Web-Scale User Modeling for Targeting

Mohamed Aly, Andrew Hatch, Vanja Josifovski, Vijay K. Narayanan
Yahoo! Research
Santa Clara, CA 95051, USA
{aly, aohatch, vanjaj, vnarayan}@yahoo-inc.com

ABSTRACT

We present the experiences from building a web-scale user modeling platform for optimizing display advertising targeting at Yahoo!. The platform described in this paper allows for per-campaign maximization of *conversions* representing purchase activities or transactions. Conversions directly translate to advertiser's revenue, and thus provide the most relevant metrics of return on advertising investment. We focus on two major challenges: how to efficiently process histories of billions of users on a daily basis, and how to build per-campaign conversion models given the extremely low conversion rates (compared to click rates in a traditional setting). We first present mechanisms for building web-scale user profiles in a daily incremental fashion. Second, we show how to reduce the latency through in-memory processing of billions of user records. Finally, we discuss a technique for scaling the number of handled campaigns/models by introducing an efficient labeling technique that allows for sharing negative training examples across multiple campaigns.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous

General Terms

Algorithms, Performance, Experimentation

Keywords

Behavioral targeting, user modeling, advertising

1. INTRODUCTION

While the emergence of online commerce and advertising has allowed advertisers to much more efficiently track the performance of their campaigns, the key goal of online advertising has not changed — advertisers are primarily interested in optimizing return on investment, i.e. getting the highest response from the users by spending as little as possible. In the online environment, ad platforms have traditionally allowed advertisers to *target* users in populations that are likely to respond positively to the provided advertising. Traditionally the methods to specify user populations rely on demographics or location info, where the advertiser chooses a set of locations and certain demographic attributes to select the population *segment* to see the ad. The demographics and location attributes can either be obtained from registration databases or inferred from past user activity. Another traditional online targeting method that uses the past user activity is *behavioral targeting*. Here, instead of bucketing the users into demographics' categories, users are put into predefined interest categories as parenting, auto, health, which are then purchased by the advertisers.

One weakness of the demographic and behavioral targeting is that the segments of users are not created using any feedback from individual advertisers and usually represent large interest groups. For example, a typical behavioral category would be "health and fitness". Such category will combine users interested in a specific brand of cross-country running shoes, as well as others interested in shock absorbent heels. Furthermore, most of the behavioral targeting methods described in the literature are based on clicks, and not on actual purchasing behavior [7, 13, 16]. Online advertising platform can easily track clicks by redirecting the ad landing page urls. However, click behavior might not correspond to commercial activity post click.

The recent trend in display advertising has been for the advertisers to instrument their web sites with embedded code that beacons the user activity on the website to the ad network, allowing thus third parties to capture the user transactions on their sites. These transactions are commonly called *conversions* and can be used to optimize the user segment generation specifically for a given advertiser [6, 3, 4, 9, 14, 2]. Using conversions enables the ad network to optimize for users that are actually inclined to perform a transaction as opposed to a casual (and in some cases random) visit to the web site through a click. In fact, in our recent work [1], we clearly showed that maximizing for clicks does not lead to maximizing for conversions, hence providing evidence for the necessity of developing models that are specifically optimized for conversions. While using advertiser data can improve the performance of campaigns, it also adds to the complexity of the optimization problem, as now the ad-targeting platform needs to find a different segment for each advertiser and thus solve orders of magnitude more optimization problems than in the case of traditional behavioral targeting.

In this paper, we frame the behavioral targeting problem as a per-campaign conversion optimization problem and describe a large-scale machine learning based approach for solving it. We focus on our experience in building a web-scale, production grade, user-modeling platform for improving behavioral targeting, and thus we will not present all the details of the modeling process. We will focus on the design choices that allows for repeatable, per-campaign customization of targeting. More specifically, we will discuss two core, and somewhat contradicting, issues we encountered in building and operating the system: 1) high data volumes and 2) sparseness of conversions.

Processing activities of billions of users on a daily basis imposes many challenges. The first and foremost is, how to build user profiles in an efficient way while efficiently expanding these profiles with the new user daily activities? A second important challenge is, how to cope with the disk input/output overhead when processing billions of users through series of successive operations, such as instance labeling and feature engineering? A third challenge consists of dealing with the problem of optimizing multiple campaigns at the same time, thus, the need for forming positive and negative training instances for each of the campaigns in an independent fashion and the possibility of repetitive processing of users exposed for different campaigns.

The conversion rarity issue requires that we parsimoniously mine the user historical online behavior. We build upon our recent findings [1] on how to represent and combine different user activities such as search queries, page views and ad clicks when modeling user interests, and the optimal user history length to be used in modeling user interests. In this work, we address further important questions needed for building a web-scale user-modeling platform, as for example, how to efficiently determine feature weights? Should user activity counts represent feature weights or more sophisticated techniques are needed? Is feature weight normalization needed? If so, how to normalize? Should all user activities be treated in a similar way regardless of their timings, or should short-term or long-term user history given a higher importance? Is it beneficial to leverage the user activity distributions in feature weighting?

Our main contributions in this paper are as follows:

- We present mechanisms for building web-scale user profiles in a daily incremental fashion.
- We explore the power of in-memory processing to efficiently process billions of user records.
- We describe a technique for scalable labeling of data across thousands of campaigns by sharing negative training examples across multiple campaigns.
- We explore different techniques for feature weighting including using activity counts for weighting and feature weighting based on feature recency. We clearly show the higher importance of recent user history over older user activities with respect to targeting.
- Finally, we show the importance of robust feature selection in optimizing the end-to-end daily running time of our platform without affecting its modeling accuracy.

2. PROBLEM DESCRIPTION

We aim to optimize existing campaigns where the advertisers pay per conversion (or action), commonly known as *cost-per-action* (CPA) campaigns. Each campaign has already been tuned manually by using traditional demographic and behavioral targeting techniques. Our objective is to refine the targeting constraints using the past behavior of the users. Through such refinement, we can improve the number of conversions per ad impression without greatly increasing the number of impressions, which increases the value of the inventory.

