Mining for Insights in the Search Engine Query Stream

Ovidiu Dan
Lehigh University
Bethlehem, PA 18015 USA
ovd209@cse.lehigh.edu

Pavel Dmitriev
Microsoft Bing
Bellevue, WA 98004 USA
padmitri@microsoft.com

Ryen W. White
Microsoft Research
Redmond, WA 98052 USA
ryenw@microsoft.com

ABSTRACT
Search engines record a large amount of metadata each time a user issues a query. While efficiently mining this data can be challenging, the results can be useful in multiple ways, including monitoring search engine performance, improving search relevance, prioritizing research, and optimizing day-to-day operations. In this poster, we describe an approach for mining query log data for actionable insights – specific query segments (sets of queries) that require attention, and actions that need to be taken to improve the segments. Starting with a set of important metrics, we identify query segments that are "interesting" with respect to these metrics using a distributed frequent itemset mining algorithm.

Categories and Subject Descriptors
H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms
Algorithms, Experimentation

Keywords
Query log analysis; frequent itemset mining; FP-growth.

1. INTRODUCTION
Web search engines store large amounts of information for each user transaction, including the content of the query, the language and location of the user, the vertical used by the user, the results the user clicked on, if the user used spelling correction or related searches, etc. However, most of this information is discarded in practice or it is used only in computing superficial descriptive statistics. Mining available metadata beyond just the text of the queries can help measure and improve search quality.

This poster focuses on mining query log metadata for actionable insights and it is motivated by the goal of better understanding the quality of search results. We segment queries into categories by using query log data and one or more metrics designed to evaluate search engine performance (e.g., [6]). We refer to the resulting categories as *segments* in the rest of the poster, not to be confused with the query segmentation line of research looking to uncover relationships between query terms [5]. A simple example of a segment is *{Market: en-US, IsNews: True, NumWords: 6, CTR: 0.3} = 10,000*, which can be read as: *of all the queries in the dataset which were issued in English in the United States, have a news intent, and which contain 6 words, 10,000 of them have a click-through rate on the search engine result page (SERP) of 0.3*. These automatically-generated patterns can be used to answer questions on the behavior of a search engines, such as: What are the types of queries for which the search engine has good performance, based on a certain metric? What are the categories of queries where the search engine needs to improve? How does the search engine stack up against competition?

WWW 2012 Companion, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1230-1/12/04.

Mining for patterns in aggregate query logs poses significant challenges. First, such logs are often terabytes or petabytes in size, and grow daily. This means that the logs cannot be stored or processed on a single machine. Second, the number of patterns can be very large. In the log dataset used here, each query had approximately 300 mostly-categorical attributes. Assuming 10 values per attribute, this means 10^{300} patterns. To mine this data in a timely manner, we developed a distributed and scalable mining algorithm.

2. DISTRIBUTED FP-GROWTH
The Frequent Itemset Mining problem was introduced by Agrawal et al. [1]. Given a set of items $I = \{i_1,\ldots,i_n\}$ and a set of transactions $D = \{t_1,\ldots,t_m\}$ the algorithm finds all subsets X of I with support(X) > T, where support(X) = $|t_i$ *containing* $X|/|D|$. Starting from a table where each row represents a unique query, and the columns represent aggregate metadata for the query, our main problem is mining for interesting patterns using a scalable algorithm. We cast the problem as frequent itemset mining by treating combinations of column names and values as items in a transaction, where each row is one transaction. Figure 1 shows an example of converting a table into transactions, then running a frequent itemset algorithm on the resulting set to generate patterns.

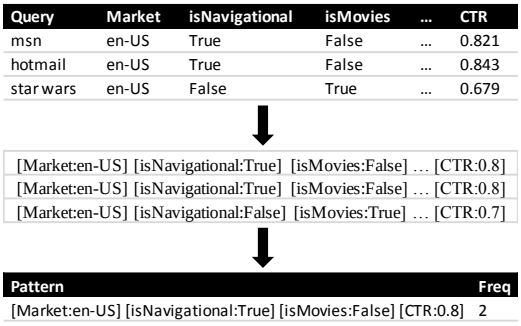

Figure 1. Example of mining for patterns.

Han et al. [3] introduced FP-Growth, an algorithm that computes the item sets in only two passes by using uses a prefix-tree structure to compress the data. Although the FP-Growth algorithm has both good space and time efficiency, the in-memory algorithm cannot handle arbitrarily large datasets. We implemented a distributed version of FP-Growth algorithm, building on a tag prediction algorithm proposed by Li et al. [4], which in turn is based on the database projection method described in the original paper [2]. The *prefix path property* described in the original paper suggests that work can be split into independent subtasks. To mine all frequent patterns with prefix a_i we only need to work on all branches which contain a_i. Furthermore, from these branches only the subpaths ending in a_i are required. Consequently, the patterns ending in a_i can be generated on one machine, independently of any other computation. The third column in Table 1 presents an example. For instance, to find the itemsets which end with *m*, we can distribute *m: a c d* and *m: b d* to one machine and continue the computation locally.

Transaction	Sorted by freq	Map outputs
p c a n	a c p n	n: a c p p: a c c: a
b	b	-
b n d	b d n	n: b d d: b
d m b	b d m	m: b d d: b
p c a q	a c p q	q: a c p p: a c c: a
m d a c	a c d m	m: a c d d: a c c: a
b a	a b	b: a

Table 1. Example transactions, transactions sorted by frequency, and distributing computation.

