Extracting User Profiles from Large Scale Data

Michal Shmueli-Scheuer, Haggai Roitman, David Carmel, Yosi Mass, David Konopnicki
IBM Research, Haifa
{shmueli,haggai,carmel,yosimass,davidko}@il.ibm.com

ABSTRACT

In this work we present the details of a large scale user profiling framework that we developed here in IBM on top of Apache Hadoop. We address the problem of extracting and maintaining a very large number of user profiles from large scale data. We first describe an efficient user profiling framework with high user profiling quality guarantees. We then describe a scalable implementation of the proposed framework in Apache Hadoop and discuss its challenges.

Categories and Subject Descriptors: H.3.4 [Systems and Software]: User Profiles

General Terms: Algorithms, Design, Experimentation, Performance

Keywords: User profile, Large scale, Hadoop

1. INTRODUCTION

In this work we present the details of a large scale user profiling framework that we developed here in IBM on top of Apache Hadoop. We address the problem of extracting and maintaining a very large number of user profiles extracted from large scale data.

In this context, a user profile is often used to classify a given user into predefined user segments (e.g., by demographics or tastes) or to capture the online behavior of the user including the user's private interests and preferences.

A user profile can be explicitly defined by the user herself, e.g., during the user's registration to some service. User profiling is usually defined as the process of implicitly learning a user profile from data associated with the user. Data sources for user profiling include among others the user's browsing sessions, the user's own generated content (e.g., the user's blog), the user's social interactions with friends in the user's social network (e.g., the user's discussions with others), click-through data extracted from search logs, or even other user profiles using collaborative filtering techniques.

User profiles may evolve over time due to possible changes in user interests and tastes, and our aim in this work is to develop a scalable user profiling solution that captures such evolution and maintains qualitative user profiles.

MDAC'10 ,April 26, 2010 Raleigh, NC, USA

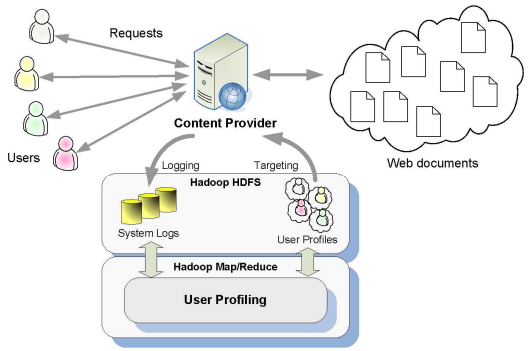

Figure 1: General setting illustration

User profiles are commonly used for purposes of personalization. For example, a user profile can be used for personalized search by reranking search results according to the query issuer's identity. User profiles are also utilized in various personalized recommendation services. For example, the popular social music service *Last.fm* utilizes user profiles for personalized music recommendations. As another example, different online stores such as *Amazon* and *Shopping.com* utilize user profiles for personalized product recommendations to potential online shoppers. Recently, in the context of online advertising campaigns, user profiles are also utilized by various behavioral targeting solutions for improving the click through rate (CTR) of disseminated ads.

In this work, we assume a general content management setting, which is further illustrated in Figure 1, where a content provider is responsible to serve multiple users that submit various requests for web documents, e.g., users that visit the content provider's website or update blogs managed by the content provider's blogging service. The provider's server logs each request in the system log and records all users interactions with their documents of interest. The provider's user profiling module consumes the system log data during scheduled time periods and is responsible to maintain user profiles over time. User profiles can be used by the content provider for targeted services to users (e.g., advertisements, recommendations) or to provide personalized services (e.g., personalization of a user's search results).

We propose a scalable user profiling framework that is based on concepts of feature selection, where user profiles

are represented by the textual content consumed or produced by different users and our aim is to weigh user profile terms according to their capability of representing the user's interests. For that purpose, we propose an efficient and scalable feature selection method based on Kullback-Leibler (KL) divergence, tailored for the user profiling task.

We then show how the proposed framework can be implemented on top of Apache Hadoop MapReduce framework. Using real-world data we evaluate the scalability of the proposed framework.

