Simple and Scalable Response Prediction for Display Advertising

OLIVIER CHAPELLE, Criteo
EREN MANAVOGLU, Microsoft
ROMER ROSALES, LinkedIn

Clickthrough and conversation rates estimation are two core predictions tasks in display advertising. We present in this article a machine learning framework based on logistic regression that is specifically designed to tackle the specifics of display advertising. The resulting system has the following characteristics: It is easy to implement and deploy, it is highly scalable (we have trained it on terabytes of data), and it provides models with state-of-the-art accuracy.

Categories and Subject Descriptors: H.3.5 [**Information Storage and Retrieval**]: Online Information Services; I.2.6 [**Artificial Intelligence**]: Learning

General Terms: Algorithms, Experimentations

Additional Key Words and Phrases: Display advertising, machine learning, click prediction, hashing, feature selection, distributed learning

ACM Reference Format:
Olivier Chapelle, Eren Manavoglu, and Romer Rosales. 2014. Simple and scalable response prediction for display advertising. ACM Trans. Intell. Syst. Technol. 5, 4, Article 61 (December 2014), 34 pages.
DOI: http://dx.doi.org/10.1145/2532128

1. INTRODUCTION

Display advertising is a form of online advertising in which advertisers pay publishers for placing graphical ads on their web pages. The traditional method of selling display advertising has been prenegotiated long-term contracts between advertisers and publishers. In the past decade, spot markets have emerged as a popular alternative due to the promise of increased liquidity for publishers and increased reach with granular audience targeting capabilities for the advertisers [Muthukrishnan 2009].

Spot markets also offer advertisers a wide range of payment options. If the goal of an advertising campaign is getting their message to the target audience (for instance, in brand awareness campaigns) then paying per impression (CPM) with targeting constraints is normally the appropriate choice for the advertiser. However, many advertisers would prefer not to pay for an ad impression unless that impression leads the user to the advertiser's website. Performance-dependent payment models, such as Cost-per-Click (CPC) and Cost-per-Conversion (CPA), were introduced to address this concern. Under the CPC model, advertisers will only be charged if users click on their ads. The CPA option reduces the advertiser's risk even further by allowing them to pay only if the user takes a predefined action on their website (such as purchasing a product or subscribing to an email list). An auction that supports such conditional

---

Most of this work was done while the authors were at Yahoo! Labs.
Authors' address: emails: o.chapelle@criteo.com, ermana@microsoft.com, rrosales@linkedin.com.

DOI: http://dx.doi.org/10.1145/2532128

payment options needs to convert advertiser bids to *Expected* price per impression (eCPM). For CPM ads, eCPM is identical to the bid. The eCPM of a CPC or CPA ad, however, will depend on the probability that the impression will lead to a click or a conversion event. Estimating these probabilities accurately is critical for an efficient marketplace [McAfee 2011].

There has been significant work in the literature on modeling clicks in the context of search and search advertising. Click and conversion prediction for display advertising presents a different set of challenges. In display advertising, the auctioneer does not readily have access to the content of the ads. Ads might not even be hosted by the auctioneers. Furthermore, the content of the ads might be generated dynamically, depending on the properties of the user. Similarly, landing pages of the ads may not be known to the auctioneer, and/or they might contain dynamically generated content. Although there have been some attempts to capture the content of the ad or the landing page recently [Cheng et al. 2012; Liu et al. 2012], this requires nontrivial effort and is not always possible. Content-related, web graph, and anchor text information are therefore absent in display advertising, leaving the auctioneer mostly with unique identifiers for representation. Despite almost 10 billion ad impressions per day in our dataset, hundreds of millions of unique user ids, millions of unique pages, and millions of unique ads, combined with the lack of easily generalizable features, makes sparsity a significant problem.

In this article, we propose a simple machine learning framework that can scale to billions of samples and hundreds of millions of parameters, and that addresses the issues just discussed effectively and with a small memory footprint. Our proposed framework uses Maximum Entropy [Nigam et al. 1999] (also known as Logistic Regression) because it is an easy to implement regression model that, as will be seen, can appropriately scale with respect to the number of features and can be parallelized efficiently. The Maximum Entropy model is advantageous also because incremental model updates are trivial and there exists an exploration strategy that can be incorporated easily. A two-phase feature selection algorithm is provided to increase automation and reduce the need for domain expertise: We use a generalized mutual information method to select the feature groups to be included in the model and *feature hashing* to regulate the size of the models.

Large-scale experimental results on live traffic data show that our framework outperforms the state-of-the-art models used in display advertising [Agarwal et al. 2010]. We believe these results and the simplicity of the proposed framework make a strong case for its use as a baseline for response prediction in display advertising.

The rest of the article is organized as follows: In Section 2, we discuss related work. In Section 3, we look at the differences between click and conversion rates with respect to features and analyze the delay between clicks and conversions. Section 4 describes the Maximum Entropy model, the features, and the hashing trick used in our framework. In this section, we also show that smoothing and regularization are asymptotically similar. We present results of the proposed modeling techniques in Section 5. Section 6 introduces a modified version of mutual information that is used to select feature groups and provides experimental results. In Section 7, motivated by our analysis, we propose an algorithm for exploration. Section 8 describes an efficient map-reduce implementation for the proposed model. Finally, we summarize our results and conclude in Section 9.

2. RELATED WORK

Learning methods developed for response prediction in computational advertising often use regression or classification models in which all factors that can have an impact on user's response are included explicitly as features or covariates.

The features used in such models can be categorized as:

—*Context features,* such as the query in sponsored search or the publisher page in content match;
—*Content features,* as discussed in Ciaramita et al. [2008] and Hillard et al. [2010] for text ads and in Cheng et al. [2012] and Liu et al. [2012] for display ads;
—*U*ser features, introduced by Cheng and Cantú-Paz [2010]; and
—*feedback features* that are generated by aggregating historical data and described in Chakrabarti et al. [2008] and Hillard et al. [2010].

Not all of these features are available in the context of display advertising, in which advertiser and publisher information is typically represented as unique identifiers. In such a space where each unique identifier is considered a feature, dimensionality becomes a major concern. Mutual information and similar filter methods [Guyon and Elisseeff 2003] are frequently used over wrapper-based methods in this domain. A recent approach to handle dimensionality reduction is the use of feature hashing as described in Weinberger et al. [2009]. The idea behind hashing is to reduce the number of values a feature can take by projecting it to a lower dimensional space. This approach has gained popularity in the context of large-scale machine learning due to its simplicity and empirical efficacy.

Logistic regression and decision trees are popular models used in the computational advertising literature. Applying decision trees to display advertising, however, has additional challenges due to having categorical features with very large cardinality and the sparse nature of the data. Kota and Agarwal [2011] use gain ratio as the splitting criterion, with a threshold on positive responses as an additional stopping criterion to avoid having too many nodes with zero positives. These authors then perform autoregressive Gamma-Poisson smoothing, top-down, to fall back to parent nodes' response rates. Decision trees are sometimes used in multilayer approaches due to high computational cost.

Logistic regression is frequently preferred because it can be parallelized easily to handle large-scale problems. Agarwal et al. [2011] present a new framework to parallelize linear models, one that is shown to reduce training times by an order of magnitude. Graepel et al. [2010] propose the use of an online Bayesian probit regression model. This approach maintains a posterior distribution over model parameters instead of point estimates. This posterior distribution is then used as the prior in the next update cycle. The authors also suggest sampling from the posterior distribution as a means of exploration (a method known as Thompson sampling), but they do not provide any analysis or empirical results using this methodology. In our framework, we employ similar techniques within the logistic regression framework, and we provide an analysis of the Thompson sampling method, as well as simulation results.

Predicting user response with sparse data has been the focus of several studies. However, most of this work was conducted on search advertising and therefore utilized either the query [Ashkan et al. 2009; Hillard et al. 2011] or the keywords that advertisers bid on [Regelson and Fain 2006; Richardson et al. 2007]. More recently, Agarwal et al. [2010] and Kota and Agarwal [2011] proposed using hierarchies present in the data to address this issue explicitly. This approach uses a baseline model trained on standard set of features to estimate baseline probabilities and then learns a correction factor by exploiting the hierarchical nature of publisher- and advertiser-related features. The model uses the assumption that the correction factors for the full advertiser and publisher paths can be estimated by employing a log-linear function of pairs of nodes, where the pairwise state parameters are modeled as Poisson distributions. It then uses noncentered parametrization to take advantage of the hierarchical correlations in estimating these state parameters. This is required because many states

will have very few observations and no positive samples. The approach in Menon et al. [2011] incorporated this hierarchical smoothing technique with the matrix factorization methods used in collaborative filtering and reported additional gains. Our approach, is to instead encode the relations between features, hierarchical or otherwise, directly in the same model as additional *conjunction* features. In Section 4.8, we show that using Tikhonov regularization with logistic regression implicitly achieves the effect of hierarchical smoothing. The framework proposed in this article therefore uses conjunction features and regularization to exploit the relationship between features and hashing to keep the model size tractable. We argue that our method requires less domain knowledge (e.g., we do not need to know the structure of ad campaigns or publisher pages to encode the relations), has better scaling properties, and is simpler to implement.

3. DATA AND FEATURES

This section provides some background information on the display advertising data used in this article. More details can be found in Agarwal et al. [2010] and Rosales et al. [2012].

3.1. Data and The Yahoo! Real Media Exchange

For this work, we collected live traffic logs from Yahoo!'s Right Media Exchange (RMX), one of the largest ad exchanges in the world, which serves around ten billion ad impressions every day. RMX follows a *network-of-networks* model in which the connections between advertisers and publishers are facilitated by intermediaries called networks. Networks can have either only publishers or advertisers, or both. Every entity in the exchange (networks, publishers, and advertisers) has a unique identifier (ID).

Publishers label their web pages with site IDs. They can also tag different parts of the same page with a different section ID. Although the exchange does not enforce any rules on the semantics of section IDs, publishers often use a different section ID for different ad slots on the same page. A relationship between a network and an advertiser or a publisher is specified by a series of attributes such as pricing type, budget, and targeting profile.

3.2. Click-Through Rate and Conversion Rate

We are interested in uncovering the fundamental properties of the click and Post-Click Conversion (PCC) events. In the context of this article, *click* refers to the event that occurs when a user interacts with an ad by means of a mouse click (or equivalent). *PCC,* conversely, is an action that the user takes after visiting the landing page of the advertiser and is regarded as valuable by that advertiser. Typical examples of conversion actions include subscribing to an email list, making a reservation, or purchasing a product. The basic quantities to measure the properties of these two types of events are the Click-Through Rate (CTR) and the Conversion Rate (CVR), respectively.

3.3. Features/Covariates in Display Advertising

In this section, we summarize the information sources utilized for modeling CTR and CVR.

We consider four sets of features that are available in most online advertising systems: *advertiser*, *publisher*, *user*, and *time* features. It is worth mentioning that the availability of the user features varies by publisher. A representative list of features for each group can be found in Table I. Some of these features are continuous rather than categorical (user age for instance) and have been quantized appropriately.

These features are obtained at the event level; that is, every time an ad is shown on a page and every time a user clicks on an ad or converts, an event will be recorded.

Table I. Sample of Features Considered Divided into Feature Families

Feature Family	Feature Members
Advertiser	advertiser (ID), advertiser network, campaign, creative, conversion id, ad group, ad size, creative type, offer type ID (ad category)
Publisher	publisher (ID), publisher network, site, section, URL, page referrer
User (when avail.)	gender, age, region, network speed, accept cookies, geo
Time	serve time, click time

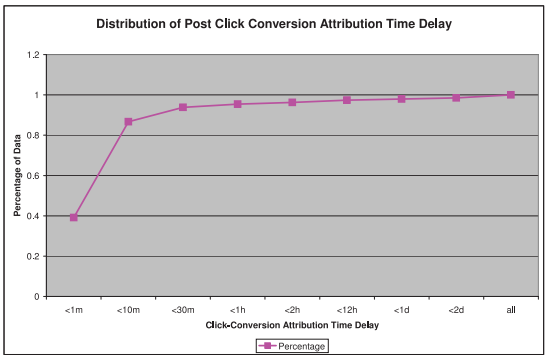

Fig. 1. Distribution of click conversion attribution time delay.

