hotter.

★★
I got them as a gift for my grandma but they look cheap and out of style. I had to return them.

★★★★
I have back-pain and I'm blown away by how good these feel! These shoes are very durable as well!

★★★★
They are my number one shoes in comfort and helping me with my back-pain issues.

Fig. 1: Two possible top search result for a query.

These two quantities have an intimate dependency to each other and the overall item score. Further, due to the interplay between similarity (which may be partially known during the execution of the algorithm) and rating of unseen or partially seen reviews, we are faced with a combinatorial number of cases for computing the terminating condition threshold. Instead, we propose a linear cost method to compute this threshold.

Moreover, in contrast to the setting of top-k algorithms [4], [6] where multi-attribute objects are ranked, we rank items that are collections of objects (reviews). We show that this cannot be handled by simply adding another level of top-k lists aggregation.

II. Definitions

An *item* I, which may be a doctor or an Amazon product, contains a set $\{r_1, r_2, ..., r_n\}$ of user reviews. A *review* $r = \{c_1, c_2, ..., c_n; rating\}$ consists of a set of concepts c_i and a numeric rating score between 0 and 1. For instance, a review may be $r = \{\text{“}cancer\text{”}, \text{“}cardiac\text{”}, \text{“}EKG\text{”}; 0.6\}$, which expresses that the concepts "cancer", "cardiac" and "EKG" are mentioned in the review, and the user rated the item with a score of 0.6.

Item Ranking Problem: Given a collection $\mathcal{I}$ of items, a *query* $Q = \{q_1, q_2, ..., q_m\}$, which is a set of concepts, and k requested top results, return the top-k items in $\mathcal{I}$ with highest score $score(Q, I)$.

The scoring function in Equation 1 is partly inspired by Zhang et al. [14], where they weigh product ratings based on the usefulness of each review. The main distinction of our method is that we average review rating weighted by their relevance to the query. If review usefulness is available, it can also be multiplied. Note that usefulness is fixed for a review, in contrast to relevance which is query-dependent.

$$score(Q, I) = \frac{\sum_{r \in I} QRSim(Q, r.concepts) \times r.rating}{\sum_{r \in I} QRSim(Q, r.concepts)}, \quad (1)$$

where $QRSim(Q, r)$ is the similarity of Q to the concepts $r.concepts$ in r.

We can extend this ranking function to include other item and review features such as number of item reviews, price or popularity or review helpfulness and freshness. Our main focus is on how to compute the top results efficiently.

The problem of computing the semantic similarity between two sets of concepts (of the item and the query) has been extensively studied in the past [12], [2], [9]. In this work, we consider Jaccard similarity, a popular and representative similarity measure.

$$QRSim(Q, r) = \frac{|Q \cap r.concepts|}{|Q \cup r.concepts|} \tag{2}$$

III. Ranking Algorithm

In this section, we present our solution to compute the top-k items for the item ranking problem. Fagin et al. [4] proposed two algorithms with different access modes – TA has random access and NRA does not – for finding the top-k multi-attribute objects, based on a monotonic aggregation function, when there is ranked list for each attribute. These algorithms perform sorted access to the attribute lists in parallel, and terminate when k objects are found whose scores are greater than the maximum possible score of other objects.

NRA-IRanker: We propose NRA-IRanker, a non-random access top-k items ranking algorithm that computes the exact top-k results.

For the proposed algorithm to work, the aggregate function in Equation 1 must be monotonic on the review ratings. That is, if the rating of any review of an item is increased, the score of that item is also increases, or stays the same. It is fairly easy to see that Equation 1 satisfies this condition.

To store the data for the algorithm, we use *Concepts index*, an inverted index that has a list for each concept. This list contains information of the reviews that include the key concept sorted by decreasing review rating. Since there is no random access made for reviews, NRA-IRanker needs to keep track of partially seen items as well as the partially seen reviews. Thus, every review in a concept list includes the following values: review-id, item-id, review rating, the total number of item's reviews and the total number of the concepts of the review.

When a query Q is issued, NRA-IRanker works as following:

While top-k items are not found, repeat:

1) Do a sequential sorted access to the query concepts lists in parallel. Every time select next review from the list with the maximum current review score. Then, update the maximum possible score of the unseen review which we denote by y_{max}.
2) For each review that has been seen in the concept list, compute the minimum and maximum possible review similarity.
3) For each item that has been seen in the concept list, compute the minimum and maximum possible item score.
4) Check termination condition by examining if there are k items with a minimum possible score greater than or equal to the y_{max} and the maximum possible score of the rest of items.

