Online Spike-and-slab Inference with Stochastic Expectation Propagation

Shandian Zhe
Purdue University
szhe@purdue.edu

Kuang-chih Lee
Yahoo! Research
kclee@yahoo-inc.com

Kai Zhang
NEC Labs America
kzhang@nec-labs.com

Jennifer Neville
Purdue University
neville@cs.purdue.edu

Abstract

We present OLSS, an online algorithm for Bayesian spike-and-slab model inference, based on the recently proposed stochastic Expectation Propagation (SEP) framework [7]. We use a fully factorized form to efficiently process high dimensional features; further, we extend the standard SEP by incorporating multiple approximate average likelihoods, each of which corresponds to a cluster of samples (e.g., positive and negative ones). This not only better summarizes the data across different regions, but also gives the flexibility to assign sample weights. On a large-scale click-through-rate (CTR) prediction task, OLSS demonstrates excellent sparsity and superior predictive performance to the popular methods in industry, including Vowpal Wabbit [6] and FTRL-Proximal [8].

1 Introduction

Sparse learning is critical to real applications with high dimensional data, for example, classification with a large number of features. On one hand, too many features will lead to a complicated model and to avoid overfitting, we have to collect a huge amount of data and use many computing resources for training, which is time consuming and computationally expensive; on the other hand, the trained model can be ponderous and not handy for real-time applications. For example, a typical online advertising system is required to perform a CTR prediction in 10-100 milliseconds; therefore, the CTR model must be parsimonious.

Spike-and-slab prior [4] is an ideal approach for Bayesian sparse learning. Compared with traditional $L_1$ regularization, the spike-and-slab prior realizes an appealing selective shrinkage property. Specifically, assume we have $d$ features; for each feature $j$, we have a weight $w_j$ and the spike-and-slab prior over $w_j$ is defined as follows:

$$p(s_j) = \text{Bern}(s_j | \rho_0) = \rho_0^{s_j}(1 - \rho_0)^{1-s_j},$$
$$p(w_j | s_j) = s_j \mathcal{N}(w_j | 0, \tau_0) + (1 - s_j)\delta(w_j)$$

where $\delta(\cdot)$ is a Dirac-delta function. Here $s_j$ — a selection indicator sampled from a Bernoulli distribution— decides what type of prior 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 $\tau_0$ (slab component), corresponding to a mild regularization; if otherwise $s_j$ is 0, meaning feature $j$ is irrelevant, the weight $w_j$ is assigned a spike prior centered at 0 (spike component), inducing a strong shrinkage effect.

Despite the amazing property, Bayesian spike-and-slab models are relatively less popular, mainly due to the computational hurdle for posterior inference, especially for large data. Conventional Markov-Chain Monte-Carlo sampling techniques converge very slowly for high dimensional problems;
standard Variational Bayes [5] or Expectation Propagation [9], although every efficient, cannot handle massive samples due to the memory limit of a single computer.

Inspired by the recent stochastic Expectation Propagation framework (SEP) [7], we develop OLSS, an online inference algorithm of Bayesian spike-and-slab models for feature selection; to the best of our knowledge, this is the first algorithm that can deal with both a huge number of samples and high dimensional features. Specifically, we first adopt a factorized form over the feature weights so as to handle high dimensions, and to save computations for sparse categorical features. Second, we extend the standard SEP, by using multiple approximate average-likelihoods, rather than one. Each average-likelihood summarizes the information from a cluster of samples. In this way, data distributions in different regions can be more accurately captured, at a negligible extra cost. In addition, it provides a flexibility of assigning weights for samples in different clusters, say, positive and negative samples.

We have applied OLSS for a real CTR prediction task. On the data with millions of samples, and hundreds of thousands features, OLSS can greatly reduce the number of features to a few hundreds, without sacrificing much prediction accuracy; on average, OLSS obtains a superior predictive performance to the state-of-the-art methods in industry, including Vowpal Wabbit [6] and FTRL-proximal [8]. Furthermore, the selected features by OLSS are proven very useful to construct more advanced, nonlinear CTR prediction models.