The interaction between user's events can be very complex. In some cases the ordering of the events is meaningful (e.g. in most cases users first buy camera and then lenses), while in other cases, the order could be reversed (e.g. running shoes and running shorts). In this work we simplify the model by assuming that the events prior to the conversion contain indications of its occurrence and we do not use any events past the conversion event in our prediction framework. As shown in Figure 1, we consider user history as a sequence of events relative to some *target time*, τ, at which time the user is being considered for targeting. We decompose the user's sequence of events around the target time τ as follows:

$$\begin{aligned} u(\tau) &= (E_F(\tau), E_T(\tau)) \\ \text{where } E_F &= \{e \mid e \in E \wedge t(e) < \tau\} \\ \text{and } E_T &= \{e \mid e \in E \wedge \tau \le t(e) \le \tau + \delta\} \end{aligned}$$

Here $E_F(\tau)$ denotes the events prior to the target time and we call it the *feature window.* $E_T(\tau)$ denotes the events that occur between τ and $\tau+\delta$ and we call it the *target window.* The granularity of both variables is in terms of days.

Hence, when we model behavioral targeting as a machine learning task where each user history forms a training example: a user is a positive example if she is credited to a conversion in the target window, a negative example otherwise.For a given campaign, a user can only be a positive or a negative example, but not both at the same time. Given training users $\{1,\ldots,m\}$, we define $T = \langle (x^1, y^1), ..(x^m, y^m) \rangle \in (\mathbb{R}^n \times \{-1,+1\})^m$ to be the training data, where x^i is a feature vector constructed from the events of the user i in the feature window, $y^i = +1$ if the i^{th} example is positive, and $y^i = -1$ otherwise. The test set is defined similarly.

While we have explored different modeling approaches, in this work, we focus on the user profile generation and report results with using a linear SVM algorithm [15], and focus on the issue of how to leverage user history efficiently. In particular, how to generate user profiles in a form of feature vectors (e.g., representing different events as features, computing their weights, incorporating timestamps) and how to construct target labels (i.e., conversions). We note that the conclusions presented here are valid in a more general context, i.e., with other modeling methods as well.

A user representation method consists of a function $\phi : U \to \mathbb{R}^n$, where $\mathbb{R}^n$ is the Euclidean space of dimension n. A target label function is defined as $\gamma : U \to \{-1,+1\}$. Given this, we extract an appropriate feature and target label set as $(x,y) = (\phi(u), \gamma(u))$. We then select a vector $w \in \mathbb{R}^n$ by solving the following optimization:

$$\arg\min_{w \in R^n} w^2 + C \sum_{i=1}^{m} L(w \,\textbf{.}\, x^i),$$

where $L(\hat{y}, y) = \max(1 - \hat{y}y, 0)$ and C is a constant that controls the balance between regularization and minimizing the loss on the training set.

In this work, we investigate different user representation

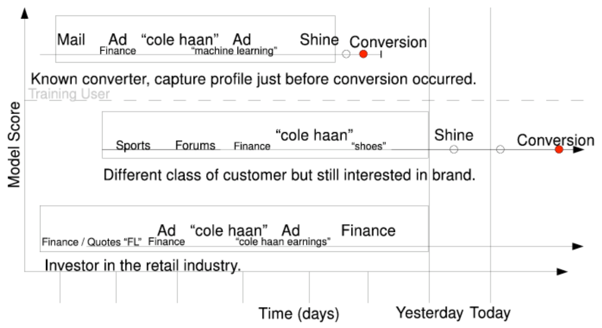
Mail
Ad Finance
"cole haan" "machine learning"
Ad
Shine
Conversion
Known converter, capture profile just before conversion occurred.
Training User
Sports
Forums
Finance
"cole haan" "shoes"
Shine
Conversion
Different class of customer but still interested in brand.
Finance / Quotes "FL"
Ad Finance
"cole haan" "cole haan earnings"
Ad
Finance
Investor in the retail industry.
Model Score
Time (days)
Yesterday
Today

Figure 1: Targeting model is trained on user histories (rectangles) as they existed prior to the start of the conversion process (open circle) that led to the conversion (solid circle). For evaluation, all users are given the same target time (yesterday) and the ad server may choose to show ads at some point in the future to start the conversion process (open circle).

operators $\phi(u)$, basically when converting events to features, computing feature weights, and incorporate timestamps.

3. SYSTEM ARCHITECTURE

We now present the high-level architecture of our platform. Our main design objective is to build a highly flexible and efficient platform to enable repeatable per-campaign customization of targeted advertising. Hence, we design our platform to concurrently customize multiple campaigns based on the advertiser objectives of each campaign. In that regard, agility in feature discovery and testing is the key challenge. Optimally, the platform should allow a fairly quick discovery process, where once initial models are generated, a loop of model evaluation and refinement takes place to tune valuable features for optimizing campaign return.

Figure 2 presents the high-level architecture of our platform. The main components of our platform are: the user profile generator, the campaign manager, the training pipeline, and the scoring pipeline. The user profile generator is the component responsible for building user profiles containing user history coming from different sources, such as content data, social data, and ad logs. User profiles are updated on a daily basis with the user activities on that day.

The campaign manager is the main campaign entry point of our system. Once an advertiser selects to optimize a current campaign through our system, the campaign is added to the campaign manager. The optimization cycle for a given campaign starts from an original user segment specified by the advertiser (possibly users expected to highly convert on this campaign). Campaign optimization can also start from the set of users previously converting for that campaign. Taking both user profiles and ad logs as inputs, the training pipeline is the component responsible for training and generating models highlighting the important features characterizing converters on this campaign. These models are passed to the scoring pipeline to be evaluated on all Yahoo! user base on a daily basis. As a result of this evaluation, the scoring pipeline produces user segments containing users with high probability of converting for this campaign. These segments are then pushed to the ad servers on a daily basis. On a daily basis, the new user ad interaction on this campaign, together with the updated user profiles, are used to tune the valuable features in the current models.