To compute y_{max}, we need to estimate the reviews of the unseen item that maximizes the score; note that we do not know which item that is yet. According to Equation 1, if an unseen item has one or more reviews, and all of them contain only the query concept (a condition which maximizes the Jaccard similarity shown in Equation 2) with maximum rating (which we assume to be equal to the last seen review's rating), then y_{max} is equal to that rating. As a result, y_{max} is equal to the maximum rating of the last seen review across all query concepts lists. Formally, if $Y = \{y_1, \cdots, y_m\}$, where y_i is the current rating of the ith concept list, then $y_{max} = max\ Y$.

As a result of non-random access pattern, there are three types of reviews: *(1) Seen reviews, (2) Partially seen reviews and (3) Unseen reviews.* A *seen* review is a review that has been seen in all possible query concept lists, except the lists that have been completely read or have a current rating smaller than the review's rating. Thus, the exact similarity of the seen reviews to the query can be computed. A *partially seen* review is the one currently seen in at least one query concept list and could be seen in more lists in the future. For every partially seen review, we maintain a minimum and a maximum possible similarity to the query.

As we mentioned earlier, the total number of reviews of an item is stored in the review entries. Therefore, the number of unseen reviews of a *partially seen* item is available for computing the minimum and the maximum possible score of a *partially seen* item. Thus, we define three types of items using the number of unseen reviews: *(1) Seen items, (2) Partially seen items and (3) Unseen items.*

Computing Query-Review Similarity Bounds: We begin by describing the concept of "unsure" list. If a partially seen review is not yet seen in a concept list and the current rating of this list is greater than or equal to the rating of the review, the presence of the review in this list is "unsure" (since concept lists are sorted by review rating). If a review r is "unsure" in z lists out of the $|Q|$ lists of the query concepts, the maximum possible similarity of this review to the query is achieved when this review is in all "unsure" lists. On the other hand, the minimum possible similarity is achieved when the review does not exist in any "unsure" list. Equations 3 and 4 define the maximum and the minimum possible similarities of the partially seen reviews to the query Q respectively.

$$XPQRSim(Q, r) = \frac{z + f}{|r| + |Q| - z - f} \quad , z + f \leq |Q| \tag{3}$$

$$MPQRSim(Q, r) = \frac{f}{|r| + |Q| - f} \quad , z + f \leq |Q| \tag{4}$$

Where f is the number of concepts lists that review is "seen" in them and z, the number of "unsure" concepts lists formally stated as $z = |\{y_i | y_i \geq r.rating, y_i \in Y, y_i \neq null\}|$. where Y is the set of current ratings of query concept lists. A $null$ value as a current rating means that the list has been read completely. Based on Equations 3 and 4, since $z = 0$ for the seen reviews, the minimum and the maximum possible similarities are equal to the exact similarity of the review. For

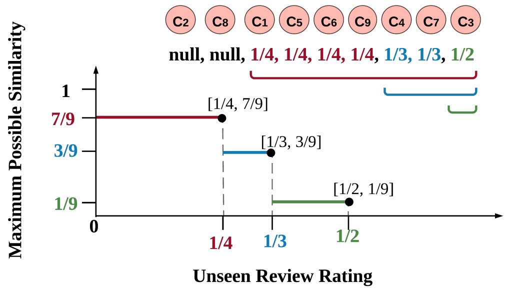


Fig. 2: An example of possible $[rating, similarity]$ pairs of an unseen review shown by black points on the diagram for a query consisting of nine query concepts.

unseen reviews, where $f = 0$, the minimum possible similarity of u, an unseen review, and Q is 0.

The maximum possible similarity of an unseen review depends on its rating, as explained by the example in Figure 2. The leftmost point $(1/4, 7/9)$ denotes that if the rating of an unseen review is equal to $1/4$, then its maximum similarity would be $7/9$, because at best it may contain seven of the nine query concepts, as the first two lists have been completely read (null) and the last six lists have current ratings greater than $1/4$. Similarly, the other points denote possible rating and maximum possible similarity combinations, $[rating, similarity]$.

Formally, we define U, the set of possible $[rating, similarity]$ pairs for $QRSim$ as follows:

$$U = \{[y_l, \frac{|l|y_o \geq y_l, y_o \in Y|}{|Q|}]|y_l \in Y\}$$

Note that if an unseen review does not include any query concept then its similarity is equal to 0 and it does not affect the item's score.

Computing Item Score Bounds: Now that we have defined the space of possible $[rating, similarity]$ values for unseen reviews, we need to decide which combination of them would maximize the score of a partially seen item. We use Theorem III.1 and Theorem III.2 to compute the maximum score of a partially seen item.