2 Stochastic Expectation Propagation

Let us first briefly review EP [9][10] and SEP [7]. Consider a probabilistic model parameterized by \( \theta \). Given the data \( D = \{z_1, \ldots, z_N\} \), the joint probability is \( p(\theta, D) = p_0(\theta) \prod_n p(z_n | \theta) \). To obtain the exact posterior \( p(\theta | D) \), we have to calculate the marginal distribution \( p(D) \), which is usually intractable. To address this problem, EP uses an exponential-family term \( f_n(\theta) \) to approximate each likelihood \( p(z_n | \theta) \), resulting in an approximate posterior \( q(\theta) \propto f_0(\theta) \prod_n f_n(\theta) \). Using the property that the exponential family are close under multiplying and dividing operations, EP cyclically refines each approximate term \( f_i \) in the following four steps: (1) calculating the calibrating distribution, \( q_{-i}(\theta) \propto q(\theta) / f_i(\theta) \); (2) constructing a tilted distribution \( t_i(\theta) \propto q_{-i}(\theta) p(z_i | \theta) \); (3) projecting \( t_i \) back into the exponential family, \( q^*(\theta) \propto \text{proj}(t_i(\theta)) \), via moment matching; (4) updating the term \( f_i \): \( f_i^{new}(\theta) \propto q^*(\theta) / q_{-i}(\theta) \).

EP often works well in practice. However, since it maintains an approximate likelihood term \( f_n(\theta) \) for every sample \( n \), it may fail when the samples are too many to be stored in memory. To address this issue and make EP scalable for large data, SEP instead uses one average-likelihood term, \( f_0(\theta) \), to summarize all the data likelihoods, and defines the approximate posterior to be \( q(\theta) \propto f_0(\theta) f_a(\theta)^N \). By only keeping and updating \( f_0 \) and \( f_a \), SEP greatly reduces the memory usage. SEP further uses an online mechanism to update \( f_a(\theta) \). Specifically, given sample \( n \), we calculate the calibrating distribution by \( q_{-n}(\theta) \propto q(\theta) / f_a(\theta) \), and follow the same way as the original EP to obtain an approximate likelihood, \( f_a(\theta) \); we then integrate \( f_a(\theta) \) into the updating of \( f_0(\theta) \), by taking the (geometric) average over the approximate data likelihoods, where the likelihood for sample \( n \) is represented by \( f_n(\theta) \) and all the others are represented by \( f_a(\theta) \). Therefore, we have \( f_a(\theta)^{new} = (f_n(\theta) f_a(\theta)^{N-1})^{1/N} \). Writing down the updates in terms of the natural parameters, we have \( \lambda_a^{new} = (1 - \frac{1}{N})\lambda_a + \frac{1}{N} \lambda_n \) where \( \lambda_n \) and \( \lambda_a \) are for \( f_a \) and \( f_n \) respectively. We can see that the natural parameters of \( f_a \) is updated by a weighted combination of the old values and the new version from the current sample. Further, we can use a mini-batch of samples \( \{z_{n_1}, \ldots, z_{n_M}\} \) to achieve a larger move: \( \lambda_a^{new} = \frac{1}{N} \sum_{j=1}^M \lambda_{n_j} + (1 - \frac{M}{N}) \lambda_a \).

3 Online Inference for Bayesian Spike-and-slab Models

Now, we present OLSS, our online inference algorithm for spike-and-slab models based on the SEP framework. We focus on sparse linear models with spike-and-slab priors. Suppose we have a dataset \( D = \{(x_1, y_1), \ldots, (x_N, y_N)\} \), where each \( x_n \) is a \( d \)-dimensional feature vector and \( y_n \) is the response. Here we consider binary responses for the classification task, i.e., \( y_n \in \{+1, -1\} \). We assume a \( d \times 1 \) weight vector \( w \), such that given \( x_n \), we have \( p(y_n | x_n, w) = \Phi(\Phi(\Phi(y_n, w^T x_n), where \( \Phi(\cdot) \) is the CDF of standard Gaussian distribution. Note that in real applications, although \( x_n \) can be high dimensional features.
After training, we select all the features that have the posterior selection probabilities bigger than 1. We use an average-likelihood, to better capture the shape of full data distribution. Second, we can vary the weights for different class of samples, through the settings of \( j \). This has two advantages: first, the summarization of the data and is fully factorized over features, where \( n \) is mainly due to the sparse categorical features, such as the product brand or the web site domain. They often have a large cardinality and we have to use a sparse, long feature vector for representation.

