Context based Re-ranking of Web Documents (CReWD)

Arijit Banerjee, Jagadish Venkatraman
Graduate Students, Department of Computer Science,
Stanford University
{arijitb@stanford.edu, jagadish@stanford.edu}

***Abstract*- In this paper, we introduce CReWD, a machine learning system for Context Based Re-ranking of Web Documents. CReWD tackles personalization by considering both the short-term and long-term history of a user when personalizing URLs to them. CReWD proposes metrics that could be used as features for re-ranking search results by future IR systems as well as systems that do rule-based re-ranking. CReWD introduces a novel user-interest model and an algorithm for candidate selection to generate a list of users who could be potentially similar with this user. We evaluate the performance of CReWD with respect to an unranked baseline and show that CReWD significantly out-performs it and improves the Normalized Discounted Cumulative Gain (NDCG) of the search engine result pages displayed to the user.**

Furthermore in CReWD, we provide support for combining several approaches to ranking search results borrowed from machine learning, information retrieval theory and evaluation into a single unified framework. CReWD approaches the problem of re-ranking as several independent modules each generating their own ranks for the URLs and then proceeds to do Rank Aggregation on each of them. We envision CReWD as a system that can be used to rapidly prototype/combine and evaluate several such approaches to re-ranking of search engine results.

I. INTRODUCTION

Users query search engines for obtaining results to their queries. Each user has his own information needs. There is a growing need to understand user search behavior. An understanding on the same will enable us build better user-interfaces. For example, for a query like 'machine learning assignment', we want a student from Stanford to be shown the cs229 web page first in his list of re-ranked results. An analyst at an investment bank might want to look up the stock quote of Facebook instead of the Facebook home page for the query on 'Facebook'. One global re-ranking algorithm would not satisfy the information needs of every user ([1][2]). One limitation of conventional Information Retrieval models is that the retrieval decision is based solely on the query and the document corpus rather than an integrated approach that takes user context into account [4]. A single query gives the IR system only very limited information (or almost no information in some cases) about a user. Hence, one must use contextual data to enhance retrieval accuracy. Context-sensitive IR is among the major challenges in today's IR research[4].

Earlier models of IR (as those in [5]) were primarily based on explicit user feedback. They relied on the users to provide feedback on whether a search result was relevant to them and specify in advance the domains in which they were interested. Not all users are enthusiastic enough to provide feedback. So, the accuracy of such models is limited. Hence, implicit feedback (relying on the user's previous history and click-through data as a measure of how effective the IR system performed) has been studied exhaustively in [4]. Previous works have actively explored several sub-fields on this research area – log analysis, probabilistic query expansion [6], scaling personalization in search [7]. These primarily apply feedback to search results and not to re-ranking, which motivates CReWD.

The remainder of this paper is structured as follows. Section II introduces the problem that CReWD tries to address and why it is relevant or challenging. We then proceed to enumerate the key contributions of this paper. The subsequent sections address each of our contributions – a user model, a similarity-detection algorithm, and combining several approaches to re-ranking. We also then demonstrate our preliminary results on running CReWD and re-ranking on a dataset of 6 million users of a search engine.

II. CREWD

CReWD (Context Based ReRanker for Web Documents) addresses the problem of re-ranking an already ranked search result. CReWD relies on implicit feedback and proposes metrics which could be used by IR systems when re-ranking their results. While a lot of work has been done on implicit feedback (those in [4][6][3]), not much has been done in unifying several approaches to re-ranking and building a system that allows developers to effectively combine several of their re-ranking algorithms coherently. We implement 3 of our proposed models in CReWD and combine ranks across all of them. CReWD is addressing a challenging problem in IR as our dataset does not have the actual words or URLs themselves but only the IDs for words and URLs. So, applying NLP based methods for query refinement and processing, stemming, using knowledge bases or ontologies to detect

surrounding context, domains, re-querying or doing query expansion by querying the actual web-pages are not possible.

CReWD takes as input a user query (as a collection of term IDs without any information on the actual terms themselves) and an already ranked list of URL IDs. We assume the underlying search engine has already applied several features when ranking the URLs – including page rank and query analysis. The problem is – given this list of ranked URL IDs from a search engine, to apply the current user's previous history and the activity of all other users to re-rank these URLs.