Conditional databases	Conditional FP-Trees
b: {a}	{} | b
c: {a / a / a}	{a:3} | c
d: {b / b / a c}	{b:2} | d
m: {b d / a c d}	{d:2} | m
n: {a c p / b d}	{} | n
p: {a c / a c}	{ a c: 2} | p
q: {a c}	{} | q

Table 2. Conditional databases and conditional trees for example transactions.

Our implementation is built on the distributed SCOPE data processing framework introduced in [2]. SCOPE runs on a distributed platform which is fault-tolerant and scales to thousands of commodity servers. Data are stored in a distributed storage system designed to reliably store extremely large sequential files.

3. QUERY STREAM EXPERIMENTS

We carried out an experiment on a large query log dataset which consists of a table with one row for each unique query issued on the Bing search engine in a one-week period ending on August 5 2011. Each row consists of 300 metadata fields obtained by aggregating data from all user sessions containing the query. The table contains information on a few hundred million unique queries. The metadata is generated by aggregating internal data, such as server logs, as well as external traffic data from toolbars.

Some of the available fields include the geographic location and language of the user, the length of the query in number of tokens, the frequency of the query in the timeframe covered by the dataset, and the output of tens of binary text classifiers used to determine the query intent of a user. The classifiers cover tens of categories, such as movies, technology, local search, auto, people search, news, navigational queries etc. Other fields include the vertical used to issue a query, the presence of specific features such as instant answers, advertisements, spelling correction, and related searches. Available click information includes the position of all the clicked results and answers, as well as the number of clicks on certain areas such as advertisements shown on the right-rail of the SERP.

The table also contains several metrics computed separately for each unique query. *Click-through rate* is the number of SERP visits with clicks divided by the total number of SERP visits. *Success-when-clicked ratio* is the ratio obtained by dividing the number of SERP visits with success by the total number of SERP visits with clicks. A SERP visit is considered to be a success if the user clicked on one of the results and did not revisit a SERP for at least 30 seconds. *Quick-back ratio* is the number of SERP visits where the user clicked on a result and returned in less than 30 seconds, divided by the total number of SERP visits with result clicks. *PSkip*, introduced in [6], estimates the probability that a user will skip the first results on the page and click on results which are further down the page. A query with lower pSkip is regarded as having better ranking quality.

We ran our algorithm separately on each metric with a minimum support of 1,000. Depending on the load on the cluster and on the way in which SCOPE built the execution plan, the different steps of the algorithm ran on as few as 250 nodes and as many as 6,000 machines or more, and finished in under an hour. Input/output bound intensive steps were usually scheduled on more machines.

Table 3 shows four categories of queries for which the Bing search engine performs well, each for a different metric. For instance, the last pattern matches all queries where the user intent is to search for show times for a particular movie in a particular location, and the quick back ratio is between 0 and 0.1. While the output of the algorithm also contains the frequency of each pattern, we do not show those values here.

[IsVideo:True] [NumWords:5] [CTR:[0.90-1.00]]
[NumWords:3] [IsCommerce:True] [success:[0.90-1.00]]
[IsUrlQuery:True] [NumWords:1] [pSkip:[0.00-0.10)]
[MovieShowtimes:True] [MovieTheater:True] [MovieTitle:True] [quickBackRatio:[0.00-0.10)]

Table 3. Frequent itemset patterns examples.

A simple modification to the experiment helped compare Bing to other search engines. Using external data from toolbar logs, we added metrics from other search engines. An artificial example of a generated pattern is *{[MovieShowtimes:True] [BingCTR:[0.90-1.00]] [OtherEngineCTR:0.00-0.10]} = 10,000*, which can be read as: *the log contains 10,000 unique queries with a MovieShowtimes intent, where click-through rate for Bing is [0.9-1.0], and the click-through rate of the other search engine for the same queries is [0-0.1]*. These types of patterns easily highlight the categories of queries where one engine outperforms another using any of the metrics, and demonstrate the value of our approach.

4. REFERENCES

[1] Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. *Proc. SIGMOD*, 207–216.

[2] Chaiken, R., Jenkins, B., Larson, P.-R., Ramsey, B., Shakib, D., Weaver, S., and Zhou, J. (2008). SCOPE: easy and efficient parallel processing of massive data sets. *Proc. VLDB Endow.*, 1(2): 1265–1276.

[3] Han, J., Pei, J., Yin, Y. and Mao, R. (2004). Mining frequent patterns without candidate generation: a frequent-pattern tree approach. *Data Mining and Knowledge Disc.*, 8(1): 53–87.

[4] Li, H., Wang, Y., Zhang, D., Zhang, M., and Chang, E.Y. (2008), PFP: parallel FP-growth for query recommendation. *Proc. RecSys*, 107–114.

[5] Tan, B. and Peng, F. (2008). Unsupervised query segmentation using generative language models and wikipedia. *Proc. WWW*, 347–356.

[6] Wang, K., Walker, T., and Zheng, Z. (2008). PSkip: estimating relevance ranking quality from web search clickthrough data. *Proc. SIGKDD*, 1355–1364