3.4. Analysis of the Click→Conversion Delay

In this article, the conversion estimation task is defined as the probability that a user will convert after they click on an ad (*PCC*). A critical aspect in PCC modeling is identifying the click events that lead to conversion actions (such as subscribing to an email list, making a reservation, or purchasing a product).

To build a conversion model, we need to attribute the conversion event to the correct click event because this represents a positive PCC example (vs. a click event without any associated conversion event). A conversion event can happen minutes, hours, or even days after a click event. The proper attribution of the conversion event to the right click event, which can be done by properly matching the click and conversion event attributes, is essential not only for PCC prediction but also for payment processing.

In general, several conversion events could be associated with the same click because advertisers may show a sequence of conversion-generating pages to the user after a click on the relevant ad. This association process faces certain practical limitations because the longer the time elapsed between click and conversion, the more logged events that need to be maintained and matched. To get a better picture of the click conversion process and to answer the question of how much data need to be utilized for matching conversions with their corresponding click events, we analyzed the properties of the time delay for conversion events.

We calculated the percentage of conversion events with different attribution time intervals, as shown in Figure 1. From the graph, we can observe that a large majority (86.7%) of conversion events are triggered within 10 minutes of the click events. Approximately half of these conversion events (39.2%) occur within 1 minute of the corresponding click event. If we look further, we observe that we can match 95.5% of the conversion events within one hour of the clicks. Because we are interested in achieving

the largest possible recall within practical boundaries, we decided to consider various days of delay. Within two days of the click, 98.5% of the conversions can be recovered.

These considerations are important for modeling and, in particular, for building proper training/test sets. The use of too short a time window would generate inaccurate datasets, whereas the use of too large a time window is impractical because the requirements for data storage and matching (click conversion) could become prohibitive. Given the just described experiments and data, we would be ignoring approximately 1.5% of the conversion events and, as a consequence, incorrectly labeling a click event as negative (no conversion) if the time frame set for post click conversion attribution is limited to two days. This was believed to be sufficient, given the practical cost of looking further in time; thus, we utilize this limit throughout the article.

In contrast to conversions, a click can be relatively easily attributed to a particular ad impression because there is a direct association between a click with the page where the ad is displayed. The large majority of ad clicks (97%) occur within a minute of the ad impression.

4. MODELING

This section describes the various modeling techniques used to learn a response predictor from clicks or conversions logs.

4.1. Logistic Regression

The model considered in this article is Maximum Entropy [Nigam et al. 1999] (a.k.a. Logistic Regression [Menard 2001]).

Given a training set $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i \in \{0,1\}^d$ a sparse binary feature vector in a d-dimensional space, and $y_i \in \{-1, 1\}$ a binary target, the goal is to find a weight vector[1] $\mathbf{w} \in \mathbb{R}^d$. The predicted probability of an example $\mathbf{x}$ belonging to class 1 is:

$$\Pr(y = 1 \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})}.$$

The logistic regression model is a linear model of the *log odds ratio*:

$$\log \frac{\Pr(y = 1 \mid \mathbf{x}, \mathbf{w})}{\Pr(y = -1 \mid \mathbf{x}, \mathbf{w})} = \mathbf{w}^\top \mathbf{x}. \tag{1}$$

The weight vector $\mathbf{w}$ is found by minimizing the negative log likelihood with an L_2 regularization term on the weight vector:

$$\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^{n} \log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i)). \tag{2}$$

Equation (2) is a convex, unconstrained and differentiable optimization problem. It can be solved with any gradient-based optimization technique. We use L-BFGS [Nocedal 1980], a state-of-the-art optimizer. A review of different optimization techniques for logistic regression can be found in Minka [2003].

4.2. Categorical Variables

All the features considered in this article are categorical. Indeed, most features are identifiers (ID of the advertiser, of the publisher, etc.). And real valued features can be made categorical through discretization.

The standard way of encoding categorical features is to use several binary features, known as *dummy coding* in statistics or the 1-of-c encoding in machine learning. If

[1]The intercept is one of the weights.

the feature can take c values, c binary features are introduced, the i^{th} one being the indicator function of the i^{th} value. For instance, a feature taking the second value out of five possibles values is encoded as

$$(0, 1, 0, 0, 0).$$

If we have F features and the f-th feature can take c_f values, this encoding will lead to a dimensionality of

$$d = \sum_{f=1}^{F} c_f.$$

4.3. Hashing Trick

The issue with the dummy coding just presented is that the dimensionality d can get very large when there are variables of high cardinality. The *hashing trick*, made popular by the *Vowpal Wabbit* learning software and first published in Weinberger et al. [2009], addresses this issue.

The idea is to use a hash function to reduce the number of values a feature can take. We still make use of the dummy coding described in the previous section, but instead of a c-dimensional code, we end up with a d-dimensional one, where d is the number of bins used with hashing. If $d < c$, this results in a compressed representation. In that case, collisions are bound to occur, but, as explained later, this is not a major concern.

When dealing with several features, there are two possible strategies:

(1) Hash each feature f into a d_f-dimensional space and concatenate the codes, resulting in $\sum d_f$ dimensions.
(2) Hash all features into the same space; a different hash function is used for each feature.

We use the latter approach because it is easier to implement.[2] A summary of the hashing trick is given in Algorithm 1. The values v_f can be of any type; the hash function only acts on their internal representation. In Algorithm 1, there is a hashing function h_f for every feature f. This can be implemented using a single hash function and having f as the seed or concatenating f to the value v_f to be hashed.

ALGORITHM 1: Hashing Trick

Require: Values for the F features, $v_1, \ldots, v_F$.
Require: Family of hash function h_f, number of bins d.
  $x_i \leftarrow 0, \quad 1 \leq i \leq d.$
  for $f = 1 \ldots F$ **do**
    $i \leftarrow [h_f(v_f) \mod d] + 1.$
    $x_i \leftarrow x_i + 1$
  end for
  Return $(x_1, \ldots, x_d)$.

There are some other ways of reducing the dimensionality of the model, such as discarding infrequent values, but they require more data processing and the use of a dictionary. A very practical appeal of the hashing trick is its simplicity: It does not require any additional data processing or data storage, and it is straightforward to implement.

[2]This is also what is implemented in Vowpal Wabbit.

4.4. Collision Analysis

We quantify in this section the log-likelihood degradation due to two values colliding into the same bin.

Consider the scenario when the first value has been observed n_1 times, all on negative examples, and the second value n_2 times, all on positive examples, and when there is only one feature in the system (no fallback on other features). If there were no collision, the weight for the value would be $-\infty$ (assuming no regularization), and the weight for the second value would be $+\infty$. This would lead to a log likelihood of zero on all the examples where either value is present.

When there is a collision, the negative log likelihood is:

$$-n_1 \log \frac{n_1}{n_1 + n_2} - n_2 \log \frac{n_2}{n_1 + n_2}.$$

This is indeed the log likelihood achieved by the solution where all the weights are at 0 except the one where there is a collision and which has a value $\log(n_2/n_1)$.

This log likelihood is large only when both n_1 and n_2 are large. This scenario can be considered a worst-case one because:

(1) The two values were extremely predictive: If these two values were not predictive, their collision would not harm the log likelihood (zero weight in all cases);
(2) There was no redundancy in features: If the system includes redundant features, a collision on one value could be mitigated by another value.

Regarding the last point, one could alleviate the collision issue by making use of multiple hash functions, in the same spirit as in the Bloom filter [Bloom 1970]. However, in practice, this does not improve the results (see Section 5.7).

This section provides only a rudimentary analysis of collisions. Given the recent interest in hashing for categorical variables, we expect and hope that a more thorough and theoretical analysis of the effect of collisions within a learning system will be developed in the years to come.

4.5. Conjunctions

A linear model can only learn effects independently for each feature. For instance, imagine a model with two categorical features, advertiser and publisher. The model would learn that some advertisers and some publishers tend to have a higher CTR than others, but it would not learn that the CTR of a `bank of america` ad is particularly high on `finance.yahoo.com`. For this, we need to introduce a new *conjunction* feature that is the cartesian product of advertiser and publisher.

A conjunction between two categorical variables of cardinality c_1 and c_2 is just another categorical variable of cardinality $c_1 \times c_2$. If c_1 and c_2 are large, the conjunction feature has high cardinality, and the use of the hashing trick is even more crucial in this case.

Note that computing conjunctions between all features is equivalent to considering a polynomial kernel of degree 2 [Schölkopf and Smola 2001] where the mapping is computed explicitly, as in Chang et al. [2010].

Due to the large cardinality of the representation, there will most likely be pairs of variable values that are unseen in the training data. And these pairs, take (advertiser, publisher) for instance, are biased by the current serving scheme because specific advertisers are selected for a given publisher. The hashing trick helps reduce the dimensionality of data but might do a poor job on these unseen pairs of values due to collisions. This can be problematic, especially in an exploration setting where predictions on infrequent values are required. A possible solution is to represent the variable values using a low-dimensional representation, for example through the use of matrix

factorization or a related approach [Menon et al. 2011]. This type of representation is not studied in this article. A promising future work direction would be to combine a low-dimensional representation with the hashing trick: The former would capture the general trend in the data, while the latter could be used to refine the model for the frequent pairs observed in the training set.

4.6. Multitask Learning

We show in this section how multitask learning, as presented in Evgeniou and Pontil [2004], is equivalent to a single model with conjunctions, as presented in this article.

Assume that we are given T learning tasks indexed by $t \in \{1, 2, \ldots, T\}$ and a training set $(\mathbf{x}_1^t, y_1^t), \ldots, (\mathbf{x}_{n_t}^t, y_{n_t}^t)$ for each of these tasks. The tasks are supposed to be different but related. A task could, for instance, learn a prediction function in a given country. Evgeniou and Pontil [2004] adapted Support Vector Machines to multitask learning by decomposing the weight vector for task t as

$$\mathbf{w}_t = \mathbf{w}_0 + \mathbf{v}_t,$$

where $\mathbf{w}_0$ can be interpreted as a global classifier capturing what is common among all the tasks, and $\mathbf{v}_t$ is a small vector modeling what is specific for that task.

The joint optimization problem is then to minimize the following cost:

$$\sum_{t=1}^{T} \sum_{i=1}^{n_t} \ell\left((\mathbf{w}_0 + \mathbf{v}_t)^\top \mathbf{x}_i^t\right) + \lambda_0 \|\mathbf{w}_0\|_2^2 + \lambda_1 \sum_{t=1}^{T} \|\mathbf{v}_t\|_2^2, \tag{3}$$

where ℓ is a given loss function.

Note that the relative value between λ_0 and λ_1 controls the strength of the connection between the tasks. In the extreme case, if $\lambda_0 \to \infty$, then $\mathbf{w}_0 = 0$, and all tasks are decoupled; on the other hand, when $\lambda_1 \to \infty$, we obtain $\mathbf{v}_t = 0$, and all the tasks share the same prediction function.

