Online Bayesian Sparse Learning with Spike and Slab Priors

Shikai Fang[†], Shandian Zhe[†], Kuang-chih Lee[‡], Kai Zhang[§], Jennifer Neville[¶]
University of Utah[†], Alibaba Inc.[‡], Temple University[§], Purdue University[¶]
{shikai,zhe}@cs.utah.edu[†], leekc307@gmail.com[‡], zhang.kai@temple.edu[§], neville@cs.purdue.edu[¶]

***Abstract*—In many applications, a parsimonious model is often preferred for better interpretability and predictive performance. Online algorithms have been studied extensively for building such models in big data and fast evolving environments, with a prominent example, FTRL-proximal [1]. However, existing methods typically do not provide confidence levels, and with the usage of L_1 regularization, the model estimation can be undermined by the uniform shrinkage on both relevant and irrelevant features.**

To address these issues, we developed OLSS, a Bayesian online sparse learning algorithm based on the spike-and-slab prior. OLSS achieves the same scalability as FTRL-proximal, but realizes appealing *selective* shrinkage and produces rich uncertainty information, such as posterior inclusion probabilities and feature weight variances. On the tasks of text classification and click-through-rate (CTR) prediction for Yahoo!'s display and search advertisement platforms, OLSS often demonstrates superior predictive performance to the state-of-the-art methods in industry, including Vowpal Wabbit [2] and FTRL-proximal.

***Keywords*-sparse learning, Bayesian methods**

I. Introduction

Many applications involve data with massive variables or features, such as web text classification, microarray analysis, click-through-rate (CTR) and conversion-rate (CVR) prediction for web advertising. In these applications, we often need to select a (small) subset of valuable features, and use them to build models for prediction. Doing so has several important reasons. First, too many features can result in complicated models; to avoid overfitting, we have to collect a huge amount of data and occupy many computing resources for training. This is time consuming and computationally expensive, especially when many online systems are required to update the models in a short time window, say, a few minutes or hours. Second, with all the features, the learned models can be ponderous and inconvenient for real-time usage. For instance, to conduct real-time bidding, a typical online advertising system is required to perform each CTR prediction within 10-100 milliseconds; hence, the CTR model must be parsimonious. Third, in feature engineering, we need to identify/evaluate relevant features, based on which we can explore or design new valuable features, to iteratively improve the models and systems.

To perform feature selection, popular approaches use sparse learning with L_1 regularization [3], where each feature weight is assigned an L_1 penalty and the weights for the irrelevant features are shrunk to 0 during training. L_1-type methods not only possess theoretical guarantees such as estimation optimality and oracle properties [3], [4], but also enjoy computational convenience — the training of many models with L_1 regularizations (*e.g.,* linear/logistic regression) is a convex optimization problem.

In practical applications, however, both the sample size and feature number can be very large, making traditional batch optimization algorithms infeasible. To address this issue, McMahan [1] has recently developed Follow-The-Regularized-Leader (FTRL) proximal, an online algorithm with mixed L_1 and L_2 regularizations, namely, the elastic net [4]. With a single machine, FTRL-proximal can quickly process enormous samples, prune massive useless features, and learn the prediction model simultaneously. FTRL-proximal has become the state-of-the-art and been applied to CTR prediction for Google's online advertising system [5].

Despite the great success, FTRL-proximal, and many L_1-type methods suffer several disadvantages. First, the L_1 regularization essentially implements uniform shrinkage. That is, no matter whether a feature is relevant or not, the corresponding feature weight has to endure a shrinkage effect of the L_1 penalty. This is not ideal, because the selected features' weights should be fully estimated from data, rather than also be shrunk by a strong penalty. Although some L_1 methods, *e.g.,* adaptive lasso [6], adopt different regularization strengths over the feature weights, those strengths are determined at the beginning and rarely close to zero. In the estimation, all the features weights are still jointly shrunk. Second, L_1-type methods typically provide point estimates — there lacks confidence information, such as selection uncertainty and weight variances. These information are important for feature evaluation, subsequent decision making, ranking and system debugging. For example, to obtain the CTR prediction confidence for optimal ads choice and displaying, McMahan et al. [5] have to invent a heuristic "uncertainty score" based on the learning rates of FTRL-proximal.

To address these issues, we propose OLSS, a Bayesian online sparse learning algorithm based on the spike-and-slab prior. The spike-and-slab prior fulfills appealing *selective* shrinkage [7]. That is, the selected features are separated from the unselected ones by binary indicator variables; while the weights of the unselected features are strongly shrunk toward zero via the spike prior, the weights of the selected features are just mildly regularized via the slab prior (equivalent to L_2 regularizations in MAP estimation) and hence can be well estimated from data. Then in the training, we jointly learn the indicator variables and feature weights. Furthermore, as a principled Bayesian approach, our algorithm seamlessly quantifies the uncertainty, including posterior inclusion probabilities, posterior means and variances of the feature weights. Hence, to obtain confidence levels, we do not need any heuristics or post processing, which can vary for different applications.

Although Bayesian spike-and-slab prior is ideal for sparse learning, it has a severe computational bottleneck in model estimation, which hinders it from large-scale applications. To overcome this problem, our algorithm, OLSS, exploits the recent stochastic Expectation Propagation (SEP) framework [8] and develops an efficient, online approximate learning approach. OLSS not only achieves the same scalability as FTRL-proximal, i.e., to both large sample size and high dimensional features, but also fulfills the more favorable selective shrinkage and uncertainty quantification. Specifically, we first design a flexible, per-feature factorized approximation form to efficiently handle high dimensional features (*e.g.,* the sparse categorical features), and to well preserve the selective shrinkage effect of the original prior. Second, we enhance the standard SEP, by estimating multiple approximate average-likelihoods. Each average-likelihood naturally summarizes the information from one type of samples. In this way, data distributions in different regions can be more accurately captured, at a negligible extra cost. In addition, it allows us to assign distinct sample weights for different types, say, positive and negative samples. Finally, OLSS sequentially processes data samples and updates the posteriors of the feature weights and selection indicators in real time; we can read out the posteriors and make predictions at any moment.

We examined OLSS in two applications, CTR prediction for online advertising and text classification. On data with millions of samples and features, OLSS can greatly reduce the feature number, say, to a few hundreds. On average, OLSS obtains a superior predictive performance to the state-of-the-art alternatives in industry, including Vowpal Wabbit [2] and FTRL-proximal [1]. In particular, we observed more evident improvement over FTRL-proximal when less features are selected — i.e., when stronger L_1 penalties are employed in FTRL-proximal. For example, when around $1,000$ features are selected, OLSS on average reduces the AUC error of FTRL-proximal by 9% and 41% for the two applications. This demonstrates the advantage of the selective shrinkage by OLSS. Finally, we analyzed the uncertainty information produced by OLSS for online advertising. We found interesting results, part of which agree with commonly used feature engineering tricks.

II. Bayesian Spike-and-Slab Models

First, let us introduce Bayesian sparse learning models with spike-and-slab priors. In this paper, we focus on the binary linear classification task, as in FTRL-proximal; it is straightforward to extend our approach to other problems, such as regression. Suppose we have a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where each x_n is a d-dimensional input feature vector and y_n is the response: $y_n \in \{+1, -1\}$. We aim to select a subset of relevant features from the input $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$, and use them to predict the responses $\mathbf{t} = [t_1, \ldots, t_n]^\top$. To this end, we introduce a $d \times 1$ weight vector $\mathbf{w}$, where each entry w_j is the classification weight for feature j. Given each (x_n, y_n), we assume a Probit regression likelihood, which is widely used in Bayesian binary classification,

$$p(y_n|\mathbf{x}_n, \mathbf{w}) = \Phi(y_n \mathbf{w}^\top \mathbf{x}_n) \tag{1}$$

where $\Phi(\cdot)$ is the cumulative density function (CDF) of the standard Gaussian distribution, i.e., $\Phi(t) = \int_{-\infty}^{t} \mathcal{N}(x|0,1)\mathrm{d}x$. To enable feature selection, we assign a spike-and-slab prior distribution over each feature weight w_j. Specifically, we first introduce a binary selection indicator $s_j \in \{0, 1\}$, sampled from the prior distribution