The key contributions of this work are the following - the CReWD model considers both short-term history (in the form of the current session of the user) and long-term history (in the form of activity by the user over a longer period of time and the interaction of similar users). The metrics of relevance proposed in our model have broad applicability across several re-ranking systems (Section III-A). CReWD proposes a novel algorithm for discovering users that are similar to a user without exhaustively searching all candidate users (Section III-B). Finally, CReWD allows combining of several algorithms for re-ranking and generates an optimal re-ranked list from all of those results.(Section III-C)

III-A. A MODEL FOR RE-RANKING

Dwell time based personalization is currently accepted as the state-of-the-art in Information Retrieval research [10, CIKM - 2013]. Informally dwell time is the amount of time the user spent on a clicked search engine result page. The dataset we use (obtained from an ongoing contest on Kaggle) contains time-stamps at which each individual query was made, the results shown to the user and the user clicks.

A search engine log record R_i, is of this form,

$$< time_i, U_i, S_i, E_i >$$

where $time_i$ is the logical time-stamp (a monotonically increasing counter) when the event was recorded. U_i is the user for which the event happened. E_i is the event meta-data descriptor that identifies the type of the event. It is set to Q in case the event is a query event and to C if the event is a click event. A Q event also contains an event-descriptor that describes the terms in the query $< term_1, term_2 .., term_i >$ and a ranked list of URLs that the search engine displayed to the user, $< \mathrm{URL_1}: \mathrm{d_1}, \mathrm{URL_2}: \mathrm{d_2} .., \mathrm{URL_n}: \mathrm{d_n} >$ where d_i is the domain of URL_i. If the event meta-data descriptor is set to C, it means that the event is the click event and it specifies the $< URL_i: d_i >$ pair the user clicked on. If the event after the C event occurred at time epoch, $time_{i+1}$, then the dwell time of the record R_i calculated as $time_{i+1} - time_i$. We define our notion of $maxDwellTime$ of a user U_i for a session S_j for a URL URL_k as the maximum dwell time of the user on the URL in the session over all records corresponding to that session. The notion of $maxDwellTime$ is useful when we want to model users who click the same link more than once in a single session.

$maxDwellTime(U_i, S_j, URL_k) = \max_R(dwellTime(U_i, S_j, URL_k))$

where R is the set of all log-records in session S_j where user U_i clicked on URL_k.

The above formalism of a $maxDwellTime(U_i, S_j, URL_k)$ allows us to associate a per-session-per-URL relevance score for every user. The CReWD model of relevance captures 3 relevance scores – 0 (in which case the , URL_k was irrelevant to U_i. A relevance score of 1 which implies that, URL_k was irrelevant to U_i and a relevance score of 2 implies that, URL_k was highly relevant to U_i. The values of RelScores are assigned based on the $maxDwellTime$ metric we defined before. The choice of these values of *maxDwellTimes* were motivated by the evaluator at Kaggle which we use for our performance evaluation.

$$RelScore(URL_i, U_{j,} S_k) = \begin{cases} 0, if\ \max DwellTime(URL_i, U_{j,} S_k) \in (0{,}50) \\ 1, if\ maxDwellTime(URL_i, U_{j,} S_k) \in [50{,}400) \\ 2, if maxDwellTime(URL_i, U_{j,} S_k) \in [400, \infty) \end{cases}$$

The $RelScore(URL_i, U_{j,} S_k)$ is set to 2 if URL_i was the last URL that user $U_{j,}$ visited in session S_k. We argue that this assumption is reasonable, since we don't have another record in the session to record the user activity, we assume that the user stayed for a long time in the last link that he clicked in his session.

Let $L_{URL_i, U_{j,} S_s}$ be an indicator random variable defined as follows:

$$L_{URL_i, U_{j,} S_s} = \begin{cases} 1, if\ URL_i\ was\ shown\ to\ U_j\ in\ session\ S_s \\ 0, otherwise \end{cases}$$

Also define V_{URL_i, U_j} to be the indicator random variable as follows,

$$V_{URL_i, U_{j,}} = \begin{cases} 1, if\ URL_i\ was\ shown\ to\ U_j\ within\ the\ last\ 30\ days \\ 0, otherwise \end{cases}$$

Note that V_{URL_i, U_j} captures only the recent URLs shown to user U_j.

Armed with the above definitions we define a cumulative relevance score of a given URL for a particular user as.

$$CumulativeRel(URL_i, U_{j,}) = \frac{\sum_{s=1}^{|S|} (RelScore(URL_i, U_{j,} S_s))}{\sum_{s=1}^{|S|} 1\{L_{URL_i, U_{j,} S_s} = 1\}}$$

The Cumulative Relevance measure defined above captures how important an URL URL_i is to user $U_{j,}$ across all sessions.