Instead of explicitly building these classifiers, an equivalent way of obtaining the same result is to optimize a single weight vector $\mathbf{w} := [\mathbf{w}_0^\top \mathbf{v}_1^\top \ldots \mathbf{v}_T^\top]^\top$ and introduce conjunction features between the task and all the other features. The following vector is thus implicitly constructed: $\tilde{\mathbf{x}}_i^t := [\mathbf{x}_i^{t\top} \mathbf{0}^\top \ldots \mathbf{x}_i^{t\top} \mathbf{0}^\top]^\top$. Both approaches are indeed equivalent when $\lambda = \lambda_0 = \lambda_1$ since $(\mathbf{w}_0 + \mathbf{v}_t)^\top \mathbf{x}_i^t = \mathbf{w}^\top \tilde{\mathbf{x}}_i^t$ and $\lambda \|\mathbf{w}\|_2^2 = \lambda_0 \|\mathbf{w}_0\|_2^2 + \lambda_1 \sum_{t=1}^{T} \|\mathbf{v}_t\|_2^2$. The case $\lambda_0 \neq \lambda_1$ can be handled by putting a weight $\sqrt{\lambda_0/\lambda_1}$ on the conjunction features.[3]

As already noted in Weinberger et al. [2009], the use of conjunction features is a powerful tool because it encompasses the multitask learning framework. Its main advantage is that a specific multitask solver is unnecessary: A single model is trained with a standard logistic regression solver.

4.7. Subsampling

Since the training data are large (around 9B impressions daily), it would be computationally infeasible to consider all impressions.[4] On the other hand, the training data are very imbalanced – the CTR is lower than 1%. For these reasons, we decided to subsample the negative class at a rate $r \ll 1$.

[3]One way to see that is to perform the change of variable $\mathbf{v}_t \leftarrow \sqrt{\lambda_1/\lambda_0}\mathbf{v}_t$.

[4]Infeasible on a single machine but would in fact be possible with the distributed learning system described later.

The model has, of course, to be corrected for this subsampling. Let us call $\Pr'$ the probability distribution after subsampling. Then:

$$\frac{\Pr(y=1 \mid \mathbf{x})}{\Pr(y=-1 \mid \mathbf{x})} = \frac{\Pr(\mathbf{x} \mid y=1)\Pr(y=1)}{\Pr(\mathbf{x} \mid y=-1)\Pr(y=-1)} \tag{4}$$

$$= \frac{\Pr'(\mathbf{x} \mid y=1)\Pr'(y=1)}{\Pr'(\mathbf{x} \mid y=-1)\Pr'(y=-1)/r} \tag{5}$$

$$= r\frac{\Pr'(y=1 \mid \mathbf{x})}{\Pr'(y=-1 \mid \mathbf{x})} \tag{6}$$

These equations rely on the fact that the class conditional distributions are not affected by the subsampling: $\Pr(\mathbf{x} \mid y) = \Pr'(\mathbf{x} \mid y)$. Combining Equations (1) and (6) results in the log odds ratio being shifted by $\log r$. Thus, after training, the intercept of the model has to be corrected by adding $\log r$ to it. Logistic regression with unbalanced classes has been well studied in the literature; more details can be found in King and Zeng [2001] and Owen [2007].

Instead of shifting the intercept, another possibility would be to give an importance weight of $1/r$ to the negatives samples correct for the subsampling. Preliminary experiments with this solution showed a lower test accuracy. A potential explanation is that generalization error bounds are worse with importance weighting [Cortes et al. 2010].

4.8. Regularization and Smoothing

4.8.1. Single Feature. We draw in this section a connection between Tikhonov regularization as used in this article and Laplace smoothing.

Suppose that our model contains a single categorical feature. Let us consider the j-th value of that feature:

$$w^* = \arg\min_w \sum_{i \in I_j} \log(1 + \exp(-y_i w)) + \frac{\lambda}{2} w^2 \tag{7}$$

with $I_j = \{i, x_i = j\}$. When there is no regularization ($\lambda = 0$), the closed form solution for w^* is:

$$w^* = \log \frac{k}{m-k} \text{ with } k = |\{i \in I_j, y_i = 1\}| \text{ and } m = |I_j|.$$

This leads to a prediction equal to k/m, which is the empirical probability $P(y = 1|x = x_j)$. We have just rederived that the logistic loss yields a Fisher consistent estimate of the output probability.

This empirical probability may have a large variance when m is small; thus, people often use a beta prior to get a biased but lower variance estimator:

$$\frac{k+\alpha}{m+2\alpha}. \tag{8}$$

This estimator is referred to as the Laplace estimator.

The regularizer in Equation (7) is another way of smoothing the probability estimate toward 0.5. These two methods are generally not equivalent, but they are related. The following proposition shows that the smoothing is similar asymptotically.

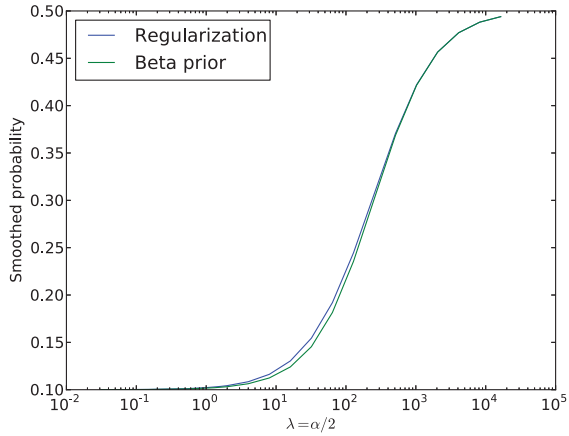

Fig. 2. Smoothed probability computed from 100 success and 1,000 trials. The smoothing is achieved by regularization, as in Equation (7) or a beta prior, as in Equation (8).

PROPOSITION 4.1. *When $\alpha \to \infty$ and $\mathbf{w}^*$ is the minimizer of Equation (7) with $\lambda = \alpha/2$, the following asymptotic equivalence holds:*

$$\frac{1}{1+\exp(-w^*)} - \frac{1}{2} \sim \frac{k+\alpha}{m+2\alpha} - \frac{1}{2}$$

PROOF. At the optimum, the derivative of Equation (7) should be 0:

$$-k\frac{\exp(-w^*)}{1+\exp(-w^*)} + (m-k)\frac{1}{1+\exp(-w^*)} + \lambda w^* = 0.$$

As $\lambda \to \infty$, $w^* \to 0$ and the above equality implies that

$$w^* \sim \frac{2k-m}{2\lambda}.$$

We also have

$$\frac{1}{1+\exp(-w^*)} - \frac{1}{2} \sim \frac{w^*}{4},$$

and

$$\frac{k+\alpha}{m+2\alpha} - \frac{1}{2} \sim \frac{2k-m}{4\alpha}.$$

Combining these three asymptotic equivalences yields the desired result. □

This proposition is illustrated in Figure 2: Regularization and smoothing with a beta prior give similar smoothed probabilities.

4.8.2. Hierarchical Feature. Consider now the case of two features having a hierarchical relationship, such as advertiser and campaign. Suppose that we have a lot of training data for a given advertiser, but very little for a given campaign of that advertiser. Because of the regularization, the weight for that campaign will be almost zero. Thus, the predicted CTR of that campaign will mostly depend on advertiser weight. This is similar to the situation in the previous section except that the output probability is not smoothed toward 0.5 but toward the output probability given by the parent feature.

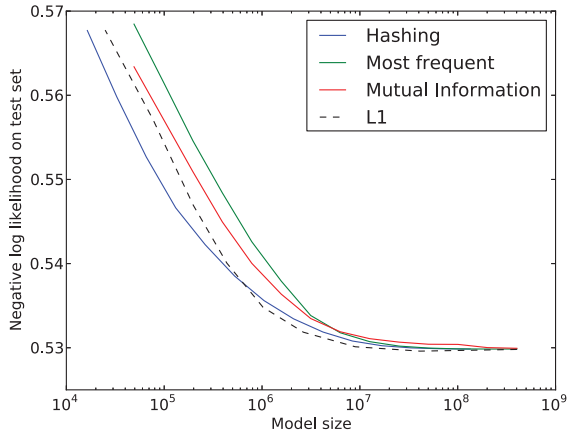

Fig. 3. Log likelihood on a test as a function of the model size for different dimensionality reduction strategies.

This kind of hierarchical smoothing is common in language models [Chen and Goodman 1999]. See also Gelman and Hill [2006] for a review of multilevel hierarchical models and Agarwal et al. [2010] for hierarchical smoothing in display advertising.

The advantage of the logistic regression approach with regularization is that it *implicitly* performs hierarchical smoothing: Unlike the works just mentioned, we do not need to specify the feature hierarchy.

4.9. Bayesian Logistic Regression

As we will see in Section 7, it is convenient to have a Bayesian interpretation of logistic regression [Bishop 2006, Section 4.5].

The solution of Equation (2) can be interpreted as the Maximum a Posteriori (MAP) solution of a probabilistic model with a logistic likelihood and a Gaussian prior on the weights with standard deviation $1/\sqrt{\lambda}$. There is no analytical form for the posterior distribution $\Pr(\mathbf{w} \mid D) \propto \prod_{i=1}^{n} \Pr(y_i \mid \mathbf{x}_i, \mathbf{w}) \Pr(\mathbf{w})$, but it can be approximated by a Gaussian distribution using the Laplace approximation [Bishop 2006, Section 4.4]. That approximation with a diagonal covariance matrix gives:

$$\Pr(\mathbf{w} \mid D) \approx \mathcal{N}(\mu, \Sigma) \text{ with } \mu = \arg\min_{\mathbf{w}} L(\mathbf{w}) \text{ and } \Sigma_{ii}^{-1} = \frac{\partial^2 L(\mathbf{w})}{\partial w_i^2},$$

and $L(\mathbf{w}) := -\log \Pr(\mathbf{w} \mid D)$ is given in Equation (2).

5. MODELING RESULTS

5.1. Experimental Setup

The experimental results presented in this section have been obtained in various conditions (training set, features, evaluation). Most of them were based on the RMX data described in Section 3, but some of the experiments—those in Figures 3, 4, and 5 and Section 5.6—have been run on logs from Criteo, a large display advertising company.

Even though the details of the experimental setups varied, they were all similar, with the following characteristics. The training and test sets are split chronologically, both with periods ranging from several days to several weeks. After subsampling the negative samples as explained in Section 4.7, the number of training samples is of the order of one billion. The number of base features is about 30, from which several

hundred conjunctions are constructed. The number of bits used for hashing was 24, resulting in a model with 16M parameters.

Depending on the experiment, the test metrics are either the negative log likelihood (NLL), root mean squared error (RMSE), area under the precision/recall curve (auPRC), or area under the ROC curve (auROC). When a metric is said to be *normalized*, this is the value of the metric relative to the best constant baseline.

5.2. Hashing Trick

The first component of our system that we want to evaluate is the use of the hashing trick. Recall that because we use features that can take a large number of values (especially the conjunction features), running the logistic regression without dimensionality reduction is infeasible.

An alternative to the hashing trick is to keep only the most important values. Coming back to the example of Section 4.2, if a feature can take five values, but only two of them are deemed important, we would encode that feature with two bits, both of them being zero when the feature value is not important.

Two simple heuristics for finding important values are:

—*Count:* Select the most frequent values.
—*Mutual information:* Select the values that are most helpful in determining the target.

A drawback of this approach is that the model needs to be stored as a dictionary instead of an array. The dictionary maps each important value to its weight. The dictionary is needed to avoid collisions and to keep track of the most important values. Its size is implementation-dependent, but a lower bound can be 12 bytes per entry: 4 for the key, 4 for the value, and 4 for the pointer in a linked list. Empirically, in C#, the size is about 20–25 bytes per entry.

Figure 3 shows the log likelihood as a function of the model size: $4d$ for hashing and $12d$ for dictionary based models, where d is the number of weights in the model. It turns out that for the same model size, the hashing based model is slightly superior. Two other advantages are *convenience* (there is no need to find the most important values and keep them in a dictionary) and *real-time efficiency* (hashing is faster than a dictionary look-up).