$$p(s_j) = \mathrm{Bern}(s_j|\rho_0) = \rho_0^{s_j}(1-\rho_0)^{1-s_j}. \tag{2}$$

Given s_j, we then sample w_j from

$$p(w_j|s_j) = s_j \mathcal{N}(w_j|0, \tau_0) + (1 - s_j)\delta(w_j) \tag{3}$$

where $\delta(\cdot)$ is a Dirac-delta function. Here, the selection indicator s_j determines the type of prior distribution over w_j: if s_j is 1, meaning feature j is selected, the weight w_j is assigned a flat Gaussian prior with variance τ_0 (slab component); if otherwise s_j is 0, namely feature j is irrelevant, the weight w_j is assigned a spike prior concentrating on 0 (spike component). The joint probability of our model is given by

$$\begin{aligned} &p(\mathcal{D}, \mathbf{w}, \mathbf{s}|\rho_0, \tau_0) \\ &\quad= \prod_{j=1}^{d} \mathrm{Bern}(s_j|\rho_0)\big(s_j\mathcal{N}(w_j|0,\tau_0) + (1-s_j)\delta(w_j)\big) \\ &\quad\cdot \prod_{n=1}^{N} \Phi(y_n \mathbf{w}^\top \mathbf{x}_n). \end{aligned} \tag{4}$$

Note that, using spike-and-slab priors yields a selective shrinkage effect [7]: the binary selection indicators $\mathbf{s} = [s_1, \ldots, s_d]$ (see (2) and (3)) actually separate the selected features from the unselected ones. The classification weights for the unselected features are directly shrunk to 0 and pruned by the spike component, i.e., the Dirac-delta prior $\delta(\cdot)$. The weights for the selected features are only mildly regularized via the slab component, i.e., the flat Gaussian prior $\mathcal{N}(\cdot|\mathbf{0}, \tau_0)$ (corresponding to L_2 regularization in MAP framework); thereby, these weights can be well estimated from the observed data, not being hurt by any strong shrinkage effect.

As a comparison, let us recall the classical L_1 regularization approach: we impose an L_1 regularization, or penalty, over each feature weight, and estimate the classification weights $\mathbf{w}$ via minimizing a cost function similar to the following form:

$$C(\mathbf{w}, \mathcal{D}) = \sum\nolimits_{n=1}^{N} L(\mathbf{w}, \mathbf{x}_n, y_n) + \lambda \sum\nolimits_{j=1}^{d} |w_j|, \tag{5}$$

where $L(\mathbf{w}, \mathbf{x}_n, y_n)$ is the loss function, *e.g.,* $-\log\big(p(y_n|\mathbf{x}_n, \mathbf{w})\big)$ in (1), and λ is the regularization strength. Note that some variants, *e.g.,* adaptive lasso [6], may assign a different λ for each w_j. Anyway, the L_1 regularizer

$\lambda|\cdot|$ enforces a strong shrinkage, and encourages every feature weight w_j to be 0 — this is a uniform shrinkage effect. Although the weights for the irrelevant features are thereby shrunk to 0 and pruned, in the mean time the weights for the selected features are shrunk (to certain degrees) as well. This is actually harmful for model estimation, because the selected features' weights should be fully learned from data, rather than be strongly shrunk. Hence, the selective shrinkage of the spike-and-slab prior is more favorable, and our experimental results (Section V) have confirmed this point.

III. Online Learning Algorithm

Given the observed data $\mathcal{D}$, the learning of our model amounts to computing the posterior distributions of the binary selection variables $\mathbf{s}$ and the classification weights $\mathbf{w}$, i.e., $p(\mathbf{s}|\mathcal{D},\tau_0,\rho_0)$ and $p(\mathbf{w}|\mathcal{D},\tau_0,\rho_0)$. This is also referred to as posterior inference. The posterior distributions not only provide the point estimates of the feature weights (i.e., the posterior means), but also contain the valuable uncertainty information, such as the posterior inclusion probabilities and weight variances, which benefit feature analysis and can be further leveraged in feature engineering.

However, the posterior inference for Bayesian spike-and-slab models is tricky. According to Bayes' rule, the exact computation of $p(\mathbf{s}|\mathcal{D},\tau_0,\rho_0)$ and $p(\mathbf{w}|\mathcal{D},\tau_0,\rho_0)$ requires the marginal probability $p(\mathcal{D}|\rho_0,\tau_0) = \sum_{\mathbf{s}} \int p(\mathcal{D},\mathbf{w},\mathbf{s}|\rho_0,\tau_0)\mathrm{d}\mathbf{w}$. Since each s_j in $\mathbf{s}$ is binary, the summation is over 2^d terms, and hence is infeasible for large d, i.e., high dimensional problems. Although standard approximate inference techniques can avoid the headache summation, they are still far from practical for large-scale applications, where both the samples and features are massive, say, millions or even billions. The Markov-Chain Monte-Carlo sampling is known to converge very slowly for high dimensional problems. Standard variational Bayes [9] and Expectation Propagation [10], though fast, perform batch inference, and need to store all the data in memory; hence they are infeasible when the data volume exceeds the memory limit.

To enable Bayesian spike-and-slab models in real-world, large-scale applications, like FTRL-proximal for L_1-type methods, we developed OLSS, an online posterior inference algorithm, by exploiting the recent stochastic Expectation Propagation (SEP) framework [8].

A. Stochastic Expectation Propagation

First, let us introduce the SEP framework [8]. To this end, we start with the classical Expectation Propagation (EP) [10]. Consider a general probabilistic model parameterized by $\boldsymbol{\theta}$. Given the data $\mathcal{D} = \{\mathbf{z}_1,\ldots,\mathbf{z}_N\}$, the joint probability is

$$p(\boldsymbol{\theta},\mathcal{D}) = p_0(\boldsymbol{\theta})\prod\nolimits_n p(\mathbf{z}_n|\boldsymbol{\theta}),$$

where $p_0(\boldsymbol{\theta})$ is the prior distribution. To obtain the exact posterior $p(\boldsymbol{\theta}|\mathcal{D})$, we have to calculate the marginal distribution $p(\mathcal{D})$, which is usually intractable. To address this issue, EP approximates $p(\boldsymbol{\theta},\mathcal{D})$ with a distribution from the easy yet flexible exponential family [11], $q(\boldsymbol{\theta},\mathcal{D}) \propto \exp(\mathbf{t}(\boldsymbol{\theta})^\top \boldsymbol{\lambda}(\mathbf{D}))$ where $\mathbf{t}(\boldsymbol{\theta})$ are the sufficient statistics and $\boldsymbol{\lambda}(\mathbf{D})$ the natural parameters. Note that many commonly used distributions belong to the exponential family, including Gaussian, Bernoulli, Gamma, etc. The motivation is that after we use an exponential family distribution $q(\boldsymbol{\theta},\mathcal{D})$ to replace the original model, the calculation of posterior distribution, $q(\boldsymbol{\theta}|\mathcal{D})$, becomes trivial and analytical. Hence the task is to find a best approximation $q(\boldsymbol{\theta},\mathcal{D})$. To do so, EP uses a fixed point iteration method to minimize an objective, EP energy function [10], which measures the similarity between $p(\boldsymbol{\theta},\mathcal{D})$ and $q(\boldsymbol{\theta},\mathcal{D})$.

Specifically, EP first introduces an exponential-family term $f_n(\boldsymbol{\theta})$ to approximate each likelihood $p(\mathbf{z}_n|\boldsymbol{\theta})$, and $f_0(\boldsymbol{\theta})$ to the prior $p_0(\boldsymbol{\theta})$. Then we have $q(\boldsymbol{\theta},\mathcal{D}) \propto f_0(\boldsymbol{\theta})\prod_n f_n(\boldsymbol{\theta})$. Because the exponential family are close under multiplications/divisions, $q(\boldsymbol{\theta},\mathcal{D})$ is guaranteed to be in this family. To optimize $q(\boldsymbol{\theta},\mathcal{D})$, EP cyclically refines each approximation term f_i with four steps:

Step 1. calculating the calibrating distribution, $q^{\backslash i}(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta},\mathcal{D})/f_i(\boldsymbol{\theta})$,
Step 2. constructing a tilted distribution $t_i(\boldsymbol{\theta}) \propto q^{\backslash i}(\boldsymbol{\theta})p(\mathbf{z}_i|\boldsymbol{\theta})$,
Step 3. projecting t_i back into the exponential family, $q^*(\boldsymbol{\theta}) \propto \mathrm{proj}(t_i(\boldsymbol{\theta}))$, via moment matching,
Step 4. updating the term f_i: $f_i^{\mathrm{new}}(\boldsymbol{\theta}) \propto q^*(\boldsymbol{\theta})/q^{\backslash i}(\boldsymbol{\theta})$.

