LASER: A Scalable Response Prediction Platform For Online Advertising

Deepak Agarwal
LinkedIn Inc.
dagarwal@linkedin.com

Bo Long
LinkedIn Inc.
blong@linkedin.com

Jonathan Traupman
LinkedIn Inc.
jtraupman@linkedin.com

Doris Xin
LinkedIn Inc.
dxin@linkedin.com

Liang Zhang
LinkedIn Inc.
lizhang@linkedin.com

ABSTRACT

We describe LASER, a scalable response prediction platform currently used as part of a social network advertising system. LASER enables the familiar logistic regression model to be applied to very large scale response prediction problems, including ones beyond advertising. Though the underlying model is well understood, we apply a whole-system approach to address model accuracy, scalability, explore-exploit, and real-time inference. To facilitate training with both large numbers of training examples and high dimensional features on commodity clustered hardware, we employ the Alternating Direction Method of Multipliers (ADMM). Because online advertising applications are much less static than classical presentations of response prediction, LASER employs a number of techniques that allows it to adapt in real time. LASER models can be divided into components with different re-training frequencies, allowing us to learn from changes in ad campaign performance frequently without incurring the cost of retraining larger, more stable sections of the model. Thompson sampling during online inference further helps by efficiently balancing exploration of new ads with exploitation of long running ones. To enable predictions made with the most recent feature data, we employ a range of techniques, including extensive caching and lazy evaluation, to permit real time, low latency scoring. LASER models are defined using a configuration language that ties together the training, validation, and inference pieces and permits even non-programming analysts to experiment with different model structures without modifications to code or interruptions to running servers. Finally, we show via extensive offline experiments and online A/B tests that this system provides significant benefits to prediction accuracy, gains in revenue and CTR, and reductions in system latency.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models; I.2.6 [**Artificial Intelligence**]: Learning; H.5.3 [**Information Systems**]: Web-based Interaction

WSDM'14, February 24–28, 2014, New York, New York, USA.

http://dx.doi.org/10.1145/2556195.2556252.

Keywords

Machine learning, response prediction, logistic regression, scalability, computational advertising

1. INTRODUCTION

Online advertising is a multi-billion dollar business and forms a large swath of the internet economy. It is practiced in various forms like sponsored search advertising, contextual advertising and display advertising [5]. These are highly automated systems that auction ad space for user visits on various publisher pages. The auction entails ranking eligible ads as a function of advertiser bid and ad click-through rates (CTR). The advertiser payoff is often determined through a second-price auction where the winner pays the minimum price required to win the auction. For performance based advertising, such a payment is only made when a user clicks on an ad or performs some advertiser specified post-click action.

Large scale online advertising systems rely heavily on machine learned models that can accurately predict CTR. This is a difficult learning problem due to the curse of dimensionality and data sparseness, but such methods have been studied extensively in the last few years in the machine learning and data mining communities. However, deploying such methods for large scale online advertising systems in a reliable and cost-effective fashion involves non-trivial challenges, beyond fitting machine learning models to large data.

An ideal system should allow for scalable model training using standard and well understood commodity clustered hardware to reduce maintenance cost; it should be easy to test a new model on live traffic without much effort on coding; offline metrics that are indicative of online performance should be available to terminate below-par models quickly; the system should also provide a mechanism for exploration to avoid ad starvation; finally, runtime computations should scale under latency constraints. The main theme of this paper is to provide such an end-to-end holistic solution. We argue that building machine learning models without consideration to all such issues is not the right approach in practice for advertising systems. Machine learning methods should not be only tied to out-of-sample predictive accuracy. Careful attention to all such practical issues is important.

Background We begin with a brief background of an online advertising system. To simplify the exposition, we assume the pay-per-click model where an advertiser pays only when a user clicks on an ad. The flow works as follows: an ad server receives a request when a user visits some publisher page. Depending on the application, we may have information about the user and also the context in

which the request was made. For instance, in sponsored search the user specified query is a necessary and important context to match ads, while user attributes might be helpful but of secondary importance. For display advertising, context information includes the device, the ad placement information, and other information about how and where the ad is shown. In this setting, user attribute information plays a more important role since user intent is typically weak. For a given request, we rank the set of eligible ads based on some function of advertiser bid and CTR (e.g., the product of bid and CTR). Such ranking has to be done for thousands of ads under tight latency constraints (tens of milliseconds).

Our **contributions** are as follows. We introduce LASER, a holistic end-to-end machine learning solution to deploy response prediction models based on logistic regression to large online advertising system. We partition the logistic regression model into a purely feature based *cold-start* component and an ad-campaign-specific *warm-start* component. The *cold-start* component is trained on commonly available commodity clustered hardware running MapReduce using the scatter-and-gather Alternating Direction Method of Multipliers (ADMM) algorithm [4], whereas the *warm-start* component learns residual campaign-level variation through frequent retraining. LASER also supports explore-exploit via Thompson sampling [19], which is an easy-to-implement yet powerful algorithm that draws samples from the posterior distribution of warm start coefficients. To permit low-latency computations at runtime, we introduce a range of techniques that take advantage of extensive caching and lazy evaluation. Our models are defined through a flexible configuration language that ties training, validation and inference pieces together, enabling testing new models without change to production code. Our system has been fully deployed on a large social media site, and we illustrate our method through extensive experiments using both offline data and online A/B tests in a real online advertising system.

2. RELATED WORK

Significant work have been done on machine learning models to accurately predict CTR for advertising applications [16, 17, 10, 1, 6, 11, 20]. However, these papers only focus on improving offline model training to improve out-of-sample predictive accuracy; scalable model training and good feature construction are the primary focus. We make a new contribution to this area by providing a scalable model fitting approach through ADMM that can scale on standard MapReduce infrastructure. Explore/exploit via Thompson sampling for advertising was recently studied by [6] but in a simulation and replay framework; no A/B test results on a live system were reported as we do in this paper. The issue of scaling runtime computations has been addressed by some papers separately through predictive indexing [2, 9], where the main idea is to perform pre-computations to do fast retrieval in an approximate fashion. Our approach is to reduce the generality of the model class (we confine ourselves to generalized linear models) but take a more general approach from a systems perspective, leveraging lazy evaluation and extensive caching for scalability. Unlike all the previous papers, to the best of our knowledge, this is the first paper that provides an end-to-end solution to deploy sophisticated machine learning models for a large scale online advertising system in an efficient, reliable and cost-effective fashion. The closest work that is similar in spirit to ours is [16], but they mainly provide an extensive summary of experience and issues when fitting large scale machine learning models in an offline fashion. The focus of that paper is building an efficient and robust offline training pipeline to minimize out-of-sample predictive accuracy metrics; issues like agile model deployment and efficient runtime computations are not the focus.

3. OUR MODEL

In this section we describe the components of the logistic regression model used by LASER. This is followed by a description of our scalable model fitting procedure and our explore/exploit strategy via Thompson sampling.

