## 1. Summary

This paper presents an approach to automatically improve information retrieval quality by mining users' clickthrough data to provide training data *implicitly* in the form of *relative* preferences. With a formulation of the information retrieval learning problem to maximize Kendall's $\tau$, support vector machine trains retrieval functions for large number of queries and features tractably and efficiently.

In Section 2, the concept, storage and usefulness of clickthrough data is introduced. It has the form of a triplet (q, r, c), where q is query, r is ranking presented to a user and c is the set of links the user clicked on. With a proxy system, clickthrough data could be collected and stored in a logfile without adding too heavy overhead to the system. Since users' usually scan only the first about 10 links being presented, clickthrough data does not convey absolute relevance judgement. Alternatively, it expresses partial *relative* relevance judgements: the links not clicked by a user while ranked higher than a link clicked by the user is considered to be less relevant than the later. Thus, an algorithm for extracting preference feedback from clickthrough data is raised to generate *training* data in an *implicit* way.

Section 3 introduces a framework for learning of retrieval functions. Aiming to return a ranking of documents per their relevance to the given query, a retrieval function is evaluated by how closely its ordering of approximates the optimal one. Binary relevance performs coarsely, while Kendall's $\tau$ seems more appropriate for this problem thanks to the introduction of *discordant* pairs. And the goal becomes to maximize the expected Kendall's $\tau$ over a distribution of queries and optimal rankings on a document collection.

Then Section 4 comes out with an SVM algorithm for learning retrieval functions. The goal is converted into maximizing empirical Kendall's $\tau$, then to minimize the number of discordant pairs between the output of learned retrieval function and the optimal ranking. By introducing a linear ranking function as the product of a *weight* vector and a *feature* vector (mapping from queries and documents), the goal is converted into find the weight vector that maximizes the number of fulfilled inequalities about the relative ordering of document pairs from training set (clickthrough data). Though the general problem is NP-hard, SVM with regularization for margin maximization to the object provides an approximate solution. The new problem (called "Ranking SVM") is equivalent to that of a classification SVM on differences of pairwise feature vectors, so similar algorithms (e.g., $SVM^{light}$) is applicable.

Section 5 covers experimental evaluation of the proposed method. To learn a retrieval function with the Ranking SVM, the author tried to design a suitable feature mapping including rank in other search engines, query/content match and popularity-attributes. Moreover, the author developed a meta-search engine named "Striver" which works by integrating query results of some famous search engines for experimental setup, and also applied the idea of presenting two rankings from different retrieval functions simultaneously for comparison. The offline experiment shows that the Ranking SVM is able to learn to predict preference constraints. Furthermore, an online experiment verifies that the learned retrieval function improves retrieval quality within statistical significance.

## 2. Comments

As the paper claims, collecting training data for retrieval systems (functions) from experts' relevance judgements is difficult and expensive. Also, users usually do not have too much time to provide explicit feedback for rankings returned by a retrieval function. Thus, it is a smart and practical idea to obtain training data via user's clickthrough data implicitly.

In addition, the conversion of the optimization problem to operating the Ranking SVM on a linear function makes the solution efficient and interpretable.

**3. Questions**

(1) How can the Ranking SVM learn the weight vector as well as the weights of the feature vector for the linear retrieval function? I mean, usually feature vectors are fixed and SVM learns weight vector. But it seems both of them are unknown and needs to be learned at the same time.

(2) In the online experiment in Section Experiment, the number of training queries being collected is 260. I was wondering if it is enough for training the retrieval function with respect to many considered features. Is there any underfitting issue for implementation?