At convergence, EP reaches a fixed point that corresponds to a stationary point of the energy function [10]. EP can be considered as a generalized belief propagation algorithm [10].

EP often works well in practice. However, since it maintains an approximate likelihood term $f_n(\boldsymbol{\theta})$ for every sample n, it may fail when the samples are too many to be stored in memory. To make EP scalable to large data, SEP instead uses only one average-likelihood term, $f_a(\boldsymbol{\theta})$, to summarize all the data likelihoods. Specifically, SEP defines the approximate distribution as

$$q(\boldsymbol{\theta},\mathcal{D}) \propto f_0(\boldsymbol{\theta})f_a(\boldsymbol{\theta})^N. \tag{6}$$

By only maintaining $f_0(\boldsymbol{\theta})$ and $f_a(\boldsymbol{\theta})$, SEP greatly reduces the memory usage. SEP further uses an online mechanism to update $f_a(\boldsymbol{\theta})$. Specifically, given sample n, we calculate the calibrating distribution by $q^{\backslash n}(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta},\mathcal{D})/f_a(\boldsymbol{\theta})$ (**Step** 1), and follow the remaining steps as in the original EP to obtain an approximate likelihood, $f_n(\boldsymbol{\theta})$ (**Step** 2-4); we then integrate $f_n(\boldsymbol{\theta})$ into the update of $f_a(\boldsymbol{\theta})$, by taking the (geometric) average over the approximate data likelihoods, where the approximation for sample n is $f_n(\boldsymbol{\theta})$ and for all the other samples $f_a(\boldsymbol{\theta})$,

$$f_a(\boldsymbol{\theta})^{\mathrm{new}} = \left(f_n(\boldsymbol{\theta})f_a(\boldsymbol{\theta})^{N-1}\right)^{\frac{1}{N}}. \tag{7}$$

Since the approximation terms are in the exponential family, we can express the update in terms of their natural parameters:

$$\boldsymbol{\lambda}_a^{\mathrm{new}} = (1-\frac{1}{N})\boldsymbol{\lambda}_a + \frac{1}{N}\boldsymbol{\lambda}_n \tag{8}$$

where $\boldsymbol{\lambda}_a$ and $\boldsymbol{\lambda}_n$ belong to $f_a(\boldsymbol{\theta})$ and $f_n(\boldsymbol{\theta})$, respectively. We can see that the natural parameters of f_a are updated by a weighted combination of the old values and the new version from the current sample. This is a typical stochastic update, as in stochastic gradient descent. Furthermore, we can use a mini-batch of samples $\{\mathbf{z}_{n_1}, \ldots, \mathbf{z}_{n_M}\}$ to achieve a larger move,

$$\boldsymbol{\lambda}_a^{\text{new}} = \frac{1}{N}\sum\nolimits_{j=1}^{M} \boldsymbol{\lambda}_{n_j} + (1 - \frac{M}{N})\boldsymbol{\lambda}_a. \tag{9}$$

Note that while using stochastic updates, SEP essentially performs fixed point iterations as well, and converges to a stationary point of an EP energy function variant that measures the similarity between the approximate and the true distributions; under certain conditions, SEP converges to the same fixed points as EP in the large data limit [8].

B. Online Learning for Bayesian Spike-and-slab Models

Now, we present OLSS, our online learning algorithm for the Bayesian spike-and-slab model defined in (4), based on the SEP framework.

1) Per-Feature Factorized Approximation: In practical applications, although each feature vector $\mathbf{x}_n$ can be extremely high dimensional, they are usually very sparse, i.e., most of the elements are zero. This is mainly due to the sparse categorical features, such as the product brand and the word dictionary. They often posses a large cardinality and we have to use a sparse, long feature vector representation. For instance, suppose there are $1K$ brands; then we need to incorporate $1K$ binary features — each feature represents whether the sample pertains to one particular brand or not. Since each sample may associate with just one brand, the $1K \times 1$ feature vector will only contain one nonzero element. Therefore, to avoid unnecessary computation regarding zero-valued features, we first rewrite the data likelihood as $p(y_n|\mathbf{x}_n, \mathbf{w}) = \Phi(y_n \mathbf{w}_{\mathrm{I_n}}^\top \tilde{\mathbf{x}}_n)$ where $\mathrm{I_n}$ are the indices of nonzero features in $\mathbf{x}_n$, and $\tilde{\mathbf{x}}_n$ the nonzero sub feature vector. Now the joint probability has a slightly changed, but totally equivalent form,

$$\begin{aligned} &p(\mathcal{D}, \mathbf{w}, \mathbf{s}|\rho_0, \tau_0) \\ &= \prod\nolimits_{j=1}^{d} \mathrm{Bern}(s_j|\rho_0)\big(s_j \mathcal{N}(w_j|0, \tau_0) + (1 - s_j)\delta(w_j)\big) \\ &\cdot \prod\nolimits_{n=1}^{N} \Phi(y_n \mathbf{w}_{\mathrm{I_n}}^\top \tilde{\mathbf{x}}_n). \end{aligned} \tag{10}$$

To enable efficient inference, we approximate the joint probability in (10) with an exponential family distribution $q(\mathbf{s}, \mathbf{w})$. To do so, we first approximate the binary summation term inside the prior distribution by the production of two exponential family terms:

$$\begin{aligned} &s_j \mathcal{N}(w_j|0, \tau_0) + (1 - s_j)\delta(w_j) \\ &\approx t_{1j} \mathrm{Bern}\big(s_j|\sigma(\rho_j)\big)\mathcal{N}(w_j|\mu_{1j}, v_{1j}) \end{aligned} \tag{11}$$

where $\sigma(x) = 1/(1 + e^{-x})$, t_{1j} is the scale factor, and $\{\rho_j, \mu_{1j}, v_{1j}\}$ are the parameters of the approximation terms that need to be estimated in our algorithm. Note that we do not need to compute the scale factor t_{1j}, because when using $q(\mathbf{s}, \mathbf{w})$ to calculate the approximate posteriors, all the scale factors are cancelled. We incorporate the logistic function $\sigma(\cdot)$ into our parameterization to improve the numerical stability. The benefit of the approximation in (11) is twofold: first, it decouples the selection variable s_j and the feature weight w_j in the joint probability, and hence enables an efficient online approximate inference, as we will present; second, it is flexible to represent the selective shrinkage effect of the original prior. For example, positive ρ_j (i.e., $\sigma(\rho_j) > 0.5$) and relatively big v_{1j} can represent that the feature j is more likely to be selected and the weight is sampled from a flat Gaussian; negative ρ_j (i.e., $\sigma(\rho_j) < 0.5$) and $\{\mu_{1j}, v_{1j}\}$ close to 0 represent that feature j is less likely to be selected and the weight is strongly shrunk toward 0.

Next, to approximate the data likelihood, we introduce two types of average-likelihood terms, $f_a^+(\mathbf{w}_{\mathrm{I}})$ and $f_a^-(\mathbf{w}_{\mathrm{I}})$ where I are the indices of the nonzero features, for the positive and negative samples respectively: $f_a^+(\mathbf{w}_{\mathrm{I}}) = \prod_{j \in \mathrm{I}} \mathcal{N}(w_j|\mu_{2j}^+, v_{2j}^+)$, and $f_a^-(\mathbf{w}_{\mathrm{I}}) = \prod_{j \in \mathrm{I}} \mathcal{N}(w_j|\mu_{2j}^-, v_{2j}^-)$. We then use

$$\Phi(y_n \mathbf{w}_{\mathrm{I_n}}^\top \tilde{\mathbf{x}}_n) \approx t_n f_a^+(\mathbf{w}_{\mathrm{I_n}})^{\mathbb{1}(y_n=1)} f_a^-(\mathbf{w}_{\mathrm{I_n}})^{\mathbb{1}(y_n=-1)} \tag{12}$$

where t_n is the scale factor, $\mathbb{1}(\cdot)$ is the indicator function, and $\{\mu_{2j}^+, v_{2j}^+, \mu_{2j}^-, v_{2j}^-\}$ are the parameters of the Gaussian approximation terms. Again we do not need to calculate the scale factor t_n .