3.1 Model Description

Notations. Throughout the paper, we will use index i to denote user i, index j to denote item j (e.g. ad campaigns), and index t to denote context t (e.g. time of day, day of week, where the item was shown, etc). The binary response (click or non-click) for a user i on an item j for context t is denoted as y_{ijt}. Since the users, items, and contexts can all have features, we denote the features for user i as x_i, the features for item (campaign) j as c_j, and the features for context t as z_t.

Model. Since the response y_{ijt} is binary, it is natural to assume a Bernoulli model with logistic link function

$$y_{ijt} \sim \text{Bernoulli}(p_{ijt}), \tag{1}$$

where

$$p_{ijt} = \frac{1}{1+\exp(-s_{ijt})}. \tag{2}$$

We model the log-odds s_{ijt} as

$$s_{ijt} = \omega + s^{1,c}_{ijt} + s^{2,c}_{ijt} + s^{2,w}_{ijt}, \tag{3}$$

$$s^{1,c}_{ijt} = x_i'\alpha + c_j'\beta + z_t'\gamma, \tag{4}$$

$$s^{2,c}_{ijt} = x_i'Ac_j + x_i'Cz_t + z_t'Bc_j, \tag{5}$$

$$s^{2,w}_{ijt} = \delta_j + x_i'\eta_j + z_t'\xi_j. \tag{6}$$

Here, ω is the global intercept, $s^{1,c}_{ijt}$ consists of all first order interactions or main effects, $s^{2,c}_{ijt}$ consists of all second order interactions among features. The parameters ω, α, β, γ, A, B and C are global coefficients that are shared across all events. We call this set of unknown coefficients $\mathbf{\Theta_c} = \{\omega, \alpha, \beta, \gamma, A, B, C\}$ the **cold-start** component, because for a new user or ad campaign, we can still use features to obtain a reasonable estimate of CTR through $s^{1,c}$ and $s^{2,c}$.

Since the number of ad-campaigns is relatively small compared to the number of users, there is heterogeneity in the sample size available across campaigns. For campaigns with large data, one could estimate residual campaign specific idiosyncrasies by adding a **warm-start** component $s^{2,w}$ in equation (3). The warm-start parameters would be denoted by $\mathbf{\Theta_w} = \{\delta_j, \eta_j, \xi_j\}$, $j = 1, \cdots, J$. This component provides a mechanism to learn campaign-specific corrections and smoothly transition from cold start to warm start based on the sample sizes of campaigns. Such a transition is possible by allowing for regularization/shrinkage on coefficients, obtained by constraining them through prior distributions that we describe below.

Priors. Since the feature set is often high-dimensional, especially for second-order interaction terms in cold-start and per-campaign coefficients in warm-start, proper shrinkage and regularization of coefficients are required to avoid over-fitting. We use Gaussian priors for all the parameters $\mathbf{\Theta} = \{\omega, \alpha, \beta, \gamma, A, B, C, \delta_j, \eta_j, \xi_j\}$. Specifically, for scalars ω and δ_j, the priors become $\omega \sim N(0, \sigma_\omega)$, and $\delta_j \sim N(0, \sigma_\delta)$. For coefficient vectors such as α, β, γ, η_j, and ξ_j, we let $\alpha \sim MVN(0, \sigma_\alpha I)$, $\eta_j \sim MVN(0, \sigma_\eta I)$ and so forth. For coefficient matrices A, B and C, we assume entries are i.i.d.

Gaussian with mean 0 and variance σ_A, σ_B and σ_C respectively. We estimate the prior variance through cross-validation using a tuning set. Also, for simplicity, in our own implementation we assume $\alpha, \beta, \gamma, A, B$, and C all share the same prior variance, the δ_j's have the same prior variance, and η_j's and ξ_j's share the same prior variance for all campaigns.

3.2 Large Scale Model Fitting

Model fitting on a single machine. When the data size is relatively small and can be loaded into the memory of a standard machine, the model fitting proceeds by using routine methods available to fit a logistic regression with Gaussian priors (L_2 regularization).

Posterior mode of $(\boldsymbol{\Theta_c}, \boldsymbol{\Theta_w})$ can be obtained via any convex optimization algorithm like conjugate gradient, L-BFGS [22], trust region [15], and others. The model can be trained frequently to rapidly update the warm-start coefficients.

3.2.1 Model Training with Large Data

For large online advertising systems where training data with hundreds of millions of observations and hundreds of thousands of coefficients (if not more) is routine, fitting logistic regression is challenging. In our system, it is also desirable to have a model fitting procedure that runs on standard distributed computing infrastructure like Hadoop. This simplifies the production workflow and provides a more reliable system.

Since the coefficient $\boldsymbol{\Theta_c}$ is more stable and changes slowly relative to $\boldsymbol{\Theta_w}$, we can have different training frequencies for cold-start and warm-start components. For the slow-changing $\boldsymbol{\Theta_c}$, we have found weekly or bi-weekly training cycles for this component to be adequate in practice. The warm-start coefficients $\boldsymbol{\Theta_w}$ needs to be re-trained more frequently since it is time sensitive and changes with the activation and termination of ad campaigns. We have found that re-training this component as frequently as possible (e.g., on the order of minutes to hours) is important.

Note that for known cold-start coefficients, the warm-start coefficients can be computed independently for each ad campaign. Thus, we are able to decompose a big logistic regression into several small logistic regressions that can be trained independently of each other. This can be done routinely on Hadoop by splitting the data using campaign id as key in the map phase, and running independent per campaign regression in the reduce phase. To train the cold-start component, we adopt the ADMM algorithm that scales through a scatter-and-gather approach. It partitions data and runs an independent regression on each partition, but obtains a consensus estimate by combining estimates across partitions after each iteration. A few iterations of such a scatter-and-gather approach are enough for convergence, provided we are careful in how we initialize and adjust the step size of the algorithm across iterations.

3.2.2 Estimating Cold-Start Coefficients on Hadoop

Traditional logistic regression model fitting approaches such as conjugate gradient or trust region require many iterations (e.g. at least in the order of 100s) to converge; running them on a Map-Reduce infrastructure like Hadoop usually introduces significant I/O overhead. This is mainly because each iteration is a new Map-Reduce job, and since the input and output of mappers and reducers are through disk, popular Map-Reduce infrastructures like Hadoop are not suitable for iterative algorithms that require large number of iterations. Although new infrastructures, such as Spark [21], have emerged recently that are more efficient for iterative machine learning algorithms, their development is still in an early stage; full integration to the Hadoop system in a production environment could have considerable costs in terms of stability, multi-user resource management, and software compatibility.

