Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine

Thore Graepel THOREG@MICROSOFT.COM
Joaquin Quiñonero Candela JOAQUINC@MICROSOFT.COM
Thomas Borchert TBORCHER@MICROSOFT.COM
Ralf Herbrich RHERB@MICROSOFT.COM
Microsoft Research Ltd., 7 J J Thomson Avenue, Cambridge CB3 0FB, UK

Abstract

We describe a new Bayesian click-through rate (CTR) prediction algorithm used for Sponsored Search in Microsoft's Bing search engine. The algorithm is based on a probit regression model that maps discrete or real-valued input features to probabilities. It maintains Gaussian beliefs over weights of the model and performs Gaussian online updates derived from approximate message passing. Scalability of the algorithm is ensured through a principled weight pruning procedure and an approximate parallel implementation. We discuss the challenges arising from evaluating and tuning the predictor as part of the complex system of sponsored search where the predictions made by the algorithm decide about future training sample composition. Finally, we show experimental results from the production system and compare to a calibrated Naïve Bayes algorithm.

1. Introduction

Sponsored search remains one of the most profitable business models on the web today. It accounts for the overwhelming majority of income for the three major search engines Google, Yahoo and Bing, and generates revenue of at least 25 billion dollars[1] per year and rising. All three major players use keyword auctions to allocate display space alongside the algorithmic search results based on a pay-per-click model in which advertisers are charged only if their advertisements are clicked by a user. In this mechanism it is necessary for the search engine to estimate the click-through rate (CTR) of available ads for a given search query to determine the best allocation of display space and appropriate payments (Edelman, Ostrovsky, & Schwarz, 2007). As a consequence, the task of CTR prediction is absolutely crucial to Sponsored Search advertising because it impacts user experience, profitability of advertising and search engine revenue.

Recognising the importance of CTR estimation for online advertising, management at Bing/adCenter decided to run a competition to entice people across the company to develop the most accurate and scalable CTR predictor. The algorithm described in this publication tied for first place in the first competition and won the subsequent competition based on prediction accuracy. As a consequence, it was chosen to replace Bing's previous CTR prediction algorithm, a transition that was completed in the summer of 2009.

The paper makes three major contributions. First, it describes the Sponsored Search application scenario, the key role of CTR prediction in general, and the particular constraints derived from the task, including accuracy, calibration, scalability, dynamics, and exploration. Second, it describes a new Bayesian online learning algorithm for binary prediction, subsequently referred to as *adPredictor*. The algorithm is based on a generalised linear model with a probit (cumulative Gaussian) link function, a factorising Gaussian belief distribution on the feature weights, and calculates the approximate posterior using message passing, providing simple, closed-form update equations with automatic feature-wise learning rate adaptation. Third, we discuss the techniques we employed to make adPredictor work in Bing's production environment, now driving 100% Sponsored Search traffic with $\mathcal{O}(10^{10} - 10^{11})$ ad impressions per year.

The paper is structured as follows. In Section 2 we describe in detail how the task of CTR prediction fits into the framework of keyword auctions and which constraints and challenges arise from the application domain of Sponsored Search. In Section 3 we describe the online Bayesian Probit Regression algorithm (adPredictor) in detail and provide a derivation of the update equations based on approximate message passing in a factor graph. In Section 4 we discuss how the algorithm operates at web scale, using accuracy controlled pruning and an implementation of parallel training. In Section 5 we discuss how predictions affect the composition of future

Appearing in Proceedings of the 27th International Conference on Machine Learning, Haifa, Israel, 2010.
[1] Source: eMarketer, April 2009

training data, and the problem of trading off exploration and exploitation. Before we conclude in Section 7 we provide experimental results from the live system comparing adPredictor's prediction accuracy with that of a calibrated Naïve Bayes classifier.

2. Sponsored Search and CTR Prediction

The Sponsored Search advertising model exploits two key aspects of web search. First, the query users enter into a search engine partly reveals their intent and can help identify appropriate ads to be displayed to the users. Second, by clicking on ads users can proceed directly to the advertisers' web pages and the business value thus generated can easily be attributed to the web search engine. The lecture notes for the *Introduction to Computational Advertising* at Stanford (Broder & Josifovski, 2009) provide an excellent introduction.

2.1. Keyword Auction

In practice, the keyword auctions work as follows (Edelman, Ostrovsky, & Schwarz, 2007). For a given product or service advertisers identify suitable keywords likely to be typed by users interested in their offering. For each of those keywords the advertisers provide a bid indicating the amount of money they would be willing to pay for a click. When a user types a query, the search engine matches the keywords of all the advertisers against the query and decides which advertisers are eligible to participate in an auction for having their ad displayed. The search engine needs to *allocate* the available ad positions to the ads in the auction and needs to determine appropriate *payments*. This is achieved by a mechanism referred to as a *Generalized Second Price (GSP) Auction*.