Further, let us assume that users are clustered into various groups. The exact details of clustering and our notion of similarity of two users follows in Section-III-B.

Let U=$\{U_1 U_2 . U_j . U_m\}$ be the set of all users that are related to User U_j. Our formulation of the CumulativeRelevance measure allows us to derive an expression for the Global Relevance of a URL, URL_i with respect to the |U| other users in his cluster.

$$GlobalRelevance(URL_i U_v) = \frac{\sum_{u=1}^{|U|}(CumulativeRel(URL_i, U_{u,}))}{\sum_{u=1}^{|U|} 1\{V_{URL_i, U_{u,}} = 1\}} \; Similarity\_score(U_u, U_v)$$

Our model expresses the *GlobalRelevance* of a user as a weighted sum of the *CumulativeRel* scores of the related individual users in the cluster. The weights are used to denote how similar users U_u, U_v are. The *GlobalRelevance* function introduced above models the user's own global search history and the history of other related users in the same cluster as U_v.

We now, give an expression for the local relevance of a user for a URL. Since our definition of $RelScore(URL_i, U_{j,} S_k)$ captures the relevance of a URL w.r.t the user for a session, it can effectively be used to track the local relevance of a URL for a user for a session.

$$LocalRelevance(URL_i, U_j, S_s) = (RelScore(URL_i, U_{j,} S_s))$$

The expressions defined above help us to formally express our *TotalRelevance* as a weighted combination of the local and global relevance scores.

$$TotalRelevance(URL_i, U_j, S_s) = (\alpha) LocalRelevance(URL_i, U_j, S_s) + (1 - \alpha) GlobalRelevance(URL_i).$$

where the parameter α represents the weight applied to the local relevance (ie, the immediate session history for the user) and the Global relevance (that captures interactions of the user and all the related users in his cluster). For each URL in the unranked set of URLs, we compute this *TotalRelevance* score for the current session. We then order the original list of URLs from the search engine by this *TotalRelevance* score. The resulting permutation of the URLs is considered to be the re-ranked list of URLs generated by this step.

The power of CReWD lies in its ability to combine multiple algorithms (for example, we propose another algorithm in Section III –C for learning the *TotalRelevance* function and how CReWD combines the permutations generated by both the algorithms). This feature in CReWD helps other developers to test or combine several algorithms and measure the effectiveness of the ranking.

III-B. COMPUTING SIMILARITY ACROSS USERS

In this section we present our implementation of Similarity based clustering across various users and how users are grouped into clusters. We also propose an optimization involving candidate selection to speed up the clustering..

Computing similarity between two items represented as feature vectors is a widely researched area in statistics. For our notion of similarity we use the Pearson's coefficient, which is a correlation based approach.

User Interest Modeling

CReWD models every user as a vector of Cumulative Relevance scores of the urls they have clicked on. We compute the *CumulativeRelevance* vector V_u,for each user u and use this as an efficient representation of u. We formalize our definition of V_u as,

$$V_u[url] = CumulativeRel(url, u)$$

Since CReWD is engineered to be a web-scale system that processes millions of URLs, users and clicks and that each user could have only clicked a few of the millions of urls, V is implemented as a sparse matrix. Now, we can compute the correlation between users 'u' and 'v' with vectors V_u and V_v with means μ_{Vu} and μ_{Vv} respectively as follows:

$$SimilarityScore(u, v) \frac{\sum_{url}(V_u\,[\text{url}] - \mu_{Vu})(V_v\,[\text{url}] - \mu_{Vv})}{\sqrt{\sum_{url}(V_u\,[\text{url}] - \mu_{Vu})^2}\sqrt{\sum_{url}(V_v\,[\text{url}] - \mu_{Vv})^2}}$$

Conventional algorithms like correlation based clustering compute the similarity score across each pair of users u and v. However, at the scale at which CReWD is engineered to operate, such an approach with O(N^2) time complexity and O(N^2) space complexity clearly would take several days to cluster users. We propose a faster algorithm for candidate selection to find similar users.

A key realization to speeding up clustering by an order of magnitude is to realize that two users U_i and U_j are potential members of the same cluster C if and only if there exists some URL, URL_ksuch that $CumulativeRel(URL_k, U_{i,}) \neq 0$ and $CumulativeRel(URL_k, U_{j,}) \neq 0$. This indicates that both U_i and U_j found URL_k relevant. The existence of one such a URL_k indicates that Ui and Uj could be potential candidates for computing similarity.