In our LASER system, we adopt the Alternating Direction Method of Multipliers (ADMM) algorithm [4] to obtain cold-start coefficient estimates, and our implementation is developed to work with the widely adopted Hadoop Map-Reduce infrastructure. ADMM can be shown to converge to the posterior mode obtained by fitting logistic regression to the entire data on a single machine. Compared to other popular fitting algorithms, such as conjugate gradient or trust region, ADMM requires significantly fewer outer iterations (i.e. Map-Reduce jobs) to converge. After applying a few modifications, like better initialization and step size selection, we have found 5-10 outer iterations of ADMM provides good predictive performance in practice. The number of inner iterations required to fit smaller logistic regression to each partition is still significant, but that does not involve I/O overhead, hence is efficient on Hadoop.

3.2.3 The ADMM Algorithm to Fit L_2 Regularized Logistic Regression

Suppose we partition the data randomly into R partitions. For each partition r, denote the data as d_r and coefficient as $\boldsymbol{\Theta}_r$. Also denote the logistic loss function (i.e. negative log-likelihood) for each partition r as $\mathscr{L}_r(\Theta_r; d_r)$, and the L_2 penalty function on Θ as $\mathscr{P}(\Theta)$. The main idea of ADMM is to assume each partition r has its own coefficient $\boldsymbol{\Theta}_r$, and assume our interest is to obtain a global consensus estimate $\boldsymbol{\Theta}$, obtained by imposing a constraint $\boldsymbol{\Theta}_r = \boldsymbol{\Theta}$ for all $r = 1 \ldots R$. The optimization problem is formally given by

$$\min \sum_{r=1}^{R} \mathscr{L}_r(\boldsymbol{\Theta}_r; d_r) + \mathscr{P}(\boldsymbol{\Theta}), \tag{7}$$

subject to

$$\boldsymbol{\Theta}_r - \boldsymbol{\Theta} = 0, r = 1 \ldots R. \tag{8}$$

The ADMM algorithm for large scale logistic regression can be described as follows: At iteration $l+1$, the estimate for $\boldsymbol{\Theta}_r$ which is denoted by $\boldsymbol{\Theta}_r^{l+1}$ can be fitted independently and in parallel by

$$\boldsymbol{\Theta}_r^{l+1} = \text{argmin}_{\boldsymbol{\Theta}_r} \mathscr{L}_r(\boldsymbol{\Theta}_r; d_r) + \rho/2 \left\| \boldsymbol{\Theta}_r - \boldsymbol{\Theta}^l + \boldsymbol{u}_r^l \right\|^2 \tag{9}$$

where ρ is the step size (learning rate) that controls the convergence rate, $\boldsymbol{\Theta}^l$ is the consensus estimate of $\boldsymbol{\Theta}$ at iteration l, and $\boldsymbol{u}_r^l$ intuitively is a measure of the partition-specific bias of the coefficients. When $\boldsymbol{\Theta}_r^{l+1}$ for all $r = 1 \ldots R$ are obtained through a Map-Reduce job, we aggregate the results and compute the mean of $\boldsymbol{\Theta}_r^{l+1}$ for all $r = 1 \ldots R$ as $\bar{\boldsymbol{\Theta}}^{l+1}$, and the mean of $\boldsymbol{u}_r^l$ for all $r = 1 \ldots R$ as $\bar{\boldsymbol{u}}^l$. The consensus estimate at $l+1$, $\boldsymbol{\Theta}^{l+1}$ can be estimated by

$$\boldsymbol{\Theta}^{l+1} = \text{argmin}_{\boldsymbol{\Theta}} \mathscr{P}(\boldsymbol{\Theta}) + (R\rho/2) \left\| \boldsymbol{\Theta} - \bar{\boldsymbol{\Theta}}^{l+1} - \bar{\boldsymbol{u}}^l \right\|^2 \tag{10}$$

and

$$\boldsymbol{u}_r^{l+1} = \boldsymbol{u}_r^l + \boldsymbol{\Theta}_r^{l+1} - \boldsymbol{\Theta}^{l+1} \tag{11}$$

Expediting convergence of ADMM. In our use ADMM we have found two ways to improve the empirical convergence rate. First, we observed that the convergence of ADMM is sensitive to the initialization of the consensus $\boldsymbol{\Theta}^0$. We found a simple initialization approach that improves the convergence of ADMM significantly. The idea is to run independent logistic regression on each partition and use the mean of the parameters from the R partitions as the initialization of $\boldsymbol{\Theta}^0$ to start ADMM.

Second, we observed that the convergence and performance of ADMM is quite sensitive to the step size ρ. One possible way to improve is to adaptively change the step size ρ over iterations. In general, a larger step size pushes the partition parameters more towards

the current estimate of the global consensus; on the other hand, a smaller step size gives each partition more freedom to explore their own data to estimate partition parameters. With the consensus initialization, we have a good starting point for the global parameter. Hence, it will be meaningful to strongly force each partition's coefficients to be close to the current global consensus at the beginning. After a couple of iterations, reducing the step size to let each partition explore more may further boost the performance. Based on this idea, we found empirically that with consensus initialization, the exponential decay of ρ over iterations is a good choice which balances the global convergence with exploration within the partitions. Specifically, we adaptively adjust ρ as follows,

$$\rho^{(l+1)} = \rho^{(0)} e^{-\tau l} \tag{12}$$

where $\rho^{(0)}$ is the base learning rate, τ is the decay the rate, and l is the number of iterations.

Combining the two approaches, we are able to significantly reduce the number of iterations until convergence of ADMM. The empirical results of how the two approaches improve the ADMM convergence on our data can be found in Section 5.2.

3.2.4 Fitting Algorithm: Putting It All Together

Our algorithm for fitting the cold start and warm start components at different frequencies is described in Algorithm 1. We note that:

- Since $\mathbf{\Theta}_c$ is purely feature driven, it changes slowly and needs to be trained less frequently (e.g., once every few days or few weeks) but since $\mathbf{\Theta}_w$ captures fast-changing item specific behaviors, it has to be updated quickly (e.g., every few minutes/hours).

- By treating $\mathbf{\Theta}_c$ as a known offset in the model, $\mathbf{\Theta}_w$ can either be obtained via running independent logistic regression for each item based on a moving window, or independent online logistic regression for each item learned on data that are split in batches. Our approach to online logistic regression utilizes the dynamic state space concept by using the posterior from the previous batch process as priors for the current batch to transition into a new state. In our current LASER system we adopt the moving window approach for simplicity. We plan to test the online logistic regression approach in the future.

- Although it may appear redundant to include $\mathbf{\Theta}_w$ when estimating $\mathbf{\Theta_c}$ using ADMM, this detail is in fact critical since without including $\mathbf{\Theta}_w$ in the fitting of $\mathbf{\Theta_c}$, we will introduce bias when estimating $\mathbf{\Theta_c}$, especially when the clicks are sparse in the data (i.e. CTR is small).