To summarize, this paper includes two main contributions. First, we describe an efficient user profiling framework with high user profiling quality guarantees. Second, we describe a scalable implementation of the proposed framework in Apache Hadoop and discuss its challenges.

The rest of the paper is organized as follows. We provide background in Section 2, followed by the details of our user profiling framework in Section 3. Section 4 presents our framework evaluation, and we conclude in Section 5.

2. BACKGROUND

In this work we extract and maintain user profiles based on the textual content of documents associated with users over time. A survey on various state-of-the-art techniques of obtaining data for user profiles and types of user profile models (e.g., content-based, hierarchical, semantic profile models) is provided in [6]. We now focus our discussion on content-based user profiles.

A common approach for representing a user profile based on content consumed or produced by the user is as a weighted vector of terms (keywords), also known as the *Bag Of Words* (BOW) model (e.g., [4, 13]). Within this model, user profile terms are usually weighted according to the well know information retrieval $tf \times idf$ measure [11].

During time, users may exhibit temporary or even constant changes in their interests. Therefore, maintaining a user profile periodically is important in order to adjust the profile to such possible changes, while keeping the history knowledge about the user. Such user profile maintenance considerations have been also extensively studied in the past, e.g., [10, 8, 13]. The most prevalent approach is to maintain user profiles using two main temporal abstraction levels, i.e., a *long-term* level to represent the history, and a *short term* level to represent the latest changes. While the long term profile may represent more general or stable interests of the user; the short term profile may represent more specific or current interests. In this context, Sugiyama et al. [13] further proposed a window-based user profile model, where keywords that appear in past and current Web pages associated with the user, are aggregated based on their relative $tf \times idf$ scores and their recency.

In this work we weigh user profile terms based on the Kullback-Leibler divergence (KL) measure, which is a non-symmetric distance measure between two given distributions, extensively used in various fields like information theory, biology, content analysis, etc. The most related works to our setting that also used KL are works on information filtering and annotation [7, 14]. The work by Yanagimoto et al. [14] aimed at filtering new documents that may not interest the user. They used the KL divergence between the documents that the user viewed and those that he did not.

Hinne et al. [7] tried to determine the most descriptive and discriminating query terms associated with a given URL in order to identify whether the URL corresponds to the user's query intent. They used the KL divergence between each pair of a query term and the associated URL.

Recently, with the growth in the amount of online generated data and users, several works have focused on developing frameworks and algorithms for targeting users in large scale systems [5, 15, 3]. Chen et al. [5] developed a behavioral targeting system over Hadoop MapReduce framework to select the ads most relevant to users. Zhou et al. [15] proposed large scale collaborative filtering techniques for movies recommendation. Finally, Cetintemel et al. [3] suggested an incremental algorithm for constructing user profiles based on monitoring and explicit user feedback. Their approach allows to trade between the profile complexity and its quality. In the context of these works, in this paper we describe how a very large scale number of content-based profiles can be extracted and maintained using Apache Hadoop MapReduce framework.

3. USER PROFILING FRAMEWORK

We now describe the details of a large-scale user profiling framework. We begin by describing the framework general setting, followed by the details of its user profile model. We then show how user profiles are extracted, and we conclude this section with a description of the framework implementation in Hadoop.

3.1 Setting

We now return back to the setting illustration of Figure 1 and assume a general content management setting with multiple users who submit various requests[1] for web documents to some content provider. The provider's server logs each request in the system log, which records all users interactions with their documents of interest. Each record in the log is a tuple $\langle u, d, context \rangle$ and captures a single user-document association where u represents the user, d represents the document associated with that user, and *context* is any additional available metadata extracted from the context of the user-document association (e.g., time, geographic location, the query text in a search session, tags used by the user to bookmark the document, etc). In this work we attach every available textual context to the associated document's content.

A user profiling module consumes the system log data during scheduled time periods (e.g., every once a day) and is responsible to maintain user profiles[2]. Both the system log and user profiles are stored in Hadoop's distributed file system (HDFS). Once user profiles are updated at some time period j, all log records are "flushed" from the log. Therefore, on each time period j only those records that were logged in the system log between the previous time period $j-1$ and the current one are needed for updating the user profiles.