Combining (11) and (12), we obtain the following approximate distribution for the model (10),

$$\begin{aligned} &q(\mathbf{s}, \mathbf{w}) \propto \prod\nolimits_{j=1}^{d} \mathrm{Bern}(s_j|\rho_0)\mathrm{Bern}\big(s_j|\sigma(\rho_j)\big)\mathcal{N}(w_j|\mu_{1j}, v_{1j}) \\ &\cdot \prod\nolimits_{n=1}^{N} f_a^+(\mathbf{w}_{I_n})^{\mathbb{1}(y_n=1)} f_a^-(\mathbf{w}_{I_n})^{\mathbb{1}(y_n=-1)}. \end{aligned} \tag{13}$$

By arranging the terms, we further obtain

$$\begin{aligned} &q(\mathbf{s}, \mathbf{w}) \propto \prod\nolimits_{j=1}^{d} \mathrm{Bern}(s_j|\rho_0)\mathrm{Bern}\big(s_j|\sigma(\rho_j)\big)\mathcal{N}(w_j|\mu_{1j}, v_{1j}) \\ &\cdot \mathcal{N}(w_j|\mu_{2j}^+, v_{2j}^+)^{n_j^+} \mathcal{N}(w_j|\mu_{2j}^-, v_{2j}^-)^{n_j^-} \end{aligned} \tag{14}$$

where n_j^+ and n_j^- are the counts of feature j being nonzero in positive and negative samples, respectively. Now it becomes clear that $q(\mathbf{s}, \mathbf{w})$ is factorized over features. Given all the parameters, $\{\rho_j, \mu_{1j}, v_{1j}, \mu_{2j}^+, v_{2j}^+, \mu_{2j}^-, v_{2j}^-\}_j$, we can immediately obtain the (approximate) posteriors for each feature j, by marginalizing $q(\mathbf{s}, \mathbf{w})$ in (14), which is trivial:

$$p(w_j|\mathcal{D}, \rho_0, \tau_0) \approx q(w_j) = \mathcal{N}(w_j|\mu_j^D, v_j^D), \tag{15}$$

$$p(s_j|\mathcal{D}, \rho_0, \tau_0) \approx q(s_j) = \mathrm{Bern}(s_j|\rho_j^D) \tag{16}$$

where $\rho_j^D = \sigma\big(\rho_j + \sigma^{-1}(\rho_0)\big)$, $v_j^D = 1/(1/v_{1j} + n^+/v_{2j}^+ + n^-/v_{2j}^-)$ and $\mu_j^D = v_j^D(\mu_{1j}/v_{1j} + n^+\mu_{2j}^+/v_{2j}^+ + n^-\mu_{2j}^-/v_{2j}^-)$. Note that unlike the standard SEP using only one average-likelihood for all the samples, we consider different sample types: for each type, we use a different average-likelihood. There are two advantages: first, the data summarization can be more accurate; in general we can cluster the data first, then for each cluster we use an average-likelihood, to better capture the shape of the entire data distribution. Second, we can vary the weights for different classes of samples, via the settings of $\{n_j^+\}_j$ and $\{n_j^-\}_j$. This can be useful for applications with imbalanced samples. Take CTR prediction as an example. The number of clicked impressions (i.e., positive samples) is far less than the non-clicks (negative samples). To save computation, we can collect all the positive samples but subsample a comparable number of negative samples; then for training, we intentionally set large $\{n_j^-\}_j$ to maintain the same positive/negative ratio in the original data. This is equivalent to duplicate the negative samples to simulate the original sample bias, but we do not need explicit duplications.

2) Stochastic Updates: Now let us look at how to estimate the approximation terms' parameters, $\{\rho_j, \mu_{1j}, v_{1j}, \mu_{2j}^+, v_{2j}^+, \mu_{2j}^-, v_{2j}^-\}_j$ to make $q(\mathbf{s}, \mathbf{w})$ as close as possible to $p(\mathcal{D}, \mathbf{w}, \mathbf{s}|\rho_0, \tau_0)$. To handle big data, we exploit the SEP framework to conduct online updates. We sequentially process training samples, each time a mini-batch. In the mini-batch, we simultaneously update the approximate likelihood for each sample n — $f_n^+(\mathbf{w}_{\mathrm{I_n}})$ if sample n is positive and $f_n^-(\mathbf{w}_{\mathrm{I_n}})$ otherwise — following the steps in Section III-A. From f_n^+ (or f_n^-), we obtain the local update of the Gaussian approximation term for each (nonzero) feature j in sample n, i.e., $\{\mathcal{N}(w_j|\mu_{2j}^+, v_{2j}^+) \text{ or } \mathcal{N}(w_j|\mu_{2j}^-, v_{2j}^-)|j \in \mathrm{I_n}\}$. We collect all the local updates in the mini-batch, then aggregate them to perform the global stochastic updates, as in (9).

Specifically, denote $\mathcal{B}$ the current mini-batch, and $\mathcal{N}(w_j|\mu_{2j}^{*,n}, v_{2j}^{*,n})$ the locally updated Gaussian term for each nonzero feature j in sample n. Suppose sample n is positive. The parameters are calculated by

$$\mu_{2j}^{*,n} = \mu_{2j}^{\backslash n} + v_{2j}^{\backslash n} \frac{\partial \log z_n}{\partial \mu_{2j}^{\backslash n}}, \tag{17}$$

$$v_{2j}^{*,n} = v_{2j}^{\backslash n} - (v_{2j}^{\backslash n})^2 \big((\frac{\partial \log z_n}{\partial \mu_{2j}^{\backslash n}})^2 - 2\frac{\partial \log z_n}{\partial v_{2j}^{\backslash n}} \big) \tag{18}$$

where

$$v_{2j}^{\backslash n} = \big((v_j^D)^{-1} - (v_{2j}^+)^{-1} \big)^{-1} \quad , \quad \mu_{2j}^{\backslash n} = v_{2j}^{\backslash n} \big(\frac{\mu_j^D}{v_j^D} - \frac{\mu_{2j}^+}{v_{2j}^+} \big),$$

$$z_n = \Phi\big(\frac{y_n \sum_{i \in \mathrm{I_n}} \mu_{2i}^{\backslash n} \tilde{x}_{ni}}{\sqrt{1 + \sum_{i \in \mathrm{I_n}} v_{2i}^{\backslash n} \tilde{x}_{ni}^2}} \big).$$

Note that μ_j^D, v_j^D are approximate posterior mean and variance of feature j, calculated from (15) based on the current $q(\mathbf{s}, \mathbf{w})$.

After collecting the local updates for all the positive samples in $\mathcal{B}$, we then perform the global updates of the Gaussian approximation terms $\{\mathcal{N}(w_j|\mu_{2j}^+, v_{2j}^+)\}_j$ according to (9), as follows,

$$v_{2j}^{+*} = \frac{n_j^+ - n_{j,B}^+}{n_j^+} \cdot \frac{1}{v_{2j}^+} + \sum_{n \in \mathcal{B}, y_n = 1, j \in \mathrm{I_n}} \frac{1}{v_{2j}^{*,n}}, \tag{19}$$

$$\mu_{2j}^{+*} = v_{2j}^{+*} \big(\frac{n_j^+ - n_{j,B}^+}{n_j^+} \cdot \frac{\mu_{2j}^+}{v_{2j}^+} + \sum_{n \in \mathcal{B}, y_n = 1, j \in \mathrm{I_n}} \frac{\mu_{2j}^{*,n}}{v_{2j}^{*,n}} \big), \tag{20}$$

where $\{\mu_{2j}^{+*}, v_{2j}^{+*}\}$ are the updated parameters, $n_{j,B}^+$ is the nonzero count of feature j in $\mathcal{B}$. Similarly, we perform the global updates of the Gaussian terms for negative samples, i.e., $\{\mathcal{N}(w_j|\mu_{2j}^-, v_{2j}^-)\}_j$. The updating equations are identical to the above except that we switch $+$ for $-$ in the superscripts. After every a few mini-batches, we update the approximate prior terms $\{\mathrm{Bern}(s_j|\sigma(\rho_j)), \mathcal{N}(w_j|\mu_{1j}, v_{1j})\}_j$ (see (11)). Since there is only one prior distribution term, we can directly use the standard moment matching as in EP:

Algorithm 1 OLSS($\mathcal{D}, \rho_0, \tau_0, M, T, \{n_j^+, n_j^-\}_j$)

```
1: Random shuffle samples in $\mathcal{D}$.
2: Initialize for each feature $j$: $\rho_j = 0, \mu_{1j} = \mu_{2j}^+ = \mu_{2j}^- = 0, v_{1j} = v_{2j}^+ = v_{2j}^- = 10^6$.
3: repeat
4:   Collect a mini-batch of size $M$.
5:   In the mini-batch, calculate the approximate likelihood for each sample and obtain the local Gaussian approximation term for every present feature ((17)(18)).
6:   Aggregate all the local updates in the min-batch and perform the global updates of $\{\mu_{2j}^+, v_{2j}^+, \mu_{2j}^-, v_{2j}^-\}$ ((19)(20)).
7:   if $T$ mini-batches are processed then
8:     Update $\{\rho_j, \mu_{1j}, v_{1j}\}_j$ using (21)(22)(23).
9:   end if
10: until all samples in $\mathcal{D}$ are processed (or certain stopping criterions are satisfied).
11: return $\{q(w_j), q(s_j)\}_j$ in (15)(16).
```

$$\rho_j^* = \log\big(\frac{\mathcal{N}(\mu_j^{\backslash 1j}|0, \tau_0 + v_j^{\backslash 1j})}{\mathcal{N}(\mu_j^{\backslash 1j}|0, v_j^{\backslash 1j})} \big), \tag{21}$$

$$\mu_{1j}^* = \sigma(\tilde{\rho}_j) \cdot \tilde{\mu}_j, \tag{22}$$

$$v_{1j}^* = \sigma(\tilde{\rho}_j)\big(\tilde{v}_j + (1 - \sigma(\tilde{\rho}_j))\tilde{\mu}_j^2 \big) \tag{23}$$

where $\{\rho_j^*, \mu_{1j}^*, v_{1j}^*\}$ are the updated parameters, $\tilde{\rho}_j = \rho_j^* + \sigma^{-1}(\rho_0)$ and

$$v_j^{\backslash 1j} = \big((v_j^D)^{-1} - v_{1j}^{-1} \big)^{-1}, \quad \mu_j^{\backslash 1j} = v_j^{\backslash 1j} \big(\frac{\mu_j^D}{v_j^D} - \frac{\mu_{1j}}{v_{1j}} \big),$$

$$\tilde{v}_j = \big((v_j^{\backslash 1j})^{-1} + \tau_0^{-1} \big)^{-1}, \quad \tilde{\mu}_j = \tilde{v}_j \frac{\mu_j^{\backslash 1j}}{v_j^{\backslash 1j}}.$$

We alternatively update the Gaussian approximation terms (19)(20) and the prior approximation terms (21)(22)(23) until all the samples have been processed (or some early stopping conditions are satisfied). At convergence, the algorithm arrives at a SEP energy function fixed point that minimizes the distance between approximate and the true distributions [8]. Finally, we select all the features that have the posterior inclusion probabilities bigger than $\frac{1}{2}$, i.e., $\{j|q(s_j = 1) > \frac{1}{2}\}$. Then we use the posterior means of the selected feature weights for prediction. The algorithm is summarized in Algorithm 1. Since at any time OLSS only stores and processes a mini-batch, the time complexity is $\mathcal{O}(Nd)$, which is linear to the number of samples and features, and the same as that of FTRL-proximal; the space complexity of OLSS is $\mathcal{O}(Md)$ (M is the mini-batch size), and is identical to that of FTRL-proximal when $M = 1$.

IV. Related Work

While proposed long time ago [12], [13], the advantages of the spike-and-slab prior are realized until recently. [7] demonstrated that the *selective* shrinkage of the spike-and-slab prior is crucial to effective feature selection in terms of risk misclassification. [14] showed improved performance of Bayesian spike and slab methods over the L_1-type methods in unsupervised settings.

A key bottleneck that limits the applications of the spike-and-slab models is the computational challenge for the posterior inference (i.e., model estimation). To tackle this problem, quite a few approximate inference algorithms have been developed. For example, [15] developed a majorization and minimization approach to obtain the MAP estimation of the spike-and-slab linear regression. [16] proposed a fast Laplace approximation approach using Nyström method. [17], [18] used the variational inference framework to approximate the posteriors. Note that the variational inference estimates the posteriors by maximizing a variational model evidence lower bound, and is different from EP — a message passing algorithm for belief propagation, which essentially conducts fixed point iterations. Despite the success of these methods, they are still insufficient for real-world large-scale applications, where both samples and features are massive. All the existing methods use batch inference procedures. That is, they load the whole data into the memory; to decide one gradient descent step or to finish one iteration, they have to go through all the samples — this not only takes huge memory, but also is computationally inefficient. To our knowledge, there are no trivial modifications to address these issues. This motives us to develop OLSS, an online spike-and-slab approximate inference algorithm.

To obtain the uncertainty information for L_1 type methods, one can employ equivalent Laplace priors over the feature weights in a Bayesian framework. However, doing so cannot prevent the uniform shrinkage effect in model estimation. One might also consider using bootstrapping to obtain confidence intervals. However, in large-scale applications, it might be too expensive to repeatedly estimate parameters from many, many large bootstrapping datasets.

In addition to the Laplace prior, other Bayesian sparse priors include ARD/t-distributions [19], [20], normal-inverse-Gaussian [21], generalized double Pareto [22], horseshoe [23], [24], *etc.* In spite of their success and elegant properties (such as heavy tails and posterior consistency), the scalable posterior inference on large-scale data remains an open and promising research direction. The theoretical justifications of the spike-and-slab prior are discussed in detail in [7], [25], [26]; the empirical comparison with other sparse priors, on small real or synthetic data, can be found in [14], [16], [27], [28].

V. Experiment

Since our algorithm is developed to enable spike-and-slab models in large-scale sparse learning tasks, we conducted evaluations in two real-world large-scale applications, click-through-rate (CTR) prediction for online advertising and text classification. Note that in small-scale real datasets or simulations, there have been many empirical validations of the spike and slab models as compared with other sparse learning approaches, including representative L_1 methods. Interested readers are referred to [14]–[16], [28].

A. Click-Through-Rate Prediction

First, we performed three groups of evaluations based on the click logs generated by two major online advertisement platforms of Yahoo!, Gemini (https://gemini.yahoo.com/advertiser/home) and BrightRoll. Note that Yahoo! was acquired by Verizon and BrightRoll has been integrated into the Oath platform (https://www.oath.com/advertising/platforms/). Gemini is designed for showing ads in Yahoo! Search Engine results; Bright-Roll is for displaying ads in large web site portals, like Yahoo! News and Sports.

GROUP 1. In the first group of evaluation, we extracted four days' click logs of Gemini, from 05/01/2015 to 05/04/2015. For training, we collected all the click impressions and subsampled a comparable number of non-click impressions in the log of 05/01/2015. Then we tested on the logs of the remaining three days. We used all the click and non-click impressions for testing. Note that training CTR prediction models with comparable clicks and non-clicks is common in online advertising systems [29]. The number of features is $1,074,917$. The sizes of training and testing datasets are $9.7M$, $546.8M$, $553.6M$, $878.7M$.

GROUP 2. In the second group of evaluation, we collected the training data from a 7 days' click logs of BrightRoll, between 07/15/2016 and 07/21/2016. We tested on the logs in 07/22/2016, 07/23/2016 and 07/24/2016. The number of features are $204,327$. For training, we collected all the click impressions and subsampled a comparable number of non-clicks; for testing data, we used all the click and non-click impressions. The sizes of training and testing data are $1.8M$, $133.7M$, $116.0M$ and $110.2M$.