For the sake of the comparison, Figure 3 also includes a method whereby the most important values are selected through the use of a sparsity-inducing norm. This is achieved by adding an L_1 norm on $\mathbf{w}$ in Equation (2) and minimizing the objective function using a proximal method [Bach et al. 2011, chapter 3]. Each point on the curve corresponds to a regularization parameter in the set $\{1, 3, 10, 30, 300, 1000, 3000\}$: the larger the parameter, the sparser the model. The L_2 regularization parameter is kept at the same value as for the other methods. Even though the resulting model needs to be stored in a dictionary, this method achieves a good tradeoff in terms of accuracy versus model size. Note, however, that this technique is not easily scalable: During training, it requires one weight for each value observed in the training set. This was feasible in Figure 3 because we considered a rather small training set with only 33M different values. But in a production setting, the number of values can easily exceed a billion.

5.3. Effect of Subsampling

The easiest way to deal with a very large training set is to subsample it. Sometimes similar test errors can be achieved with smaller training sets, and there is no need for large-scale learning in these cases.

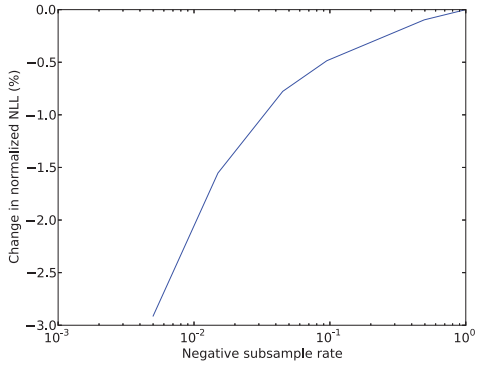

Fig. 4. Change in log likelihood by subsampling the negatives. The baseline is a model with all negatives included.

Table II. Test Performance Drop as a Function of the Overall Subsampling Rate

	1%	10%
auROC	−2.0%	−0.5%
auPRC	−7.2%	−2.1%
NLL	−3.2%	−2.3%

Two different levels of subsampling be done: Subsampling the negatives, as discussed in Section 4.7, or subsampling the entire dataset. Both speed up the training considerably, but can result in some performance drop, as shown later. The choice of the subsampling rates is thus an application-dependent tradeoff to be made between computational complexity and accuracy.

5.3.1. Subsampling the Negatives. In this experiment, we keep all the positives and subsample the negatives. Figure 4 shows how much the model is degraded as a function of the subsampling rate.

We found empirically that a subsample rate of 1% was a good tradeoff between model training time and predictive accuracy. Unless otherwise noted, the negatives are subsampled at 1% in the rest of this paper.

5.3.2. Overall Subsampling. The entire dataset has been subsampled in this experiment at 1% and 10%. The results in Table II show that there is a drop in accuracy after subsampling.

In summary, even if it is fine to subsample the data to some extent—the negatives in particular—the more data, the better, and this motivates the use of the distributed learning system presented in Section 8.

5.4. Comparison with a Feedback Model

An alternative class of models used for response prediction is the *feedback* models described in Hillard et al. [2010, Section 4.1]. In that approach, response rate is not encoded in the weights but in the feature values via the use of *feedback features*. This approach relies on the observation that the average historical response rate of an ad is a good indicator of its future response rate.

Feedback features are derived by aggregating the historical data along various dimensions (such as ads, publishers, or users) at varying levels of granularity. The corresponding response rate (CTR or CVR) and its support (i.e., number of impressions or clicks) are the feature values used in modeling. When the support is under some

Table III. Comparison with a Click Feedback Model

auROC	auPRC
+0.9%	+1.3%

threshold, the historical response rate is typically considered to be undefined because the estimate would be unreliable.

The response rate of an ad might change depending on the publisher page it is shown on or the users who see it. To capture the variation of response rate across different dimensions, multiple attributes along different dimensions could be used to generate composite feedback features (similar to conjunction features), such as publisher-advertiser and user-publisher-creative composites.

Feedback features are quantized using a simple k-means clustering algorithm [MacQueen 1967] (with a special cluster ID indicating the undefined values) before they are fed to the logistic regression algorithm. The size of the final models is therefore typically small. However, additional memory is needed to store the feature ID to feedback feature value mappings. Note that feedback features are often refreshed regularly by updating the statistics with the latest historical information available.

One potential weakness of a feedback feature is that it may give an incorrect signal because of confounding variables. Thus, it is preferable, as advocated in this article, to directly model the response as a function of all variables and not to perform any aggregation.

However, from a practical standpoint, a model based on feedback features may be preferred in cases where the cost of updating the model is substantially higher than the cost of updating the feedback features.

We used a relatively small number of features to do the comparison between our proposed model based on categorical features and a model based on feedback features. Results in Table III show that both models are similar, with a slight advantage for our proposed model. The difference would likely be larger as the number of features increases: The model based on categorical variables would better model the effect of confounding variables.

5.5. Comparison with a Hierarchical Model

Next, we compare our approach to the state-of-the-art Log-linear Model for Multiple Hierarchies (LMMH) method [Agarwal et al. 2010] that has been developed in the same context: CTR estimation for display advertising with hierarchical features. The LMMH approach exploits the hierarchical nature of the advertiser and publisher attributes by explicitly encoding these relations. It splits the modeling task into two stages: First, a feature-based model is trained using covariates, without hierarchical features. In the second stage, the publisher (e.g., publisher type→publisher ID) and advertiser hierarchies (e.g., advertiser ID→campaign ID→ ad ID) are used to learn the correction factors on top of the baseline model. Expected number of clicks, E, for each publisher and advertiser hierarchy is calculated by summing up the baseline probabilities over all the samples in which that publisher and advertiser hierarchy is present. Correction factors of the full hierarchies are modeled as log-linear functions of pairs of advertiser and publisher nodes (e.g., {publisher type, advertiser ID}):

$$\lambda_{h_i} = \prod_{a_i \in h_{i,ad}} \prod_{p_j \in h_{i,pub}} \phi_{a_i,p_j} \tag{9}$$

where λ_{h_i} is the multiplicative correction factor for the hierarchy pair h_i, $h_{i,ad}$ and $h_{i,pub}$ are advertiser and publisher hierarchies for that pair, and ϕ_{a_i,p_j} is the state parameter

Table IV. Comparison with LMMH

auROC	auPRC	NLL
+3.1%	+10.0%	+7.1%

Table V. Improvement in Log Likelihood on the Training Set as Function of the Number of Hash Functions, Total Number of Bits Remaining Constant

2	4	8	12
−0.21%	+0.44%	+0.13%	−0.14%

associated with the advertiser node a_i and publisher node p_j. The clicks (or conversions) per hierarchy pair, C_{h_i} are assumed to be conditionally independent, given ϕ and E_{h_i}, and are modeled using a Poisson distribution. The log likelihood of ϕ under the Poisson assumption is given by

$$l(\phi) = \sum_{h_i} (-E_{h_i}\lambda_{h_i} + \log(\lambda_{h_i})C_{h_i}) + constant. \tag{10}$$

The LMMH approach takes further advantage of the hierarchical structure by sharing ancestor parameters among descendant nodes when estimating ϕ. The closer the child nodes are to each other, the closer their correction factors will be to the parent.

Results presented in Table IV show improvements in all metrics. This result could be explained by our model's ability to take advantage of all relations, jointly.

5.6. Value of Publisher Information

We now provide an analysis of the relevance of publisher information for both click-through and post-click conversion predictions.

For this, we built, both for CTR and CVR predictions, a model without any publisher features, and we compared it with a model including publisher features.

The model with publisher features improves the normalized negative log likelihood by 52.6% for CTR prediction, but only by 0.5% for CVR prediction. It makes sense that the publisher features are very useful for CTR prediction because it helps in ad matching. But once the user clicks on the ad, the publisher's page does not have much impact on that user's conversion behavior. Potentially, the PCC could thus be considerably simplified because no publisher-side information would be needed.

Our results differ from those reported in Rosales et al. [2012]. The PCC experiments in that study show larger improvements (5.62% in auROC). One hypothesis explaining this behavior is that the coverage and reliability of the user features are different in these two datasets. The dataset used in the previous analysis includes many samples in which the user features were not available. The dataset used in the current experiments, by contrast, contain more reliable user information. In the absence of explicit user attributes, publisher information might serve as a proxy for the user features. The website for Disney games will surely attract more kids than adults with no kids.

5.7. Multiple Hash Functions

A collision between two frequent values can lead to a degradation in the log likelihood. One idea to alleviate this potential issue is to use several hash functions, similarly to what is done in a Bloom filter [Bloom 1970]: Each value is replicated using different hash functions. A theoretical analysis of learning with multiple hash functions is provided in Shi et al. [2009, theorem 2].

Table V shows that using multiple hash functions does not result in any significant improvements. This could be explained by the fact that the conjunctions already induce redundancies. All other experiments in this article use a single hash function.

6. FEATURE SELECTION

We tackle in this section the problem of feature selection for categorical variables.

Let us first distinguish two different levels of selection in the context of categorical variables:

—*Feature selection.* The goal is to select some of the features (such as age, gender, or advertiser) to be included in the learning system. This is the purpose of this section.
—*Value selection.* The goal is to reduce the number of parameters by discarding some of the least important values observed in the training set. Section 5.2 compares various schemes for this purpose.

L_1 regularization is an effective way of reducing the number of parameters in the system and is thus a *value selection* method. But it does not necessarily suppress all the values of a given feature and cannot therefore be used for *feature selection*. For this, a possible regularization is the so-called ℓ_1/ℓ_2 regularization or group lasso [Meier et al. 2008]. However, group lasso would be computationally expensive at the scale of data at which we operate. We present in this section a cheaper alternative.

As for mutual information and other *filter* methods for feature selection [Guyon and Elisseeff 2003], we are looking for a criterion that measures the utility of a feature in a binary classification task. However, the goal is to do so conditioned on already existing features; that is, we want to estimate the additional utility of a feature when added to an already existing classifier [Koepke and Bilenko 2012].

6.1. Conditional Mutual Information

We assume that we already have an existing logistic regression model with a base set of features. Let s_i be the score predicted by this model.

The negative log likelihood of the current model is:

$$\sum_{i=1}^{n} \log(1 + \exp(-y_i s_i)). \tag{11}$$

Let us approximate the log-likelihood improvement we would see if we were to add that feature to the training set.

A first approximation is to say the weights already learned are fixed and that only the weights associated with the new features are learned. For the sake of simplicity, let's say the new feature takes d values and that these values are $\mathcal{X} = \{1, \ldots, d\}$. We thus need to learn d weights $w_1, \ldots, w_d$, each of them corresponding to a value of the feature. The updated prediction with the new feature on the i-th example is then: $s_i + w_{x_i}$.