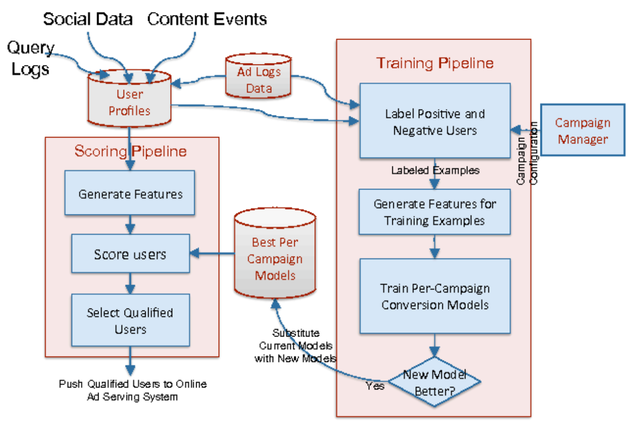

Figure 2: System Architecture

Our platform fully runs on Apache Hadoop [1]. In our design, we determine two granularities of operations: workflows and tasks. We define a task to be a contiguous set of operations to be applied on one (ore more) input data set(s) resulting in one (or more) output data set(s), where both the input and output data sets are stored on Hadoop Data File System (HDFS). The workflow is simply defined as being a set of tasks whose dependence upon each other can be represented as a Directed Acyclic Graph (DAG). For example, the training pipeline can represent one workflow and the scoring pipeline can represent another workflow.

4. USER PROFILE GENERATION

The first challenge for our platform lies in handling the huge amounts of user histories. The ability of handling billions of user records in an efficient way on a daily basis is a key for our platform to successfully train and score per-campaign models for improving targeting in a daily fashion. In this section, we highlight our techniques for building web-scale user profiles in an efficient manner.

4.1 User Representation

We start by describing the main types of user activity in our platform. When collecting the user's historical online behavior, we consider both *active* and *passive* events. Passive events include viewing ads and visiting pages in which an action is not specifically required upon seeing the page. Active events include issuing search queries and clicking ads, i.e., when users actually perform an action on the page. Browsing event is somewhat active because the user took action to visit the page, but we argue that the activity level is less than specifically typing a search query or clicking on an ad. Previously, we showed that the collection of active and passive user events is stronger in predicting the user's propensity to convert on a set of advertisements than any type separately [1].

Events are associated with both a timestamp and metadata. For example, if an event is a visit to a finance web page, the metadata associated with the event would be the content of the page, which is logged separately and then

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

joined with the event along with the anonymized identifier of the user and the time. We consider several different events, each with a corresponding feature extraction method.

- *Pages visited*: Website pages are clustered into several smaller subdirectories. Some features extracted are the id of the cluster and the category of the page from an existing hierarchical page categorizer.
- *Search queries*: Searches issued, clicks on search links, clicks on search advertising links. Some features extracted are the query category, the click information and the unigrams/bigrams in the query. Furthermore, we use the category of the search query from an existing hierarchical query categorizer.
- *Graphical Ads*: Views and clicks on ads. Some features extracted are the ad category, the click information, and the id of the ad. We also use ad view and ad click categories resulting from feeding views and clicks, respectiviely, to an existing hierarchical ad categorizer.

4.2 Incremental User Profile Generation

User profiles are not static — users perform new observable activity daily. To provide up-to-date targeting, the user profiles need to be updated when the new activity is recorded. In addition to adding new activity, due to improve performance and satisfy legal and engineering constraints, activity that is deemed too long in the past is removed from the user profile. A simple way to update the profile is to regenerate it every day. However, this approach would require processing all the raw data for the duration of the observation period and thus it would be prohibitively expensive. Therefore we designed an incremental mechanism for updating the user profiles where user history events are aggregated on the daily level. Here, activities of each day form a daily user profile. In the daily data set, only the count of each specific event is stored, rather than the actual timestamps of the different event occurrences. We develop a user data model consisting of an efficient representation of the user, both in-memory on on-disk. For the on-disk user representation, we use the JSON format to represent the user activities. Each user record is represented by a JSON object, with a set of key-value pairs. Within this object, daily activities are represented through a JSON array, where each day is assigned a JSON object with key-value pairs representing the individual activities that the user did on that day together with their counts. We allow meta-data key-value pairs on both the user level and the activity level. As for the in-memory representation, we use C++ maps (internally implemented as balanced binary search tree) to represent activities, thus maintaining the in-memory activity order. We also maintain the activity order when writing user records to disk.

Recall that we use all user history falling in the feature window, $E_F(\tau)$, in training. Thus, keeping user profiles aggregated on the daily level means that we need to feed $u(\tau)$ (i.e., the sum of the feature window and the target window) daily user profiles as input to the training pipeline on a daily basis. This is definitely very infeasible. For that reason, we developed an efficient mechanism for keeping user profiles aggregated on multiple time granularities, daily, weekly, and monthly. Furthermore, we set each of the user profiles of larger time granularities to apply a daily round robin mechanism to drop the JSON object of the oldest day and substitute it with the one representing today's user activities. Based on the training date, we take the minimum number of user profiles covering the training period, $u(\tau)$, while giving preference to user profiles from larger time granularities in addition to minimizing the coverage intersection (in terms of days) between the different user profiles selected. So, instead of having $u(\tau)$ user records to be joined at the beginning of the training pipeline, we end up joining 2-3 records at most.

The final technique in our incremental user profile generation methodology is to efficiently join user records. As newly generated user daily profiles are in need to be joined daily with user profiles of larger granularities, e.g., weekly and monthly, efficient joining of user records represent a key challenge. For that goal, we leverage Hadoop's map-reduce platform in aggregating users with the same key in one reducer. This can be achieved by setting the number of reducers of the user profile generation task to be constant across all days and applying any type of consistent hashing on the user key to map a given user to the same reducer at every new daily run (i.e., the one with the same number). Once all records of a given user are available at the same reducer, we merge all activities from the different input user records into one (in-memory) user record. We then optimize the round-robin activity maintenance effect to avoid parsing all activities of all user records. Instead, we directly access the first day of history covered by the activities map ($O(\log n)$) and delete it from the map. This drops the overall running time of the round robin process from linear time to logarithmic time.