User profiles can be used by the content provider for targeted services to users (e.g., advertisements, recommendations) or to provide personalized services (e.g., personalization of a user's search results). In the rest of this paper, we

[1] We use the term "request" to represent either a single user data consumption or production activity, i.e., by submitting a `HTTP-GET` or `HTTP-POST` request to the provider's server.

[2] For simplicity, in this work we schedule profile maintenance between fixed time intervals. Methods to determine optimal intervals also exist [12].

focus on the aspects of our user profile model and its large scale implementation in Hadoop.

3.2 User Profile Model

3.2.1 Basic notations

Given some time period j, we denote by D_j the "*community snapshot*", defined as the set of all requested documents d that were recorded in the system log during that time period. Given a user u, we also denote by $D_j(u)$ the "*user snapshot*", defined as the subset of documents $d \in D_j$ that were requested by user u during time period j.

A user profile is represented by the textual content of the documents associated with the user over time. In this work we adopt the *Bag Of Words* (BOW) model for representing user profiles. Given a vocabulary of terms $V = \{t_1, t_2, \ldots, t_m\}$, the user profile of each user u, at a given time period j (denoted by $p_j(u)$), is then defined as a weight vector:

$$p_j(u) = (w_j^u(t_1), w_j^u(t_2), \ldots, w_j^u(t_m))$$

Each weight $w_j^u(t)$ corresponds to the relative profile importance of a *unique* term $t \in V$ at time period j, which may appear in the textual content of the associated documents $D_j(u)$. If the user profile content does not contain term t, then $w_j^u(t) = 0$. Each term can be either a word, a phrase, or a lexical affinity.

3.2.2 User profile maintenance

User profiles are maintained over time, and we capture the evolution of user profiles using the following recursive update rule:

$$\tilde{p}_j(u) \leftarrow \alpha_j \cdot p_{j-1}(u) + (1 - \alpha_j) \cdot p_j(u) \qquad (1)$$

α_j is a learning parameter and it controls the system's relative preference between the fresh user profile snapshot $p_j(u)$ and the history profile $p_{j-1}(u)$, where $p_0(u)$ is initiated with zero weights. In this work we use exponential decay smoothing and set $\alpha_j = \alpha_0 \cdot \exp^{-j/C}$. The larger α_j is, the more we rely on the user profile learned from history (model *exploitation*) and the less we rely on the fresh user profile (model *exploration*). The C parameter controls the rate of learning where larger values imply a slower learning process. We learn both parameters from sample data, and in this work we set $\alpha_0 = 0.1$ and $C = 100$.

Figure 2 further demonstrates an example of a single evolving user profile $p_j(u)$ of some user u during different time periods j. At every time period, the user profile is represented in Figure 2 by a single "tag-cloud", where more fresh user profile snapshots are represented by larger clouds. The community snapshot D_j and the user snapshot $D_j(u)$ at every time period j are also illustrated in Figure 2 by an ellipse (up) and a circle (down).

3.3 Weighting User Profile Terms

Weighting user profile terms is strongly related to *feature selection* which is the process of selecting a subset of the terms for text representation, and is frequently applied by text categorization and text clustering methods [11]. Common approaches for feature selection evaluate terms according to their ability to distinguish the given text from the whole text. For the user profiling task, given time period j, we capture user u latent interests by weighting the terms in

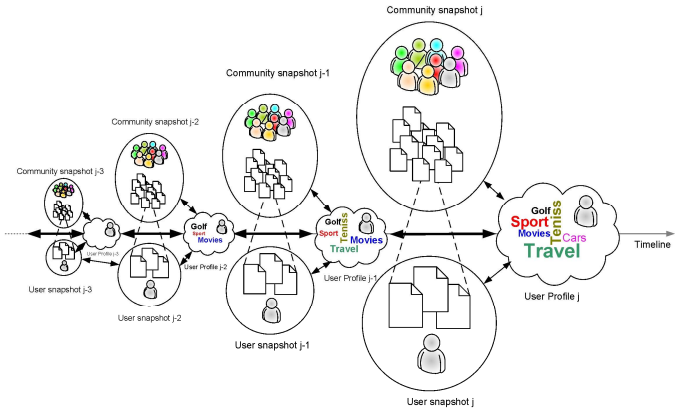

Figure 2: Demonstration of a single user profile evolution over time

$p_j(u)$ according to their contribution to the *separability* of the current user snapshot $D_j(u)$ from the entire community snapshot D_j.