They are found by minimizing the new log likelihood:

$$\sum_{i=1}^{n} \log(1 + \exp(-y_i(s_i + w_{x_i}))),$$

which can be decomposed as:

$$\sum_{k=1}^{d} \sum_{i \in I_k} \log(1 + \exp(-y_i(s_i + w_k))), \quad \text{with } I_k = \{i, x_i = k\}. \tag{12}$$

This is an easy optimization problem since all w_k are independent of each others:

$$w_k = \arg\min_w \underbrace{\sum_{i \in I_k} \log(1 + \exp(-y_i(s_i + w)))}_{L_k(w)}.$$

There is, however, no closed form solution for w_k. A further approximation is to set w_k to the value after one Newton step starting from 0:

$$w_k \approx -\frac{L'_k(0)}{L''_k(0)}. \tag{13}$$

The first and second derivatives of L_k are:

$$L'_k(0) = \sum_{i \in I_k} p_i - \frac{y_i + 1}{2} \quad L''_k(0) = \sum_{i \in I_k} p_i(1 - p_i) \quad \text{with } p_i = \frac{1}{1 + \exp(-s_i)}.$$

Once the w_k are computed, the log-likelihood improvement can be measured as the difference of Equations (12) and (11) or using the second-order approximation of the log likelihood:

$$\sum_{k=1}^{d} w_k L'_k(0) + \frac{1}{2} w_k^2 L''_k(0).$$

Special case. When s is constant and equal to the log odds class prior, there is a closed form solution for w_k. It is the value such that

$$\frac{1}{1 + \exp(-s + w_k)} = \frac{1}{|I_k|} \sum_{i \in I_k} \frac{y_i + 1}{2} = \Pr(y = 1 \mid x = k).$$

With that value, the difference in the log likelihoods in Equations (12) and (11) is:

$$\sum_{i=1}^{n} \log \frac{\Pr(y_i \mid x_i)}{\Pr(y_i)}, \tag{14}$$

which is exactly the *mutual information* between the variables x and y. The proposed method can thus be seen as an extension of mutual information to the case where some predictions are already available.

6.2. Reference Distribution

As always in machine learning, there is an overfitting danger: It is particularly true for features with a lot of values. Such features can improve the likelihood on the training set, but not necessarily on the test set. This problem has also been noted with standard mutual information [Rosales and Chapelle 2011]. To prevent this issue, a regularization term, λw_k^2, can be added to L_k. And as in Rosales and Chapelle [2011], instead of computing the log-likelihood improvement on the training set, it can be done on a separate reference set.

The issue of computing the mutual information on the training set can be illustrated as follows: Consider a feature taking unique values (it can, for instance, be an event identifier), then the mutual information between that feature and the target will be maximum since the values of that feature can fully identify the data point and therefore its label. Formally, Equation (14) turns out to be in this case:

$$\sum_{i=1}^{n} \log \frac{1}{P(y_i)} = H(Y),$$

since $\Pr(y_i \mid x_i) = 1$ if all x_i are unique. And, of course, the mutual information can never be larger than the entropy. This example shows that the mutual information is not a good indicator of the predictiveness of a feature because it is biased toward features with high cardinality. Computing the mutual information on a validation set addresses this issue.

6.3. Results

We utilized the method just described for determining feature relevance for click prediction. Our main motivation was decreasing model complexity because the available number of possible features is too large to be used in their entirety during prediction/modeling. Practical considerations, such as memory, latency, and training time constraints, make feature selection a clear requirement in this task.

The evaluation is divided into two parts: We first verify that computing the mutual information using a reference distribution gives more sensible results than the standard mutual information; then, we evaluate the use of the *conditional* mutual information in a forward feature selection algorithm.

6.3.1. Use of a Reference Distribution. For these experiments, we considered one day of data for the training distribution and one day for the reference distribution.

Our goal is to identify predictive features in the most automated manner possible (reducing time spent by people on this task). Thus, practically all the raw (unprocessed) data attributes available were included in the analysis. These features are a superset of those in Table I and include identifiers for the actual (serve/click/conversion) event, advertiser, publisher, campaign, cookies, timestamps, advertiser/publisher specific features, related URLs, demographics, user-specific features (identifiers, assigned segments), and the like. We consider conjunctions of any of these features, giving rise to about 5,000 possible compound features in practice. Each feature in turn can take from two to millions of possible values.

We remark that wrapper methods are rarely appropriate in this setting because they require training using a large set of variables; this is usually impractical except for some simple models. It is in this setting that filter methods, such as the MI methods described here, can be more advantageous.

We applied the Standard MI (SMI) ranking algorithm for feature selection. The results, summarized in Table VI (top) reflect our main concern. The spurious features, or features that are informative about the data point *per se,* rank substantially high. The calculated MI score is correct in that it reflects the information content of these features; however, these features are too specific to the training data distribution.

The proposed extension of the MI score utilizing a reference distribution (RMI) provides a more appropriate ranking, as shown in Tables VI (mid-bottom) because the information content is calculated with respect to (expectations on) the reference distribution; thus, feature values that are not seen in the new distribution are basically considered less important and their impact on the information score is reduced.

More specifically, attributes such as `event_guid` that identifies the data point have maximal information content according to the training distribution (SMI), but near zero information content when calculated with a reference distribution (RMI). A similar effect was observed for other features that have low relevance for prediction, such as `query_string` and `receive_time` which, unless parsed, are too specific; `xcookie` and `user_identifier`, which clearly do not generalize across users (but could be quite informative about a small fraction of the test data); and `user_segments`, which indexes user categories. The results for other features are more subtle but follow the same underlying principle in which a reference distribution is utilized to avoid spurious dependencies often found when utilizing empirical distributions.

Table VI. Top Features for Click Prediction Along with Their Mutual Information

Single Feature	SMI (bits)
event_guid	0.59742
query_string	0.59479
xcookie	0.49983
user_identifier	0.49842
user_segments	0.43032
Single feature	**RMI (bits)**
section_id	0.20747
creative_id	0.20645
site	0.19835
campaign_id	0.19142
rm_ad_grp_id	0.19094
Conjunction feature	**RMI (bits)**
section_id x advertiser_id	0.24691
section_id x creative_id	0.24317
section_id x IO_id	0.24307
creative_id x publisher_id	0.24250
creative_id x site	0.24246
site x advertiser_id	0.24234
section_id x pixeloffers	0.24172
site x IO_id	0.23953
publisher_id x advertiser_id	0.23903

First table: standard mutual information; second and third table: modified mutual information (RMI). Bottom section contains the top conjunction features.

6.3.2. Learning Performance Results. Here, we explore the question of automatically finding new conjunction features and whether these features actually offer any performance gains. For this, we use the conditional mutual information within a forward feature selection algorithm:

(1) Start with a set of base features and no conjunction features;
(2) Train a model with all the selected features;
(3) Compute the conditional mutual information for all conjunctions not yet selected;
(4) Select the best conjunction;
(5) Go back to (2).

The results of this procedure are shown in Figure 5. It can be seen that selecting 50 features in this way has a twofold advantage over including the 351 possible conjunctions: It results in a model with fewer features (faster to train and evaluate), and it generalizes better.

7. NONSTATIONARITY

Display advertising is a nonstationary process because the set of active advertisers, campaigns, publishers, and users is constantly changing. We first quantify these changes in Sections 7.1 and 7.2, then show in Section 7.3 how to efficiently update the model to take into account the nonstationarity of the data. We discuss Thompson sampling in Section 7.4 and later as a way to address the explore/exploit tradeoff necessary in this kind of dynamic environment.

7.1. Ad Creation Rates

Response prediction models are trained using historical logs. When completely new ads are added to the system, models built on past data may not perform as well,

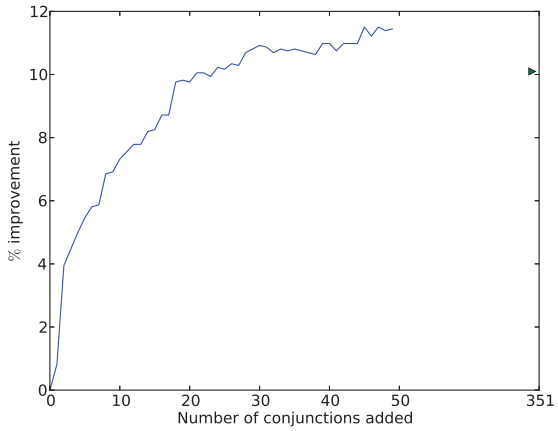

Fig. 5. A base model was trained with 26 base features. The method of Section 6 was then used to determine the best conjunctions to add to the model. The y-axis shows the relative improvement in normalized RMSE on a test set. Adding in all conjunctions (green triangle) results in 351 new features but does not yield better performance than the top 50 conjunction features.

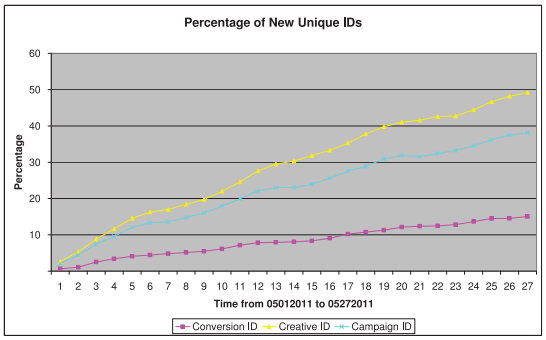

Fig. 6. Percentage of new unique conversion identifiers, creatives, and campaigns emerging for each day in one month (relative to those existing in the previous month).

particularly for those new ads (this clearly depends on the generalization power of the features utilized for modeling). Keeping the models fresh can therefore be critical to achieving sustained performance.

To design a model update mechanism (i.e., to be able to cope with the dynamic nature of campaign generation) with a reasonable tradeoff between model performance and computational cost, it is important to investigate the rate at which new ads are introduced. To this end, we used one month of data as the reference period and computed the percentage of new ads introduced on every day of the following month. We considered three levels of identification for ads: conversion, creative, and campaign identifiers (these also turned out to be the most informative advertiser-based features in the models studied).

Figure 6 shows that the percentage of unique new ads in the data is increasing steadily day-by-day for all three identifiers. However, there is a difference in the magnitude of change. Creatives are observed to change most frequently. There are 19.7% new creatives after 10 days and 39.8% after 20 days. New ad campaigns also increase steadily, but at a slower rate, by approximately 16% after 10 days and 31% after 20 days. Conversion identifiers are the most stable of the three, with only 5.4% of new unique

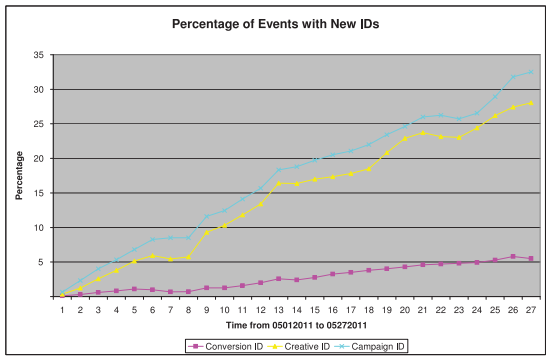

Fig. 7. Percentage of events with new conversion identifiers, creatives, and campaigns for each day in one month (relative to those existing in the previous month).

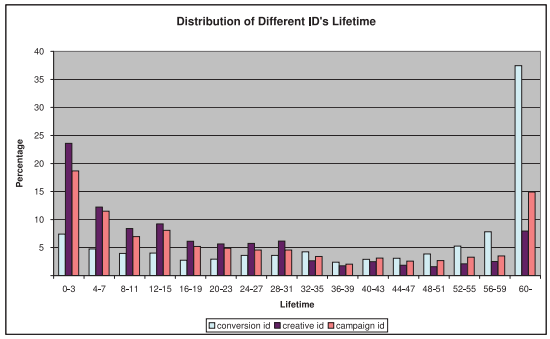

Fig. 8. Distribution of lifetime for different IDs.

IDs after 10 days and 11.2% after 20 days. This indicates that advertisers tend to use the same identifiers to track conversions even after they create new ad creatives and campaigns.

Similar conclusions can be reached by looking at the percentage of events instead of unique identifiers. Figure 7 shows that only 1.3% of the events contain new conversion identifiers after 10 days and 4.3% after 20 days. This is considered to be a relatively small increase for practical purposes because even if a model is not updated for as long as 10 days, it will still know about almost all the conversion identifiers.

This is beneficial to some extent to PCC modeling because a model that relies on conversion identifiers would be relatively stable over time.

