Computing Information Retrieval Performance Measures Efficiently in the Presence of Tied Scores

Frank McSherry and Marc Najork

Microsoft Research, Mountain View CA 94043, USA

Abstract. The Information Retrieval community uses a variety of performance measures to evaluate the effectiveness of scoring functions. In this paper, we show how to adapt six popular measures — precision, recall, F1, average precision, reciprocal rank, and normalized discounted cumulative gain — to cope with scoring functions that are likely to assign many tied scores to the results of a search. Tied scores impose only a partial ordering on the results, meaning that there are multiple possible orderings of the result set, each one performing differently. One approach to cope with ties would be to average the performance values across all possible result orderings; but unfortunately, generating result permutations requires super-exponential time. The approach presented in this paper computes precisely the same performance value as the approach of averaging over all permutations, but does so as efficiently as the original, tie-oblivious measures.

1 Introduction

One of the fundamental problems in Information Retrieval is the ranking problem: Ordering the results of a query such that the most relevant results show up first. Ranking algorithms employ scoring functions that assign scores to each result of a query, where the score is an estimate of the result's relevance to the query at hand. So, ranking the results of a query consists of assigning a score to each result and then sorting the results by score, from highest to lowest.

Ranking algorithms are typically evaluated against a test collection consisting of a set of queries. Each query in the test collection has a set of results, and the results were arranged into a (partial or total) order by a human judge. In order to evaluate a ranking algorithm, the algorithm is applied to the result set of each query, the distance between the computed ranking and the "optimal" ordering determined by the judge is measured, and the distances are averaged over the entire test collection. Coming up with suitable distance metrics (or performance measures) has been the subject of considerable research, and there are numerous such metrics.

Typically these performance measures assume that a ranking algorithm arranges the results of a query into a total ordering, *i.e.* that no two results to a query have the same score. This assumption is reasonable for scoring functions that map a rich set of features of the result document to a real-valued score, but

C. Macdonald et al. (Eds.): ECIR 2008, LNCS 4956, pp. 414–421, 2008.

it is less warranted for evaluating the performance of a single discrete feature, *e.g.* page in-degree, click count, and page visits.

We stress that we are not concerned with evaluating the final results of a ranking system, which will almost certainly combine many features into a real-valued score, but rather the internals of such a system, where the performance of individual, potentially discrete features needs to be assessed. A typical modern search engine will use hundreds of features and combine the evidence provided by these features using *e.g.* a neural network. An important first step in training is feature selection, to restrict the number of inputs and avoid overfitting. A natural approach is to treat each feature as a scoring function in its own right and assess its performance under various IR metrics.

Despite the fact that performance evaluation of IR systems is a mature field and that much thought has gone into devising appropriate measures to compare different ranking systems, not much work has been done on adapting these measures to evaluate the performance of ranking algorithms that impose only a partial ordering on the result set. The impact of ties in ranked result sets on measures of retrieval effectiveness was first considered by Cooper [2], who proposed *expected search length* as a performance measure robust to ties. Raghavan and Jung [5] investigated the problem of ties in the context of precision and recall. Their focus is on precision at varying levels of recall, and they develop approaches that are sensible in the presence of ties. In contrast, our approaches focus on a larger space of evaluation measures, including F1, RR, AP, and NDCG, but are aimed at settings with fixed document cut-off values.

The simplest approach to dealing with the problem (which is in fact the approach that was taken by TREC competitions; see for example [4]), is to arbitrarily pick one of the valid (that is, well-sorted) orderings of a result vector and evaluate it. However, in our own experiments on large-scale test sets for web collections, and using scoring functions prone to produce tied results, we found that different well-sorted permutations of result vectors can have appreciably different performance values, large enough to affect the relative ordering of several of the scoring functions we compared.

A more disciplined approach is to average over all possible orderings. A naive realization of this approach to dealing with tied scores would entail generating the possible orderings of a result set by generating all permutations of each subset of tied results. This approach, while straightforward to implement, is computationally very expensive; its time complexity is super-exponential to the number of tied results. This is especially troublesome in cases where the distance metric is the cost function of an optimization algorithm, such as a dynamic optimization program to determine optimal parameters of a scoring function.