We next describe a feature selection method that is based on few simple principles that were previously proposed for the tasks of website findability [2] and cluster labeling [1]. This method was further shown in [1] to provide a relatively very high quality text features for representing a cluster of documents compared to other state-of-the-art feature selection methods [11].

We first discuss the set of basic features that we use; then, we discuss how user profile terms are actually weighted using the proposed feature selection method which is based on Kullback-Leibler (KL) divergence [9].

3.3.1 Term features

Our user profiling method is based on several low level IR-fashion term features extracted from both the community snapshots D_j and user snapshots $D_j(u)$ over time. Given a term $t \in V$ and a document $d \in D_j(u)$ the first feature type is the term t frequency in document d, denoted by $tf(t, d)$. The second feature type is the term t document frequency, defined as the number of documents in a given document collection that contain that term. For time period j, we denote by $df(t, D_j)$ and $df(t, D_j(u))$ the term t document frequency in the j-th community snapshot and j-th user snapshot.

The intuition behind the selection of these basic features is that a good representative term t should be such that is highly frequent in documents $d \in D_j(u)$, should appear as much as possible in user snapshot $D_j(u)$ documents and as less as possible in the community snapshot documents D_j. Leaning on this simple intuition, given some term $t \in V$, and snapshots D_j and $D_j(u)$, the initial weight we assign to the term is given by:

$$w_j^u(t) = \overline{tf}(t, D_j(u)) udf(t, D_j(u)) idf(t, D_j) \qquad (2)$$

where $\overline{tf}(t, D_j(u)) = \frac{1}{|D_j(u)|} \sum_{d \in D_j(u)} tf(t, d)$ is the (average) term frequency in $D_j(u)$'s centroid, $udf(t, D_j(u)) = \frac{df(t, D_j(u))}{|D_j(u)|}$ is the maximum likelihood estimation of the probability to find term t in $D_j(u)$, and $idf(t, D_j) = \log\left(1 + \frac{|D_j|}{df(t, D_j)}\right)$ is the inverse document frequency of the term t, determined

by that term relative document frequency in the j-th community snapshot D_j. A naïve approach for weighting the user profile terms is according to Eq. 2. Next we propose a more sophisticated method we coin the "KL method" which is based on the initial weights of Eq. 2.

3.3.2 The KL method

Recall that, given some time period j, we wish to capture each user u's latent interests by weighting the terms in $p_j(u)$ according to their contribution to the separability of the current user snapshot $D_j(u)$ from the entire community snapshot D_j.

Such separability can be measured using the Kullback-Leibler (KL) divergence [9] between the user snapshot $D_j(u)$ term distribution and that of the entire community snapshot D_j. For two distributions $P_1(t)$ and $P_2(t)$ over the terms in the collection $t \in V$, the KL divergence is defined as:

$$\mathfrak{D}_{KL}\left(P_1||P_2\right) = \sum_{t \in V} P_1(t) \log \frac{P_1(t)}{P_2(t)} \quad (3)$$

For a given term $t \in V$, let $P(t|D_j(u))$ and $P(t|D_j)$ denote the marginal term distributions over the set of documents in $D_j(u)$ and D_j, respectively. We now describe how each of the two marginal distributions is generated.

Given community snapshot D_j, its marginal distribution with respect to term t is calculated as:

$$P(t|D_j) = \overline{tf}(t, D_j) cdf(t, D_j) N_j \quad (4)$$

where $cdf(t, D_j) = \frac{df(t, D_j)}{|D_j|}$ is the maximum likelihood estimation of the probability to find term t in D_j, and $N_j = \left(\sum_{t \in V} P\left(t|D_j\right)\right)^{-1}$ is the probability normalization factor.