Theorem III.1. *Let I be an item with h unseen reviews and a set R of seen reviews. If there is a set U of candidate $[s, w]$ (s stands for the rating and w stands for similarity) pairs, in order to maximize the score of I, we can reuse a single pair in U for all h unseen reviews.*

Similarly, to minimize the score of item I, we use one pair that is $[0, \max w_j]$ for all h unseen reviews. Proof is omitted to conserve space.

Next, we describe how we compute the minimum and the maximum possible scores of partially seen items based on the weighted mean of review ratings for items as defined in Equation 1 using the minimum and the maximum possible similarities of the partially seen and unseen reviews.

It is not practical to check all combinations of query-review similarities in order to compute the maximum score. Instead, we invoke Theorem III.2 to compute the maximum possible score of a partially seen item with an optimal greedy solution, which has linear complexity on the number of item reviews.

Theorem III.2. *Given an item I with a set R of n partially seen reviews, where each review r has a known $r.rating$, minimum similarity $MPQRSim(Q, r)$, and maximum similarity $XPQRSim(Q, r)$, there is a review $r_i \in R$ such that the score of I is maximized if we assign similarity $XPQRSim(Q, r)$ to all reviews with rating greater than or equal to $r_i.rating$ and $MPQRSim(Q, r)$ to the rest.*

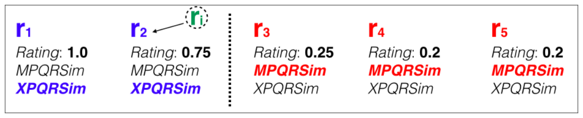


Fig. 3: An example case of Theorem III.2. Here we show a list of an item's reviews sorted by review rating in descending order. The maximum possible score of the item is achieved when the first two reviews are weighted by the maximum possible similarity, and the last three reviews are weighted by the minimum possible similarity.

Figure 3 shows a list of five reviews sorted by rating, and r_2 is the divider review (review r_i as stated in Theorem III.2). We omit the proof to conserve space.

In the presence of unseen reviews with unknown ratings, we compute a locally maximum item score for every possible pair of $[rating, similarity]$ of unseen reviews. Therefore, the maximum possible score of the item is the maximum of all local maximum scores (for a $[rating, similarity]$ pair).

Similarly, in order to calculate the minimum possible score of a partially seen item, there is only one way to partition the sorted list of partially seen reviews by rating such that query-review similarity is maximized for the first partition and it is minimized for the second partition.

IV. Evaluation

In this section we evaluate the run-time performance of our proposed algorithm. We start by describing experimental settings followed by the experimental results. Then, we compare the run-time performance of our proposed algorithms against two baselines and discuss the results in detail.

1) Experimental Setting:
Baseline Algorithm: We compare our solution to a baseline algorithm that uses the same *Concepts* index as used in NRA-IRanker to get all the reviews that contain at least one of the query concepts. Then, we group the reviews of each item and compute the weighted average based on Equation 2.

Dataset: In order to evaluate our algorithm, we use healthcare provider reviews that are crawled from the vitals.com and ucomparehealthcare.com websites. We merge items and reviews of these two websites into a single dataset and in the following we will refer to the combined dataset as the *Doctors* dataset. Using the MetaMap tool [1], we extracted medical concepts from the textual content of the reviews.

In addition to *Doctors* dataset, we use *Amazon* product reviews dataset [8]. This dataset is collected from Amazon.com and is a fairly comprehensive collection of English language product data in a wide variety of categories such as books, clothing and movies. Each product is provided with its reviews including textual reviews and star ratings. Table I summarizes the characteristics of the datasets used in this study.

Experimental Setup and Parameters: The *Concepts index* is stored in a Cassandra data store [7] on a single node. We chose a NoSQL store, because we only need an efficient mechanism

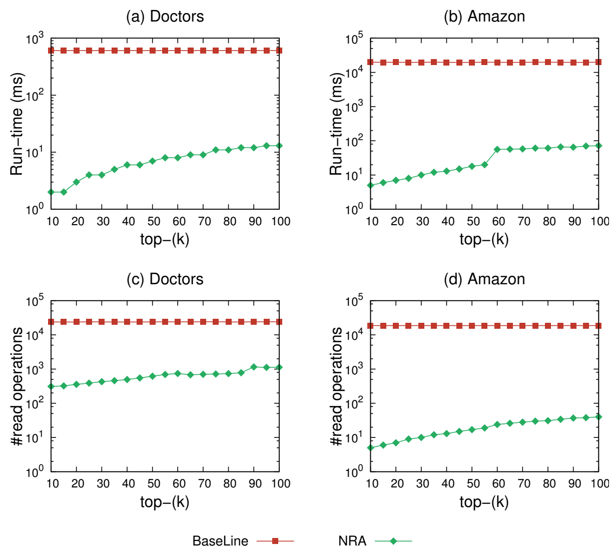


Fig. 4: Time and number of read operations vs. number of top items for queries of size $|Q| = 3$.