7.2. Ad Lifetime

Another important factor that could effect model performance is the churn rate of the ads. Figure 8 plots the distribution of the lifetime of ads at three levels: conversion, campaign, and creative. 37.4% of conversion IDs lives longer than 2 months, which is quite significant compared to 8% for creative IDs and 14.9% for campaign IDs. 23.6% of creative IDs and 18.7% of campaign IDs have a lifetime shorter than 3 days, whereas this number for conversion ID is only 7.4%. It is notable that conversion ID lives much longer than the creative ID and campaign ID, which is consistent with the conclusion reached in Section 7.1.

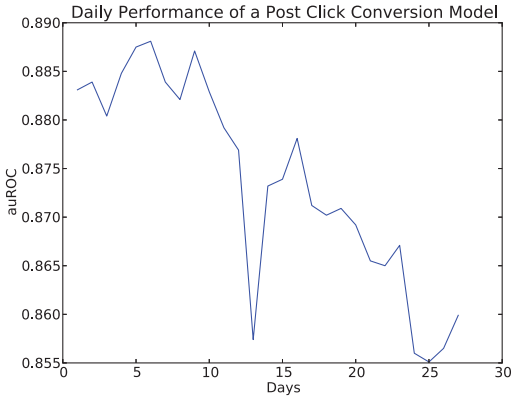

Fig. 9. The performance (auROC) of the model degrades with time.

7.3. Model Update

The results in Sections 7.1 and 7.2 indicate that there is a nontrivial percentage of new ads entering the system every day. In this section, we first illustrate the impact that this has on a static post-click conversion model, and later provide an algorithm to update our models efficiently.

In this analysis, data collected from a given month are used for training, and data of the following month are used as test data.

To evaluate the impact of the new ads on the model performance, we divided the test data into daily slices. In Figure 9, we present the performance of the static model. The x-axis represents each day in the test data, and the y-axis indicates the auROC. We can see from Figure 9 that the performance of the model degrades with time.

It is clear that the degradation in performance is closely related to the influx rate of new ads in the test data, as elucidated in previous analyses (Sections 7.1–7.2). For this reason, we cannot use a static model; we need to continuously refresh it with new data, as explained below.

The Bayesian interpretation of logistic regression described in Section 4.9 can be leveraged for model updates. Remember that the posterior distribution on the weights is approximated by a Gaussian distribution with diagonal covariance matrix. As in the Laplace approximation, the mean of this distribution is the mode of the posterior, and the inverse variance of each weight is given by the curvature. The use of this convenient approximation of the posterior is twofold. It first serves as a prior on the weights to update the model when a new batch of training data becomes available, as described in Algorithm 2. And it is also the distribution used in the exploration/exploitation heuristic described in the next section.

To illustrate the benefits of model updates, we performed the following experiment. We considered 3 weeks of training data and 1 week of test data. A base model is trained on the training data. The performance with model updates is measured as follows:

(1) Split the test set in $n = 24 \times 7/k$ batches of k hours;
(2) For $i = 1, \ldots, n$:
    —Test the current model on the i-th batch
    —Use the i-th batch of data to update the model as explained in Algorithm 2.

The baseline is the static base model applied to the entire test data. As shown in Table VII, the more frequently the model is updated, the better its accuracy.

Table VII. Influence of the Update Frequency (Area under the PR Curve)

1 day	6 hours	2 hours
+3.7%	+5.1%	+5.8%

ALGORITHM 2: Regularized Logistic Regression with Batch Updates

Require: Regularization parameter $\lambda > 0$.
$m_i = 0,\ q_i = \lambda$. {Each weight w_i has an independent prior $\mathcal{N}(m_i, q_i^{-1})$}
for $t = 1, \ldots, T$ **do**
Get a new batch of training data $(\mathbf{x}_j, y_j),\ j = 1, \ldots, n$.
Find $\mathbf{w}$ as the minimizer of: $\frac{1}{2}\sum_{i=1}^{d} q_i(w_i - m_i)^2 + \sum_{j=1}^{n} \log(1 + \exp(-y_j \mathbf{w}^\top \mathbf{x}_j))$.
$m_i = w_i$
$q_i = q_i + \sum_{j=1}^{n} x_{ij}^2 p_j(1 - p_j),\ p_j = (1 + \exp(-\mathbf{w}^\top \mathbf{x}_j))^{-1}$ {Laplace approximation}
end for

7.4. Exploration/Exploitation Tradeoff

To learn the CTR of a new ad, it needs to be displayed first, leading to a potential loss of short-term revenue. This is the classical exploration/exploitation dilemma: Either exploit the best ad observed so far, or take a risk and explore new ones resulting in either a short-term loss or a long-term profit depending on whether that ad is worse or better.

Various algorithms have been proposed to solve exploration/exploitation or bandit problems. One of the most popular one is the *Upper Confidence Bound* (UCB) [Lai and Robbins 1985; Auer et al. 2002] for which theoretical guarantees on the regret can be proved. Another representative is the Bayes-optimal approach of Gittins [1989] that directly maximizes expected cumulative payoffs with respect to a given prior distribution. A less known family of algorithms is so-called *probability matching*. The idea of this heuristic is old and dates back to Thompson [1933]; thus, this scheme is also referred to as *Thompson sampling* in the literature. The idea of Thompson sampling is to randomly draw each arm according to its probability of being optimal. In contrast to a full Bayesian method like the Gittins index, one can often implement Thompson sampling efficiently.

7.5. Thompson Sampling

In this section, we present a general description of Thompson sampling and provide some experimental results in the next one. More details can be found in Chapelle and Li [2011].

The contextual bandit setting is as follows: At each round, we have a context x (optional) and a set of actions $\mathcal{A}$. After choosing an action $a \in \mathcal{A}$, we observe a reward r. The goal is to find a policy that selects actions such that the cumulative reward is as large as possible.

Thompson sampling is best understood in a Bayesian setting as follows: The set of past observations D is made of triplets (x_i, a_i, r_i) and are modeled using a parametric likelihood function $P(r|a, x, \theta)$ depending on some parameters θ. Given some prior distribution $P(\theta)$ on these parameters, the posterior distribution of these parameters is given by the Bayes rule, $P(\theta|D) \propto \prod P(r_i|a_i, x_i, \theta)P(\theta)$.

In the realizable case, the reward is a stochastic function of the action, context, and the unknown true parameter θ^*. Ideally, we would like to choose the action maximizing the expected reward, $\max_a \mathbb{E}(r|a, x, \theta^*)$.

Of course, θ^* is unknown. If we are just interested in maximizing the immediate reward (exploitation), then one should choose the action that maximizes $\mathbb{E}(r|a,x) = \int \mathbb{E}(r|a,x,\theta)P(\theta|D)d\theta$.

But in an exploration/exploitation setting, the probability matching heuristic consists in randomly selecting an action a according to its probability of being optimal. That is, action a is chosen with probability

$$\int \mathbb{I}\left[\mathbb{E}(r|a,x,\theta) = \max_{a'} \mathbb{E}(r|a',x,\theta)\right] P(\theta|D)d\theta,$$

where $\mathbb{I}$ is the indicator function. Note that the integral does not have to be computed explicitly: It suffices to draw a random parameter θ at each round, as explained in Algorithm 3. The implementation is thus efficient and straightforward in our application: Since the posterior is approximated by a Gaussian distribution with diagonal covariance matrix (see Section 4.9), each weight is drawn independently from a Gaussian distribution.

ALGORITHM 3: Thompson Sampling

$D = \emptyset$
for $t = 1, \ldots, T$ **do**
  Receive context x_t
  Draw θ^t according to $P(\theta|D)$
  Select $a_t = \arg\max_a \mathbb{E}_r(r|x_t, a, \theta^t)$
  Observe reward r_t
  $D = D \cup (x_t, a_t, r_t)$
end for

In addition to being a good heuristic for explore/exploit, the advantage of Thompson sampling is that the randomized predictions it generates are compatible with a generalized second price auction, as typically used in ad exchanges. In other words, the auction is still incentive-compatible even though the predictions are randomized [Meek et al. 2005, Example 2]. It would be unclear which generalized second price to charge with other explore/exploit strategies.

7.6. Evaluation of Thomson Sampling

Evaluating an explore/exploit policy is difficult because we typically do not know the reward of an action that was not chosen. A possible solution is to use a *replayer,* in which previous, randomized exploration data can be used to produce an *unbiased* offline estimator of the new policy [Li et al. 2011]. Unfortunately, this approach cannot be used in our case because it reduces the effective data size substantially when the number of arms K is large, yielding too high variance in the evaluation results.

For the sake of simplicity, therefore, we considered in this section a simulated environment. More precisely, the context and the ads are real, but the clicks are simulated using a weight vector $\mathbf{w}^*$. This weight vector could have been chosen arbitrarily, but it was in fact a perturbed version of some weight vector learned from real clicks. The input feature vectors $\mathbf{x}$ are thus as in the real-world setting, but the clicks are artificially generated with probability $P(y = 1|\mathbf{x}) = (1 + \exp(-\mathbf{w}^{*\top}\mathbf{x}))^{-1}$.

About 13,000 contexts, representing a small random subset of the total traffic, are presented every hour to the policy, which then has to choose an ad among a set of eligible ads. The number of eligible ads for each context depends on numerous constraints set by the advertiser and the publisher. It varies between 5,910 and 1, with a mean of 1,364 and a median of 514 (over a set of 66,373 ads). Note that in this experiment the

Table VIII. CTR Regrets for Different Explore/Exploit Strategies

Method	TS			LinUCB			ε-greedy				
Parameter	0.25	0.5	1	0.5	1	2	0.005	0.01	0.02	Exploit	Random
Regret (%)	4.45	3.72	3.81	4.99	4.22	4.14	5.05	4.98	5.22	5.00	31.95

number of eligible ads is smaller than what we would observe in live traffic because we restricted the set of advertisers.

The model is updated every hour, as described in Algorithm 2. A feature vector is constructed for every (context, ad) pair and the policy decides which ad to show. A click for that ad is then generated with probability $(1+\exp(-\mathbf{w}^{*\top}\mathbf{x}))^{-1}$. This labeled training sample is then used at the end of the hour to update the model. The total number of clicks received during this one-hour period is the reward. To eliminate unnecessary variance in the estimation, we instead computed the expectation of that number since the click probabilities are known.

Several explore/exploit strategies are compared; they only differ in the way the ads are selected—all the rest, including the model updates, is identical, as described in Algorithm 2. These strategies are:

—*Thompson sampling*. This is Algorithm 3, in which each weight is drawn independently according to its Gaussian posterior approximation $\mathcal{N}(m_i, q_i^{-1})$ (see Algorithm 2). We also consider a variant in which the standard deviations $q_i^{-1/2}$ are first multiplied by a factor $\alpha \in \{0.25, 0.5\}$. This favors exploitation over exploration.
—*LinUCB*. This is an extension of the UCB algorithm to the parametric case [Li et al. 2010]. It selects the ad based on mean and standard deviation. It also has a factor α to control the exploration/exploitation tradeoff. More precisely, LinUCB selects the ad for which $\sum_{i=1}^{d} m_i x_i + \alpha\sqrt{\sum_{i=1}^{d} q_i^{-1} x_i^2}$ is maximum.
—*Exploit-only*. Select the ad with the highest mean.
—*Random*. Select the ad uniformly at random.
—*ε-greedy*. Mix between exploitation and random: With ε probability, select a random ad; otherwise, select the one with the highest mean.

Results. A preliminary result is about the quality of the variance prediction. The diagonal Gaussian approximation of the posterior does not seem to harm the variance predictions. In particular, they are very well calibrated: When constructing a 95% confidence interval for CTR, the true CTR is in this interval 95.1% of the time.