For user snapshot $D_j(u)$, we use Eq. 2 and measure its marginal distribution with respect to a given term t according to the relative initial weight of that term as follows:

$$P(t|D_j(u)) = (1-\lambda)\left(\frac{w_j^u(t)}{\sum_{t \in V} w_j^u(t)}\right) + \lambda P(t|D_j) \quad (5)$$

λ is a smoothing parameter used to smooth $P(t|D_j(u))$ with $P(t|D_j)$, which is a common technique in language models [11]. In this work we set $\lambda = 0.001$.

Finally, we weigh each user profile term t by its marginal contribution to the maximization of KL divergence between the user snapshot and community snapshot distributions, calculated as follows:

$$\tilde{w}_j^u(t) = P(t|D_j(u)) \log \frac{P(t|D_j(u))}{P(t|D_j)} \quad (6)$$

3.4 Large Scale Implementation

We now describe an efficient large scale implementation of our user profiling solution on top of Apache Hadoop MapReduce framework[3]. The implementation flow pseudo-code is further provided in details in Algorithm 1.

The amount of parallelization of the user profiling framework closely depends on two computational bottlenecks that dictate the capability to process data without the need to wait for the completion of some unprocessed sub-flows. The first bottleneck is attributed to the calculation of the probability normalization factor N_j, used to normalize the community snapshot probabilities in Eq. 4[4]. The second bottleneck is attributed to the calculation of the weights normalization factor $\sum_{t \in V} w_j^u(t)$ in Eq. 5 due to a similar reason.

Algorithm 1: User profiling

Input: Log records: $[u, d]$, Document contents: $[d, text]$
Output: Userprofile weights: $[(u, t), \tilde{w}_j^u(t)]$

Phase (1) [*count the number of documents on each user snapshot $D_j(u)$ and calculate intermediate community snapshot D_j features*]
begin
  MapReduce flow: `countUserDocs`
  begin
    `Map`$[u, d] \rightarrow [u, 1]$
    `Reduce`$[u, \{1\}] \rightarrow$ `Store`$[u, |D_j(u)|]$
  MapReduce flow: `calcCommunityFeatures`
  begin
    `Map`$[d, text] \rightarrow [t, (d, 1)]$
    `Reduce`$[t, \{(d, 1)\}] \rightarrow$ `Store`$[(t, d), tf(t, d)]$
    `Map`$[(t, d), (tf(t, d), |D_j|)] \rightarrow [t, (tf(t, d), |D_j|, 1)]$
    `Reduce`$[t, \{(tf(t, d), |D_j|, 1)\}] \rightarrow$
    `Store`$[t, (\overline{tf}(t, D_j) cdf(t, D_j), idf(t, D_j))]$
    `Map`$[t, (\overline{tf}(t, D_j) cdf(t, D_j))] \rightarrow$
    $[$`norm`$, (\overline{tf}(t, D_j) cdf(t, D_j))]$
    `Reduce`$[$`norm`$, \{(\overline{tf}(t, D_j) cdf(t, D_j))\}] \rightarrow$ `Store`$[N_j]$

Phase (2) [*calculate community snapshot D_j marginal term distribution and the initial user profile term weights*]
begin
  MapReduce flow: `calcComSnapshotProb`
  begin
    `Map`$[t, (\overline{tf}(t, D_j) cdf(t, D_j), N_j)] \rightarrow [t, P(t|D_j)]$
    `Reduce`$[t, P(t|D_j)] \rightarrow$ `Store`$[t, P(t|D_j)]$
  MapReduce flow: `calcInitialTermWeights`
  begin
    `Map`$[(u, t, d), (tf(t, d), |D_j(u)|, idf(t, D_j))] \rightarrow$
    $[(u, t), (tf(t, d), |D_j|, idf(t, D_j), 1)]$
    `Reduce`$[(u, t), \{(tf(t, d), |D_j|, idf(t, D_j), 1)\}] \rightarrow$
    `Store`$[(u, t), w_j^u(t)]$
    `Map`$[(u, t), w_j^u(t)] \rightarrow [u, w_j^u(t)]$
    `Reduce`$[u, \{w_j^u(t)\}] \rightarrow$ `Store`$[u, \sum_{t \in V} w_j^u(t)]$