For tractable inference, we first approximate the prior term, \( s_j \mathcal{N}(w_j|0, \tau_0) + (1 - s_j)\delta(w_j) \), with \( \mathcal{N}(s_j|\mu|\mu_j, v_j) \). Then, we use two average-likelihood terms, \( f_a^+(w_1) \) and \( f_a^-(w_1) \), defined by \( f_a^+(w_1) = \prod_{j=1}^d \mathcal{N}(w_j|\mu_j^{\tilde{y}_j}, v_j^{\tilde{y}_j}) \) and \( f_a^-(w_1) = \prod_{j=1}^d \mathcal{N}(w_j|\mu_j^{-y_j}, v_j^{-y_j}) \), for the positive and negative samples respectively. We then have the approximate posterior to be \( q(s, w) \propto \prod_{j=1}^d \text{Bern}(s_j|\rho_j) \mathcal{N}(s_j|\mu_j, v_j) \prod_{n=1}^N f_a^+(w_{I_n})^{y_n} f_a^-(w_{I_n})^{1-y_n} \). Hence, \( q(w, s) \propto \prod_{j=1}^d \text{Bern}(s_j|\rho_j) \mathcal{N}(w_j|\mu_j, v_j) \prod_{n=1}^N f_a^+(w_{I_n})^{y_n} f_a^-(w_{I_n})^{1-y_n} \). And \( \mathcal{N}(w_j|\mu_j, v_j) \) is the logitic function.

The algorithm, OLSS, sequentially processes data, each time a mini-batch. In each mini-batch, we update the weights for different class of samples, through the settings of \( j \). This can be very useful for applications with unbalanced samples. Take the online advertising as an example. The number of clicked impressions (i.e., positive samples) are 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 \( \{ n^{-y}_j \} \) 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.

The algorithm, OLSS, sequentially processes data, each time a mini-batch. In each mini-batch, we approximate the average-likelihood for each positive and negative samples in parallel, then update the corresponding average-likelihood terms for each feature \( j \), i.e., \( \mathcal{N}(w_j|\mu_j^{\tilde{y}_j}, v_j^{\tilde{y}_j}) \) and \( \mathcal{N}(w_j|\mu_j^{-y_j}, v_j^{-y_j}) \), following the way mentioned in Section 2. After every a few mini-batches, we update the prior weight terms, \( \{ \text{Bern}(s_j|\rho_j) \mathcal{N}(w_j|\mu_j, v_j) \}_{j,t} \), with the current average-likelihoods. The derivation of the updates is pretty standard, hence we omit the details to save space. The algorithm is summarized in Algorithm 1.

After training, we select all the features that have the posterior selection probabilities bigger than \( \frac{1}{2} \), i.e., \( \{ j|q(s_j = 1) > \frac{1}{2} \} \). Then we use the selected feature weights for prediction.

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

Random shuffle samples in \( D \).

Initialize for each feature \( j \): \( \rho_j = 0.5, \mu_j = \mu_j^{+} = \mu_j^{-} = 0, v_j^{+} = v_j^{-} = 10^6 \). 

repeat

Collect a mini-batch of samples \( B_t \), with size \( M \), where \( B_t^{+} \) and \( B_t^{-} \) denote the positive and negative samples, and \( b_{ij}^{+} \) and \( b_{ij}^{-} \) denote the appearance counts of feature \( j \) in \( B_t^{+} \) and \( B_t^{-} \). Calculate the approximate likelihood for each sample in \( B_t \): \( \{ \mathcal{N}(w_j|\mu_j, v_j) \}_{j,t} \), with the current average-likelihoods.

Update the Gaussian terms for the average-likelihoods:

\[
\begin{align*}
\mu_j^{+} &\leftarrow \frac{\mu_j^{+} - \frac{1}{n_j^{+}} n_j^{+} b_{ij}^{+} v_j^{+}}{\frac{1}{n_j^{+}} n_j^{+}} \quad \mu_j^{-} &\leftarrow \frac{\mu_j^{-} + \frac{1}{n_j^{-}} n_j^{-} b_{ij}^{-} v_j^{-}}{\frac{1}{n_j^{-}} n_j^{-}} \\
v_j^{+} &\leftarrow \frac{n_j^{+} + b_{ij}^{+} v_j^{+}}{n_j^{-} + b_{ij}^{-} v_j^{-}} \quad v_j^{-} &\leftarrow \frac{n_j^{-} + b_{ij}^{+} v_j^{+}}{n_j^{-} + b_{ij}^{-} v_j^{-}} \\
\end{align*}
\]

If \( T \) mini-batches have been processed, update \( \{ \rho_j, \mu_j, v_j \}_{j,t} \) for the approximate prior terms.

until all samples in \( D \) is processed.