One of the great advantages of the new efficient technique for incrementally building user profiles on a daily basis is that it allows our platform to build richer and denser user profiles. As user profiles are aggregating logs from different systems/products (e.g., user logs of Yahoo! News, Yahoo! Finance, etc), the ability to join user history from the different sources on a daily basis requires processing of large amounts of data. Using our new scalable user profile generation technique, we are currently able to aggregate all user data into a single user profile in an incremental fashion. From one side, this resulted in richer user profiles as this increased the profile density (we define the *Profile Density* to be the total number of days of history spanned by a user profile). On the other hand, it substantially reduced the total number of profiles to be processed by our platform. Figure 3 compares the profile density distribution of the old and the new versions of our system when building user profiles for two-months worth of user histories. The comparison clearly shows the large effect of our new technique in increasing the profile density of a large amount of our users. This is accompanied by a 32.46% decrease in the total number of resulting user profiles.

5. THE MODULE FRAMEWORK

One of the major challenges imposed by the web-scale volumes of user profiles is the cost of disk input/output. Considering all operations of any typical tasks in our platform, the cost of I/O to HDFS is by far the most expensive in terms of running time. This definitely represents a great roadblock, especially when trying to run our pipelines on a daily basis processing billions of user records. One important observation in that regard is that, most of our operations are independent, that is, an operation is applied on one user record at a time without dependence on any other user records. In light of this observation, we devised a mech-

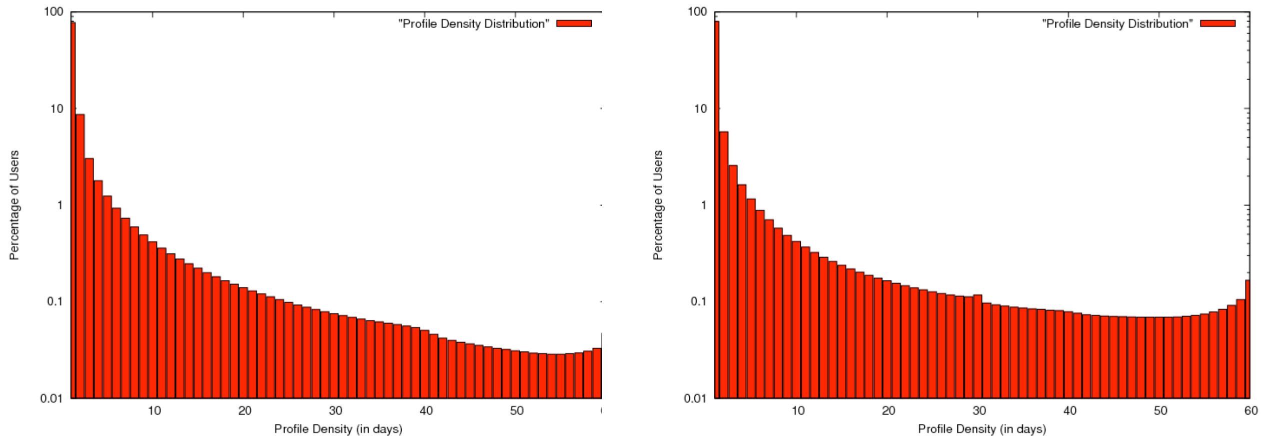

Figure 3: A Comparison of Profile Density Distribution without (left) and with (right) the Incremental Profile Generation Technique

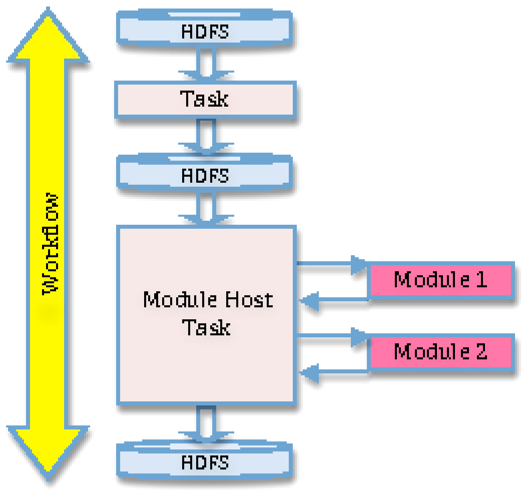

Figure 4: The Module Framework

anism for in-memory processing to minimize the disk I/O operations in any given task.

Figure 4 presents a high level overview of our module framework. Basically, we are introducing a new processing granularity to our platform, which is the *module*. A module is a unit of processing that modifies records in-memory. We defined a standard API for modules to standardize the input and output formats of the module to be a set of one or more records of our standard in-memory user representation. A module is executed by a module-host task that is responsible for loading users from disk, passing them to the module, merging the module output with the original user records (in-memory), and storing the resulting user records back to disk. The module-host task is equipped with the capability of running multiple modules in series and pipelining user records through these modules while maintaining the module running order.

	1 Module	2 Modules	3 Modules
1k users	0.0%	3.0%	5.0%
10k users	9.29%	10.5%	11.2%
100k users	98.5%	100.2%	102.4%

Table 1: Relative difference in running time performance for applying module framework on multiple numbers of modules and multiple sizes of user sets

The module framework has a lot of advantages. The obvious advantage is to reduce workflow running times substantially, due to the ability of performing multiple operations on the same set of user records in-memory while reducing the number of disk I/O operations by a factor of n, where n is the number of modules run in series. Table 1 presents an empirical comparison of running times of our module framework when run on 1, 2, and 3 module with sets of user records of multiple sizes. In this setting, each of the modules performs a typical set of operations requiring the iterative access to all user daily activity objects, as well as to all key-value pairs in each of these daily activities. The numbers clearly show the ability of the module framework to exponentially reduce processing time with the addition of new modules and the increase of the number of processed user profiles. The second major advantage for the module framework is to simplify the plug-in of new operations (mainly feature extraction ones) in our platform with no need for large code changes, e.g., adding the support for new data sources, plugging-in new analysis techniques for user data. This allows our platform to support rapid innovation and shortens the path between experimentation of new techniques and actually getting successful ones into production.