GROUP 3. Real-world click data are extremely imbalanced: the click actions are very rare. For example, in Gemini logs, the clicks only take 0.6% of the total impressions. In the first and second groups of evaluation, we used downsampling techniques to obtain balanced training datasets so as to avoid the training being dominated by large portion of nonclicks (i.e., negative samples), and to reduce the computational cost. This trick has been proven very useful and adopted by Yahoo! Ad science and production team. However, to ensure that OLSS is also robust to imbalanced cases, we performed a third evaluation using a set of highly imbalanced impressions for training. Specifically, from Gemini we randomly sampled a subset of click log in 05/01/2015 as the training dataset, which consists of $798,152$ samples; there are only $5,004$ positive samples, i.e., clicks. We then randomly sampled another subset of click log in 05/02/2015 for testing, which consists of $547,043$ samples with $3,688$ clicks. The number of features are $617,258$.

Methods. We compared with two state-of-the-art methods widely used in industry: online logistic regression in Vowpal Wabbit (VW) [2], and FTRL-proximal (FTRLp) [1] which performs sparse online logistic regression based on the L_1/L_2

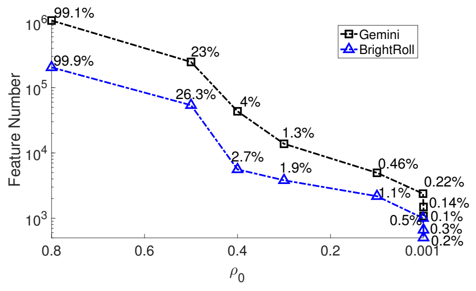

Fig. 1. The sparsity levels achieved by OLSS: number of selected features *v.s.* setting of ρ_0. Numbers on data points show the feature selection ratio.

regularization, i.e., elastic net. Note that the training of logistic regression with elastic net is known to be convex, and has a unique, global solution. Hence different algorithms will produce the same results. We also compared OLSS with other algorithms on small-scale datasets and obtained similar comparison results, to what we will present as follows.

Parameter settings. While VW supports the L_1/L_2 regularization as well, we found its implementation is problematic, and cannot provide sparse estimations as in FTRLp. Hence we turned off the L_1/L_2 options. We adopted the default parameters, which turns out to performs best. FTRLp has four parameters, α, β, λ_1 and λ_2, where $\{\alpha, \beta\}$ control the per-coordinate learning rate, and $\{\lambda_1, \lambda_2\}$ are the strengths for L_1 and L_2 regularizations. The performance of FTRLp is very sensitive to the setting of the four parameters. Therefore, we conducted the following parameter selection procedure on the validation datasets. We first fixed $\lambda_1 = \lambda_2 = 1.0$ and then fine tuned α, β from $\{0.005, 0.0075, 0.01, 0.025, 0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1\} \times \{0.005, 0.0075, 0.01, 0.025, 0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1\}$ to find the best parameters α^*, β^*. Next we fixed $\lambda_1 = 1.0$, $\alpha = \alpha^*$, $\beta = \beta^*$ and tuned λ_2 from $\{0.1, 0.5, 1, 2, 3, 4, 5\}$; we then obtained the best parameter λ_2^*. Finally, we fixed $\alpha = \alpha^*$, $\beta = \beta^*$, $\lambda_2 = \lambda_2^*$, varied the L_1 regularization strength λ_1 to change the sparsity level and examined the predictive accuracy accordingly on the test datasets. For our approach OLSS, we fixed the mini-batch size to 100 and set T to 1 (see Algorithm 1). OLSS has two hyper-parameters: τ_0 and ρ_0 where τ_0 is the variance of the slab component and ρ_0 is the prior belief of the feature selection ratio. We fixed $\rho_0 = 0.5$ and used the same validation dataset to tune τ_0 from $\{1, 3, 5, 10, 50, 100, 1\text{K}, 5\text{K}\}$. Note that for OLSS we only need to determine two hyper-parameters while for FTRLp we need to tune four; therefore OLSS is more convenient for practical usage. We then varied ρ_0 to adjust the sparsity level, under which we examined the predictive accuracy on each test dataset. While we implemented OLSS with Python, both VW and FTRLp are integrated in the Vowpal Wabbit binary executable, which were implemented by C++ and highly optimized. Note that Vowpal Wabbit package includes a collection of commonly used machine learning algorithms. For brevity, we simply refer its online logistic regression algorithm to as VW. We ran all the methods on a single Linux server with 32GB memory and Intel Xeon E5-2690 processors.

In the first group of evaluation, we randomly sampled $500K$ impressions from Gemini 05/02/2015 log as the validation dataset; in the second one, we randomly sampled $100K$ impressions from BrightRoll 07/23/2016 log as the validation dataset. For the third group of evaluation, we randomly split the training data, where $118K$ samples were used for validation; we used the remaining $680K$ samples for training and selected the best parameters according to their performance in the validation dataset. Given the best parameters, we then ran FTRLp and OLSS on the whole training datasets to obtain the models for testing.

First, we examined how much sparsity OLSS can yield when varying ρ_0. In Fig. 1, we show the number of features selected under different ρ_0, in the first and second groups of evaluation. We can see that bigger ρ_0 encourages a larger number of features to be selected; when we decreased ρ_0, OLSS quickly pruned massive features. Finally, the number of features can be reduced to around 0.1% of the entire feature set.

Next, we tested the predictive accuracy of OLSS and FTRLp under different sparsity levels. We report the Area Under ROC Curve (AUC) for all the test datasets. This is the most commonly used metric in the offline evaluation for CTR prediction. The results are shown in Fig. 2, 3 and 4e. In general, the prediction accuracy of OLSS and FTRLp gradually decreased when selecting less and less features, except that on the 05/04 test dataset in the first evaluation group (Fig. 2c), OLSS and FTRLp obtained improvement when discarding features; but later the prediction accuracy decreased after even more features were dropped.

As we can see, in most cases, OLSS outperforms FTRLp at all sparsity levels (except in a few cases of Fig. 2c, OLSS is slightly worse). The improvement is particularly evident when smaller numbers of features were selected — i.e., when FTRLp employed stronger L_1 regularization strengths. Note that, due to L_1 regularization's uniform shrinkage, the weights of the selected features have to bear strong penalties (as the unselected features' weights do); hence the model estimation of FTRLp can be suboptimal and the predictive performance can be hurt, especially when we employ stronger regularization strengths to obtain more concise models. The improved prediction accuracy of OLSS therefore demonstrates the advantages of the selective shrinkage from the spike-and-slab prior.

Compared with VW, OLSS kept a superior predictive performance until the feature number became too small. However, the accuracy drop of OLSS is much less and gentler than FTRLp. Furthermore, it is interesting to note that while VW obtains excellent predictive performance by using all the features, the advantage diminishes when it is tested on the impressions longer afterwards. From Fig. 2, we can see that VW's prediction accuracy is close to OLSS preserving all the features on 05/02 test impressions (Fig. 2a), but inferior on 05/03(2b) and more inferior on 05/04 (2c); similarly, in Fig. 3, the gap between VW and OLSS using all the features is farther on 07/24 (Fig. 3c) than on 07/21 (3a). Therefore, compared with VW, OLSS's performance is less sensitive to the time stamps of test impressions and hence more robust.

The average running time of OLSS are 122.7, 20.5 and 10.8 minutes on the three training datasets from GROUP 1 to 3. VW and FTRLp used much less time (less than 10 minutes), as expected, because they are implemented with highly optimized C++ source code and have been complied to binary executables. Our Python implementation is straightforward, and there are much room for engineering improvement, such as feature hashing, numerical acceleration, multi-threading, and C/C++ reimplementation.

B. Text Classification

In addition to CTR prediction, we examined OLSS in another typical application, text classification. We used two public datasets, News20 (http://qwone.com/ jason/20Newsgroups/) and RCV1 [30]. News20 is collection of $19,996$ news documents originally categorized into 20 groups. We used a two-class variant [31], where the sizes of the positive and negative samples are $9,999$ and $9,997$, respectively. We randomly chose $10,000$ samples for training and used the remaining ones for testing; the number of features are $1,103,456$, which is much larger than the number of samples. RCV1 is an archive of $804,414$ Reuters newswire stories which were manually categorized. We downloaded a preprocessed, binary version of the data from http://hunch.net/∼vw/rcv1.tar.gz. We randomly chose $23,149$ stories for training and the remaining $781,265$ stories for testing. The number of features are $43,001$.