For each URL, URL_k, CReWD maintains an index of a list of users who have clicked on the URL (this approach is scalable as since each click event is completely independent, this index could be computed in parallel by a pipeline of Map-Reduce jobs). Having obtained the index of a list of users who have clicked on each URL, CReWD scans each URL in the list. Let S=$\{U_1, U_2 .., U_i, .. U_n\}$ be the list of users who have clicked on a URL, URL_k. Let C_i of each user U_i be the set of candidate users similar to U_i. For each pair of users U_i, U_j in S, add U_i to the candidate set of U_j. Since in practice, S is expected to be small, we expect this approach to perform better. Our algorithm is efficient in time as only $|C_i|$ users are examined

for potentially similar candidates instead of running an exhaustive search over all the N users.

For each user, U_i CReWD computes the top 'k' users who have them highest similarity measure with U_i by only examining $|C_i|$ potential candidates. Thus, we derive the list of users similar to U_i.

A further optimization can be made here. Some links are visited by a lot of users and on a daily basis. Since, a lot of people visit such links, these URLs do not contribute any value to the clustering. Hence, we neglect those URLs having more than a threshold number of users visiting them.

III-C. LEARNING A URL RELEVANCE FUNCTION FOR PERFORMING RERANKING

In this section, we describe a supervised learning based approach to learn a relevance function for a (URL, User) pair. The next section combines the ranks generated from this approach and the ranks generated from III-A, using rank aggregation.

We cast ranking as a supervised learning problem of learning a relevance function of a URL u, as a weighted sum of the individual feature values using a classifier.

$$TotalRelscore(url,u) = \sum_{i=1}^{num\_features} feature_i W_i$$

Feature Selection

CReWD uses a mix of dwell time information, number of clicks at a session level and cluster level (note that we cluster user from our algorithm in Section-III-B) as features. We list the features and an intuition for having each of them.

- **# clicks by User u on the URL:** Models a click event on the same URL before. A common use-case is people forget the full URL of a site and instead, frame a query to a search engine to get a SERP and clicking on their desired URL. For example, a cs229 student visits the assignments page multiple times and might use a search engine to find it.
- **# clicks by User u on the domain:** Captures if a user had clicked on the same domain before. A common use case is a stock analyst who is interested in finance.yahoo.com/facebook and not in facebook.com, when he searches for facebook. Similar domains visited before are an indication that they should be ranked higher in the search result.
- **Average Dwell time of the User u on the domain:** This feature models the time that the user usually spends on the domain of the URL. We want URLs with higher dwell times to be ranked higher.
- **Average Dwell time of all similar users on the URL domain.**
- **Average Dwell time of all similar users on the URL:** The above two capture a global similarity across related users.
- **URL clicked on the same session:** Session level feature to model user-interaction. (Also captures immediate local similarity)

Here similar users refers to the list of users who are in the same cluster as 'u' as determined from the previous step. We performed supervised learning on our dataset to learn whether a document was relevant or not given the above features. The coefficients of the hyperplane resulting for classification gives us the individual weights of the features which can then be used to calculate TotalRelscore.

The training data was generated by scanning through the search engine logs and extracting the above features. URLs were given a classification label of either Relevant or Irrelevant based on the dwell time of the user on that URL in a particular session. URLs not clicked on during the session were considered irrelevant.

C was performed using the **scikit-learn** library provided in Python. We used a linear SVM performing L2-regularization for classification. We mention our results in a later section.

III-D COMBINING SEVERAL RANK PERMUTATIONS

In this section, we describe how several permutations generated in previous steps (III-A and III-C) can be combined together. Rank Aggregation is a method of combining several rank outputs resulting from previous stages. The idea of Rank Aggregation is that we can consider the output from each independent re-ranking algorithm as a signal and output a permutation closest to all these permutations under some metric. CReWD implements a simple Rank Aggregation algorithm called the Borda ranking method.

Assume that there are 'n' ranking algorithms each with their own permutation of URLs each of the permutation of size k. Let S= $\{URL_1, URL_2 .., URL_i, .. URL_k\}$;|S|=k; be the unranked permutation and let $P_1, P_2, . P_n$ be the 'n' permutations of S.

We update Rank_Score of an URL per permutation P_i as follows, If an URL appears in rank 'r' in any permutation then we set rank_score[url] to k-r.

Similarly, we estimate scores of each of the URLs. We then order the URLs by the decreasing order of their scores and generate a new permutation. The generated permutation gives us an ordering which is the combined rank of the 'n' individual permutations.