Algorithm 1 LASER Model Fitting Algorithm

```
for every D days do
  Obtain the most recent data and use ADMM to fit Θ_c and Θ_w
  simultaneously by treating item-id as features.
  for every M minutes do
    Obtain the most recent data
    Split the data by item id
    for each item j in parallel do
      Treat Θ_c as a given offset and fit (or update) {δ_j, η_j, ξ_j}
      (i.e. Θ_w for item j)
    end for
  end for
end for
```

3.3 Explore-Exploit with Thompson Sampling

The goal in online advertising is to estimate CTR to maximize revenue; improving out-of-sample predictive accuracy is not our main objective. This is an explore/exploit problem, and there is a positive utility associated with serving ads that have low empirical means but high variances. We exploit ads that are known to be good based on current knowledge for near term gains, but also explore those that could be potentially good for longer-term gains.

Several explore/exploit schemes have been used in the literature. Examples include epsilon-greedy, the upper confidence bound approach (UCB) [3, 14], Gittins index [8] and Thompson sampling [19, 18, 6]. Of these, Thompson sampling works well when the posterior distribution of parameters is available. [6] showed that Thompson sampling is better than other schemes like epsilon-greedy, UCB, and exploit-only approaches through simulation studies conducted on an advertising application. They however did not test it on a real advertising system. In our LASER system, we adopt Thompson sampling and find it to work well in practice.

The main idea of Thompson sampling is to draw a sample from the posterior of parameters $(\mathbf{\Theta_c}, \mathbf{\Theta_w})$, and compute CTR based on the drawn sample instead of using the posterior mode. Intuitively, this introduces more exploration for ad campaigns that have high posterior variances due to small sample sizes. Since $\mathbf{\Theta_c}$ is estimated using large amounts of data, we assume the posterior is peaked and always use the mode as a plug-in estimate. However, we do obtain the posterior of $\mathbf{\Theta_w}$ and draw samples from it at serving time. We assume this posterior is Guassian centered around the estimated mean and posterior covariance. For simplicity and fast runtime computation, the posterior covariance matrix is assumed to be diagonal.

3.4 Efficient Computation of Interaction Terms

For computing CTR according to model described in Equation (3), we have to compute quadratic terms of the form $x_i' A c_j$, where x_i and c_j are feature vectors and A is a matrix of coefficients. Computing these interaction terms at runtime can be expensive. We reduce this cost by pre-computing

$$\tilde{A}_j = A c_j$$

or the equivalent product with x_i. In general, the number of ads is smaller than number of users, hence it is more beneficial to compute $\tilde{A}_j$.

At the time of inference, the scoring system only computes the inner product $x_i' \tilde{A}_j$, saving considerable computation. This technique is particularly effective when the set of all c_j is much smaller than the set of all x_i, since the offline pre-computation is not too costly and the set of $\tilde{A}_j$ coefficients can be effectively cached. The full set of $\tilde{A}_j$ values only needs to be computed when the original coefficients A are updated, but new and changed items will need their $\tilde{A}_j$ values re-computed, either on demand or on a frequent schedule.

4. CONFIGURATION AND INFERENCE

In addition to achieving high performance on the typical metrics used to benchmark click prediction systems, the LASER system is designed to satisfy additional requirements needed for deployment in a web-scale advertising system. In the previous section, we described using ADMM to scale training on commodity cluster systems. In this section, we present our solutions to two additional challenges posed by our application: flexible configuration and real-time inference.

4.1 Flexible Configuration

Models rarely stay constant: we are always working to improve performance, introduce new features, and accommodate changes in the training data. To facilitate rapid model experimentation, LASER specifies the feature construction and transformation structure using a JSON-based [7] configuration language that drives both the training process and online inference. The model's feature processing is expressed through the arrangement of three types of components: *sources*, *transformers*, and *assemblers*, whose configurations can be changed and refined without code modifications or service restarts. We have developed a rich library of components that can be used without modification as well as standard interfaces for the implementation of application-specific functionality.

4.1.1 Sources

Source components translate features from external sources into a numeric feature vector representation. The specification for sources is deliberately left rather vague: features can come from almost anywhere and be originally in any conceivable format. So long as the component can translate these features into a numerical vector, it can be used as a feature source.

For ad click prediction, we created sources for the user, ad campaign, and context features as described Section 3. The user and ad sources pull features off of the user profile and ad copy respectively, including both standardized features, like known entities, and less structured features, like the text of an ad. The context source encodes all information we have about where and when the ad is displayed, such as the time of the request, the page where it is to be shown, the displayed size, and the formatting of the ad.

The actual implementations of these sources are clearly specific to a given environment, but can often be reused widely within a site. For example, our user profile features are standardized across many applications and generated in a common format, so we can use the user source built for ad click models in all response prediction tasks that use user profile data as an input.

4.1.2 Transformers

Like sources, transformers emit feature vectors; however, they are further constrained to only consume other feature vectors in the system as input (i.e. feature vector outputs from sources or other transformers). Transformers implement vector-to-vector functions that modify the feature vectors into a form that is more useful for a specific prediction task. Because they both consume and emit a common datatype, transformers are even more easily reused than sources.

Some examples of the basic transformers included in the LASER system are:

Subset transformer. Sources are typically written for maximum reuse, so they often include features that may not be useful in a given application. The subset transformer allows the modeler to specify exactly which features are needed (usually based on feature selection results), so that the dimensionality of the feature space can be reduced.

Bucketing transformer. Sometimes discretizing a numerical feature into several categorical features can provide better model performance. For example, we might have the time of a request encoded as seconds since midnight, but it's more useful to encode it as one of eight discrete 3 hour day parts. The bucketing transformer allows the model writer to specify how to encode a real valued feature into set of ordinal or categorical binary features, each representing a range of the original real value.

Disjunction transformer. The disjunction transformer combines several fine-grained binary features into a single coarser binary feature using the logical-OR operation.

Interaction transformer. The interaction transformer operates on a pair of inputs and computes their outer product. As a practical implementation matter, the interaction transformer also performs a subset operation using results from feature selection, since it is rare that all interaction features in the outer product are useful, and it is far less expensive to never compute them than to subset them in a subsequent transformer.

Function transformer. The function transformer applies a user-specified function to its input vector. Among other applications, this permits feature linearization and basis expansions.

4.1.3 Assembler

The final component in the LASER feature transformation pipeline is the assembler. Unlike the other types, there is typically only a single assembler in the end of a model configuration file. The assembler takes inputs from each transformer or source output that is to be used in the model, and packages them together into a single block feature vector that is then used for model training, validation and inference.

4.1.4 Example Model

In Figure 1 we provide a simple example of a LASER model with two feature sources and three transformers. This model obtains its input features from the user and the ad campaign sources. The feature vectors produced by each of these sources are fed through subset transformers, where we keep the country and occupation features from the user, and the title and body text features from the ad campaign. These vectors then become the input to an interaction transformer that creates interaction terms between the user's country and occupation and the ad campaign's title and body text. The assembler finally gathers the output feature vectors from the subset and interaction transformers and packages them for training and inference.