Phase (3) [*calculate user snapshot $D_j(u)$ marginal term distribution and weight user profile terms using the KL method*]
begin
  MapReduce flow: `calcUserProfileTermWeights`
  begin
    `Map`$[(u, t), (P(t|D_j), w_j^u(t), \sum_{t \in V} w_j^u(t), \tilde{w}_{j-1}^u(t), \alpha_j)] \rightarrow$
    $[(u, t), \tilde{w}_j^u(t)]$
    `Reduce`$[(u, t), \tilde{w}_j^u(t)] \rightarrow$ `Store`$[(u, t), \tilde{w}_j^u(t)]$

Therefore, the implementation flow in Algorithm 1 is partitioned into three main phases that are serially executed. Each phase begins by fetching the intermediate results of the previous phases from HDFS, followed by the parallel execution of the phase logic using few MapReduce sub-flows, and ends by storing back the phase intermediate results into HDFS for use by the following phases. We now describe the flow of each phase in more details. The notations `Map`$[(key1), (val1)] \rightarrow [(key2), (val2)]$ and `Reduce`$[(key2), \{(val2)\}] \rightarrow [(key3), (val3)]$ in Algorithm 1 denote a single MapReduce sub-flow, while the notation `Store`$[(key), (val)]$ further denotes the storage of intermediate results in HDFS.

[3] http://hadoop.apache.org/

[4] This bottleneck can be resolved by using approximation approach and it is considered as a future work.

The first phase executes two parallel MapReduce flows. The first flow, `countUserDocs`, gets the input of user-document pairs and counts for each user the number of documents in that user's snapshot. The second flow, `calcCommunityFeatures`, begins with a MapReduce sub-flow that calculates the term frequency of each term in each document $tf(t,d)$, which output is then both stored in HDFS for later use in the next phase and is piped into another MapReduce sub-flow that calculates the community snapshot features of each term. Its output is also both stored in HDFS for later use in the next phase and is piped into the last MapReduce sub-flow that calculates the value of the normalization factor N_j and store it in HDFS for the next phase.

The second phase also executes two parallel MapReduce flows. The first flow, `calcComSnapshotProb`, gets the intermediate community snapshot features that were calculated in the previous phase and calculates for each term $t \in V$ the probability $P(t|D_j)$ according to Eq. 4. The probability values are then stored in HDFS for later use in the last phase. The second flow, `calcInitialTermWeights`, begins with a MapReduce sub-flow that gets as an input several user snapshot features and calculates the initial user profile weights $w_j^u(t)$. The weight values are then both stored in HDFS for the next phase and piped into a second MapReduce sub-flow that aggregates these values and then stores their aggregated value in HDFS for use in the next phase.

Finally, the last phase has a single MapReduce flow, `calcUserProfileTermWeights`, that gets all intermediate values from the previous phases and the history user profile terms $\tilde{w}_{j-1}^u(t)$ kept in HDFS from the previous time period $j-1$, and outputs for each user-profile term, the final term KL weight according to Eq. 6.

4. EXPERIMENTS

We now provide empirical evaluation of our user profiling framework. We first describe the datasets and experimental setting followed by quality evaluation and scalability analysis of the proposed framework.

4.1 Datasets and Experimental Setup

We used two real-world datasets, data extracted from the Open Directory Project (ODP)[5] and blog data extracted from Blogger.com[6]. The ODP data was used for the quality analysis and used to simulate a user content (resulting in 100 users with 100 documents each), and was generated as follow; we randomly selected 100 different categories from the ODP hierarchy. From each category we then randomly selected up to 100 documents, resulting in a collection size of about 10,000 documents. The categories were manually labeled. These ground-truth "correct" labels were later used to evaluate the KL features selection.

For scalability analysis we used the blog data to simulate user document requests to the content provider (in this use-case, Blogger.com); In this dataset, each blog is a collection of posts, where each post may further have some comments attached to it. Users usually do not read all blog posts, but rather few posts from each blog. Therefore, we consider each post with its comments as a single document. Each post is further associated with a unique userid (the post author); in addition, each comment on a post is also associated with some user (commenter). Hence, we consider both the post author and the set of its commenters as users that are associated with that post. In addition, the content of the post is given by the concatenation of the original post with its comments. We crawled 973,518 blog posts from March 2007 until January 2009, with a total collection size of $5.45GB$.