Let us refer to the bid of advertiser i as b_i and the probability of click (CTR) of advertiser i at the top display position as p_i. The allocation of ads to display positions is determined by their so-called rank score $p_i b_i$, which can be interpreted as expected revenue for ad i if displayed in the top position[2]. The indices i are chosen according to that ranking, such that for all ads i we have: $p_i b_i \geq p_{i+1} b_{i+1}$. The payments c_i in a GSP auction are designed to avoid dynamic bidding behaviour because the charge per impression for ad i depends on the value per impression of ad $i+1$ such that $c_i = b_{i+1} p_{i+1} / p_i$.

It can be seen that the estimated click-through rate p_i plays a crucial role in determining both allocation and payments, and that it will have a crucial effect on the user experience, the advertiser value and the general health and income of the ad marketplace.

[2] The calculation of the rank score may also involve other criteria such as relevance of the ad landing page etc.

2.2. Input Features

We refer to an ad shown to a particular user in a particular page view as an ad impression. One of the key questions is the availability of suitable input features or predictor variables that allow accurate CTR prediction for a given impression (Richardson, Dominowska, & Ragno, 2007). These can generally be grouped into three categories: *Ad features* include bid phrases, ad title, ad text, landing page URL, landing page itself[3], and a hierarchy of advertiser, account, campaign, ad group and ad. *Query features* include the search keywords, possible algorithmic query expansion, cleaning and stemming. *Context features* include display location, geographic location, time, user data and search history.

Of course, these are only the base features which serve as the building blocks for more complex features modelling the interaction between ad, query and context. These more complex features can, e.g., be constructed by taking the Cartesian product of base features. As in most machine learning problems, constructing and selecting good features is one of the core challenges. For the learning algorithm one of the resulting challenges is the requirement to be able to handle discrete features of very different cardinalities, e.g., a two-valued feature such as gender and a billion-valued feature such as user ID.

2.3. Domain-Specific Challenges

2.3.1. Evaluation

An important question is how to evaluate a predictor within the context of a given application domain. Broadly speaking, the performance of a predictor can be evaluated in isolation or as part of the larger system.

To evaluate a predictor in isolation, the machine learning community has developed a number of reasonable measures such as the log-likelihood of test data under the model or the area under the receiver-operator (RO) curve (AUC). In the experimental section we will use these measures to evaluate adPredictor in comparison to calibrated Naïve Bayes. However, it is clear that these measures can only act as a proxy for the performance of the predictor in the larger system.

Ultimately, the predictor is part of a larger system that serves a purpose different from predicting user behaviour, namely the selection of ads. The ad selection system must be designed to balance the utilities of different players participating in the transaction: advertisers, users, and the search engine. These three types of players have different, even contradictory objectives. *Advertisers* are interested in maximising their return on investment at high volume. *Users* would like to see maximally relevant ads that help

[3] The user only gets to see the landing page once the click has been made. Over time, however, its quality can impact the perception of the advertiser and hence CTR.

them pursue their intent. *The search engine* would like to maximise revenue and growth.

Internally, these conflicting goals are mapped to different key performance indicators (KPIs) that are used to tune the ad selection system. However, these KPIs are influenced by a large number of other subsystems such as fraud detection, query expansion, keyword-query matching, etc. Furthermore, there are a large number of parameters influencing the KPIs including reserve prices and rank-score parameters. So, while the ultimate test of a CTR predictor lies in its performance as part of the ad selection system, in a modular architecture it is often best to identify isolated performance measures as proxies for in-system performance.

2.3.2. Dynamics and Exploration

The Web itself and the behaviour of people on the web is by no means static and it is therefore necessary to devise a dynamic CTR predictor which is able to track changes in CTR over time. Such changes can be the result of seasonal variation, gradual changes in taste or interest, changes in web content, economic conditions etc. Online algorithms are particularly suited to the task because they can adapt to the dynamics of the impression-click sequence. Batch learning algorithms can be trained on windows shifted over the time series.

While the prediction of CTR is essentially an inference problem, the performance of the ad selection system will be measured in terms of the decisions made. Moreover, since the CTR estimates of the CTR predictor are used to select ads for display through the keyword auction, the output of the CTR predictor effectively determines the ads present in its future training sample. Hence, the ad selection mechanism must somehow address the exploration/exploitation trade-off (Sutton & Barto, 1998): Greedy ad selection according to CTR will result in a locally optimal selection policy that ignores the long-term benefits of exploring the full ad inventory.

2.3.3. Computational Cost and Scale

The global business of Sponsored Search has vast proportions (Broder & Josifovski, 2009). There are millions of different ads that need to be stored, curated, updated, and indexed. There are billions of users whose behaviour needs to be tracked in accordance with their privacy preferences. Many 10s of millions of ad impressions per hour need to be served with acceptable response times below 100ms per request, and many more are evaluated. In addition, with each request requiring considerable CPU time and data residing in RAM, there is a significant cost associated with running the business.