Should we wish to modify what feature terms are used in this model or how they are generated — e.g. by adding a third source, changing the features kept by the subset transformers, or modifying how feature vectors flow through the pipeline — we only need to change this JSON file. Extensive model changes can thus be made without modifying code in either the training or inference systems.

4.2 Real-time Inference

Given a trained model, predicting the CTR for a new event may appear trivial, but performing such inference in real-time at web scale presents a host of challenges. Each user visit generates one or more requests to LASER to score and rank between several hundred and a few thousand ads, with each request requiring features for the user, context and each ad. LASER must gather these features and score these requests quickly — typically within 10-20 milliseconds — to prevent delays in dependent systems.

```
{
  objects: [{
    "name": "user_source",
    "class": "org.laser.source.UserSource",
    "parameters": { ... }
  }, {
    "name": "ad_source",
    "class": "org.laser.source.AdSource",
    "parameters": { ... }
  }, {
    "name": "user_subset",
    "class": "org.laser.transformer.Subset",
    "parameters": {
      "input": "user_source",
      "features": ["country", "occupation"]
    }
  }, {
    "name": "ad_subset",
    "class": "org.laser.transformer.Subset",
    "parameters": {
      "input": "ad_source",
      "features": ["title", "body"]
    }
  },{
    "name": "user_ad_int",
    "class": "org.laser.transformer.Interaction",
    "parameters": {
      "inputs": [ "user_subset", "ad_subset" ],
      "interactions": [["country", "occupation"], ["title", "body"]]
    }
  }, {
    "name": "assembler",
    "class": "org.laser.assembler.Assembler",
    "parameters": {
      "inputs": ["user_subset", "ad_subset", "user_ad_int"]
    }
  }]
}
```

Figure 1: A simple LASER model

4.2.1 *"Better Wrong Than Late"*

It often takes time to compute a feature vector. For example, the ad and user sources must retrieve feature data from a remote data store that may not be able to respond within the time window LASER has to process a request, particularly if we have to query for many items. The same situation would arise in the case of a transformer that has to perform a computationally expensive operation on its input.

One advantage of the logistic regression models we use is that they degrade gracefully when features are missing. Offline, we can compute the mean of each term in the model using an unbiased sample of impressions. Then if features are missing for a given term, we can substitute this expected value as a reasonable approximation to the term. LASER exploits this mechanism to ensure that scoring remains quick by treating as missing any features that are not immediately available.

We call this approach the "better wrong than late" philosophy, since slightly less accuracy is preferable to not being able to generate a prediction because data takes too long to arrive. Any component in a LASER model that expects to block or take excessively long to compute its output vector (typically, but not always, sources) is written such that it returns an empty value if the actual vector is not immediately available. When these empty values reach the assembler, the system will interpret them using the missing values for that part of the model. Processing of these vectors proceeds in a background thread and upon completion, the created feature vector is stored in a cache for instant lookup on subsequent requests that need the same data.

4.2.2 *Caching And Lazy Evaluation*

All terms in a LASER model take the form of a dot product between a coefficient vector and a feature vector, both of which may be indexed by one or more of the entities present in the request being scored. A typical model has 10 or more such terms. We often observe that these terms recur unchanged across different items that are being scored. For example, in a single request, we may be asked to score multiple items for the same user i, so the term $x_i'\alpha$ can be reused. Or we may have a popular item j that we need to score for all users visiting the site during a given period of time; in this case the term $c_j'\beta$ can be shared across all the users from different requests.

Partial Results Cache. We can save considerable time by caching the scalar values of each of these terms. Subsequent requests can use these cached values, saving the overhead of repeatedly computing the dot product. We call the structure that manages this operation the *Partial Results Cache (PRC)*, as it stores pieces of the final score. The entries in the PRC, which are simply these scalar dot products, are referenced using keys that uniquely identify the feature and coefficient vectors in the term. Because both the keys and values are small — unlike the feature vectors and coefficients — the PRC can be made very large, improving its effectiveness.

Keyed Feature Vectors. In order to index into the PRC, all sources and transformers emit *keyed feature vectors*, a combination of a string key and the actual feature vector. These keys obey the following two constraints:

1. For each component, the output vector's key must be computed from only information passed in on the request and/or the *keys*, but not the actual content, of any input vectors.

2. If two keyed feature vectors have equal keys, they must also have equal values.

Keyed feature vectors are implemented using *lazy evaluation* [12]. On each request, LASER components generate only the keys, which can be computed very quickly. The vector content is a *thunk* [13]: a function that can compute the actual vector on demand. On a PRC hit, the thunk is never evaluated so we do not pay the cost of constructing the actual feature vector.

How the sources and transformers obey these constraints is up to the implementor. In our application, we construct keys by combining the unique name of the source or transformer with identifying informatiom about the vector, such as the ID of the underlying item in the case of sources or the value of the input keys in transformers.

Using the PRC for real-time inference. To illustrate how the PRC helps with real-time inference, consider the user main effect term, $x_i'\alpha$, in the example model from Figure 1. Upon receiving a request, LASER passes request information into the pipeline, causing the user source to emit a keyed feature vector with a fully constructed key, which takes a form like "user_source:i". Provided all objects in the pipeline follow similar naming conventions, this form of key obeys the necessary constraints: because source and transformer names are unique, no other component will generate the same key for a different vector even if they refer to the same user i. Likewise, the keyed feature vectors for two different users will be different due to their different values of i. Finally, this form of key does not require any information other than what is available from the request.

This keyed feature vector then passes to the user subset transformer, which emits its own keyed feature vector. This vector has the key, "user_subet:user_source:i", which again follows the constraints by relying on the uniqueness of both the transformer name and input key to guarantee uniqueness of the output key. At this time, neither the user source nor the user subset transformer has done any computation to create the actual vector values.

When scoring this section of the model, the inference engine will use the keyed feature vector's key and a key that identifies the coefficient vector[1] to look up the scalar value of the term in the PRC.

[1]Because coefficients do not flow through an abitrary transformer pipeline, keying them is considerably easier. In our implementation, we simply use the identity of the coefficient term, α in this case, along with an identifier of the model, such as a unique name or version number.

If the PRC contains that value, then no further work needs to be done: all scoring for this section has been reduced to some tiny string manipulations and a cache lookup.

On a PRC miss, we first execute the thunk in the keyed feature vector produced by the user subset transformer. This thunk performs the actual subset computation, but first needs the original user feature vector, which it obtains by evaluating the thunk in the user source's keyed feature vector. Once all the lazy computations complete, the inference engine can use the final vector value along with the coefficients to compute the dot product, which it then puts into the PRC for reuse.

PRC effectiveness Figure 2(a) demonstrates the effectiveness of these techniques using a LASER model with around 20K binary features in the cold-start section. The number of features in the warm-start section depends on the total number of campaigns we train. We typically see 20-100 non-sparse entries in the feature vector for a single item in a request. For this test, we took a single instance of LASER running in a production ad serving environment and disabled the Partial Results Cache for a period of about 30 minutes. During this test, we saw response times leap 49% from a baseline of 9.4ms to a peak of 14.0ms. Upon reactivating the caches, performance immediately dropped back to pre-test levels. Below, we show the marginal cost of scoring an item using LASER with the PRC is approximately 6μs, so the additional 5.6ms latency is equivalent to scoring an additional 930 items. During this test, we were scoring 927 items on average, so disabling the PRC is equivalent to doubling the size of the request. Despite only caching computation — slow, off-process communication is handled by additional caching layers — the PRC is a major contribution to LASER's speed and scalability.

4.2.3 Performance Evaluation

To evaluate the scalability of the LASER system, we gathered one week's worth of performance data at two hour increments from an instance serving live ad traffic. Figure 2(b) plots LASER's average time to score a request against the number of requests received per second. As the request volume increases, LASER's latency is largely unaffected: we observe no significant correlation between the number of requests per second and the time it takes to process each one. This is not surprising since there are no explicit dependencies between requests (i.e. LASER inference is "embarrasingly parallel"), so additional requests can be handled easily up until machine resources are exhausted or lock contention on shared caches becomes an issue. In our experience, this tipping point occurs at around 300-350 requests per second per server with existing hardware, beyond which latencies rise dramatically.

Figure 2(c) shows LASER's time per request against the average number of items being scored per request. This graph shows a very clear ($R^2 = 0.98$) linear correlation between items scored and response time: each additional item adds about 6μs to the request time. LASER can thus easily be scaled to rank thousands of items per request while still maintaining real-time responsiveness. If the number of items increases to the point that latency becomes an issue, it is usually straightforward to break them apart so that they can be serviced by separate threads or even servers.

5. EXPERIMENTS

In this section we describe our experimental results on the following aspects: (a). A convergence study of ADMM comparing the original ADMM to the improved version that includes better initialization and adaptive selection of step size ρ. (b). An offline experiment that compares the full model in Equation (1) - (3) with a couple of baseline models. (c). An online A/B test result showing the performance of our LASER models with and without Thompson sampling.

We denote the models considered in our experiments as follows:

- CONTROL is a per-campaign CTR counting model that gets updated per 15 minutes. For each item (campaign) j, it obtains the number of clicks NC_j and number of views NV_j from the entire campaign history, and the predicted CTR is equal to NC_j/NV_j.
- COLD-ONLY is the cold-start-component-only model that models s_{ijt} by

$$s_{ijt} = \omega + s^{1,c}_{ijt} + s^{2,c}_{ijt}. \tag{13}$$

- LASER is the full model that is specified in Equation (1) - (3) that contains both the cold-start component and the warm-start component. At serving time LASER uses the posterior mean of $\mathbf{\Theta}_w$ to score, which means for new campaigns with no data in the previous batches, all the warm-start coefficients are 0.
- LASER-EE is the LASER model that uses Thompson sampling in the serving time. To be conservative we adjust the posterior variance of $\mathbf{\Theta}_{\boldsymbol{w}}$ by 0.1. We note that the adjustment of the posterior variance in Thompson sampling is a common practice that may provide better performance [6].

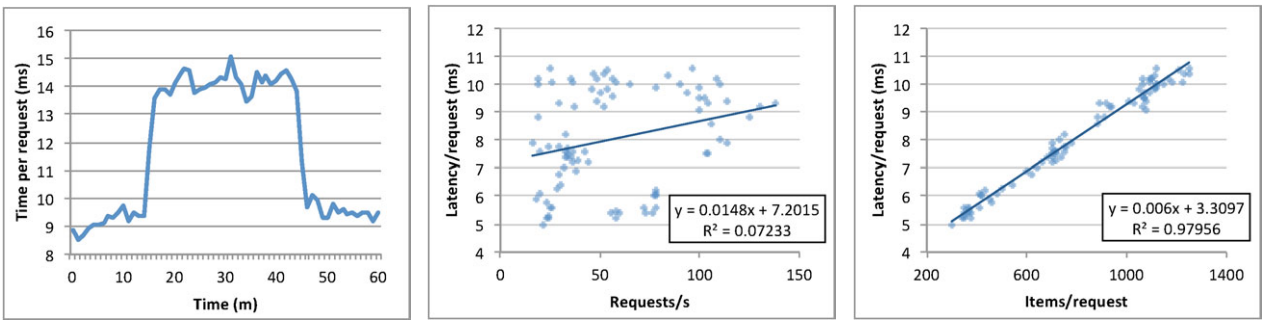
5.1 The Data

Our data for offline experiments consist of 45 days of advertising event logs for a major social network site from July 16 to August 29, 2012. We use the first 30 days of data as training and the last 15 days data as test. To fit models, we down-sampled the negatives but re-adjusted the probabilities to obtain CTR estimates on the original scale. After downsampling, both the training and test sets consist of hundreds of millions of events.

We use thousands of features for x_i and c_j for both user and item (ad campaign), and hundreds of features for the context feature z_t. For instance, the ad campaign features include n-grams, categories, advertiser characteristics, among others. The context features include time of day, day of week, the page id that the ad is shown, and the format id that the ad is placed with. We use mutual information and the minimum support criteria to do feature selection on these features and the two-way interactions between all pairs. Finally we obtain a cold-start feature set that contains roughly 20K binary features. We note that 20K is just the dimensionality of the cold-start features; the total number of warm-start features that occur in the model depends on the total number of campaigns. To predict CTR for a user, campaign and context triplet, roughly 20-100 such binary features occur per example.

5.2 The Study of ADMM Convergence

In Figure 3 we compare the convergence speed of the original ADMM algorithm [4] with the improved ADMM algorithm using the two variations described in Section 3.2: the initialization of the consensus $\mathbf{\Theta}^{\mathbf{0}}$ at the first iteration, and an exponential decay of step size ρ over iterations. In Figure 3 the average test log-likelihood versus the number of iterations is shown. We use ADMM to denote the original algorithm, ADMM-M to denote the ADMM algorithm with only the consensus initialization but without adaptive ρ and ADMM-MA to denote the ADMM algorithm with both consensus initialization and adaptive ρ. We note that the consensus initialization significantly boosts the convergence rate of ADMM in terms of the test log-likelihood. With consensus initialization, the test log-likelihood of the first iteration of ADMM-M is close to the test



(a) Effect of disabling PRC (b) Latency versus request frequency (c) Latency versus request size

Figure 2: LASER performance tests

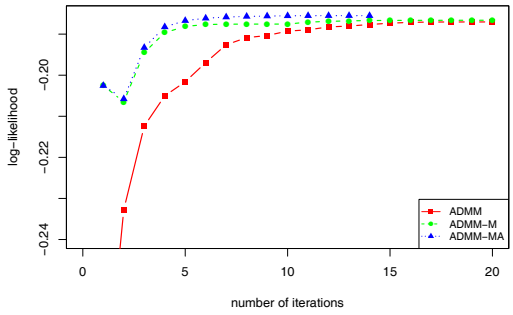


Figure 3: Convergence of the original ADMM compared to ADMM with initialization and adaptive ρ.

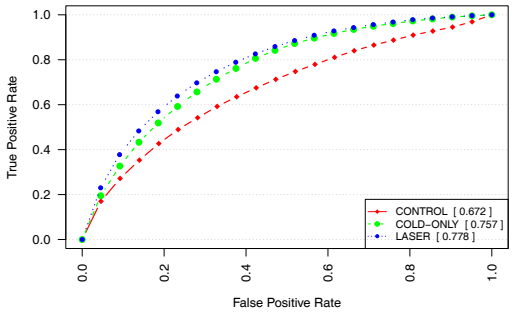


Figure 4: Offline AUC of the LASER model compared to CONTROL and COLD-ONLY baseline models.

log-likelihood from the 6th iteration of ADMM. Making ρ to adaptively evolve across iterations makes the convergence even faster. Figure 3 shows that it only takes ADMM-MA 6 iterations to reach the same log-likelihood that ADMM takes 20 iterations to reach. The two modifications to ADMM above reduced the model training time by approximately 70% for our data.

5.3 Offline Experiments

Using the data described in Section 5.1 we compare the performance of our LASER model to two baselines: the CONTROL model which only uses the campaign-CTR as the score (i.e. similar to warm-start only model), and the COLD-ONLY model which only contains the cold-start component. The ROC curves along with the AUC values (numbers in the legend of the figure) for the three models are shown in Figure 4. It is obvious that our LASER model that consists of both cold start and the warm start components provide the best performance in terms of the predictive accuracy.

5.4 Online A/B Test Results

In this section we show the result of an online A/B test experiment that was run from July 31 to August 6, 2013 in the advertising system of a major social network site. The site traffic is split randomly by user id into three buckets: CONTROL (10%), LASER (85%) and LASER-EE (5%). In general, we routinely run several such tests in our system and ramp new models that perform better than the status quo.

Overall performance. In this example, the daily performance lifts over CONTROL for LASER and LASER-EE in terms of two major business-related metrics CTR and CPM (cost per impression) are shown in Figure 5 and 6 (The overall improvements are shown

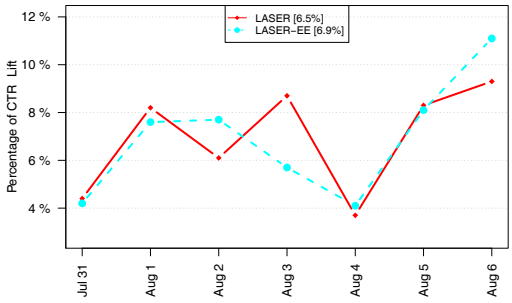


Figure 5: Percent improvement in CTR for LASER and LASER-EE versus CONTROL, Jul. 31–Aug. 6.

in the figure legends). We note that for both metrics, LASER and LASER-EE are significantly and consistently better than CONTROL (more than 6.5% CTR lift and more than 10.5% CPM lift over the 7 days). On the other hand, the overall performance of LASER and LASER-EE are similar. This is interesting, since one would intuitively expect the latter to be inferior due to the exploration cost, because both models share the same training data.

Segmented analysis by campaign warmness. To better understand the difference between LASER and LASER-EE, we split the observed data into 8 different segments based on the "degree of warmness" of the campaigns. The **degree of warmness** for a campaign at a given time is defined as the number of training samples in the training data before the specified time that contain the campaign. We are unable to reveal the actual splitting criteria of the segments, but they are obtained through logarithmic binning on campaign sample size. We also note that segment #1 contains

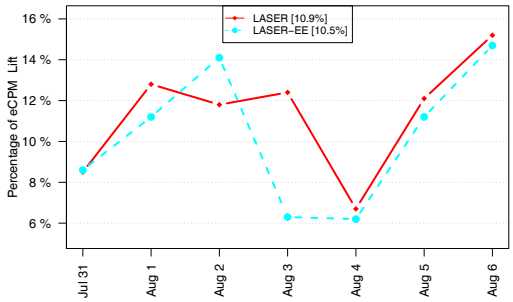


Figure 6: Percent improvement in CPM for LASER and LASER-EE versus CONTROL, Jul. 31–Aug. 6.

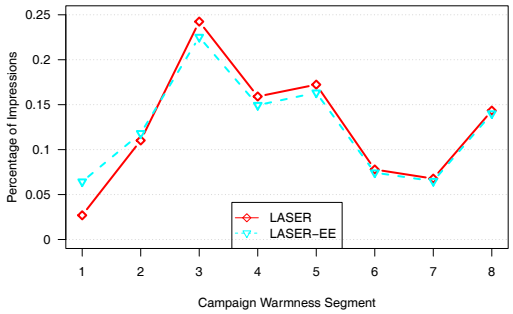


Figure 7: Fraction of impressions served by LASER-EE and LASER for the 8 campaign warmness segments.

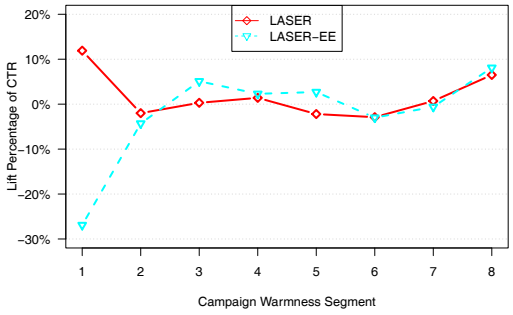


Figure 8: Percent improvement in CTR for LASER and LASER-EE versus CONTROL by campaign warmness segment.

the campaigns with almost no data in training, while segment #8 consists of campaigns that are served most heavily in the previous batches so that their CTR can be estimated quite accurately. The distributions of the number of impressions over the 8 segments served by LASER and LASER-EE are shown in Figure 7, and the CTR and CPM lifts of LASER and LASER-EE over CONTROL for the 8 campaign warmness segments are shown in Figure 8 and 9. From these figures we note the following interesting points:

- For the "cold-start" campaigns that have very little data to learn from (i.e. segment #1), the CTR and CPM performance of LASER-EE are significantly worse than LASER model and CONTROL due to its exploration property. Due to the same reason, more impressions are also served in this segment for LASER-EE relative to LASER, due to higher posterior variances of the warm-start coefficients.

- For segment #1 LASER is much better than CONTROL. This is because the cold-start features are helpful in providing a good CTR estimate for relatively new campaigns. The fact that cold-start features help is also corroborated by the AUC numbers in offline experiments.

- LASER-EE provides significantly better performance for both CTR and CPM than LASER in segments #3 to #5, and from the impression distribution plot it is clear that these three segments are the segments with the largest amount of traffic. This is expected since winners in these segments were able to win despite the optimism Thompson sampling provided to new campaigns that were participating in these auctions, hence only the best would win. This is not the case for LASER where the cold-start campaigns use an average CTR estimate. It is interesting to note that for LASER-EE the losses due to exploration are compensated by the increased performance in these segments.