The regrets of the different explore/exploit strategies can be found in Table VIII. Thompson sampling achieves the best regret, and, interestingly, the modified version with $\alpha = 0.5$ gives slightly better results than the standard version ($\alpha = 1$).

Exploit-only does pretty well, at least compared to random selection. This seems at first a bit surprising given that the system has no prior knowledge about the CTRs. A possible explanation is that the change in context induces some exploration, as noted in Sarkar [1991]. Also, the fact that exploit-only is so much better than random might explain why ε-greedy does not beat it: Whenever this strategy chooses a random action, it suffers a large regret on average, which is not compensated by its exploration benefit.

Finally, Figure 10 shows the regret of three algorithms across time. As expected, the regret has a decreasing trend over time.

8. LARGE-SCALE LEARNING

Because of the large amount of data, we use the distributed Hadoop-based learning system [Agarwal et al. 2011]. This section presents a summary of this system with some experimental results.

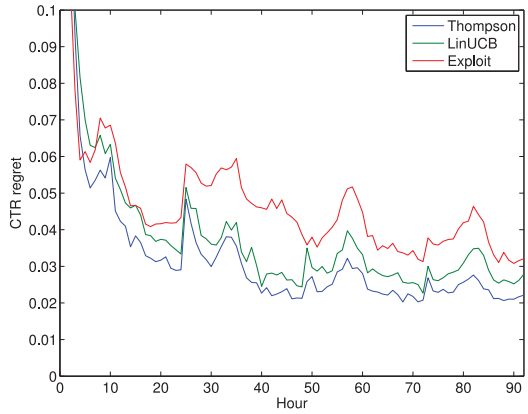

Fig. 10. CTR regret over the 4-day test period for three algorithms: Thompson sampling with $\alpha = 0.5$, LinUCB with $\alpha = 2$, Exploit-only. The regret in the first hour is large, around 0.3, because the algorithms predict randomly (no initial model provided).

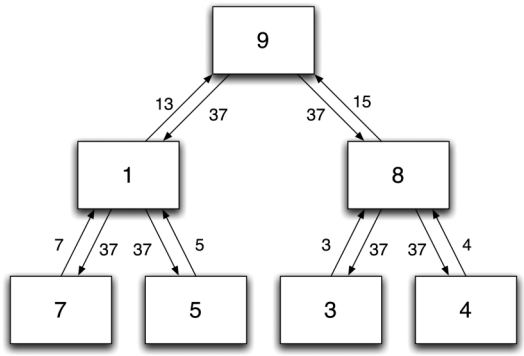

Fig. 11. AllReduce.

Map-Reduce [Dean and Ghemawat 2008] and its open-source implementation Hadoop[5] have become the overwhelmingly favorite platforms for distributed data processing in general. However, the abstraction is rather ill-suited for machine learning algorithms, as several researchers in the field have observed [Low et al. 2010; Zaharia et al. 2012], because it does not easily allow iterative algorithms, such as the typical optimization algorithms used to minimize the objective function (Equation (2)).

8.1. Hadoop-compatible AllReduce

AllReduce is a more suitable abstraction for machine learning algorithms. AllReduce is an operation in which every node starts with a number and ends up with the sum of the numbers at all the nodes. A typical implementation is done by imposing a tree structure on the communicating nodes—numbers can be summed up the tree (this is the *reduce* phase) and then broadcasted down to all nodes—hence the name AllReduce. See Figure 11 for a graphical illustration. When doing summing or averaging of a long vector, such as the weight vector $\mathbf{w}$ in the optimization in Equation (2), the reduce and broadcast operations can be pipelined over the vector entries, hence the latency of going up and down the tree becomes negligible on a typical Hadoop cluster.

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

For problems of the form in Equation (2), AllReduce provides straightforward parallelization—we just accumulate local gradients for a gradient-based algorithm like gradient descent or L-BFGS. In general, any statistical query algorithm [Kearns 1993] can be parallelized with AllReduce with only a handful of additional lines of code. This approach also easily implements averaging parameters of online learning algorithms.

An implementation of AllReduce is available in the MPI package. However, it is not easy to run MPI on top of existing Hadoop clusters [Ye et al. 2009]. Moreover, MPI implements little fault tolerance, with the bulk of robustness left to the programmer.

To better address the reliability issues, we developed an implementation of AllReduce that is compatible with Hadoop. Implementation of AllReduce using a single tree is clearly less desirable than MapReduce in terms of reliability because if any individual node fails, the entire computation fails. To deal with this, we make use of the speculative execution of Hadoop, which makes AllReduce reliable enough to use in practice for computations of up to 10K node hours.

8.2. Proposed Algorithm

The main algorithm is a hybrid online+batch approach. To understand the motivations for this, we start by discussing the pros and cons of purely online or batch learning algorithms common in machine learning. An attractive feature of online learning algorithms is that they can optimize the sample risk to a rough precision quite fast, in just a handful of passes over the data. The inherent sequential nature of these algorithms, however, makes them tricky to parallelize. Batch learning algorithms such as Newton and Quasi-Newton methods (e.g., L-BFGS), on the other hand, are great at optimizing the sample risk to a high accuracy once they are in a *good neighborhood* of the optimal solution. But the algorithms can be quite slow in reaching this good neighborhood. Generalization of these approaches to distributed setups is rather straightforward, only requiring aggregation across nodes after every iteration, as also noted in some prior works [Teo et al. 2007].

We attempt to reap the benefits and avoid the drawbacks of both these approaches through our hybrid method. We start with each node making one online pass over its local data according to adaptive gradient updates [Duchi et al. 2010; McMahan and Streeter 2010]. We notice that each online pass happens completely *asynchronously* without any communication between the nodes, and we can afford to do so since we are only seeking to get into a good neighborhood of the optimal solution rather than recovering it to a high precision at this first stage. AllReduce is used to average these local weights.

This solution $\bar{\mathbf{w}}$ is used to initialize L-BFGS [Nocedal 1980] with the standard Jacobi preconditioner, with the expectation that the online stage gives us a good *warmstart* for L-BFGS. At each iteration, the local gradients are summed up using AllReduce, whereas all the other operations can be done locally at each node. The algorithm benefits from the fast reduction of error initially that an online algorithm provides and the rapid convergence in a good neighborhood guaranteed by Quasi-Newton algorithms. We again point out that the number of communication operations is relatively few throughout this process.

Note that the implementation is open source in Vowpal Wabbit [Langford et al. 2007] and is summarized in Algorithm 4.

8.3. Results

We provide in this section some experimental results using AllReduce and compare them to other methods for distributed learning. The entire set of results can be found in Agarwal et al. [2011].

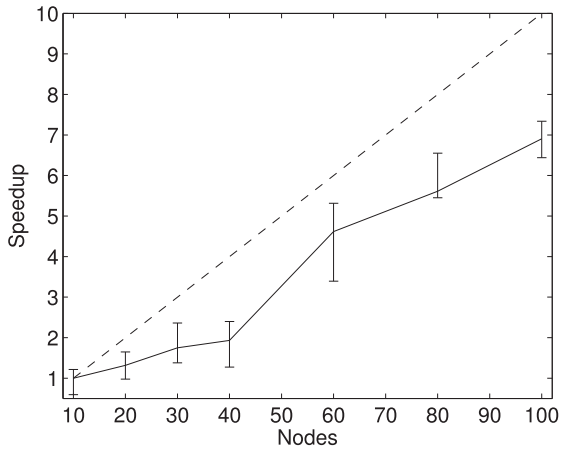

Fig. 12. Speedup for obtaining a fixed test error relative to the run with 10 nodes, as a function of the number of nodes. The dashed line corresponds to the ideal speedup, the solid line is the average speedup over 10 repetitions, and the bars indicate maximum and minimal values.

ALGORITHM 4: Sketch of the Proposed Learning Architecture

Require: Data split across nodes
  for all nodes k **do**
    $\mathbf{w}^k$ = result on the data of node k using stochastic gradient descent.
  end for
  Compute the average $\bar{\mathbf{w}}$ using AllReduce.
  Start a preconditioned L-BFGS optimization from $\bar{\mathbf{w}}$.
  for $t = 1, \ldots, T$ **do**
    for all nodes k **do**
      Compute $\mathbf{g}^k$ the (local batch) gradient of examples on node k
      Compute $\mathbf{g} = \sum_{k=1}^{m} \mathbf{g}^k$ using AllReduce.
      Add the regularization part in the gradient.
      Take an L-BFGS step.
    end for
  end for

8.3.1. Running Time. We study the running time as a function of the number of nodes. We varied the number of nodes from 10 to 100 and computed the speedup factor relative to the run with 10 nodes. In each case, we measured the amount of time needed to get to a fixed test error. Since there can be significant variations from one run to the other—mostly because of the cluster utilization—each run was repeated 10 times. Results are reported in Figure 12. The speedup appears to be close to linear.

8.3.2. Online and Batch Learning. We investigate the number of iterations needed to reach a certain test performance for different learning strategies: batch, online, and hybrid.

Figure 13 compares two learning strategies—batch with and without an initial online pass—on the training set. It plots the optimality gap, defined as the difference between the current objective function and the optimal one (i.e., minimum value of the objective, as in Equation (2)), as a function of the number iterations. From this figure, one can see that the initial online pass results in a saving of about 10–15 iterations.

Figure 14 shows the test auPRC as a function of the number of iterations for four different strategies: Only online learning, only L-BFGS learning, and two hybrid methods

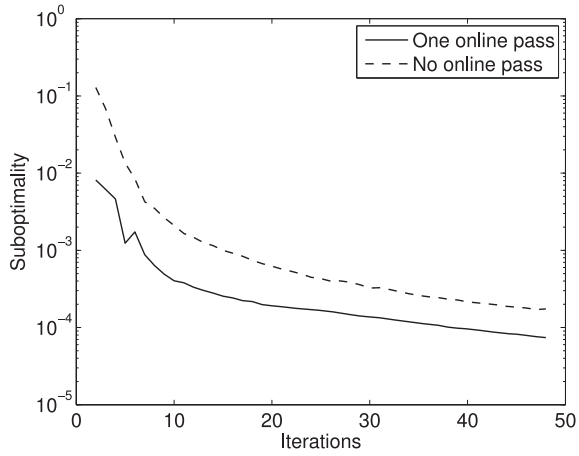

Fig. 13. Effect of initializing the L-BFGS optimization by an average solution from online runs on individual nodes.

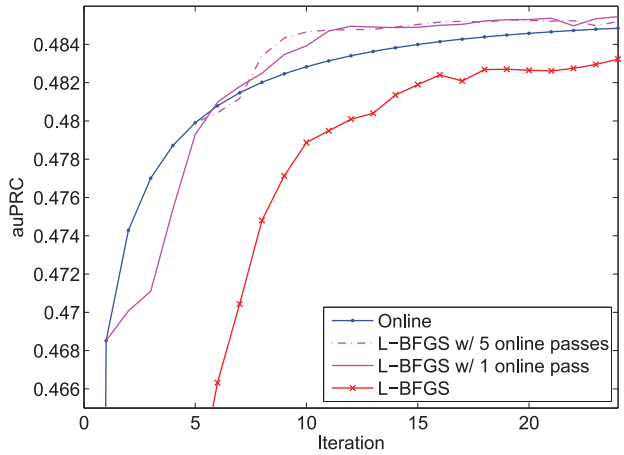

Fig. 14. Test auPRC for four different learning strategies.

consisting of one or five passes of online learning followed by L-BFGS optimization. L-BFGS with one online pass appears to be the most effective strategy.