For the task of CTR prediction this means that we require a fast, parallelisable learning algorithm that yields a predictor with low computational costs. The training algorithm needs to be able to handle features that can take potentially billions of different values, and it must be able to handle highly correlated input features as might be present in the nodes of the ad hierarchy (advertiser, account, campaign, etc.) Furthermore, the prediction algorithm itself needs to have a bounded memory footprint in RAM to be able to run continuously in the production system.

3. Online Bayesian Probit Regression

The new algorithm presented here is a general Bayesian online learning algorithm for the prediction of binary outcomes. However, in the context of this paper, we will use terminology related to the task of CTR prediction.

3.1. Task and Notation

We aim to learn a mapping $X \to [0,1]$ where X denotes the set of ad impressions as represented by their feature descriptions, and the interval $[0,1]$ represents the set of possible CTRs (probabilities of click). In this application, we consider the case of impressions that are described by N discrete multi-valued features, with feature $i \in \{1, \dots, N\}$ taking M_i different values. To simplify notation we represent that collection of features for a given impression in terms of a sparse binary feature vector $\boldsymbol{x} := (\boldsymbol{x}_1^T, \dots \boldsymbol{x}_N^T)^T$ where each vector $\boldsymbol{x}_i$ represents a binary 1-in-N encoding of the corresponding discrete feature value such that each vector $\boldsymbol{x}_i$ has exactly one element with value 1 and the remaining values 0, i.e. for all $i \in \{1, \dots, N\}$ we have $x_{i,j} \in \{0,1\}$ and

$$\boldsymbol{x}_i := \begin{pmatrix} x_{i,1} \\ \vdots \\ x_{i,M_i} \end{pmatrix}, \qquad \sum_{j=1}^{M_i} x_{i,j} = 1 \tag{1}$$

For notational convenience, we will denote the outcome or label click/non-click by $y \in \{-1, +1\}$ where -1 represents a non-click, and $+1$ represents a click.

3.2. Probability Model and Factor Graph

Our starting point is a generalised linear model with a probit link function. The sampling distribution of this model is given by.

$$p(y|\boldsymbol{x}, \boldsymbol{w}) := \Phi\left(\frac{y \cdot \boldsymbol{w}^T \boldsymbol{x}}{\beta}\right) \tag{2}$$

Here $\Phi(t) := \int_{-\infty}^{t} \mathcal{N}(s; 0,1)\, ds$ is the standardized cumulative Gaussian density (probit function) which serves as the inverse link function mapping the output of the linear model in $[-\infty, \infty]$ to $[0,1]$. The parameter β scales the steepness of the inverse link function.

In order to arrive at a Bayesian online learning algorithm we postulate a factorising Gaussian prior distribution over the weights of the model:

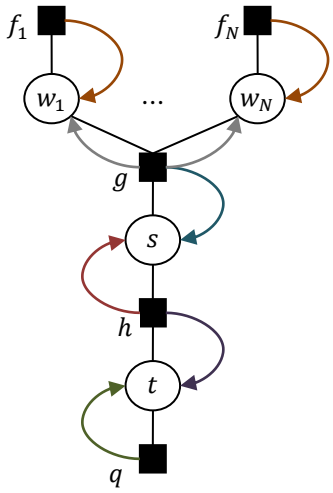

Figure 1: Factor graph model of Bayesian probit regression with message flow. Only active weights are shown.

$$p(\boldsymbol{w}) = \prod_{i=1}^{N} \prod_{j=1}^{M_i} \mathcal{N}\left(w_{i,j}; \mu_{i,j}, \sigma_{i,j}^2\right) \qquad (3)$$

Given the sampling distribution $p(y|\boldsymbol{x}, \boldsymbol{w})$ and the prior $p(\boldsymbol{w})$ it remains to calculate the posterior.

$$p(\boldsymbol{w}|\boldsymbol{x}, y) \propto p(y|\boldsymbol{x}, \boldsymbol{w}) \cdot p(\boldsymbol{w}) \qquad (4)$$

The exact posterior over weights $\boldsymbol{w}$ can neither be represented compactly nor calculated in closed form. We therefore resort to approximate message passing. In order to bring out the full factorial structure of the likelihood, we introduce two latent variables s, t, and consider the equivalent joint density function $p(y, t, s, \boldsymbol{w}|x)$ which factorises as

$$p(y \mid t) \cdot p(t \mid s) \cdot p(s \mid \boldsymbol{x}, \boldsymbol{w}) \cdot p(\boldsymbol{w}) \qquad (5)$$

This distribution can be understood in terms of the following generative process, which is also reflected in the factor graph in Figure 1.

- Factors f_i: Sample weights $\boldsymbol{w}$ from the Gaussian prior $p(\boldsymbol{w})$.
- Factor g: Calculate the score s for $\mathbf{x}$ as the inner product $\boldsymbol{w}^T\boldsymbol{x}$, such that $p(s|\boldsymbol{x}, \boldsymbol{w}) := \delta(s = \boldsymbol{w}^T\boldsymbol{x})$.
- Factor h: Add zero-mean Gaussian noise to obtain t from s, such that $p(t|s) := \mathcal{N}(t; s, \beta^2)$.
- Factor q: Determine y by a threshold on the noisy score t at zero, such that $p(y|t) := \delta(y = \text{sign}(t))$.