EVALUATION AND RESULTS

Below are the feature weights learned by our linear classifier after training on 10 million log records. The weights we obtained for the following features were as follows: (classification accuracy was 96%)

No. of clicks by user on URL (f1)	0.77
No. of clicks by user on domain (f2)	0.06
Average dwell time on URL (f3)	3.65

A record was found to be relevant iff **0.77 * f1 + 0.06 * f2 + 3.65 * f3 – 3.27 >= 0**

Our results suggest that average dwell time is a much stronger metric for relevance determination followed by number of clicks on the URL. Surprisingly, number of clicks on the domain does not seem to be as important.

We evaluated the performance of our supervised learning algorithm combined with rank aggregation and clustering on a Kaggle dataset consisting of 70 million unique URLs, 64 million clicks, 21 million unique Queries, 0.5 Million unique users, 34.5 Million training sessions and 0.79 Million test sessions. (Our experiments were carried out on the madmax machines in the Infolab with Intel Xeon processors, 2.4GHz, 1 TB RAM, 80 virtual cores).

The metric used by Kaggle for evaluation was the Normalized Discounted Cumulative Gain (NDCG), a common metric used for search engine result evaluation[3]. In the DCG method the relevance contribution of a document is decreased by an amount proportional to the log of the position of the result in the page. The measure allows for more grades of relevance as opposed to a binary grade – as relevant or not.
The DCG of a particular re-ranked test sample is computed using the formula,

$$DCG_p = \sum_{i=0}^{p} \frac{2^{Rel(URL_i)} - 1}{\log(i+1)}$$

ALGORITHM	NDCG
Random Baseline	0.47972
Default Ranking Baseline	0.79056
Rule Based Classifier with Local History α=0.75	0.79125
Supervised Learning With Clustering and Rank Aggregation **(CReWD)**	**0.79457**
Top position in Kaggle Leaderboard	0.80278

We evaluate our NDCG performance against a default ranking baseline and a random baseline. The default baseline does not do any re-ranking. Instead it relies just on features at the query level and features like page-rank to do ranking. For the default ranking baseline, the re-ranked list is the same as the ranked list. The random baseline returns an arbitrarily random permutation of an URL. The top NDCG in the Kaggle leaderboard is the maximum NDCG obtained by any contestant. We implement Borda's Rank Aggregation (implemented by combining the re-ranking algorithms in Section IIIA and Section IIIC), with different values of α for tuning local and global history weights.

We see that the NDCG obtained by our algorithm (**0.79457**) is not that far from the top ranked NDCG (**0.802**).

Due to memory constraints, we could not perform all pairs correlation and find the users most similar to a given user, we sampled 10^5 users randomly for each user and chose the top 100 while calculating correlation. After clustering, the number of URLs we could re-rank went up from **5.3%** to **14.7%** since we had more information about a user's preferences. However, we expect that our re-ranked results could have been more relevant had we been able to perform an all pairs correlation computation to find the most similar users for every user.

CONCLUSION

We described CReWD a system for reranking ranked search engine result pages. We also introduced novel metrics (Section III A) that could be leveraged by modern IR systems or machine learning systems that perform ranking. We also proposed an optimization for candidate selection to avoid an exhaustive while finding similar users. We finally described how CReWD performs rank aggregation on both our supervised and unsupervised approaches. We see that our rank-aggregated result gives a higher NDCG than rule based learning.

FUTURE WORK

It would be interesting to perform an all pairs correlation computation between users. We are also curious about how other rank aggregation schemes such as those based on inversion distance would perform. We may implement some of these approaches in the Winter break to see if our NDCG increases further.

REFERENCES

1.Hongning Wang, Xiaodong He, Ming-Wei Chang, Yang Song, Ryen W. White, and Wei Chu. 2013. Personalized ranking model adaptation for web search. In Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval (SIGIR '13).

2.R. Fidel and M. Crandall. Users' perception of the performance of a Filtering system. In SIGIR'97, volume 31, pages 198–205.ACM, 1997.

3.Belkin, N.J., Ingwersen, P. and Leong, M.-K , IR evaluation methods for retrieving highly relevant documents,. (eds.) Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, NY: ACM, pp. 41–4

4. X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In SIGIR `05, pages 43-50, 2005.

5. J. Rocchio. Relevance feedback information retrieval. In The Smart Retrieval System-Experiments in Automatic Document Processing, pages 313–323, Kansas City, MO, 1971.Prentice-Hall

6.H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma. Probabilistic query expansion using query logs. In Proceedings of WWW 2002, 2002

7. G. Jeh and J. Widom. Scaling personalized web search. In Proceeding of WWW 2003, 2003