6. TRAINING EXAMPLE LABELING

Our system uses support vector machines (SVMs) to train conversion models for each ad campaign. Each conversion model is trained on a set of positive training examples and on a separate set of negative examples. The positive examples are composed of "converters", that is, users who viewed an ad from a given ad campaign and later converted, while the negative examples are drawn from the set of users who did not convert. In our original system, the negative examples were drawn from the following two groups:

1. "non-converters": Users who viewed an ad but did not convert
2. "non-viewers": Users who did not view an ad from the given ad campaign

While the set of "converters" for a given ad campaign is typically on the order of several thousand unique users in a one-month period, the set of "non-converters" and "non-viewers" is typically on the order of hundreds of millions of unique users. Generating large numbers of examples and training on them can be computationally expensive; hence, the large number of negative examples leads to a potential bottleneck in system scalability and run-time. In experiments, we have found that model accuracy improves as we increase the number of negative examples; however, this improvement typically tops-out at around 200k negative examples. Hence, we can improve system run-time without sacrificing classification accuracy by aggressively subsampling the negative examples. We can further reduce computation time and memory requirements by replacing the set of "non-viewers", which are advertiser dependent, with a random sample of Yahoo! users. Note that this arrangement is somewhat non-ideal, since a random sample will contain a small percentage of "converters", which are used as positive examples. However, we have not seen any appreciable degradation in accuracy from replacing "non-viewers" with a random sample of users.

7. EXPERIMENTAL EVALUATION

We study the performance of our platform compared to the previous version of the system that we developed in [1]. We mainly compare modeling performance in terms of the area under the ROC curve (AUC). To assess the efficiency of the novel engineering techniques applied in our platform (those presented in Sections 4,5,6), we compare both systems in terms of the end to end running time performances of the training and scoring pipelines. Before presenting the performance comparison between the two systems, we present further details about our platform.

7.1 Dataset

In our experiments, we generate user profiles for $7.7B$ users (i.e., browser cookies) spanning two months worth of user history.. We collected 4 weeks of ad data (i.e., impressions, clicks, and conversions) for 1776 campaigns over this set of users. Each campaign is treated as a separate targeting task. 66% of the data is used for training, while the remaining 34% is used for scoring. The train/test split is performed in a random fashion (rather than based on event times such as in [1]). This helps in removing any data dependency between the train and test sets. As our user profiles span 2 months of user history, each training/scoring example is preceded by at least 4 weeks of user events. Negative examples are down-sampled using the same technique presented in Section 6. This benchmark data set enables us to do rigorous offline experiments. The empirical evaluation is based on our baseline. Unless otherwise specified, all metrics are measured as conversion-weighted average of AUC across all campaigns in the benchmark set. For simplicity, we denote the conversion-weighted average of AUC as *Weighted AUC*.

7.2 Model Parameters and Feature Engineering

Our system consists of the application of the support vector machine classifier on a per-campaign basis. The training pipeline is responsible for building the positive/negative training instances for each of the campaigns, then for running the classifier to produce the per-campaign models. Based on some simple experiments, we arrived at the following classification parameters. Because we have a large number of mostly irrelevant features, we choose a strong regularization parameter of $C \in [0.001, 0.05]$. Next, since we have a highly imbalanced class distribution, we choose a weighting parameter for each of the positive and negative classes of any campaign as the reverse of the number of instances from that class for that campaign. In order to have a single parameter configuration for all learning tasks, we sample the negative examples such that the class ratio is always satisfied.

For all feature types, we use a common feature weighting operator $\phi : E^\star \to \mathbb{R}^n$ which we call relative frequency bag of events. The frequency of an event is defined as the number of days in which the user has performed the event. We consider each type of events separately (e.g., page visits, search queries). For example we concentrate on page visits and denote by event p the "visit to any e-mail page" and other page view events by q and r. If the user had visited the pages p, q, and r in a sequence over four days as follows:

$$e = (p_1, p_1, p_2, q_2, r_3, q_4)$$

where each event p_i denotes the visit on page p on day i, then the frequency bag of events representation for page visits would be: $F_p(e) = (p : 2, q : 2, r : 1)$ where we use the convention of $p : n$ to mean that the feature p has the value n in the feature vector. Here, the page p was visited 3 times but on just 2 distinct dates. The relative frequency representation normalizes within each feature type such that the final feature vector is defined as follows:

$$\phi(e) = \left(\frac{F_1(e)}{\|F_1(e)\|}, \dots, \frac{F_n(e)}{\|F_n(e)\|} \right)$$

where n is the number of feature types such as page views, ad views, ad clicks, *etc.*

Another important aspect related to features is that of *feature selection*. In a platform processing billions of user events on a daily basis, there needs to be a mechanism for determining the importance of the different features in terms of targeting. For example, events representing the browsing history corresponding to the email checks are, with a high probability, not of any benefit for advertisers, thus, may be dropped from the user profiles without causing any effect on the modeling accuracy of our platform. To model the importance of the different events, i.e., features, for targeting, we apply feature selection techniques for only including the most *discriminative* features in terms of classification. For our baseline, we include a straightforward feature selection technique that we denote as *Example Existence Thresholding*, where we only include features available in *at least* p negative training examples and q positive training examples.

	Old System	Current Platform
ΔAverage AUC	0.0%	3.1%
ΔTraining Time	0.0%	-70.3%
ΔScoring Time	0.0%	-65.62%

Table 2: Platform Modeling and Running Time Performance

Based on a simple parameter tuning, we select $p = 100$ and $q = 1$ as the thresholding parameters for our baseline.

7.3 Overall Performance

We first study the overall performance of our platform compared to the system we originally developed in [1]. Table 3 presents both the modeling and the running time performances of our platform compared to our old system. As for the modeling performance, it has been computed through an offline stress test analysis on data corresponding to 226 current advertising campaigns. The comparison of the two systems in terms of AUC shows the superiority of our platform over the existing system. As for running times, the analysis shows that our new efficient techniques for user profile generation and handling as well as for the generation of negative instances result in the reduction of the training pipeline running time by 70.3%. As for the scoring pipeline, our new techniques for efficient user profile joining, as well as the power of in-memory processing through the module framework result in a 65.62% reduction in its running time. We deployed our platform to production and achieved a comparable boost in online metrics, such as eCPA, compared to the old system (we don't share these results for privacy reasons).

7.4 Feature Weighting

We now move on to explore the effect of different feature engineering techniques on the performance of our platform. All the experiments are based on using the dataset presented in Section 7.1. Our first track of exploration consists of feature weighting. Driven by the rarity of targets, we study the most efficient techniques for weighting features to optimize the targeting accuracy of our platform.