TABLE I: Datasets' characteristics.

Dataset	#Items	#Reviews	Reviews per Item	Concepts per Review
Doctors	248580	726996	Min: 1 Max: 249 Mean: 2.9 Median: 5	Min: 0 Max: 121 Mean: 3.29 Median: 7
Amazon	9743974	82037337	Min: 1 Max: 25260 Mean: 8.4 Median: 2	Min: 1 Max:1719 Mean: 26.65 Median: 15

to look up index lists and no SQL capabilities. Other key-value stores can also be used.

We partition all the index lists into 8 KB chunks to allow for incremental reading of the lists. That is, each index list is a row in Cassandra, and each chunk in the list is a column in this row.

Query workload generation: For the *Doctors* datasets, we generate queries by choosing a set of query concepts, where each concept is selected with probability proportional to its frequency in the reviews. For every experiment, we present results averaged over 100 queries. The queries for the Amazon dataset are selected from the queries suggested by the website in different categories. For example, "spray face child sun screen" is a search query suggested by amazon.com.

Parameters: We evaluate the performance of our algorithms by varying the number of top items and the query size.

2) Experimental Results: In this section, we compare the run-time performance of our algorithm against the baseline using several experiments on *Doctors* and *Amazon* datasets.

Figure 4 shows the experimental run-time for *Doctors* and *Amazon* datasets for different values of k, the number of top items, given queries of size 3. Figures 4(a) and 4(b) show the average retrieval time of different algorithms for different datasets. In Figures 4(c) and 4(d) we demonstrate the total number of read operations made to *Concepts index*.

The results of our experiment show that NRA-IRanker is significantly faster than the baseline that process complete lists to generate same exact top-k result and it needs to make much less read accesses to the database.

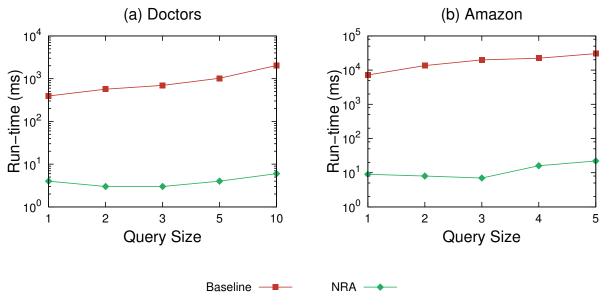


Fig. 5: Time vs. query size for top-20 items.

We also show the query response time for different query sizes in Figure 5. In this experiment, the query time of the baseline method increases with increase in query size because it reads more review lists per query concept and need to process more data. For *Amazon* dataset, we use product queries that have 5 terms at most. That explains the different range of query size for Amazon dataset. The results clearly indicates the significant run-time improvement using NRA-IRanker for querying real world large scale datasets.

ACKNOWLEDGMENT

Partially supported by NSF grants IIS-1216007, IIS-1447826 and IIS-1619463.

REFERENCES

[1] A. R. Aronson. Effective mapping of biomedical text to the umls metathesaurus: the metamap program. In *Proceedings of the AMIA Symposium*. American Medical Informatics Association, 2001.

[2] M. Batet, D. Sánchez, and A. Valls. An ontology-based measure to compute semantic similarity in biomedicine. *Journal of biomedical informatics*, 2011.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ACM ICML*, 2005.

[4] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 2003.

[5] M. Hu and B. Liu. Mining and summarizing customer reviews. In *ACM SIGKDD*, 2004.

[6] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *CSUR*, 2008.

[7] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 2010.

[8] J. McAuley, R. Pandey, and J. Leskovec. Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2015.

[9] G. B. Melton, S. Parsons, F. P. Morrison, A. S. Rothschild, M. Markatou, and G. Hripcsak. Inter-patient distance metrics using snomed ct defining relationships. *Journal of biomedical informatics*, 2006.

[10] A. Mukherjee, B. Liu, and N. Glance. Spotting fake reviewer groups in consumer reviews. In *ACM WWW*, 2012.

[11] A.-M. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *Natural language processing and text mining*. Springer, 2007.

[12] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on*, 1989.

[13] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009.

[14] Z. Zhang. Weighing stars: Aggregating online product reviews for intelligent e-commerce applications. *IEEE Intelligent Systems*, 2008.