In this paper, we show how to compute six of the most popular performance measures efficiently in the presence of tied scores. Our approach is arguably superior to the two aforementioned approaches: It is completely deterministic just like the second approach (and in fact produces precisely the same results), and at the same time not substantially more expense to compute than the first approach of computing the traditional measures for a single permutation. We

have implemented all of the tie-aware measures described, and are routinely using them as cost functions (*i.e.* the inner loop) of dynamic optimization systems.

The remainder of the paper is structured as follows: section 2 adapts six well-established (and tie-oblivious) performance measures to handle result vectors with tied scores in a robust and efficient manner. In section 3 we assess the performance impact of tie-awareness. Finally, section 4 offers concluding remarks.

2 Dealing with Tied Scores

We will now show how six standard performance measures can be adapted to deal with ties in a robust, deterministic matter, while still being about as efficient as the standard tie-oblivious definitions. The approach we take is to consider all possible consistent orderings, but rather than explicitly producing these orderings we analytically derive their average, which in each case we can easily compute.

We will first introduce some mathematical notation to help us describe performance measures, and then develop the tie-aware versions of these measures.

A typical ranking algorithm applies a *scoring function* s to all result documents retrieved in response to a query q, and sorts the results by decreasing score. This produces a result vector $V = \langle v_1, \cdots, v_n \rangle$, where $s(v_i) \geq s(v_{i+1})$ for all $1 \leq i < n$. Document v_1 is the highest-scoring result.

Scoring functions are evaluated using test collections of queries and associated result sets, where the results have been labeled by human judges as to their relevance. Labels can be binary (*e.g. relevant* or *irrelevant*), drawn from a small range (*e.g. excellent*, *good*, *fair*, *bad*), or fine-grained enough to impose a total ordering on the results (*e.g. best*, *second-best*, *etc.*). Five of the six performance measures described in this paper assume that the judges have used a binary labeling scheme, *i.e.* have marked the results in the test collection as relevant or irrelevant to their associated query. We write $rel(v_i)$ to denote the relevance of document v_i in a result set; $rel(v_i)$ is 1 if v_i is relevant to the query and 0 otherwise. In order to evaluate the performance of a scoring function s, we iterate over the query/result-set pairs in a test collection, use s to rank the results associated with each query, use a performance measure to quantify how much the ranking imposed by the scoring function diverges from the assessment of the human judges, and average these quantities over all queries in the test collection. This approach is often called *macro-averaging*.

2.1 Precision

The *precision* measure [1] is based on the observation that users of an IR system tend to peruse only the first k results of a search. It measures what fraction of these k results are relevant to the query on average. The value k is commonly called the *document cut-off value*. For the rest of the paper we assume that $k \leq n$. The precision at k is defined as:

$$P@k(V) = \frac{1}{k} \sum_{i=1}^{k} rel(v_i)$$

In order to deal with ties, we introduce a *tie-vector* $T = \langle t_1, \cdots, t_{m+1} \rangle$ whose first element t_1 is 0 and whose remaining elements are the ending indices of the m equivalence classes in V, so that $V_i = \langle v_{t_i+1}, \cdots, v_{t_{i+1}} \rangle$ all have the same score. These classes need not have more than a single element, as in the case that there is not a tie. We use the notations r_i and n_i to reference the number of relevant and total elements in V_i, respectively. We use R_i to denote the number of relevant elements that precede V_i in V. Furthermore, we assume *w.l.o.g.* that the document cut-off occurs in sub-vector V_c, or putting it differently, that k is in the half-open interval $(t_c, t_{c+1}]$.

In order to compute the precision at k in the presence of ties, we sum up the expected number of relevant results contained in each sub-vector V_i of tied results. For any sub-vector preceding V_c (the sub-vector where the cut-off occurs), the contribution is exactly the same as in the tie-oblivious case, since any permutation of tied results does not change the number of relevant results in the sub-vector. For any sub-vector succeeding V_c, the contribution is 0, since per definition none of its results fall below the document cut-off. Finally, the sub-vector V_c contains n_c results, r_c of which are relevant, and it has $k - t_c$ "slots" within the document cut-off window. So, V_c contains on average $(k - t_c)\frac{r_c}{n_c}$ relevant results. This leads to the following tie-aware definition of precision at k:

$$P@k(V) = \frac{1}{k}\left(R_c + \frac{k - t_c}{n_c} r_c\right) = \frac{1}{k}\left(\sum_{i=1}^{t_c} rel(v_i) + \frac{k - t_c}{n_c} \sum_{i=t_c+1}^{t_{c+1}} rel(i)\right)$$

The time complexity for computing $P@k(V)$ is $O(k)$ in the tie-oblivious case, since we only need to examine the first k elements of the result vector V, and $O(t_{c+1})$ in the tie-aware case, since we need to scan the result vector up to and including the sub-vector V_c overlapping the document cut-off specified by k.

2.2 Recall

The *recall* measure [1] quantifies what fraction of all the relevant results was ranked to fall within the first k documents. The recall at k is defined as:

$$R@k(V) = \frac{\sum_{i=1}^{k} rel(v_i)}{\sum_{i=1}^{n} rel(v_i)}$$

Recall can be adapted to results with tied scores in much the same way as precision. Again, we sum up the contributions of each sub-vector V_i of tied results, *i.e.* the expected number of relevant results contained in each sub-vector. But while precision normalizes this sum by the document cut-off value k, recall normalizes it by the total number of relevant results in the entire result vector. Neither normalization factor is influenced by permutations of results with tied scores. So, we arrive at the following tie-aware definition of recall at k:

$$R@k(V) = \frac{R_c + (k - t_c)\frac{r_c}{n_c}}{\sum_{i=1}^{n} rel(v_i)} = \frac{\sum_{i=1}^{t_c} rel(v_i) + \frac{k-t_c}{n_c} \sum_{i=t_c+1}^{t_{c+1}} rel(i)}{\sum_{i=1}^{n} rel(v_i)}$$

The time complexity for computing $R@k(V)$ is $O(n)$ in both cases, since we need to iterate over the entire result vector V to find out how many relevant results there are in total. If the number of relevant results in V is known, $R@k(V)$ can be computed in $O(k)$ time in the tie-oblivious case and $O(t_{c+1})$ in the tie-aware case.

2.3 F1 Measure

A common combination of precision and recall is the F1 measure, defined as the harmonic mean of precision and recall, which can be rewritten so as to avoid division by zero in the absence of any relevant results:

$$F1@k(V) = \frac{2}{\frac{1}{P@k(V)} + \frac{1}{R@k(V)}} = \frac{2\sum_{i=1}^{k} rel(v_i)}{k + \sum_{i=1}^{n} rel(v_i)}$$

The denominator of the rewritten equation is independent of ties in V, and the numerator is exactly as we have seen in both precision and recall. We thus adapt F1 similarly, replacing $\sum_{i=1}^{k} rel(v_i)$ with $R_c + (k - t_c)\frac{r_c}{n_c}$, the average number of relevant documents over all possible ties.

$$F1@k(V) = \frac{2(R_c + (k - t_c)\frac{r_c}{n_c})}{k + \sum_{i=1}^{n} rel(v_i)}$$

The time complexity of computing the F1 measure is no more than precision or recall in either case; the numerator can be computed in $O(k)$ or $O(t_{c+1})$ time, and the denominator, if unavailable, can be computed in $O(n)$ time.

2.4 Average Precision

The *average precision* measure computes a precision for every relevant result in a result vector, and averages these precision values. More precisely, the *average precision* at k is defined to be:

$$AP@k(V) = \frac{\sum_{i=1}^{k} P@i(V) rel(v_i)}{\sum_{i=1}^{n} rel(v_i)} .$$

The numerator is the only quantity that has the opportunity to vary based on the choice of ordering given ties. To simplify presentation, when considering a position j, we will bind i to be the index of the tie that contains j; *i.e.* the value of i such that $t_i < j \leq t_{i+1}$. To analyze the average contribution of a position j in a tie V_i, we note that position j is relevant in a $\frac{r_i}{n_i}$ fraction of orderings. When an element is relevant, the average number of relevant documents preceding it in the tie is $\frac{r_i - 1}{n_i - 1}$ times the number of available slots in the tie, $j - t_i - 1$. We thus define the tie-aware variant of AP as:

$$AP@k(V) = \frac{\sum_{j=1}^{k} \frac{r_i}{n_i} \left(R_i + (j - t_i - 1)\frac{r_i - 1}{n_i - 1} + 1) \right) \frac{1}{j}}{\sum_{j=1}^{n} rel(v_j)}$$

Computing $P@i(V)$ and R_i for increasing values of i does not require two nested loops, but can be done in a single loop running from 1 to k. Hence, the time complexity to compute $AP@k(V)$ is $O(k)$ in the tie-oblivious case and $O(t_{c+1})$ in the tie-aware case, since all of the terms in the above formulas can be evaluated in a single pass through the first k or t_{c+1} elements.

2.5 Reciprocal Rank

The *reciprocal rank* measure [6] favors scoring functions that rank relevant results highly. The value of the measure is inversely proportional to how far a user has to go down the ranked list of results on average to find the first relevant result:

$$RR@k(V) = \begin{cases} \frac{1}{i} & \text{if } \exists i \leq k : rel(v_i) = 1 \wedge \forall j < i : rel(v_j) = 0 \\ 0 & \text{otherwise} \end{cases}$$

The tie-aware variant requires attention only in the case where the first relevant result in the partial order occurs in a tie with at least one other object. In this case, we must determine the average value of $\frac{1}{i}$ for the first relevant result in that set of ties. Additionally, if this set of tied results crosses the imposed document cut-off value of k, we must consider the possibility that all relevant results are ranked beyond k, yielding no score at all.

To compute the tie-aware reciprocal rank, we first identify the first group V_i containing a relevant result. For each of the values j from t_i+1 up to $\min(t_{i+1}, k)$, we compute the fraction of orderings in which the first relevant result occurs at exactly that position. Multiplying this fraction by $\frac{1}{j}$ and accumulating over j gives the correct answer.

We compute the fraction of orderings with the first relevant result at position t_i+x by computing for each t_i+x the fraction of orderings whose first x elements are irrelevant, and then computing the difference between adjacent fractions. Taking those orderings whose first x elements are relevant, minus those whose first $x+1$ elements are irrelevant, gives the fraction whose first relevant element is at x. The fraction $f(x, r, n)$ of the orderings of r out of n relevant elements for which the first x are irrelevant follows as simple recursive definition:

$$f(x, r, n) = \begin{cases} 1 - \frac{r}{n} & \text{if } x = 1 \\ (1 - \frac{r}{n-x+1}) f(x-1, r, n) & \text{otherwise} \end{cases}$$

Intuitively, each ordering that contributes to $f(x-1, r, n)$ will contribute to $f(x, r, n)$ if the next element is irrelevant, which occurs when none of the r relevant results are chosen from the set of $n - x + 1$ remaining results.

Letting V_i be the first group containing a relevant result,

$$RR@k(V) = \sum_{j=t_i+1}^{\min(t_{i+1},k)} \frac{f(j - t_i, r_i, n_i)}{j}$$

The time complexity for computing $RR@k(V)$ is $O(k)$ in the tie-oblivious case, since it requires a linear scan of at most the first k elements of the result

vector V, and at most $O(t_{c+1})$ in the tie-aware case, as we need only scan as far as the end of the last possible tied group. Once the first relevant group V_i is identified, the dynamic program takes $O(n_i)$ time to compute the fractions and accumulate the weighted reciprocals, which is at most $O(t_{c+1})$.