3.3. Inference

The factor graph in Figure 1 allows us to break down the computation of the posterior over weights $\boldsymbol{w}$ into local computations referred to as messages (Kschischang, Frey, & Loeliger, 2001). In fact, since the exact posterior calculation is intractable, we maintain an approximation of the posterior in the same family of distributions as the prior (3). The approximate message passing algorithm used is expectation propagation (Minka, 2001) in the mode of assumed Gaussian density filtering.

There are two inference tasks corresponding to two types of marginal distributions to be computed on the factor graph in Figure 1.

- Given training example $(\boldsymbol{x}, y)$ and prior $p(\boldsymbol{w})$, infer the new posterior $p(\boldsymbol{w}|\boldsymbol{x}, y)$ (upward messages).
- Given posterior $p(\boldsymbol{w}|\boldsymbol{x}, y)$ and feature vector $\boldsymbol{x}$ infer predictive distribution $p(y|\boldsymbol{x})$ (downward messages).

We represent the Gaussian beliefs over weights $\boldsymbol{w}$ by sparse vectors only storing values different from the prior.

- A vector of means $\boldsymbol{\mu} := \left(\mu_{1,1}, \ldots, \mu_{N,M_N}\right)^T$
- A vector of variances $\boldsymbol{\sigma^2} := \left(\sigma_{1,1}^2, \ldots, \sigma_{N,M_N}^2\right)^T$.

We will not provide a detailed derivation of the inference equations because adPredictor can be seen as a special case of the TrueSkill™ rating algorithm for games (Herbrich, Minka, & Graepel, 2007). The input feature vector in adPredictor corresponds to a team of players, with each active weight in adPredictor corresponding to the skill of a player in the team. Inference on the weights in adPredictor is equivalent to inference on the player skills in TrueSkill after a hypothetical match against a team with known skill of zero. Given the factor graph in Figure 1 together with Table 1 in the above paper the update equations can be derived.

3.3.1. Update Equations for Online Learning

The update equations represent a mapping from prior to posterior parameter values based on input-output pairs $(\boldsymbol{\mu}, \boldsymbol{\sigma^2}, \boldsymbol{x}, y) \mapsto (\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}^2)$. In terms of Figure 1, the calculation can viewed as following the message passing schedule towards the weights $\boldsymbol{w}$. We define the total variance for a given input $\boldsymbol{x}$ as

$$\Sigma^2 := \beta^2 + \boldsymbol{x}^T\boldsymbol{\sigma}^2 \qquad (6)$$

The update for the posterior parameters is given by:

$$\tilde{\mu}_{i,j} = \mu_{i,j} + y x_{i,j} \cdot \frac{\sigma_{i,j}^2}{\Sigma} \cdot v\left(\frac{y \cdot \boldsymbol{x}^T\boldsymbol{\mu}}{\Sigma}\right) \qquad (7)$$

$$\tilde{\sigma}_{i,j}^2 \leftarrow \sigma_{i,j}^2 \cdot \left[1 - x_{i,j} \cdot \frac{\sigma_{i,j}^2}{\Sigma^2} \cdot w\left(\frac{y \cdot \boldsymbol{x}^T\boldsymbol{\mu}}{\Sigma}\right)\right]. \qquad (8)$$

The functions v and w (see also Figure 2) are given by

$$v(t) \coloneqq \frac{\mathcal{N}(t;0,1)}{\Phi(t;0,1)}, w(t) \coloneqq v(t) \cdot [v(t)+t]. \quad (9)$$

About these update equations. i) The amount of change depends on how "surprising" the observation is. For example, in the regime $y \cdot \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\mu} < 0$ the function $v(\cdot)$ grows almost linearly. ii) The amount of change is proportional to the variance $\tilde{\sigma}_{i,j}^{\boldsymbol{2}}$ relative to all other active weight beliefs. Hence $\sigma_{i,j}^2/\Sigma$ is a weight-specific learning rate and weights with low variance act as an "anchor". iii) Every observation leads to a reduction in variance, and positive examples increase the means of active weights whereas negative examples decreases them.

These update equations lead to a natural online learning algorithm in which the parameters of the prior weight distribution $(\boldsymbol{\mu}, \boldsymbol{\sigma^2})$ are initialised to reflect any prior information (such as historical average CTR). For the first training example $(\boldsymbol{x}, y)$, the posterior parameters $(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\sigma}}^{\boldsymbol{2}})$ are calculated. After that, the previously obtained posterior is used as the prior for the next update, i.e., $\boldsymbol{\mu} \leftarrow \tilde{\boldsymbol{\mu}}$ and $\boldsymbol{\sigma^2} \leftarrow \tilde{\boldsymbol{\sigma}}^{\boldsymbol{2}}$.

