Distributed Ranking Under Temporal Constraints

Aman Sawhney

***Abstract*—Ranking is the central problem for information retrieval. "Ranking documents by their relevance to queries is a common way for search engines to return retrieval results to users. In doing so, a single relevance score must be calculated based on all the measurable evidence. In traditional informational retrieval models (e.g. vector space model), relevance is represented by the similarity metric between a document and a query. The assumptions underlying these models are that (1) the content of a document can be represented by a vector of concept-bearing terms.."**

"This paper introduces the notion of temporally constrained ranked retrieval, which, given a query and a time constraint, produces the best possible ranked list within the specified time limit. Naturally, more time should translate into better results, but the ranking algorithm should always produce some results. This property is desirable from a number of perspectives: to cope with diverse users and information needs, as well as to better manage system load and variance in query execution times. We propose two temporally constrained ranking algorithms based on a class of probabilistic prediction models that can naturally incorporate efficiency constraints: one that makes independent feature selection decisions, and the other that makes joint feature selection decisions. Experiments on three different test collections show that both ranking algorithms are able to satisfy imposed time constraints, although the joint model outperforms the independent model in being able to deliver more effective results, especially under tight time constraints, due to its ability to capture feature dependencies."

Keywords-

I. Introduction

Our Contribution.

Related Work. [1]

Organization.

II. Wang et. al Indep model

Let q be the query placed by a user, d be the set of documents in the index, F be the feature set, where

$$F = \{f_1,\ f_2,\ \ldots,\ f_n\}$$

then

$$score(q,\ d)\ = \sum_i (\lambda_i . f_i(q,\ d))$$

where λ_i is the weight associated with feature f_i
The features f_i can be either term features(unigram) or term proximity features(bigrams).

If the model is temporally constrained with a limit of $T(q)$, i.e. the time associated with the query q then

$$score(q,\ d)\ = \sum_{f_i(q,\ .):\ S_i\ =1} (\lambda_i . f_i(q,\ d))$$

s.t.

$$\sum_{f_i(q,\ .):\ S_i\ =1} C(f_i(q,\ .))\ <= T(q)$$

Here, $C(f_i(q))$ is the cost associated with feature i. $S_i\ = 1$ implies that the feature i is selected.

$$\hat{S} = \ argmax_S\ \sum_i S_i\ .\ \lambda_i(q)$$

s.t.

$$\sum_i S_i\ C(f_i(q,\ .))\ <= T(q), \quad S_i\ \in \{0,\ 1\}$$

Cost is computed by the document frequency that the query generates for the a particular feature.

So, in this model there is a profit(λ) associated with the selection of a feature, a weight($C(f)$) that has to be borne and a total weight constraint(time constraint $T(q)$). Hence, the problem can viewed as a classic *knapsack problem*. The authors solve the problem using a greedy approach: sort the features in decreasing ratio of profit to weight density, add features one by one until you exceed the time constraint or run out of of features.

III. Problem

In this section, we model the problem of temporally constrained ranking with multiple processors, where there is a time constraint $T(q)$ for a query q.

Linear ranking functions are a class of effective ranking models used for information retrieval. A linear ranking function is a characterized by a set of features $F = \{f_1, \ldots, f_N\}$ and their corresponding model parameters $\Lambda = \{\lambda_1, \ldots, \lambda_N\}$, where each feature f_i is a function mapping a query-document pair (q, d) to a real value. The relevance of document d to query q, $score(q, d)$, is defined as a weighted linear combination of the feature values computed over the document and the query. As a result, $score(q, d)$ is defined as follows:

$$score(q, d) = \sum_{i=1}^{N} \lambda_i(q) f_i(q, d) \qquad (1)$$

where $\lambda_i(q)$ is calculated based on meta-features defined over query q for each feature i, and other

Since using all features in the linear ranking function may exceed the time constraint, we need to change the efficiency characterises of the ranking function by choosing only a subset of the features in order to meet its time requirement. The resulting linear ranking function is a temporally constrained linear ranking function.

The problem can be formulated as an integer program, as follows:

$$\text{Maximize } score(q,d) = \sum_{i=1}^{N}\sum_{p=1}^{P} \lambda_i(q) \cdot x_{ip} \tag{2}$$

Subject to:

$$\sum_{p=1}^{P} x_{ip} \leq 1, \forall i = 1, \ldots, N \tag{3}$$

$$\sum_{i=1}^{N} w_i C(f_i(q,\cdot)) x_{ip} \leq T(q), \forall p = 1, \ldots, P \tag{4}$$

$$x_{ip} \in \{0,1\}, \forall i = 1, \ldots, N, \forall p = 1, \ldots, P \tag{5}$$

The decision variables x_{ip} are defined as follows:

$$x_{ip} = \begin{cases} 1 & \text{if } f_i \text{ is allocated to } p, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

where P is the number of computers and N is the number of features.

The objective function is to maximize the total score of the selected features. Constraints (3) ensure that each feature is selected at most once. Constraints (4) guarantee that the computational cost of selected features for each processor does not exceed the time requirement for any processor. Constraints (5) represent the integrality requirements for the decision variables.

Notice that the time constraint for all P machines is the same ($T(q)$) because these can all be run in parallel.
This can be modeled in a similar greedy manner.

A. homogeneous machines

Murthy et. al [2] provide an algorithm that solves the multiple knapsack problem in a distributed manner. An adaptation of that algorithm for our problem follows.

Algorithm 1 Greedy Distributed (homogeneous machines) algorithm with $T(q)$ time constraint for each machine

1: $ItemList \leftarrow ItemList.SortDecreasingBy(\frac{\lambda(f_i,.)}{C(f_i,.)})$
2: i=0
3: $\forall j, r_j = T(q)$
For the source p_s
4: **for** feature f_i; i=1 to n **do**
5: Broadcast $\langle C(f_i,.)\rangle$ to all p_j
6: Recieve$\langle j\rangle$
7: Assign i to j
8: **end for**
For the processor p_j**:**
9: Receive $\langle C(f_i,.)\rangle$ from p_s
10: **if** $r_j >= C(f_i)$ **then**
11: Broadcast $\langle j,\ r_j\rangle$ to all p'_j
12: **else**
13: Broadcast $\langle j,\ \perp\rangle$ to all p'_j
14: **end if**
15: Receive $\langle j, r_j\rangle\ from\ all\ p'_j$ best = $argmax'_j(r'_j)$ from p_s
16: **if** best = j **then**
17: Send $\langle j\rangle$ to all p_s
18: $r_j \leftarrow r_j - C(f_i,.)$
19: **end if**
procedure Final():
20: **for** j =1 to m **do**
21: Pick $max(p_j, max_{i:C(f_i)<=T(q)}(\lambda_i))$
22: **end for**

B. heterogeneous machines

Min-Min and Max-Min heuristics

REFERENCES

[1] L. Wang, D. Metzler, and J. Lin, "Ranking under temporal constraints," in *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM, 2010, pp. 79–88.

[2] A. Murthy, C. Yeshwanth, and S. Rao, "Distributed approximation algorithms for the multiple knapsack problem," *arXiv preprint arXiv:1702.00787*, 2017.