RankFP: A Framework for Supporting Rank Formulation and Processing

Hwanjo Yu
University of Iowa
hwanjoyu@cs.uiowa.edu

Seung-won Hwang, Kevin Chen-Chuan Chang
University of Illinois
{shwang5,kcchang}@uiuc.edu

Data retrieval – finding relevant data from large databases – has become a serious problem. Consider user Amy, who is looking for a house in Chicago. She searches *realtor.com* with a few constraints on city, price range, beds, baths, which returns 3,581 matching houses. Or, Amy may realize that she must “narrow” her query – However, on this extreme, and equally undesirable, she may as well get *no hits* at all. She will likely manually “oscillate” between these extremes, before eventually managing to complete her data retrieval task, if at all.

Relational databases offer little support for such retrieval tasks. Traditional Boolean-based query models like SQL are based on “hard” criteria (*e.g.*, price < $100,000) while users often employ “soft” criteria for their specific senses of “relevance” or “preference.” Unlike flat Boolean results, these fuzzy criteria naturally calls for *ranking*, to indicate how well the results match. Such ranking is essential for data retrieval, by ordering answers according to their matching “scores.” Thus, on one hand, there will not be *too many* matches, since ranking focuses users on the *best matches*. On the other hand, neither will there be *no hits*, since ranking will return even *partial matches*. While such ranking has been the norm for “text” retrieval [7] (*e.g.*, search engines like Google), it is critically missing in relational database systems for supporting similar “data” retrieval.

To enable *ad-hoc ranking* for data retrieval, we observe two major barriers: First, *usability*: Ad-hoc ranking should be “user friendly,” for *ordinary users* to easily specify their ranking criteria– Note that, as motivated earlier, data retrieval must accommodate everyday users. Second, *efficiency*: Ad-hoc ranking should be “database friendly,” to be amenable to efficient processing. Note that data retrieval, with many interesting scenarios *online*, must essentially achieve responsive processing.

Supporting ranking has been researched in both databases and machine learning communities with different aspects of interests: On one hand, the databases community has studied top-k query processing [1, 2, 3, 4], focusing on designing rank query models and efficient algorithms. However, they clearly lack the support for intuitively formulating ranking in the first place, to accommodate everyday users like Amy. On the other hand, the machine learning community has studied learning ranking from examples (*i.e.*, partial orders), using sophisticated techniques, *e.g.*, SVMs [5, 6]. However, such learning is mostly “offline”– 1) it requires pre-labeled training examples, and 2) it requires per-object evaluation of the learned classifier function–For our goal of ad-hoc ranking, both training and classification must be online.

This paper proposes a new framework such that: 1) to achieve *usability*, it allows users to *qualitatively and intuitively* express their preferences by partial orders on selected examples, from which it effectively learns a *quantitative* global ranking function, and (2) to achieve *efficiency*, it integrates the front-end machine learner with a back-end top-k query processor to evaluate the learned functions.

First, to support efficient query processing, our framework assumes the *score-based ranking model*. Such a model is both expressive and amenable to efficient query processing. To see why, consider a data retrieval scenario, where queries capture preference.

Example 1 Amy, who is looking for a house, prefers those somehow $cheap$, $large$, and in a $safe$ area. Assume these “predicates” or “features” are specified (*e.g.*, $cheap$ below), each as a *soft* predicate returning a matching score.

Predicate - $cheap$(h.price):
If (h.price $> 500,000$) **Then Return** $1.0 - \frac{\text{h.price}}{MAX_PRICE}$
Else Return 1.0

To *rank* results in the order of preference, she may formulate a query, combining features with min as the ranking function.

Query Q:
select h.id, h.address **from** Houses h
where h.city = "Chicago"
order by $min(f_1{:}cheap$(h.price), $f_2{:}large$(h.size), $f_3{:}safe$(h.zip)$)$

■

Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)
1084-4627/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

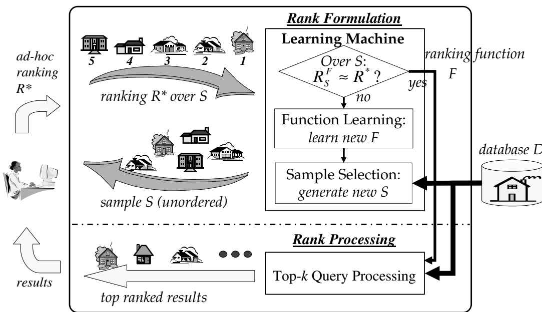

Figure 1. Framework *RankFP*: Rank formulation and processing for data retrieval.