Experiments were run using a 4-nodes commodity machines cluster (each machine with 4GB RAM, 60GB HD, 4 cores), each node with Linux Ubuntu operating system and Hadoop version 0.20.1 installations.

4.2 Quality Analysis

We start with quality analysis of our proposed user profile model, which is based on the KL feature selection method as described in section 3.3.2.

We compared the KL method with two other state-of-the-art feature selection methods namely, mutual information (MI) and χ^2 [11]. We further compared the KL method with the naïve approach which weighs user profile terms according to Eq. 2 (denoted here as tf-udf-idf). We followed the evaluation framework of Carmel et al. [1], where a proposed feature (label) for a given cluster is considered correct if it is identical, an inflection, or a Wordnet synonym of the correct label. We evaluated the features selection performance using the **Match@K** measure, which measures the probability to get at least one correct feature from the set of top-K proposed features (labels) [1]. Figure 3 shows the **Match@K** obtained by the different feature selection methods. We observe that the KL method outperforms the other methods. For example, in at least 50% of the cases, the first label proposed by the KL method was correct (at least 20% better than the other methods).

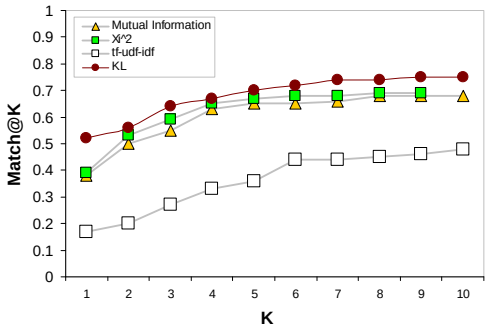

Figure 3: Quality performance of KL, MI, χ^2, and tf-udf-idf feature selection methods.

4.3 Scalability Analysis

We now analyze the scalability of our framework with respect to several parameter settings. We identify two main factors that may affect the performance of Algorithm 1, namely, the number of user profiles we are required to maintain, and the number of documents that are associated with users in the system (data size).

4.3.1 Number of User Profiles

We first analyze the performance with respect to the number of user profiles that the framework needs to maintain. It is worth noting that, when the number of user profiles increases, we expect the number of documents to increase as well. We varied the number of user profiles between 20,000

[5] http://www.dmoz.org/
[6] http://www.blogger.com/

to 120,000, resulting with about 40,000 to 500,000 documents with a total data size of 250MB and 3GB, respectively. We used the 4-nodes cluster and measured the total runtime (in minutes) of the profiling task. Table 1 summarizes the setting and the resulting runtime; Figure 4 further provides relative ratios with respect to the smallest data set (the first row, #1). For example, the user profiles, documents and times ratios for dataset #3 with respect to dataset #1 are 6.60, 4.68 and 4.55, respectively. Interestingly, we can observe that the running times ratio is correlated with the user profiles ratio rather than with the documents ratio, which means that the number of user profiles rather than the number of documents dictates the performance of the algorithm.

Dataset	# user profiles	# documents	Runtime (min.)
#1	17,786	39,596	39.21
#2	34,186	88,563	67.76
#3	83,344	261,509	170.71
#4	124,782	493,490	272.39

Table 1: Runtime performance with respect to increasing number of user profiles.

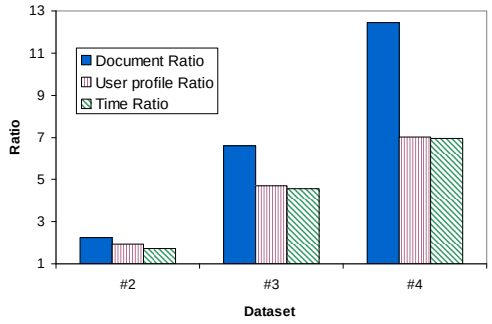

Figure 4: Documents, user profiles and times ratios (with respect to dataset #1).

4.3.2 Data Size

Finally, we analyze the scalability of our framework with respect to the data size, given by the number of documents need to be processed by the framework. We chose a subset of about 180,000 users that existed in the dataset during the two months of March-April 2007, and extracted their associated documents between the months March to August 2007, such that the total number of documents varied between 41,227 to 157,330[7], resulting in a total data size of 250MB to 1.2GB, respectively. We used the 4-nodes cluster and measured the total runtime of the profiling task. Figure 5 provides the runtime for different data sizes. We can observe that the runtime linearly increases with the increase in data size. While here we observe that the data size has linear affect on the runtime, this effect is negligible when the number of user profiles increases as was shown in the first analysis above.

5. CONCLUSIONS

In this paper we proposed a scalable user profiling solution, implemented on top of Hadoop MapReduce framework. Future work will include the extension of our framework with other profile models such as hierarchical or semantic models. We also intend to incorporate structured data sources into our framework.

[7]Please note that some of these users only comment on others blog posts.

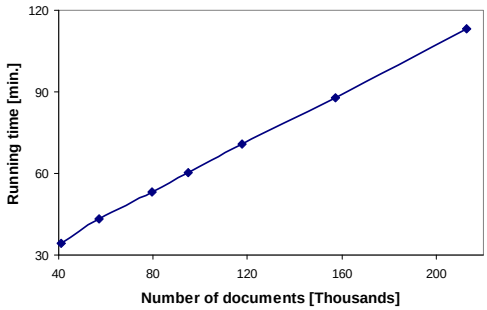

Figure 5: Runtime performance with respect to increasing data size.

Acknowledgment

We thank Avishay Livne for his assistance with the implementation of the initial framework version.

6. REFERENCES

[1] D. Carmel, H. Roitman, and N. Zwerdling. Enhancing cluster labeling using wikipedia. In *SIGIR '09*, pages 139–146, New York, NY, USA, 2009. ACM.

[2] D. Carmel, E. Yom-Tov, A. Darlow, and D. Pelleg. What makes a query difficult? In *SIGIR '06*, pages 390–397. ACM Press, 2006.

[3] U. Cetintemel, M. J. Franklin, and C. L. Giles. Self-adaptive user profiles for large-scale data delivery. In *ICDE*, pages 622–633, 2000.

[4] L. Chen and K. Sycara. Webmate: a personal agent for browsing and searching. In *AGENTS '98*, New York, NY, USA, 1998. ACM.

[5] Y. Chen, D. Pavlov, and J. F. Canny. Large-scale behavioral targeting. In *KDD '09*, New York, NY, USA, 2009. ACM.

[6] S. Gauch, M. Speretta, A. Chandramouli, and A. Micarelli. User profiles for personalized information access. In *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*. Berlin, Heidelberg, 2007.

[7] M. Hinne, W. Kraaij, S. Raaijmakers, S. Verberne, T. van der Weide, and M. van der Heijden. Annotation of urls: more than the sum of parts. In *SIGIR '09*, pages 632–633, New York, NY, USA, 2009. ACM.

[8] H. R. Kim and P. K. Chan. Learning implicit user interest hierarchy for context in personalization. In *In Proc. of International Conference on Intelligent User Interface (IUI)*, pages 101–108, 2003.

[9] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[10] L. Li, Z. Yang, B. Wang, and M. Kitsuregawa. Dynamic adaptation strategies for long-term and short-term user profile to personalize search. In *APWeb/WAIM*, pages 228–240, 2007.

[11] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[12] H. Roitman, D. Carmel, and E. Yom-Tov. Maintaining dynamic channel profiles on the web. *Proc. VLDB Endow.*, 1(1):151–162, 2008.

[13] K. Sugiyama, K. Hatano, and M. Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. In *WWW*, pages 675–684, 2004.

[14] Yanagimoto and S. H. Omatu. User profile creation using genetic algorithm with kullback leibler divergence. *IEEJ Transactions on Electronics, Information and Systems*, 126:389–394, 2006.

[15] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *AAIM '08*, pages 337–348, Berlin, Heidelberg, 2008. Springer-Verlag.