3.3.2. Predictive Distribution

The prediction equation represents a mapping from a set of posterior parameters and an input $\boldsymbol{x}$ to a predictive distribution over y, $(\boldsymbol{\mu}, \boldsymbol{\sigma^2}, \boldsymbol{x}) \mapsto p(y)$. In terms of Figure 1, the calculation can viewed as following the message passing schedule towards the factor q which encompasses the label y. Alternatively, given equations (2) and (3) the predictive distribution can be derived as the integral:

$$p(y|\boldsymbol{x}) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} p(y|\boldsymbol{x}, \boldsymbol{w}) \cdot p(\boldsymbol{w})\, d\boldsymbol{w}. \quad (10)$$

This can be solved exactly and in closed form to give:

$$p(y|\boldsymbol{x}) = \Phi\left(\frac{y \cdot \boldsymbol{x}^T\boldsymbol{\mu}}{\Sigma}\right) \quad (11)$$

Comparing this expression to the sampling distribution (2) shows that the additional variance from the posterior over the active weights has the effect of pushing the predictive probability towards 0.5, and hence to increase its binary entropy. In the limiting case of zero variance on the weights, the sampling distribution (2) is recovered.

3.3.3. Dynamics

So far the model assumes a stationary data distribution $p(y|\boldsymbol{x})$ and hence in the limit of infinitely many observations the variances $\boldsymbol{\sigma}^2$ would converge towards zero and learning would come to a halt. We have investigated models of dynamics that can account for changes in the environment.

The simplest approach uses the idea of a Kalman filter with covariance $\tau^2\boldsymbol{I}$ per unit time on the weight vectors

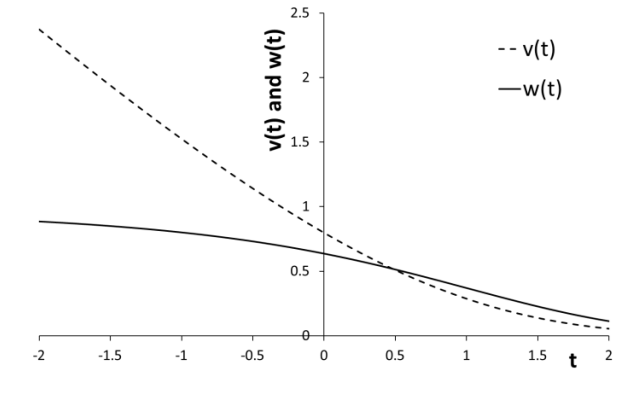

Figure 2: Plot of learning step size functions $v(\cdot)$ (for the mean update) and $w(\cdot)$ (for the variance update). The function w is the negative first derivative of v.

leading to an effective dynamics correction at each time step according to

$$\tilde{\sigma}_{i,j}^2 = \sigma_{i,j}^2 + \tau^2. \quad (12)$$

The dynamics model above is unsatisfactory in that the variance of the posterior of a weight that is rarely active can grow larger than the prior variance. As a consequence the corresponding feature can become a "dark horse" and can receive very large updates if it becomes active for a surprising training example.

We devised an alternative model of dynamics which converges back to the prior rather than to the uniform distribution in the limit of zero data and infinite time. It is based on the idea of gradually forgetting the influence for past data $\mathcal{D} \coloneqq (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_T, y_T)$. In the IID case, the posterior distribution $p(\boldsymbol{w}|\mathcal{D})$ is the product of the prior $p(\boldsymbol{w})$ and the product of likelihood terms $\prod_{t=1}^{T} p((\boldsymbol{x}_t, y_t|\boldsymbol{w})$. Suppose now that per unit time the likelihood of the data seen so far is subjected to a noise process with parameter $\varepsilon \ll 1$ decreasing its influence on the posterior:

$$p(\boldsymbol{w}|\mathcal{D}) \propto \left[\prod_{t=1}^{T} p((\boldsymbol{x}_t, y_t|\boldsymbol{w})\right]^{1-\varepsilon} \cdot p(\boldsymbol{w}). \quad (13)$$

In the case of Gaussian message passing it is possible to store the prior $p(\boldsymbol{w})$ in terms of its parameters $\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0^{\boldsymbol{2}}$ and recover the approximate likelihood from the approximate posterior using Bayes' rule, yielding the following dynamics corrections:

$$\tilde{\sigma}_{i,j}^2 = \frac{\sigma_{0,i,j}^2 \sigma_{i,j}^2}{(1-\epsilon)\sigma_{0,i,j}^2 + \epsilon\sigma_{i,j}^2}. \quad (14)$$

$$\tilde{\mu}_{i,j} = \tilde{\sigma}_{i,j}^2 \cdot \left[(1-\epsilon)\frac{\mu_{i,j}}{\sigma_{i,j}^2} + \epsilon\frac{\mu_{0,i,j}}{\sigma_{0,i,j}^2}\right]. \quad (15)$$

In the limit of zero data and infinite time, this dynamics converges back towards the prior.