8.3.3. Large Experiment and Comparison with Sibyl. We experimented with an 8 times larger version of the data (16B examples). Using 1,000 nodes and 10 passes over the data, the training took only 70 minutes.[6] Since each example is described by 125 nonzero features, the average processing speed was

$$16\text{B} \times 10 \times 125 \text{ features}/1000 \text{ nodes}/70 \text{ minutes} \;=\; 4.7 \text{ M features/node/s}.$$

The overall learning throughput was

$$16\text{B} \times 125 \text{ features}/70 \text{ minutes} \;=\; 470 \text{ M features/s}.$$

[6]As mentioned earlier, there can be substantial variations in timing between different runs; this one was done when the cluster was not much occupied.

Table IX. Average Training Time (in Seconds) per Iteration of an Internal Logistic Regression Implementation Using Either MapReduce or AllReduce for Gradients Aggregation

	Full size	10% sample
MapReduce	1690	1322
AllReduce	670	59

We briefly compare this with a result reported for Sibyl for a run on 970 cores [Canini et al. 2012, slide 24]. The run was done over 129.1B examples, with 54.61 nonzero features per example. The reported processing speed was 2.3M features/core/s (which is a factor of two slower than our achieved processing speed). The reported number of iterations is 10–50, which would lead to final learning throughput in the range of 45–223 M features/s (i.e., the results appear to be slower by a factor of 2–10).

8.4. Comparison with MapReduce

The standard way of using MapReduce for iterative machine learning algorithms is the following [Chu et al. 2007]: Every iteration is an M/R job in which the mappers compute some local statistics (such as a gradient) and the reducers sum them up. This is ineffective because each iteration has large overheads (job scheduling, data transfer, data parsing, etc.). We have an internal implementation of such a M/R algorithm. We updated this code to use AllReduce instead and compared both versions of the code in Table IX. This table confirms that Hadoop MapReduce has substantial overheads since the training time is not much affected by the dataset size. The speedup factor of AllReduce over Hadoop MapReduce can become extremely large for smaller datasets and remains noticeable even for the largest datasets.

It is also noteworthy that *all* algorithms described in Chu et al. [2007] can be parallelized with AllReduce, as well as further algorithms such as parameter averaging approaches.

9. CONCLUSION

In this article, we presented a framework for modeling response prediction in display advertising. Our approach has several advantages over the alternatives:

—*Simplicity.* The proposed framework is easy to implement, trivial to update, and lightweight to use on real servers. The ad servers need only to load model files that can easily fit in memory and implement a model applier that uses only a hash function as feature generator.
—*Scalability.* Our algorithms are easy to parallelize. The map-reduce architecture presented in this article allows us to train models using many billions of samples in one hour, a significant improvement over existing methods.
—*Efficiency.* The results presented in this article clearly show that our framework is as good as or better than the state-of-the-art alternatives.

The experiments reported in this study were conducted on data from two large (and distinct) display advertising companies. The stability and repeatability of our results suggest strongly that the presented framework could serve as a strong baseline in this domain.

ACKNOWLEDGMENTS

We would like to thank our colleagues Kannan Achan, Haibin Cheng, Dmitry Pavlov, Benoît Rostykus, and Huang Xu for providing us with valuable feedback, as well as for running some of the experiments in this paper.

REFERENCES

A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. 2011. A reliable effective terascale linear learning system. *CoRR* abs/1110.4198 (2011).

D. Agarwal, R. Agrawal, R. Khanna, and N. Kota. 2010. Estimating rates of rare events with multiple hierarchies through scalable log-linear models. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 213–222.

Azin Ashkan, Charles L. A. Clarke, Eugene Agichtein, and Qi Guo. 2009. Estimating ad clickthrough rate through query intent analysis. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*.

P. Auer, N. Cesa-Bianchi, and P. Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47, 2 (2002), 235–256.

Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. 2011. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning* 4, 1 (2011), 1–106.

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc.

B. H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 13, 7 (1970), 422–426.

K. Canini, T. Chandra, E. Ie, J. McFadden, K. Goldman, M. Gunter, J. Harmsen, K. LeFevre, D. Lepikhin, T. L. Llinares, I. Mukherjee, F. Pereira, J. Redstone, T. Shaked, and Y. Singer. 2012. Sibyl: A system for large scale supervised machine learning. (2012). Presentation at MLSS Santa Cruz, http://users.soe.ucsc.edu/niejiazhong/slides/chandra.pdf.

D. Chakrabarti, D. Agarwal, and V. Josifovski. 2008. Contextual advertising by combining relevance with click feedback. In *Proceedings of the 17th International Conference on World Wide Web*. 417–426.

Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. 2010. Training and testing low-degree polynomial data mappings via linear SVM. *The Journal of Machine Learning Research* 11 (2010), 1471–1490.

O. Chapelle and L. Li. 2011. An empirical evaluation of thompson sampling. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. Bartlett, F. C. N. Pereira, and K. Q. Weinberger (Eds.). 2249–2257.

S. F. Chen and J. Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* 13, 4 (1999), 359–393.

Haibin Cheng and Erick Cantú-Paz. 2010. Personalized click prediction in sponsored search. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining*. 351–360.

Haibin Cheng, Roelof van Zwol, Javad Azimi, Eren Manavoglu, Ruofei Zhang, Yang Zhou, and Vidhya Navalpakkam. 2012. Multimedia features for click prediction of new ads in display advertising. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 777–785.

C. T. Chu, S. K. Kim, Y. A. Lin, Y. Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. 2007. Map-reduce for machine learning on multicore. In *Proceedings of the 2006 Conference on Advances in Neural Information Processing Systems*, Vol. 19.

Massimiliano Ciaramita, Vanessa Murdock, and Vassilis Plachouras. 2008. Online learning from click data for sponsored search. In *Proceedings of the 17th International Conference on World Wide Web*. 227–236.

C. Cortes, Y. Mansour, and M. Mohri. 2010. Learning bounds for importance weighting. In *Advances in Neural Information Processing Systems*, Vol. 23. 442–450.

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

John Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12 (2010), 2121–2159.

Theodoros Evgeniou and Massimiliano Pontil. 2004. Regularized multi-task learning. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 109–117.

A. Gelman and J. Hill. 2006. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.

John C. Gittins. 1989. *Multi-armed Bandit Allocation Indices*. John Wiley & Sons.

T. Graepel, J. Quinonero Candela, T. Borchert, and R. Herbrich. 2010. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's bing search engine. In *Proceedings of the 27th International Conference on Machine Learning*. 13–20.

I. Guyon and A. Elisseeff. 2003. An introduction to variable and feature selection. *The Journal of Machine Learning Research* 3 (2003), 1157–1182.

D. Hillard, E. Manavoglu, H. Raghavan, C. Leggetter, E. Cantú-Paz, and R. Iyer. 2011. The sum of its parts: Reducing sparsity in click estimation with query segments. *Information Retrieval* (2011), 1–22.

D. Hillard, S. Schroedl, E. Manavoglu, H. Raghavan, and C. Leggetter. 2010. Improving ad relevance in sponsored search. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining*. 361–370.

Michael Kearns. 1993. Efficient noise-tolerant learning from statistical queries. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*. 392–401.

G. King and L. Zeng. 2001. Logistic regression in rare events data. *Political Analysis* 9, 2 (2001), 137–163.

H. Koepke and M. Bilenko. 2012. Fast prediction of new feature utility. In *Proceedings of the 29th International Conference on Machine Learning*. 791–798.

Nagaraj Kota and Deepak Agarwal. 2011. Temporal multi-hierarchy smoothing for estimating rates of rare events. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1361–1369.

T. L. Lai and H. Robbins. 1985. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics* 6 (1985), 4–22.

J. Langford, L. Li, and A. Strehl. 2007. Vowpal Wabbit Open Source Project. https://github.com/JohnLangford/vowpal_wabbit/wiki. (2007).

L. Li, W. Chu, J. Langford, and R. E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*. 661–670.

L. Li, W. Chu, J. Langford, and X. Wang. 2011. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining*. 297–306.

Yandong Liu, Sandeep Pandey, Deepak Agarwal, and Vanja Josifovski. 2012. Finding the right consumer: Optimizing for conversion in display advertising campaigns. In *Proceedings of the 5th ACM International Conference on Web Search and Data Mining*. 473–482.

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2010. GraphLab: A new framework for parallel machine learning. In *The 26th Conference on Uncertainty in Artificial Intelligence*.

James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, Berkeley, CA, 281–297.

R. P. McAfee. 2011. The design of advertising exchanges. *Review of Industrial Organization* (2011), 1–17.

H. B. McMahan and M. Streeter. 2010. Adaptive bound optimization for online convex optimization. In *Proceedings of the 23rd Annual Conference on Learning Theory*. 244–256.

C. Meek, D. M. Chickering, and D. Wilson. 2005. Stochastic and contingent payment auctions. In *Workshop on Sponsored Search Auctions, ACM Electronic Commerce*.

Lukas Meier, Sara Van De Geer, and Peter Bühlmann. 2008. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 70, 1 (2008), 53–71.

S. Menard. 2001. *Applied Logistic Regression Analysis*. Vol. 106. Sage Publications, Inc.

Aditya Krishna Menon, Krishna-Prasad Chitrapura, Sachin Garg, Deepak Agarwal, and Nagaraj Kota. 2011. Response prediction using collaborative filtering with hierarchies and side-information. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 141–149.

T. P. Minka. 2003. *A Comparison of Numerical Optimizers for Logistic Regression*. Technical Report. Microsoft Research. Retrieved from http://research.microsoft.com/en-us/um/people/minka/papers/logreg/.

S. Muthukrishnan. 2009. Ad exchanges: Research issues. In *Proceedings of the 5th International Workshop on Internet and Network Economics*. 1–12.

K. Nigam, J. Lafferty, and A. McCallum. 1999. Using maximum entropy for text classification. In *IJCAI-99 Workshop on Machine Learning for Information Filtering*, Vol. 1. 61–67.

J. Nocedal. 1980. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation* 35, 151 (1980), 773–782.

A. B. Owen. 2007. Infinitely imbalanced logistic regression. *The Journal of Machine Learning Research* 8 (2007), 761–773.

Moira Regelson and Daniel C. Fain. 2006. Predicting click-through rate using keyword clusters. In *Proceedings of the Second Workshop on Sponsored Search Auctions*.

M. Richardson, E. Dominowska, and R. Ragno. 2007. Predicting clicks: Estimating the click-through rate for new ads. In *Proceedings of the 16th International Conference on World Wide Web*. New York, NY, 521–530.

R. Rosales and O. Chapelle. 2011. Attribute selection by measuring information on reference distributions. In *Tech Pulse Conference, Yahoo!*. Retrieved from http://people.csail.mit.edu/romer/papers/RosChaTP11.pdf.

R. Rosales, H. Cheng, and E. Manavoglu. 2012. Post-click conversion modeling and analysis for non-guaranteed delivery display advertising. In *Proceedings of the 5th ACM International Conference on Web Search and Data Mining*. ACM, 293–302.

J. Sarkar. 1991. One-armed bandit problems with covariates. *The Annals of Statistics* (1991), 1978–2002.

B. Schölkopf and A. J. Smola. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and SVN Vishwanathan. 2009. Hash kernels for structured data. *The Journal of Machine Learning Research* 10 (2009), 2615–2637.

C. Teo, Q. Le, A. Smola, and SVN Vishwanathan. 2007. A scalable modular convex solver for regularized risk minimization. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 727–736.

William R. Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 3–4 (1933), 285–294.

K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 1113–1120.

Jerry Ye, Jyh-Herng Chow, Jiang Chen, and Zhaohui Zheng. 2009. Stochastic gradient boosted distributed decision trees. In *Proceeding of the 18th ACM Conference on Information and Knowledge Management*. 2061–2064.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. 15–28.

Received December 2012; revised April 2013; accepted August 2013