7.4.1 Weighting by Activity Counts

As described in Section 7.2, our platform uses a common per-feature-type feature representation (i.e., weighting) operator $\phi : E^{\star} \rightarrow \mathbb{R}^n$ which we call relative frequency bag of events. The frequency of an event is defined as the number of days in which the user has performed the event. We now question this feature weighting technique by considering two different techniques that are based on actual user daily activity counts. The two techniques are:

- Weighting by the Sum of Daily Activity Counts: Assigning each feature a weight equal to the total number of times that user performed the activity across all days in the example history period. Thus, $w_f = \sum_{\forall d<td} w_{f,d}$, where w_f is the weight of history event f, d represents any day in which event f is available in the user history, td is the target date of the instance, i.e., the impression date in case of a negative instance and the conversion date in case of a positive instance, and $w_{f,d}$ is the total number of times the user performed event f on day d.

	ΔAUC
Baseline	0.0%
Weighting by the Sum of Daily Activity Counts	-4.26%
Weighting by the Activity Count Logarithm	-0.57%

Table 3: Targeting Accuracy Comparison of the Activity Count Feature Weighting Techniques with the Baseline

- Weighting by the Activity Count Logarithm: Weighting by the logarithm of the total number of times that the user performed the activity throughout the example history period. Thus, $w_f = \sum_{\forall d<td} \log w_{f,d} + 1$. Note that we add 1 to cope for the zero log for the $\forall_{d<td} w_{f,d} = 1$ case.

Table 3 presents the comparison of the baseline performance with those of the activity count feature weighting techniques. The comparison shows that the current relative frequency weighting scheme beats both activity count weighting schemes. Modeling performance is highly degraded when trying to weight features based on the sum of *raw* activity counts throughout the example history periods with a percentage decrease of -4.26%. This is mainly due to the fact that this scheme results in a very wide distribution for feature weights. This complicates the task of the classifier in identifying the discriminative features for each campaign. The rarity of our targets further complicates the classifier task and results in the severe witnessed performance degradation. On the other hand, adding the logarithm operator to the raw activity count helps in reducing the range of possible feature weights. That's why the decrease in performance is much less than in the first case (only -0.57%). Overall, the relative frequency weighting technique performs the best as it bounds the feature weights by the total number of days spanned by the user profiles (In our scheme, that number is L2 normalized to a value $\in [0, 1]$).

7.4.2 Recency-Based Feature Weighting

In this experiment, we move on to study another feature weighting technique. Basically, we investigate whether the recency of the user activity makes a difference in the activity value. For instance, a user may have searched for the query "toyota cars" two days before the target date, while she may have visited "www.orbitz.com" a week before the target date. The question is, if the targeting system is to show this user an advertisement, will it be a car-related one or a travel-related one? We investigate this problem by trying to give more weight to the recent activity in comparison to the older ones (or to older ones over recent ones). Our goal is to understand which history type is more important when it comes to targeting, short-term history or long-term history.

As described before (Section 7.2), we set the weight of each feature in user profile based on the number of days on which the feature is present. Hence, in our baseline, we do not distinguish between the features which occurred in recent days and those which had occurred earlier. For this experiment, to give more weights to recent features we set the weight of feature f to:

$$\sum_{t_i \in <f,u>} \alpha^{-(\tau - t_i)}$$

where τ is the target time (when the prediction is being

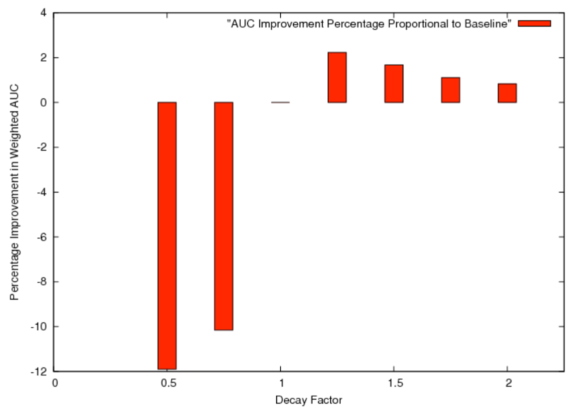

Figure 5: Effect of Recency-Based Feature Weighting on Targeting Accuracy

made), α is the decay factor (or in other words, the decay base) and $<f, u>$ is the sequence of days on which feature f is present in the history of user u. When $\alpha = 1$ the above weighting approach reduces to uniform weighting. By setting α to a larger constant, we can give more weight to recent user activities in comparison to the older ones. Similarly, by setting α to a smaller constant, we favor older user activities over recent ones. Note that the weights are then normalized within each feature type, as before.

In Figure 5, we show the effect of decay factor α on the prediction performance. We report performance with respect to the baseline (i.e., $\alpha = 1$). Results show that the best performance is achieved by $\alpha = 1.25$, that is, when we give a slightly higher preference to recent history over older history. Performance improves by around 2.3% (in terms of weighted AUC) over the baseline, where we are giving similar weight to all activities regardless of their occurrence dates. The performance gain decreases when we increase the value of the decay factor. This shows that, although recent history is more important than older history, we still need to include older history to get the most complete idea about the user. This makes perfect sense as for example, a user who is a great long-term basketball fan will be a perfect candidate to be targeted with offers on game tickets during the time of the NBA playoffs though she may have most of her recent activity related to travel as she is planning for a future vacation. Furthermore, our results show the large performace loss incurred in favoring long-term history over short-term history. This is obvious as the recent history clearly communicates with a high probability the current interests of the user. So, neglecting such history type is definitely not a good idea.

This result contradicts with our previous findings [1] where we have seen that the uniform wheighting of history events, regardless of the event date, performs better than giving more weight to neither recent nor older events. The main explanation for this contradition lies in the improved density of the user profiles achieved by our new platform due to the incremental profile generation technique presented in Section 4.2. As Figure 5 shows, our new scalable profile generation technique helps in improving the density distribution of user profiles. Basically, increasing the history length means we know more about the user. This gives us the power to better model user interests. In our case, this helps us understand better facts about the relative importance of the user history, something that was impossible to achieve with sparse user profiles. This represents a great lesson and a perfect example for the boosted targeting accuracy that could be achieved when applying sophisticated engineering solutions to improve the overall system scalability.