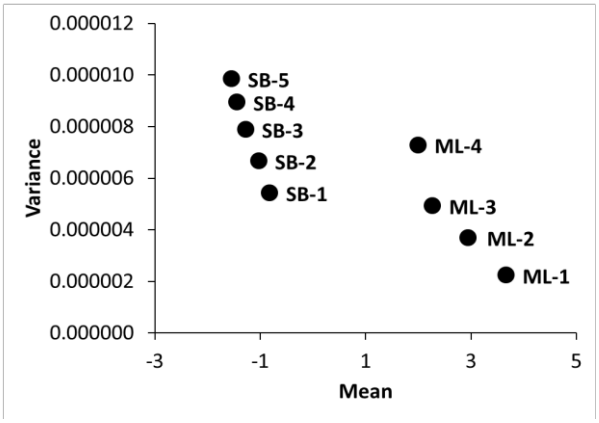

Figure 3: Scatter plot of the means and variances of the display position feature bag. ML refers to mainline, SB to sidebar (also known as right rail).

Computationally, the dynamics updates for both cases can be delayed and applied cumulatively at the next data-driven update of the respective weight component.

4. Web Scale Implementation

4.1. Model Size and Pruning

In order for a large scale predictive model to be deployed in practice it is necessary to limit its memory footprint. Due to the sparse encoding of features, adPredictor's complexity (and hence memory footprint) very much depends on the support of the input distribution. For example a user ID feature might allow for, say, six billion different values, but the number of weights that differ from the prior would only correspond to those user IDs that have actually been observed in the training sample, and it is only those weights that require explicit memory.

Due to the heavy-tailed nature of the distribution of items observed on the web such as users, queries, ads etc., the model's complexity grows with the number of training examples used. On the one hand it is necessary to track each newly discovered feature value in case it will reappear frequently. On the other hand, many of those values will only re-occur very rarely and the model should thus not waste memory on them. We have therefore devised a pruning criterion that allows model compression without losing much prediction accuracy.

The criterion for pruning a particular weight (i.e., resetting its parameters to the prior) is based on the influence that weight has on the prediction of a completely unknown input $\boldsymbol{x}_0$ with that feature value on. We assume that the prior parameters are constant within feature bags but may differ across, $\forall j\colon (\mu_{0,i,j}, \sigma^2_{0,i,j}) = (\mu_{0,i}, \sigma^2_{0,i})$. Then the pruning criterion is given by the Kullback-Leibler divergence between two Bernoulli variables with success probabilities corresponding to the

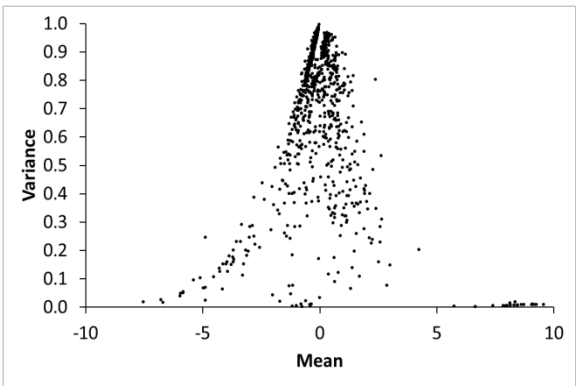

Figure 4: Scatter plot of means and variances of weights for user ID feature bag. The peak of the triangle represents the prior.

predictions using the prior and posterior weight distributions, respectively. For weight $w_{k,l}$ we have:

$$KL\left(\Phi\left(\frac{\sum_{i=1}^{N}\mu_{0,i}}{\beta^2+\sum_{i=1}^{N}\sigma_{0,i}^2}\right)\middle|\Phi\left(\frac{\sum_{i\neq k}^{N}\mu_{0,i}+\mu_{k,l}}{\beta^2+\sum_{i\neq k}^{N}\sigma_{0,i}^2+\sigma_{k,l}^2}\right)\right)$$

In other words, if changing the weight parameters from posterior to prior does not make a noticeable difference, the weight can be pruned.

4.2. Parallel Training

The process of prediction is inherently parallelisable: load the model parameters on the set of machines and predict in parallel. Training is more problematic because the adPredictor algorithm is inherently an online learning algorithm where learning from the next example builds on what has been learnt from the previous examples.

Nonetheless, an elegant parallel version of the learning algorithm can be devised based on the message passing formulation from the previous section. We consider the case of data-parallelism (Chu, et al., 2007) where training examples can be distributed across compute nodes. Clearly, off-the-shelf parallelisation using locks on shared memory will be inefficient for low-cardinality features that will be active in many training examples.

The idea is to construct a new distributed factor graph that is formally equivalent to the full factor graph that includes all the training examples as factors connected to the weights, i.e., replicating the structure below the weights in Figure 1 for each training example. The online learning algorithm from Section 3 can be viewed as a particular message passing schedule in this graph where each training example is taken in turn, receiving one message from the prior and sending one message back towards the weights. The new distributed factor graph has replicated weight variables on each compute node, which are connected to a master instance of the weight variables.

The parallel version of the algorithm can now be viewed as a message passing schedule on this new, distributed

factor graph. The simplest instance sends down the prior message from the head node to the compute nodes, each of which runs the online learning algorithm from Section 3, then sends messages back from its weight variables to the master weight variables, which accumulate the results from all the compute nodes.

5. Closing the Loop

One of the most intriguing aspects of integrating a CTR prediction algorithm into a complex ad delivery system is the dynamic feedback loop thus created: The algorithm's predictions influence the selection of ads to be shown and hence determine the future composition of the training sample for the algorithm. This appears to be a rather general phenomenon, and a similar phenomenon has been described in (Herbrich, Minka, & Graepel, 2007) for the TrueSkill rating system.

One problem caused by this feedback loop is that prediction becomes harder if the algorithm predicts successfully and leads to the selection of ads with higher CTR: Typical CTRs are below 50% and hence increasing CTR leads to higher source entropy (17), an effect which could lead to the misconception that the performance of the algorithm is degrading over time.

The second problem is the trade-off between exploration and exploitation. In order to be able to estimate the CTR of a new ad, it is necessary to present the ad to users and observe their click/non-click response. At the same time it is in the interest of everyone involved to show high-CTR ads to the user based on what is already known. The exploration problem can be addressed by exploiting the fact that adPredictor maintains uncertainty about the weights $\boldsymbol{w}$, and hence about the CTR of any particular ad impression $\boldsymbol{x}$. Instead of always feeding the expected CTR to the ad auction, the system can sample from the posterior weight distribution $p(\boldsymbol{w}|\mathcal{D})$ when evaluating the prediction using (2), an idea that goes back to Thompson (Thompson, 1933). This has the effect of bubbling up ads about whose CTR the system has a high degree of uncertainty left.

6. Numerical Results

6.1. Posterior Weight Parameters

In order to illustrate the kind of results obtained from training on the production system, consider scatter plots of posterior weight parameters.

Figure 3 shows a scatter plot for the display position feature which takes nine different values. Each dot represents the posterior mean and variance of one of the feature weights. The mainline (ML) weights have higher means than the right-rail or sidebar (SB) weights as would be expected from the page layout. Also, the variances for the positions that are more frequently observed and have higher CTR on average are lower.

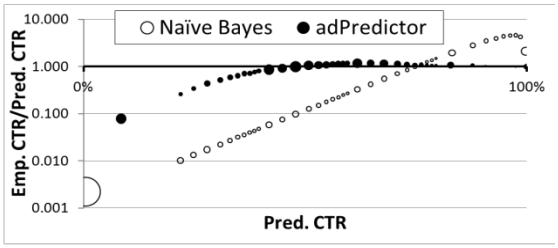

Figure 5: Calibration plot of Naïve Bayes (NB) and adPredictor before calibration. NB is ill-calibrated by orders of magnitude for small CTR.

Figure 4 shows a scatter plot for a feature representing user ID. This is an extremely high cardinality feature and as a consequence, the residual variance is much higher. Note that the top of the triangle laid out by the dots corresponds to the weight prior. Dots further down and out correspond to feature values that have been frequently observed and have therefore moved away from the prior. Extreme outliers to the right can be considered bots.

6.2. Comparison with Naïve Bayes

We now present an evaluation of the predictive performance of adPredictor. We focus on a natural alternative for web-scale classification: the Naïve Bayes (NB) classifier (Hand & Yu, 2001). The training for NB is extremely light-weight and only requires counting feature values conditional on the label, a process that can easily be parallelised. However, NB makes the strong assumption that the feature values are independent of each other given the label and hence tends to be non-calibrated. We therefore used isotonic regression (Zadrozny & Elkan, 2001) to calibrate NB. We evaluate the algorithms along three dimensions: prediction accuracy, calibration, and ranking. The training sample consists of a stream of ad impressions together with a click/non-click label extracted from 14 days of production data. The subsequent day is split into two subsamples, one of which is used for the calibration using isotonic regression, the other half is used for testing. We cannot disclose the exact feature set used to represent the impressions, but refer the reader to the general discussion of features in Subsection 2.2.