As in the CTR prediction task, we examined the predictive performance of OLSS and FTRLp at various sparsity levels. In addition to AUC, we evaluated all the methods in terms of F1 score, an important measurement in text classification [30]. We employed the same parameter selection procedure as in Section V-A; to construct the validation datasets, we randomly split the training data and used $2,000$ samples for News20 and $3,149$ samples for RCV1.

As shown in Fig. 4a-d, OLSS outperforms FTRLp in terms of both AUC and F1 score, especially when selecting less features — this is consistent with the comparison results in CTR prediction. We can also observe the trade-off between the predictive accuracy and the sparsity degree: starting with large numbers of features, both OLSS and FTRLp obtained better or the same predictive accuracy of VW; when more and more features were pruned, the performance degraded. However, FTRLp's performance dropped more steeply.

C. Feature Analysis

TABLE I
CTR FEATURE SET IN BRIGHTROLL DATASET

Type	Examples
User	age, gender, local_hour, local_day,...
AD	line_id, publisher_id, ad_position_id,...
Web page	TLD (top level domain), subdomain, layout_id, ...
Combination	TLD×line_id, ad_position_id×layout_id, ...

Finally, we looked into the uncertainty information produced by OLSS for feature analysis. We focused on the BrightRoll dataset in Section V-A and ran OLSS with $\rho_0 = 10^{-7}$. We selected 504 out of $204,037$ features. The whole feature set are described in Table I. We investigated the posterior inclusion probability and weight variance for each feature.

First, as shown in Fig. 4f, the posterior inclusion probabilities are correlated to the posterior means of the feature weights: features with posterior means close to 0 usually correspond to tiny inclusion probabilities, meaning those features are very unlikely to be selected; features with large (absolute) posterior means often have big inclusion probabilities (say, close to 1), meaning these features are selected and important. This is consistent with L_1-type methods which only perform weight shrinkage. However, there are many cases between the two extremes, *e.g.,* the posterior weight means are around 0.2 but the posterior inclusion probabilities are between 0.4 and 0.6. It is interesting to know how and why the corresponding features act in this way. Second, we noticed that the posterior weight variance is directly correlated to the cardinality of the feature group (See Fig. 5). For example, TLD features of small websites (*e.g.,* TLD_soompi.com and TLD_smartsmania.it) have much higher posterior weight variances than layout_id or ad_position_id. The reason is that the whole TLD category has much higher cardinality, and each specific TLD appears much less frequently; the posterior variance is larger with less observed data. This might explain why empirically account managers choose low cardinality feature groups to set up targeting attributes. Similarly, feature combination, i.e., the cross features (see Table I) can generate more fine-grained features, and are often observed with higher (posterior mean) weights; however, the (posterior) variances increase as well. To mitigate this issue, we often instead use their real-valued versions (*e.g.,*TLD_CTR and (TLD×line_id)_CTR), to reduce the variances (See the variances of those features in Fig. 5).

VI. CONCLUSION

We have presented OLSS, an online Bayesian sparse learning algorithm for large-scale, high dimensional feature selection problems. OLSS exhibits an amazing selective shrinkage effect and is able to quantify the uncertainty information. OLSS has shown its effectiveness in CTR prediction and text classification applications. While general linear models (*e.g.,* logistic regression) are popular in online applications, such as web advertising and online recommendation, recent research explore deep neural networks to automatically extract more discriminative, nonlinear features (*e.g.,* [32]–[34]). A very successful approach is Wide & Deep model [35] — it jointly learns a linear model (the wide component) and a neural network (the deep component) to include both the manually crafted and automatically extracted features for prediction. Like FTRLp, our approach can be further used to contribute to the wide component of this powerful and flexible model.

REFERENCES

[1] H. B. McMahan, "Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization." in *AISTATS*, 2011, pp. 525–533.

[2] J. Langford, "Vowpal wabbit," *URL http://hunch.net/vw*, 2013.

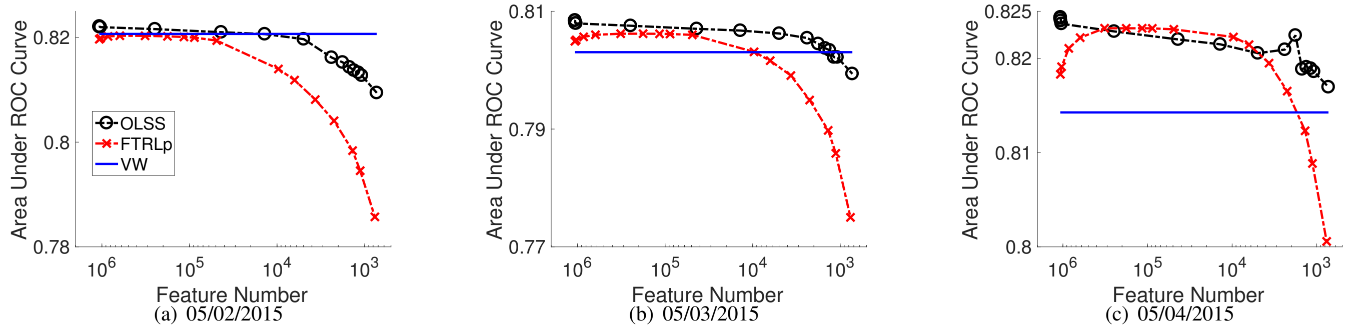

Fig. 2. Prediction accuracy *v.s.* the number of features on CTR data from Yahoo! Gemini Search Ads platform. Note that VW uses all the features and does not perform feature selection.

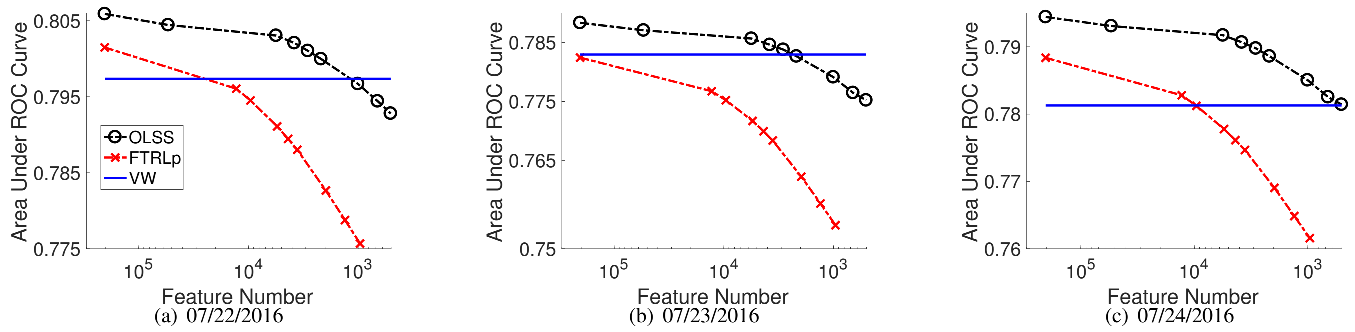

Fig. 3. Prediction accuracy *v.s.* the number of features on CTR data from Yahoo! BrightRoll Display Ads platform.