7.5 Feature Selection

Next, we study the effect of feature selection in terms of both modeling accuracy and resulting model sizes. Throughout this section, we denote the "model size" of any campaign as the total number of features available in the trained model for this campaign. We compare different feature selection mechanisms in terms of the conversion-weighted average model sizes. For simplicity, we denote the conversion-weighted average model size for any feature selection scheme as the "weighted model size" for that scheme. Considering the architecture of our platform, we note that a reduction in the weighted model size would result in a comparable reduction in the running time of our daily scoring pipeline (assuming the number of users to be scored is the same). Furthermore, forming a *feature white list* including the top discriminative features and filtering all user events based on such list further reduces the daily scoring time.

A straightforward feature selection technique is to filter feature types with no real value in terms of advertising. As described in Section 4.1, we mainly have three types of features, together with sub-types related to each of them: pages visited, search queries, and graphical ads. For each feature type, sub-types included both *raw* features (actual user events) together with *categorical* features, resulting from feeding events from that feature type to a corresponding hierarchical categorizer and using the resulting categories instead of the actual features. In this experiment, we apply two feature selection techniques based on completely dropping all features belonging to one or more feature sub-types.

Based on an analysis of the feature type sizes, we realized that ad views represent a fairly large percentage of the available user events. This is due to the fact that, with every single page view, many ads are shown to the user. One fact about ad views is that they represent passive user behavior. Furthermore, we can assume the following hypothesis. In the case of a user viewing an ad and interacting with it through clicks/conversions, click features should give the same signal as the ad views. In contrary, if the user just views the ad, this usually tells small information about the user interests. Hence, the first filtering technique that we apply consists of dropping all raw ad views. As for the second filtering technique we apply, we drop all raw features and only keep categorical features.

Table 4 shows the comparison of the two feature type filtering techniques with the baseline. Results show that each of the two techniques is able to achieve more than 70% reduction in weighted model size while dropping the weighted AUC measure by 3.69% and 4.26%, respectively. This indeed verifies that many of our raw features are completely non-discriminative. However, a small percentage of these features are actually important in terms of targeting accu-

	ΔAUC	Δ Weighted Model Size
Baseline	0.0%	0.0%
No Ad Views	-3.69%	-71.41%
Only Categorical Features	-4.26%	-70.88%

Table 4: Targeting Accuracy and Model Size Changes for Different Feature Type Filtering Techniques.

	ΔAUC	Δ Weighted Model Size
Baseline	0.0%	0.0%
(500neg.&1pos.)	-2.22%	-64.52%
(1000neg.&10pos.)	-4.26%	-78.77%

Table 5: Targeting Accuracy and Model Size Changes for Different Example Existence Thresholing Versions

	ΔAUC	Δ Weighted Model Size
Baseline	0.0%	0.0%
MI (top 100)	-10.93%	-99.81%
MI (top 500)	-4.83%	-99.11%
MI (top 1000)	-2.98%	-98.21%
MI (top 10,000)	-0.56%	-82.31%
MI (top 20,000)	-0.28%	-64.81%
MI (top 30,000)	-0.14%	-47.38%
MI (top 40,000)	-0.14%	-30%
MI (top 50,000)	-0.14%	-12.67%
MI (top 100,000)	0%	73.19%

Table 6: Targeting Accuracy and Model Size Changes with Different Mutual Information Versions (Various K Values)

racy. Hence, we move on to consider more advanced feature selection mechanisms.

In the next experiment, explore further the thresholding technique presented in Section 7.2, to filter features in a more rigorous fashion. We consider two versions of the technique by using the following parameter settings: ($p = 500, q = 1$) and ($p = 1000, q = 10$), respectively. Figure 5 shows the performance of the two techniques compared to the baseline. Similar to the feature type filtering results, the performance measures of the two techniques show that a considerable amount of features could be dropped without highly affecting the targeting accuracy.

7.6 Filtering by Mutual Information

Based on the results achieved with the previous two types of feature selection, we move on to study a more standard feature selection technique, which is feature selection through considering the mutual information between features and labels and including discriminative features with a large mutual information with labels. We use the following score to evaluate the importance of a feature:

$$\begin{aligned} I(y, [x]_i) &= H(y) - H(y|[x]_i) \qquad (1) \\ &= H(y) + \sum_{[x]_i} \left[\sum_{y} p(y|[x]_i) \log p(y|[x]_i) \right] \end{aligned}$$

Whenever $y, [x]_i$ have finite (small) joint support, this can be approximated efficiently by using empirical probability estimates. We use the latter as a criterion to order features, on a per campaign basis, then take the top k features in the ordered list to be included in the feature white list. The final feature white list represents the union of the top-k lists for all campaigns modeled by the platform. When training and scoring models, we only include features belonging to the feature white list.

Table 6 shows the results of various versions of the mutual information selection technique based on using different values for parameter k. Results show that, with a small k value of only 500, we can perform 4.83% less than the baseline in terms of weighted AUC while reducing the weighted model size by 99.11%. Increasing the value of k to be a multiple of $10,000$ achived almost the same tagreting accuracy as the baseline with a drop in weighted AUC ranging from 0.56% for $k = 10,000$ to 0.14% with $k \in [20,000, 50,000]$, while achieving a reduction in weighted model size ranging from 82.31% to 12.67%, respectively. This result shows that, increasing the feature white list size beyond a specific threshold achieves very little improvement in modeling accuracy. Finally, increasing the k value to be $100,000$ actually reaches an equal targeting accuracy to that of the baseline while increasing the weighted model size by 73.19%. This clearly shows that a huge number of features were added on top of the ones already selected by the baseline without achieving any improvement in terms of targeting accuracy.

Note that, in addition to mutual information, we applied feature selection based on another standard feature selection technique, namely ℓ_1 regularization. Results achieved by the latter in terms of both targeting accuracy and weighted model size were comparable to the results achieved by feature selection through mutual information. However, the end-to-end running times of the feature selection through mutual information were slightly better. In production, we select to use a mixture of both feature selection techniques where we apply a coarse feature selection through mutual information, then we apply a rigorous feature selection through ℓ_1 regularization. This turns out to achieve the best performance in terms of both targeting accuracy and end-to-end pipeline running times.