The task in the *score-based model* is to rank a database of n objects $\mathcal{D} = \{u_1, \ldots, u_n\}$ (*e.g.*, Houses in Example 1). For each object u, some m soft *predicates* $f_1, \ldots, f_m$ evaluate u to scores in [0:1], which are then aggregated by some *ranking function* $\mathcal{F}$, *i.e.*, $\mathcal{F}(f_1, \ldots, f_m)[u]$ = $\mathcal{F}(f_1[u], \ldots, f_m[u])$. All objects are then ranked, highest first, by their ranking *scores* $\mathcal{F}(f_1, \ldots, f_m)[u]$ or $\mathcal{F}[u]$ for short. For instance, Query Q in Example 1 uses $min(f_1, f_2, f_3)$ as the ranking function.

This score-based model matches well our goal of supporting ad-hoc ranking: First, such a ranking function is *compatible* with and thus already expressible in SQL (*e.g.*, as the Query Q in Example 1). Second, it is simple yet *expressive*, by determining a global ordering with a single formula. Third, it is amenable to *efficient* top-*k* query processing, for which we can leverage several existing techniques.

While the score-based ranking is both expressive and efficient, formulating underlying ranking functions is challenging to users. It is far from trivial for the user to articulate how she evaluates each and every object into an absolute numeric score, that is, to express her preference with the query language and soft predicates. Note that, unlike relational database systems (where "canned" transactions or queries seem to be the dominant usage), common users for data retrieval tasks are ordinary people like Amy. Thus, to accommodate such users, rank formulation must be essentially supported– without which ranking is not usable.

We thus develop the *RankFP* framework, aiming at integrating a "front-end" for learning-based rank query formulation to a "back-end" for score-based rank query processing. As Figure 1 illustrates, first, with the "iterative learning" front-end (at the top), our *RankFP* framework supports users to formulate queries in a process that is both exploratory (as the system iteratively shows database sample objects) and intuitive (by specifying only partial ordering on those samples). Second, with the score-based top-*k* query processing back-end (bottom), our framework supports online "classification" (*i.e.*, to evaluate the learned function) as query processing to return ranked answers efficiently.

Rank Formulation: The rank formulation module (Figure 1, top) iteratively interacts with the user to learn the desired ranking function. This process operates in *rounds*. In each round, the learning machine selects a sample S of a small number of l objects (for $l \ll |\mathcal{D}|$; *e.g.*, $l = 5$ in our study). The user orders these examples by her desired ranking R^*; thus she "labels" these examples as training data. The learning machine will thus construct a function $\mathcal{F}$ from the training examples so far; let $R_S^{\mathcal{F}}$ be the induced ranking over the latest sample S. At *convergence*, *i.e.*, when $R_S^{\mathcal{F}}$ is sufficiently close to R^* (*i.e.*, when $\mathcal{F}$ is *accurate* on S), the learner will halt and output $\mathcal{F}$ as the learned ranking function. We use SVM for learning ranking, which has shown high learning performance.

Rank Processing: The rank processing module (Figure 1, bottom) carries out the "classification" by the learned function $\mathcal{F}$ as online query processing over the entire database. We develop the *dual* views of such $\mathcal{F}$ to realize the integration of the rank learning front-end and the rank processing back-end: We connect a classifier function (on object pairs) from the learner, to a global ranking function (on per-object) for the query processor. With this connection, we effectively transform the classification task into a standard $\mathcal{F}$-based top-k query processing problem, allowing us to adopt existing algorithms.

We perform our experiments on a real-estate system. with real-life house dataset used in [2].[1] To evaluate the framework extensively, we synthetically generate queries with various complexity and measure the performance. We then evaluate the framework with real-life queries, collected from our user study. For the real preferences, our rank query framework formulates an accurate (i.e., accuracy $\geq$ 90%) ranking function in a couple of iterations ($\leq$ 2) with a user quickly (i.e., response time $\leq$ 10 millisec.) and automatically (without any parameter tuning).

In conclusion, we have developed framework *RankFP* aiming at integrating rank formulation and processing to support ad-hoc data retrieval.

References

[1] M. J. Carey and D. Kossmann. On saying "enough already!" in SQL. In *SIGMOD*, pages 219–230, 1997.

[2] K. C.-C. Chang and S.-W. Hwang. Minimal probing: Supporting expensive predicates for top-k queries". *SIGMOD 2002*, 2002.

[3] S. Chaudhuri and L. Gravano. Evaluating top-*k* selection queries. In *VLDB*, 1999.

[4] R. Fagin, A. Lote, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[5] R. Herbrich, T. Graepel, and K. Obermayer, editors. *Large margin rank boundaries for ordinal regression*. MIT-Press, 2000.

[6] T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD 2002*, 2002.

[7] G. Salton. *Automatic Text Processing*. Addison-Wesley, Reading, Mass., 1989.

[1]The dataset is available at http://aim.cs.uiuc.edu

1084-4627/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY