

# A Note on Search based Forecasting of Ad Volume in Contextual Advertising

Xuerui Wang  
University of Massachusetts  
Amherst, MA, USA  
xuerui@cs.umass.edu

Marcus Fontoura  
PUC-Rio  
Rio de Janeiro, Brazil  
mfontoura@inf.puc-rio.br

Andrei Broder  
Yahoo! Research  
Santa Clara, CA, USA  
broder@yahoo-inc.com

Vanja Josifovski  
Yahoo! Research  
Santa Clara, CA, USA  
vanjaj@yahoo-inc.com

## ABSTRACT

In contextual advertising, estimating the number of *impressions* of an ad is critical in planning and budgeting advertising campaigns. However, producing this forecast, even within large margins of error, is quite challenging. We attack this problem by simulating the presence of a given ad with its associated bid over historical data, involving billions of impressions. This apparently enormous computational task is reduced to a search task involving only the set of distinct pages in the data. Furthermore the search is made more efficient using a two-level search process. Experimental results show that our approach can accurately forecast the expected number of impressions of contextual ads in real time.

## Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—*data mining*; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

## General Terms

Algorithms, Experimentation, Measurement, Performance

## 1. INTRODUCTION

Web advertising has become a major industry. *Sponsored Search* (SS), which displays ads on the result pages from a search engine, and *Contextual Advertising* (CA), which places ads within the content of a Web page, have been the main advertising channels used to distribute online ads. Different from traditional media, the prevalent pricing model for online ads is that the advertisers pay a certain amount for every click on the advertisement (pay-per-click or PPC).

In CA, rather than placing generic ads, it seems preferable to have ads related to page content to match the user interest and thus to increase the probability of clicks, called *click-through-rate* (CTR), which can be approximated by a similarity score  $\text{Sim}_{a,p}$  between ad  $a$  and page  $p$ . Given page  $p$ , we place the ads that maximize  $\text{Score}_{a,p} \stackrel{\text{def}}{=} \text{Bid}_a \times \text{Sim}_{a,p}$ , where  $\text{Bid}_a$  is ad  $a$ 's bid in the PPC model and  $\text{Sim}_{a,p} \stackrel{\text{def}}{=}$

$\sum_{f \in a \cap p} w_{f,a} \times w_{f,p}$ , where the weights  $w_{f,p}$  and  $w_{f,a}$  are the weights of feature  $f$  on ad  $a$  and page  $p$ , respectively. Thus, advertisers eager to reach more customers have two options: crafting their ads to improve  $\text{Sim}_{a,p}$  or increasing their bids. In either case, the advertisers are interested to know what is the effect of these changes. A simple approach is to try a test ad and analyze the effect after a long while, which is inefficient and has a huge turn-around. Therefore, a real-time forecasting system has obvious applications for ad selection, campaign budgeting, and advertising strategy.

We propose to forecast ad impressions in real time by replaying historical page views with the associated ads. Given ad  $a$  and its bid  $\text{Bid}_a$ , we consider all CA opportunities over, say, the last week, and check how often ad  $a$  would have been shown if it were in the system, i.e., how often  $\text{Score}_{a,p}$  would have beaten the score of the lowest scoring ad historically shown on  $p$ ,  $\text{minScore}_p$ . The impression rate is then used to predict the impression volume over, say, the next week.

The billions of past CA opportunities can be collapsed into a relatively small number of distinct pages and furthermore we devise an efficient method to greatly reduce the number of pages to be considered. Using a standard PC with 2GB memory, it takes about 10 ms for our system to forecast the impression volume of an ad, over 700M CA opportunities.

## 2. METHOD

We build an inverted index for historical page views, and collect the number of page impressions  $\text{IMP}_p$  and  $\text{minScore}_p$  *offline*. Test ads are sent *online* to query the page index via a two-level search process: at the first level, an inexpensive approximate evaluation is conducted to identify the set of candidate pages on which the ad could have been shown. In detail, we check if ad  $a$  could be shown on page  $p$  by comparing an upper bound of  $\text{Score}_{a,p}$  with  $\text{minScore}_p$ , i.e., if  $\text{Bid}_a \times \sum_{f \in a \cap p} w_{f,a} \times \text{maxWeight}_f > \text{minScore}_p$ , where  $\text{maxWeight}_f$  represents the maximum weight of the feature  $f$  in any page. For efficiency, the value of  $\text{maxWeight}_f$  can be pre-calculated offline as well during index building for each feature  $f$ . At the second level, a full evaluation is done for each candidate page and the forecast counter is increased accordingly if a test ad's score beats  $\text{minScore}_p$  of the candidate page  $p$ . The first level filtering does not alter the final results—we would obtain exactly the same results even if we fully evaluate every page  $p$  at the second level.

We implement the two-level search process by adapting the WAND operator [1]. At the first level test, we evaluate

$$\text{WAND}(X_{f_1}, w_{f_1,a} \times \max_{\text{Weight}_{f_1}}, X_{f_2}, w_{f_2,a} \times \max_{\text{Weight}_{f_2}}, \dots, X_{f_{|a|}}, w_{f_{|a|},a} \times \max_{\text{Weight}_{f_{|a|}}}, \frac{\text{minScore}_p}{\text{Bid}_a}),$$

where  $X_{f_i}$  is an indicator variable for the presence of query feature  $f_i$  in page  $p$  and  $|a|$  is the number of features for ad  $a$ . If WAND is true, a full evaluation is performed.

The original WAND iterator [1] is used to obtain a ranked list of documents relevant to a query. To do ordinary search using WAND, the threshold (the last argument of WAND) is set dynamically to the minimum score of the top results found so far as the posting list cursor  $C_f$  advances. The threshold is fixed for pages between two consecutive full evaluations. However, in ad impression forecasting, the threshold  $\text{minScore}_p/\text{Bid}_a$  is page dependent since  $\text{minScore}_p$  is different for each page and, for a given page, the threshold does not change at all no matter how the cursors move.

To take advantage of the page difference in  $\text{minScore}_p$ , we index the pages in increasing order of  $\text{minScore}_p$  (i.e., pages with smaller PIDs (page identifier) have lower  $\text{minScore}_p$ ), so that it becomes more and more difficult to have the test ad shown when the posting list cursors move to larger PIDs.

In our new WAND iterator, the core method `nextCandidate()` repeatedly advances the individual feature cursors until it finds a candidate page to return for full evaluation.

```

1. Function nextCandidate()
2.   repeat
3.     sort(features)
4.     pivotFeature ← findPivotFeature(features)
5.     if (pivotFeature = null) return (LastID)
6.     pivotPID ← CpivotFeature.PID
7.     if (pivotPID = LastID) return (LastID)
8.     if (pivotPID = currentPage) //pivotPID tested?
9.       f ← pickFeature(features[1, ..., pivotFeature])
10.      Cf.beyond(pivotPID + 1)
11.   else
12.     if (C0.PID= pivotPID)
13.       currentPage ← pivotPID
14.       return (currentPage)
15.   else
16.     f ← pickFeature(features[1, ..., pivotFeature])
17.     Cf.beyond(pivotPID)
18.   end repeat

```

The `nextCandidate()` method invokes three helper functions: `sort()`, `findPivotFeature()` and `pickFeature()`. The first helper, `sort()`, sorts the features in non-decreasing order of their current PIDs, and `findPivotFeature()` returns *pivotFeature*—the first feature in the sorted order for which the accumulated (weighted by  $w_{f,a}$ ) upper bounds of all features preceding it, including it, exceed  $\text{minScore}_p/\text{Bid}_a$ . The last helper, `pickFeature()`, receives as input a set of features and selects the feature whose cursor is to be advanced.

Note that if we indexed the pages **not** in increasing order of  $\text{minScore}_p$ ,  $\text{minScore}_p/\text{Bid}_a$  for a larger PID could be actually lower than for a smaller PID and valid candidate pages might be incorrectly skipped for full evaluation.

### 3. EXPERIMENTAL RESULTS

We collected two weeks of actual impression events from a random set of hosts participating in Yahoo!’s ad network, for about 1.3B impressions, and 10M textual ads were chosen at random from Yahoo!’s ad database. The data were split into (half/half) a training set and a test set. Naturally, many

**Table 1: Average absolute relative error of forecasted impressions of at different impression levels.**

#Impressions	#Ads	Error	90% Interval
0-2000	2307	26364.2%	[-100%,1e5%]
2001-5000	759	157.7%	[-100%,498%]
5001-10000	768	69.7%	[-99.7%,196%]
10001-20000	406	61.0%	[-79.7%,156%]
20001-50000	708	54.6%	[-60.4%,140%]
50001-100000	287	48.0%	[-54.5%,112%]
100001-200000	186	32.0%	[-62.6%,69.8%]
200001-500000	185	32.8%	[-60.3%,75.4%]
500001-1000000	102	28.6%	[-53.3%,55.4%]
1000001-10000000	156	15.7%	[-31.3%,28.8%]

pages appear both in the training set and test set. However, due to the dynamic nature of the Web and to traffic variations, those pages are not identical and they differ with respect to both composition and impression counts.

We evaluate our system by randomly picking approximately 6000 sample test ads and compare the forecasted impressions based on the training set with the actual impressions from the test set. We split the test ads according to the impression level and report the number of sample ads at that level, the average absolute relative error and the corresponding 90% confidence interval in Table 1. The 90% interval is a range such that 90% of the test ads have a relative error in that range. The average absolute relative error becomes smaller when the number of impressions becomes larger, and the confidence interval becomes tighter. Our approach can also display an impression volume vs. bid curve to help advertisers make cost-effective decision on bids.

In terms of efficiency, experimentally, our approach is demonstrated to respond in sub-seconds. On an Intel Xeon 3GHz PC with 2GB RAM, with the inverted page index and other offline information kept in memory, the average evaluation time for a (test ad, bid) pair is 10.1 ms and we require 377.1 ms for curve plotting with 100 different bid levels.

### 4. CONCLUSIONS

We have presented a search-based method for ad impression forecasting of new ads in contextual advertising. To make the search feasible, we first reduce the search over the past impressions into a search over the unique Web pages in those impressions. Then we search the space of pages using a two-level search process: at the first level, an inexpensive approximate evaluation is performed to identify candidate pages on which a test ad could possibly have been shown; at the second level, the candidates are fully evaluated and their contribution is counted. Experimentally, we demonstrate that our approach can accurately forecast impressions of ads in the higher range of views per day in real time. As such, it can be used by mid-size and large advertisers that run campaigns with millions of ad views per day. Our approach can also be used for bid suggestion, and ad evaluation.

### 5. REFERENCES

- [1] A. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 12th International Conference on Information and Knowledge Management*, pages 426–434, 2003.