The empirical cross entropy or log-score is given by

$$CE := \frac{1}{T}\sum_{t=1}^{T} y_t \log \hat{p}_t + (1-y_t)\log(1-\hat{p}_t). \quad (16)$$

We measure the prediction accuracy in terms of relative information gain (RIG) on a test set. Given the empirical CTR of the data, $\bar{p} := \sum_{t=1}^{T} y_t / T$ we define the information gain as $IG := CE + H(\bar{p})$ where H is the entropy defined by

$$H(\bar{p}) := -(\bar{p}\log\bar{p} + (1-\bar{p})\log(1-\bar{p})). \quad (17)$$

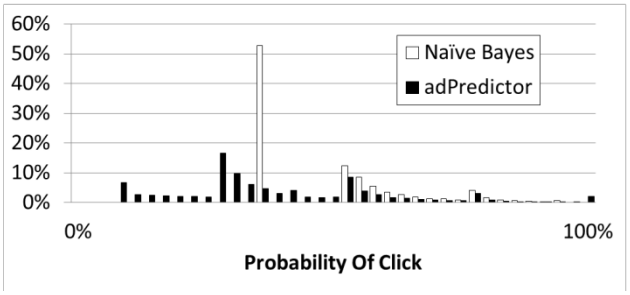

Figure 6: Histogram over impressions of predicted probabilities for calibrated Naive Bayes and adPredictor. Clearly adPredictor makes more spread-out and hence more informative predictions.

We define the relative information gain as the ratio $RIG := IG/H(\bar{p})$. Since $H(\bar{p})$ is the maximally attainable value of IG it quantifies the information gain relative to the source entropy.

To quantify the quality of the ranking that results from ordering the test examples according to the predicted probability, we also compare the algorithms' areas under the RO curve (AUC).

Algorithm	RIG	AUC
adPredictor	61.24%	95.6%
adPredictor (calibrated)	61.35%	95.6%
Naïve Bayes	-41.54%	89.4%
Naïve Bayes (calibrated)	33.86%	89.3%

It can be seen that in terms of RIG, adPredictor has a clear edge on NB, which performs worse than the grand average CTR predictor, but can gain if calibrated. In terms of AUC, adPredictor is clearly superior to NB, and calibration does not make a difference.

Results for the calibration of adPredictor and Naïve Bayes can be seen in Figure 5 and Figure 6. Impressions have been grouped according to predicted CTR and the vertical axis shows the ratio of empirical versus predicted CTR on a log scale with bubble size indicating number of impressions. Figure 5 shows how un-calibrated NB is in comparison to adPredictor. After calibration by isotonic regression, it can be seen in the histogram of Figure 6 that adPredictor–with its prediction spread out further—makes much more informative predictions than NB.

7. Conclusions and Future Work

We presented adPredictor, a simple, powerful Bayesian online learning algorithm used for CTR prediction in Bing's Sponsored Search advertising. We are currently exploring its use for related tasks within Bing including organic search, display and contextual advertising, based on other target signals such as conversion. We are also investigating the use of alternative models, such as the feature-based collaborative filtering method Matchbox (Stern, Herbrich, & Graepel, 2009) for personalisation.

Acknowledgements

We would like to thank Guy Dassa, Ewa Dominowska, Oleg Isakov, Deepak Pawar, Siddhartha Sinha, Jill Goldschneider, Patrice Simard, and many others in adCenter without whom this work would not have been possible. We would also like to thank Onno Zoeter, Tom Minka, Anton Schwaighofer and David Stern.

References

Agarwal, D., & Chen, B.-C. (2009). Regression based Latent Factor Models. *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Broder, A., & Josifovski, V. (2009). *Lecture Introduction to Computational Advertising*. Stanford University, Computer Science. Online Lecture Notes.

Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., et al. (2007). Map-Reduce for Machine Learning on Multicore. *Neural Information Processing Systems (NIPS) 19* .

Edelman, B., Ostrovsky, M., & Schwarz, M. (2007). Internet Advertising and the Generalized Second Price Auction: Selling Billions of Dollars Worth of Keywords. *American Econ. Rev.*, *97*(1), 242-259.

Hand, D. J., & Yu, K. (2001). Idiot's Bayes - Not so stupid after all? *International Statistical Review, 69*(3), 385-398.

Herbrich, R., Minka, T., & Graepel, T. (2007). TrueSkill: A Bayesian Skill Rating System. *Advances in Neural Information Processing Systems 20* (pp. 569-576). The MIT Press.

Kschischang, F., Frey, B. J., & Loeliger, H.-A. (2001). Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, *47*, 498--519.

Minka, T. (2001). *A family of algorithms for approximate Bayesian inference*. PhD thesis, MIT, CSAIL.

Richardson, M., Dominowska, E., & Ragno, R. (2007). Predicting Clicks: Estimating the Click-Through Rate for New Ads. *World Wide Web (WWW) conference*, (pp. 521-529).

Stern, D., Herbrich, R., & Graepel, T. (2009). Matchbox: Large Scale Bayesian Online Recommendations. *World Wide Web (WWW) conference*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning - An Introduction*. MIT Press.

Thompson, W. R. (1933). On the likelihood that one unknown exceeds another in view of the evidence of two samples. *Biometrika, 25*, 285-294.

Zadrozny, B., & Elkan, C. (2001). Obtaining calibrated probability estimates from decision trees and naive Bayes classifiers. *International Conference on Machine Learning (ICML)*, (pp. 609-616).