8. RELATED WORK

Targeting users who will convert is a difficult problem and often the problem is divided into predicting clicks, and predicting the probability that the click will convert [9, 6, 14]. The advantage of this division relates to business logic: the publisher (such as Yahoo! or Google) has data about how likely users are to follow various paths towards clicks on advertisements on their site. On the other hand, advertisers have more information about the paths of users on their website.

In this paper, we focused on building predictive models for user conversions. We compared our approach with existing behavioral targeting methods (such as [7, 16]) which optimize for click-through rates and showed how optimizing directly for conversions can lead to improved performance. Compared to previous work on conversion optimization [6, 3, 4, 9, 14], our work makes several new contributions: we look into understanding the effect of different user activities on prediction, give insights about the temporal aspect of user behavior (recency vs. long-term trends) and explore different variants (user representation and target label) through large offline and online experiments.

9. CONCLUSION

Behavioral targeting platforms use user's historical activity to find responsive audiences for display advertising campaigns. Unlike the traditional model of crafting audiences based on the data available to the publishers as page views and search queries, a new trend has recently emerged where advertiser specific information is used to customize the audience selection for each advertiser. In this paper, we presented some of the lessons learned from building a web-scale user-modeling platform for audience selection. Our main design objective is to build a highly flexible and efficient platform to enable repeatable (per-campaign) customization of targeted advertising. Our platform faces two major challenges:1) the web-scale volumes of user history to be processed on a daily basis; and 2) the rarity of targets, i.e., conversions.

To address these challenges, we first develop an efficient technique for building user profiles in an incremental fashion. Additionally, we develop a user processing pipeline where the users are partitioned among machines and each user data is passed in-memory through a series of modules. Last but not least, we develop an efficient technique for reusing negative instances across campaigns. Both our offline and online results show that these techniques bring a substantial improvement in the efficiency of the pipeline.

We use a large scale real world benchmark to show how the proposed approach scales in terms of number of customized campaigns and conducted a rigorous empirical study of different modeling techniques aiming at addressing the main challenges faced by our platform. We model the performance of different feature weighting schemes such as feature weighting based on actual user daily activity counts and feature weighting based on recency (to favor short term or long term history). Our results show the relative higher importance of short-term over long-term user history when it comes to targeting. Additionally, we develop a feature selection mechanism based on both mutual information and ℓ_1 regularization that achieves a substantial reduction in both the resulting model sizes and the end-to-end running times without affecting the targeting accuracy of our platform.

Acknowledgment

The authorship of this paper reflects the contributions of the authors to the paper. The work presented in this paper is a result of a large team effort across multiple organizations at Yahoo!, including the Yahoo! Labs, Yahoo! Advertising Engineering and Product Management groups. We are thankful for the opportunity to have worked with those teams to deploy this state-of-the-art display advertising platform to production.

10. REFERENCES

[1] S. Pandey, M. Aly, A. Bagherjeiran, A. O. Hatch, P. Ciccolo, A. Ratnaparkhi, and M. Zinkevich. Learning to Target: What Works for Behavioral Targeting. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management*, 2011.

[2] A. Ahmed, Y. Low, M. Aly, V. Josifovski, and A. J. Smola. Scalable Distributed Inference of Dynamic User Interests for Behavioral Targeting. In *Proceedings of the of the 17th SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011.

[3] A. Bagherjeiran, A. O. Hatch, and A. Ratnaparkhi. Ranking for the Conversion Funnel. In *Proceeding of the 33rd SIGIR conference on Research and development in information retrieval*, 2010.

[4] A. Bagherjeiran, A. O. Hatch, A. Ratnaparkhi, and R. Parekh. Large-Scale Customized Models for Advertisers. In *ICDM Workshops*, 2010.

[5] A. Hatch, A. Bagherjeiran, and A. Ratnaparkhi. Clickable Terms for Contextual Advertising. In *ADKDD*, 2010.

[6] N. Archak, V. S. Mirrokni, and S. Muthukrishnan. Mining Advertiser-Specific User Behavior using Adfactors. In *Proceedings of the 19th International World Wide Web Conference*, 2010.

[7] Y. Chen, D. Pavlov, and J. Canny. Large-Scale Behavioral Targeting. In *Proceedings of KDD*, 2009.

[8] I. Click Forensics. Click Fraud Index. `http://www.clickforensics.com/resources/click-fraud-index.html`, 2010.

[9] Google, Inc. Google Analytics. `http://www.google.com/analytics`.

[10] I. Nielsen Company. Nielsen Claritas PRIZM. `http://en-us.nielsen.com/tab/product_families/nielsen_claritas/prizm`.

[11] Y. Peng, L. Zhang, M. Chang, and Y. Guan. An Effective Method for Combating Malicious Scripts Clickbots. In *Proceedings of the 14th European Symposium on Research in Computer Security*, 2009.

[12] B. J. Pine. Mass Customizing Products and Services. *Strategy & Leadership*, 21(4):6 – 55, 1993.

[13] F. Provost, B. Dalessandro, R. Hook, X. Zhang, and A. Murray. Audience Selection for Online Brand Advertising: Privacy-Friendly Social Network Targeting. In *Proceedings of the 15th SIGKDD international conference on Knowledge discovery and data mining*, 2009.

[14] B. Rey and A. Kannan. Conversion rate based bid adjustment for sponsored search auctions. In *Proceedings of the 19th International World Wide Web Conference*, 2010.

[15] X.-R. Wang, K.-W. Chang, C.-J. Hsieh, R.-E. Fan, G.-X. Yuan, H.-F. Yu, F.-L. Huang, and C.-J. Lin. Liblinear – A Library for Large Linear Classification. `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`.

[16] J. Yan, N. Liu, G. Wang, W. Zhang, Y. Jiang, and Z. Chen. How Much can Behavioral Targeting Help Online Advertising? In *Proceedings of the 18th International Conference on World Wide Web*, 2009.

[17] L. Zhang and Y. Guan. Detecting Click Fraud in Pay-Per-Click Streams of Online Advertising networks. In *Proceedings of the 28th IEEE International Conference on Distributed Computing Systems*, 2008.