return \( q(w, s) = \prod_j \mathcal{N}(w_j|\mu_j, v_j) \text{Bern}(s_j|\rho_j) \), where \( v_j = (v_j^{-1} + n_j^{+} v_j^{+} + n_j^{-} v_j^{-})^{-1} \), \( \mu_j = v_j \left( \frac{n_j^{+} \mu_j^{+}}{n_j^{+} v_j^{+}} + n_j^{-} \mu_j^{-} \frac{v_j^{-}}{v_j^{+}} \right) \), \( \alpha_j = \sigma(\sigma^{-1}(\rho_j) + \sigma^{-1}(\mu_j)) \) (\( \sigma(\cdot) \) is the logitic function).
4 Experiment

We examined OLSS in a real CTR prediction task. We collected the training data from a 7 days’ click logs generated by Yahoo! Display Ads platform, between 07/15/2016 and 07/21/2016. Then we tested on the logs in 07/22/2016, 07/23/2016 and 07/24/2016. The number of features are 204, 327; the size of training and testing data are 1.8M, 133.7M, 116.0M and 110.2M. For training, we collected all the click impressions and subsampled a comparable number of non-clicks, while for testing data, we used all the click and non-click impressions. Note that training CTR prediction models with comparable clicks and non-clicks is common in online advertising systems [1].

We compared with two state-of-the-art methods widely used in industry: online logistic regression in Vowpal Wabbit (VW) without feature selection, and FTRL-proximal (FTRLp) with online feature selection. For our approach, we set τC to 1.0, M to 100 and T to 1. We varied ρ0—the prior belief about the ratio of selected features—to adjust the sparsity level; for VW we adopted the default parameters, which turned out to perform best in prediction; FTRLp has four parameters, α, β, λ1 and λ2, where \{α, β\} are used to control the per-coordinate learning rate, and \{λ1, λ2\} are the strengths for L1 and L2 regularization; to choose the best parameters, we fixed L1 = 1.0 and L2 = 1.0 and fine tuned \{α, β\} in a validation dataset sampled from the log in 07/23/2016. The best settings are α = β = 0.1. Then we fixed λ2 to 1.0, adjusted λ1 and examined the sparsity and the predictive accuracy (in our application, different choices of λ2 have little effect to the predictive performance).

First, we examined how much sparsity OLSS can yield when varying ρ0. From Table 1 we can see that big ρ0 encouraged a large number of features to be selected; when we decreased ρ0, OLSS quickly pruned massive features, as expected. Finally, the number of features can be reduced to a few hundreds, taking only 0.2% of the entire feature set.

Table 1: The number of selected features v.s. the setting of ρ0.

<table>
<thead>
<tr>
<th>ρ0</th>
<th>0.8</th>
<th>0.5</th>
<th>0.4</th>
<th>0.3</th>
<th>0.1</th>
<th>10^{-4}</th>
<th>10^{-5}</th>
<th>10^{-6}</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature number ratio (%)</td>
<td>204,080</td>
<td>35,827</td>
<td>5,591</td>
<td>3,010</td>
<td>2,174</td>
<td>1,004</td>
<td>663</td>
<td>504</td>
</tr>
<tr>
<td>99.9%</td>
<td>26.3%</td>
<td>2.7%</td>
<td>1.9%</td>
<td>1.1%</td>
<td>0.5%</td>
<td>0.3%</td>
<td>0.2%</td>
<td></td>
</tr>
</tbody>
</table>

Next, we examined the predictive performance of OLSS and FTRL when selecting different number of features. We report the area-under-curve (AUC) for all the three test datasets. As shown in Figure 1a-c, the prediction accuracy decreased when using less and less features for both OLSS and FTRL. However, OLSS always outperformed FTRLp, in all sparsity levels. This is more significant when smaller number of features were selected. In addition, compared with VW using all the features, our method, OLSS, kept a superior predictive performance until the feature number became too small. However, the accuracy drop of OLSS is much less than FTRLp.

Finally, to confirm the usefulness of the selected features, we trained a nonlinear classification model, Gradient Boosting Tree (GBT) [2, 3], based on the 504 features selected by OLSS (when setting ρ0 = 10^{-7}). GBT has an excellent performance for CTR prediction [11] but is not scalable for high dimensional features. We compared GBT with OLSS on the same 504 features, and with VW using all the 204, 037 features. As shown in Figure 1d, GBT outperformed both OLSS and VW on all the three test datasets. Therefore, the selected features by OLSS are useful not only for linear classification models, but also for the advanced, nonlinear models. This enlightens another application of sparse learning—that is, we can choose a small set of useful features, then based on which we can construct feasible and more powerful models to further improve the prediction tasks.
References