Number of campaigns served. Although from our A/B test experiments LASER-EE does not provide better performance than LASER in terms of overall CTR and CPM, it provides valuable exploration data that can be used by the exploit-only model LASER. It also adds more diversity in terms of number of campaigns served to a user segment, which is good for the long-term marketplace health. This can be seen in Figure 10, the percentage increase in the number of campaigns served by LASER-EE over LASER. Note that the bucket size of LASER-EE is only 5% while LASER's bucket size is 85%. For a fair comparison we did 20 bootstraps within the LASER bucket to obtain 20 sampled 5% buckets. We then take an average of the number of campaigns and the number of impressions served over the 20 bootstrap samples for each campaign warmness segment for LASER. It is obvious from the figure that for almost all segments LASER-EE is serving significantly more campaigns than LASER, in every campaign segment.

Although it is difficult to compare LASER-EE and LASER since the latter learns from exploration data generated by the former, the results above show that in our system, LASER-EE, with an explore/exploit scheme built-in, is better than its exploit-only counterpart. In general, we recommend running Thompson sampling and its exploit only counterpart together and analyzing the explore/exploit behavior. It is worthwhile to start with the explore/exploit model on a relatively smaller fraction (e.g. 5%-10%) and gradually ramp up, based on performance.

Observed vs expected ratio. Another offline metric that we have found very useful before launching a new model on live traffic is the ratio of observed versus expected number of clicks (o/e ratio). The observed number of clicks is simply the number of clicks in the data and the expected number of clicks can be computed by the sum of the predicted CTR for each impression. Since in serving time only the top-ranked campaigns are served for each impression and there are often thousands of campaigns to be ranked in the auction, selection bias tends to choose campaigns with over-estimated CTR (also referred to as winner's curse). Therefore, a good CTR prediction model should have lower curse, reflected in an o/e ratio that is close to unity. In Figure 11 we show such ratios for LASER, LASER-EE and CONTROL. It is obvious that LASER and LASER-EE in general have a much better o/e ratio than CONTROL, and it is also not surprising that LASER-EE has the worst o/e ratio for the segment #1 ("cold-start") since it is mainly exploring there. We have found this offline metric to be a better indicator of online model performance than standard out-of-sample predictive accuracy metrics like AUC and log-likelihood that are used to measure quality of a supervised learning algorithm.

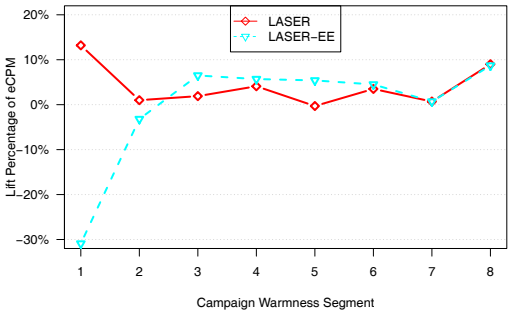


Figure 9: Percent improvement in eCPM for LASER and LASER-EE versus CONTROL by campaign warmness segment.

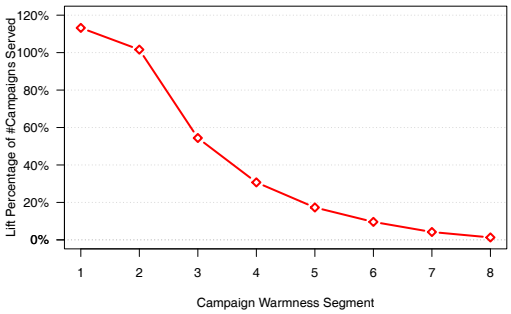


Figure 10: Increase in number of campaigns served by LASER-EE versus LASER by campaign warmness segment.

6. CONCLUSION

We introduced LASER, a scalable response prediction platform used as a part of a social network advertising system. Although the logistic regression model we used is standard, the major contribution of this paper comes from integrating several key components to obtain a scalable, reliable and accurate end-to-end system. These include: a large-scale model fitting algorithm that uses ADMM with modifications, a novel system design that enables flexible configuration and fast inference with low latency, and an explore-exploit serving scheme using Thompson sampling with interesting insights illustrated through A/B tests on a live system.

7. REFERENCES

[1] D. Agarwal, R. Agrawal, R. Khanna, and N. Kota. Estimating rates of rare events with multiple hierarchies through scalable log-linear models. In *KDD*, pages 213–222. ACM, 2010.

[2] D. Agarwal and M. Gurevich. Fast top-k retrieval for model based recommendation. In *WSDM*, pages 483–492, 2012.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[5] A. Broder. Computational advertising. In *SODA*, pages 992–992, 2008.

[6] O. Chapelle and L. Li. An empirical evaluation of thompson sampling. In *NIPS*, pages 2249–2257, 2011.

[7] D. Crockford. RFC 4627: The application/json media type for JavaScript object notation (JSON), June 2006.

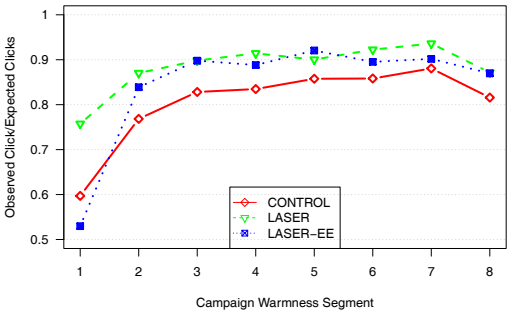


Figure 11: Ratio of observed to expected clicks served by LASER-EE and LASER by campaign warmness segment.

[8] J. C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 148–177, 1979.

[9] S. Goel, J. Langford, and A. L. Strehl. Predictive indexing for fast search. In *NIPS*, pages 505–512, 2008.

[10] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich. Web-Scale Bayesian Click-Through rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. In *ICML'10*, pages 13–20, 2010.

[11] D. Hillard, S. Schroedl, E. Manavoglu, H. Raghavan, and C. Leggetter. Improving ad relevance in sponsored search. In *WSDM*, pages 361–370, 2010.

[12] P. Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):359–411, September 1989.

[13] P. Ingerman. Thunks: a way of compiling procedure statements with some comments on procedure declarations. *Communications of the ACM*, 4(1):55–58, January 1961.

[14] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, pages 661–670. ACM, 2010.

[15] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region newton method for logistic regression. *JMLR*, 9:627–650, 2008.

[16] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica. Ad click prediction: a view from the trenches. In *KDD*, 2013.

[17] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: estimating the click-through rate for new ads. In *WWW*, 2007.

[18] S. L. Scott. A modern bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

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

[20] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. In *ICML*, pages 1113–1120, 2009.

[21] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[22] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.