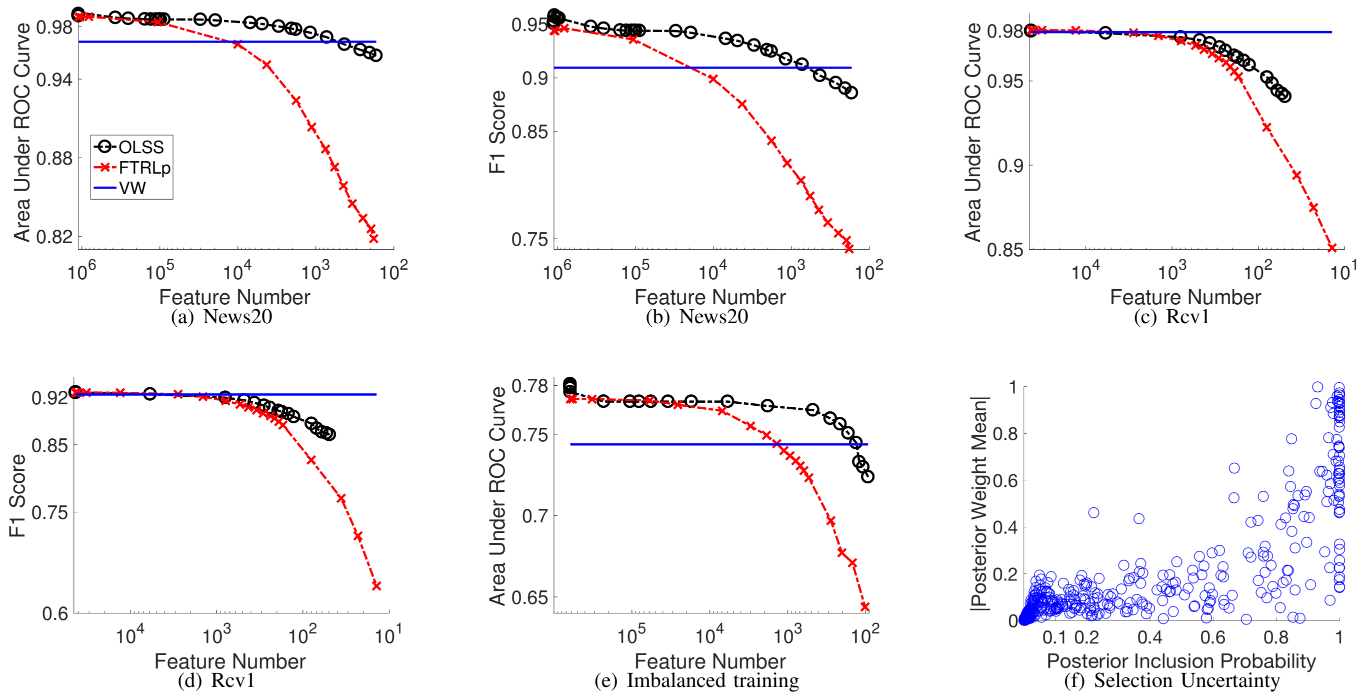

Fig. 4. Predictive performance *v.s.* the number of features on text classification data News20 and Rcv1 (a-d), and on imbalanced CTR training data from Yahoo! Gemini Search Ads platform (e); feature inclusion probabilities *v.s.* weight means on BrightRoll dataset (f). Note that VW uses all the features and does not perform feature selection.

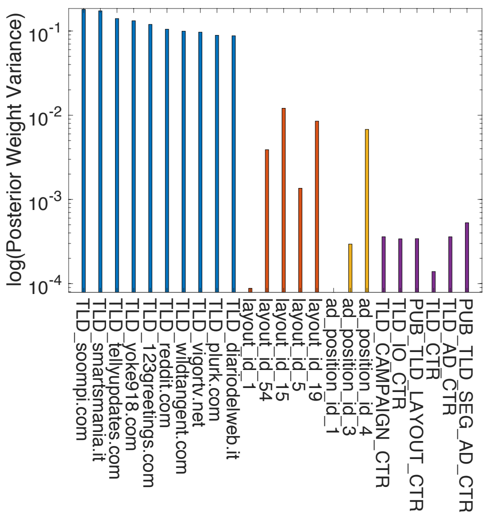

Fig. 5. Posterior variances of the selected features' weights.

[3] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[4] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[5] H. B. McMahan, G. Holt *et al.*, "Ad click prediction: a view from the trenches," in *KDD*, 2013, pp. 1222–1230.

[6] H. Zou, "The adaptive lasso and its oracle properties," *Journal of the American statistical association*, vol. 101, no. 476, pp. 1418–1429, 2006.

[7] H. Ishwaran and J. S. Rao, "Spike and slab variable selection: frequentist and bayesian strategies," *Annals of statistics*, pp. 730–773, 2005.

[8] Y. Li, J. M. Hernández-Lobato, and R. E. Turner, "Stochastic expectation propagation," in *NIPS*, 2015, pp. 2323–2331.

[9] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine learning*, vol. 37, no. 2, pp. 183–233, 1999.

[10] T. P. Minka, "Expectation propagation for approximate Bayesian inference," in *UAI*, 2001, pp. 362–369.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2007.

[12] T. J. Mitchell and J. J. Beauchamp, "Bayesian variable selection in linear regression," *Journal of the American Statistical Association*, vol. 83, no. 404, pp. 1023–1032, 1988.

[13] E. George and R. McCulloch, "Approaches for Bayesian variable selection." *Statistica Sinica*, vol. 7, no. 2, pp. 339–373, 1997.

[14] S. Mohamed, K. Heller, and Z. Ghahramani, "Evaluating Bayesian and L_1 approaches to sparse unsupervised learning," in *ICML*, 2011.

[15] T.-J. Yen, "A majorization–minimization approach to variable selection using spike and slab priors," *The Annals of Statistics*, vol. 39, no. 3, pp. 1748–1775, 2011.

[16] S. A. Z. Naqvi, S. Zhe, Y. Qi, Y. Yang, and J. Ye, "Fast Laplace approximation for sparse Bayesian spike and slab models," in *IJCAI*, 2016, pp. 1867–1973.

[17] P. Carbonetto, M. Stephens *et al.*, "Scalable variational inference for bayesian variable selection in regression, and its accuracy in genetic association studies," *Bayesian Analysis*, vol. 7, no. 1, pp. 73–108, 2012.

[18] M. K. Titsias and M. Lázaro-Gredilla, "Spike and slab variational inference for multi-task and multiple kernel learning." in *NIPS*, vol. 24, 2011, pp. 2339–2347.

[19] D. J. MacKay, "Bayesian methods for backpropagation networks," in *Models of neural networks III*. Springer, 1996, pp. 211–254.

[20] M. E. Tipping, "Sparse bayesian learning and the relevance vector machine," *Journal of machine learning research*, vol. 1, no. Jun, pp. 211–244, 2001.

[21] O. E. Barndorff-Nielsen, "Normal inverse gaussian distributions and stochastic volatility modelling," *Scandinavian Journal of statistics*, vol. 24, no. 1, pp. 1–13, 1997.

[22] A. Armagan, D. Dunson, and J. Lee, "Bayesian generalized double pareto shrinkage," *Biometrika*, 2010.

[23] C. M. Carvalho, N. G. Polson, and J. G. Scott, "Handling sparsity via the horseshoe," in *Artificial Intelligence and Statistics*, 2009, pp. 73–80.

[24] ——, "The horseshoe estimator for sparse signals," *Biometrika*, vol. 97, no. 2, pp. 465–480, 2010.

[25] I. Castillo, J. Schmidt-Hieber, A. Van der Vaart *et al.*, "Bayesian linear regression with sparse priors," *The Annals of Statistics*, vol. 43, no. 5, pp. 1986–2018, 2015.

[26] V. Rovcková *et al.*, "Bayesian estimation of sparse signals with a continuous spike-and-slab prior," *The Annals of Statistics*, vol. 46, no. 1, pp. 401–437, 2018.

[27] R. Pong-Wong, "Estimation of genomic breeding values using the horseshoe prior," in *BMC proceedings*, vol. 8, no. 5. BioMed Central, 2014, p. S6.

[28] H. Li and D. Pati, "Variable selection using shrinkage priors," *Computational Statistics & Data Analysis*, vol. 107, pp. 107–119, 2017.

[29] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang, "Laser: A scalable response prediction platform for online advertising," in *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM, 2014, pp. 173–182.

[30] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *Journal of machine learning research*, vol. 5, no. Apr, pp. 361–397, 2004.

[31] S. S. Keerthi and D. DeCoste, "A modified finite newton method for fast solution of large scale linear svms," *Journal of Machine Learning Research*, vol. 6, no. Mar, pp. 341–361, 2005.

[32] J. Chen, B. Sun, H. Li, H. Lu, and X.-S. Hua, "Deep ctr prediction in display advertising," in *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 2016, pp. 811–820.

[33] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," *arXiv preprint arXiv:1708.05123*, 2017.

[34] B. Edizel, A. Mantrach, and X. Bai, "Deep character-level click-through rate prediction for sponsored search," in *ACM SIGIR*. New York, NY, USA: ACM, 2017, pp. 305–314.

[35] H.-T. Cheng *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 2016, pp. 7–10.