2.6 Normalized Discounted Cumulative Gain

The *discounted cumulative gain* measure [3] assumes that judges have assigned labels to each result, and accumulates across the result vector a gain function G applied to the label of each result, scaled by a discount function D of the rank of the result. A common example uses integer labels, the gain function $G(l) = 2^l - 1$, and discount funtion $D(i) = \frac{1}{\log(1+i)}$. We define the *discounted cumulative gain* at k as follows:

$$DCG@k(V) = \sum_{i=1}^{k} G(label(v_i))D(i)$$

We normalize $DCG@k(V)$ into the range [0,1] by dividing by the DCV of an "ideal" result vector I (produced by a hypothetical clairvoyant scoring function that maximizes $DCG@k(I)$):

$$NDCG@k(V) = \frac{DCG@k(V)}{DCG@k(I)}$$

NDCG can be adapted fairly easily to deal with ties, as the normalization requires no special attention, and discounted cumulative gain is a simple sum over the returned results. Notice that for each position in a tied group, the average gain at that position is the average of the gain function across tied elements. As the discount function is multiplicative, we need only multiply it by this average gain at each position:

$$DCG@k(V) = \sum_{i=1}^{m} \left(\left(\frac{1}{n_i} \sum_{j=t_i+1}^{t_{i+1}} G(label(v_j)) \right) \sum_{j=t_i+1}^{\min(t_{i+1},k)} D(j) \right)$$

As mentioned above, the normalization step is independent of ties in a candidate partial ordering, and is computed and applied as usual.

Computing $NDCG@k(V)$ involves computing $DCG@k(V)$ and computing $DCG@k(I)$. Computing $DCG@k(V)$ takes $O(k)$ time in the tie-oblivious case and $O(t_{c+1})$ in the tie-aware case, as the average gain can be computed for each V_i in time $O(n_i)$, with the discounted gain accumulated for that group in a similar amount of time. As $DCG@k(()I)$ is independent of the ordering of V, its computation time is unaffected.

3 Implementation

We have built C# implementations[1] of both tie-oblivious and tie-aware versions of the performance measures described in this paper. We compared the wall-clock

[1] Available at http://research.microsoft.com/research/sv/tie-aware-measures

running times of the tie-oblivious and tie-aware variants of each method. In order to account for the effects of disk latency, we first loaded the vector of ranks and scores for a test set of 28,043 queries into main memory. The scoring function used was the in-degree of the web page. We found that for most of the performance measures the overhead is negligible. The exception is reciprocal rank, where the overhead is roughly 25%. Reciprocal rank is the only measure we considered that can be computed without sorting the result vectors; consequently, the overhead of the tie-awareness is much more noticeable, as it is not drowned out by the cost of sorting.

4 Conclusions

This paper addressed the issue of defining deterministic performance measures for scoring functions that are prone to assign identical scores to many results in a result set. Our approach is inspired by the idea of evaluating the performance of all possible well-ordered permutations of the result set and averaging the performances, but it avoids the factorial time complexity that would go along with such an approach, despite the fact that it produces precisely the same performance values as averaging over all well-ordered permutations does. We have applied our approach to six well-established performance measures: recall, precision, F1, average precision, reciprocal rank, and normalized discounted cumulative again. For these six measures, computing the tie-aware measures is not appreciably slower than computing the standard, tie-oblivious performance measures.

References

1. Cleverdon, C.W., Mills, J.: The testing of index language devices. Aslib Proceedings 15(4), 106–130 (1963)
2. Cooper, W.: Expected Search Length: A Single Measure of Retrieval Effectiveness Based on the Weak Ordering Action of Retrieval Systems. American Documentation 19(1), 30–41 (1968)
3. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems 20(4), 422–446 (2002)
4. National Institute of Standards and Technology. TREC 2005 Robust Track Guidelines (2005), `http://trec.nist.gov/data/robust/05/05.guidelines.html`
5. Raghavan, V., Jung, G.: A Critical Investigation of Recall and Precision as Measures of Retrieval System Performance. ACM Transactions on Information Systems 7(3), 205–229 (1989)
6. Voorhees, E.M., Harman, D.K.: TREC: Experiment and Evaluation in Information Retrieval. MIT Press, Cambridge (2005)