Online Spike-and-slab Inference with Stochastic Expectation Propagation

Shandian Zhe
Purdue Univesity
szhe@purdue.edu

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

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

Jennifer Neville
Purdue Univesity
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:

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

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 τ_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;

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

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 $\boldsymbol{\theta}$. Given the data $\mathcal{D} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, the joint probability is $p(\boldsymbol{\theta}, \mathcal{D}) = p_0(\boldsymbol{\theta}) \prod_n p(\mathbf{z}_n|\boldsymbol{\theta})$.To obtain the exact the posterior $p(\boldsymbol{\theta}|\mathcal{D})$, we have to calculate the marginal distribution $p(\mathcal{D})$, which is usually intractable. To address this problem, EP uses 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 $p_0(\boldsymbol{\theta})$, resulting an approximate posterior $q(\boldsymbol{\theta}) \propto f_0(\boldsymbol{\theta}) \prod_n f_n(\boldsymbol{\theta})$. 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, $q_{-i}(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta})/f_i(\boldsymbol{\theta})$; (2) constructing a tilted distribution $t_i(\boldsymbol{\theta}) \propto q_{-i}(\boldsymbol{\theta}) p(\mathbf{z}_i|\boldsymbol{\theta})$; (3) projecting t_i back into the exponential family, $q^*(\boldsymbol{\theta}) \propto \mathrm{proj}(t_i(\boldsymbol{\theta}))$, via moment matching; (4) updating the term f_i: $f_i^{\mathrm{new}}(\boldsymbol{\theta}) \propto q^*(\boldsymbol{\theta})/q_{-i}(\boldsymbol{\theta})$.

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 address this issue and make EP scalable for large data, SEP instead uses one average-likelihood term, $f_a(\boldsymbol{\theta})$, to summarize all the data likelihoods, and defines the approximate posterior to be $q(\boldsymbol{\theta}) \propto f_0(\boldsymbol{\theta}) f_a(\boldsymbol{\theta})^N$. By only keeping and updating f_0 and f_a, SEP greatly reduces the memory usage. SEP further uses an online mechanism to update $f_a(\boldsymbol{\theta})$. Specifically, given sample n, we calculate the calibrating distribution by $q_{-n}(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta})/f_a(\boldsymbol{\theta})$, and follow the same way as the original EP to obtain an approximate likelihood, $f_n(\boldsymbol{\theta})$; we then integrate $f_n(\boldsymbol{\theta})$ into the updating of $f_a(\boldsymbol{\theta})$, by taking the (geometric) average over the approximate data likelihoods, where the likelihood for sample n is represented by $f_n(\boldsymbol{\theta})$ and all the others are represented by $f_a(\boldsymbol{\theta})$. Therefore, we have $f_a(\boldsymbol{\theta})^{\mathrm{new}} = \left(f_n(\boldsymbol{\theta}) f_a(\boldsymbol{\theta})^{N-1}\right)^{\frac{1}{N}}$. Writing down the updates in terms of the natural parameters, we have $\boldsymbol{\lambda}_a^{\mathrm{new}} = (1 - \frac{1}{N})\boldsymbol{\lambda}_a + \frac{1}{N}\boldsymbol{\lambda}_n$ where $\boldsymbol{\lambda}_a$ and $\boldsymbol{\lambda}_n$ 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 $\{\mathbf{z}_{n_1}, \ldots, \mathbf{z}_{n_M}\}$ to achieve a larger move: $\boldsymbol{\lambda}_a^{\mathrm{new}} = \frac{1}{N}\sum_{j=1}^{M} \boldsymbol{\lambda}_{n_j} + (1 - \frac{M}{N})\boldsymbol{\lambda}_a$.

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 $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_n)\}$, 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 $\mathbf{w}$, such that given $\mathbf{x}_n$, we have $p(y_n|\mathbf{x}_n, \mathbf{w}) = \Phi(y_n \mathbf{w}^\top \mathbf{x}_n)$, where $\Phi(\cdot)$ is the CDF of standard Gaussian distribution. Note that in real applications, although $\mathbf{x}_n$ can

be extremely high dimensional, they are often very sparse, i.e., most of the elements are zero. This 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. Therefore, to avoid the unnecessary computation, we rewrite $p(y_n|\mathbf{x}_n, \mathbf{w}) = \Phi(y_n \mathbf{w}_{I_n}^\top \hat{\mathbf{x}}_n)$ where I_n is the indexes of nonzero elements in $\mathbf{x}_n$ and $\hat{\mathbf{x}}_n$ is the corresponding nonzero subvector. We further assign the spike-and-slab prior over $\mathbf{w}$ (see (1)), and obtain the joint probability as follows: $p(\mathcal{D}, \mathbf{w}, \mathbf{s}|\rho_0, \tau_0) = \prod_{j=1}^d \mathrm{Bern}(s_j|\rho_0)\big(s_j \mathcal{N}(w_j|0, \tau_0) + (1 - s_j)\delta(w_j)\big) \prod_{n=1}^N \Phi(y_n \mathbf{w}_{I_n}^\top \hat{\mathbf{x}}_n)$.

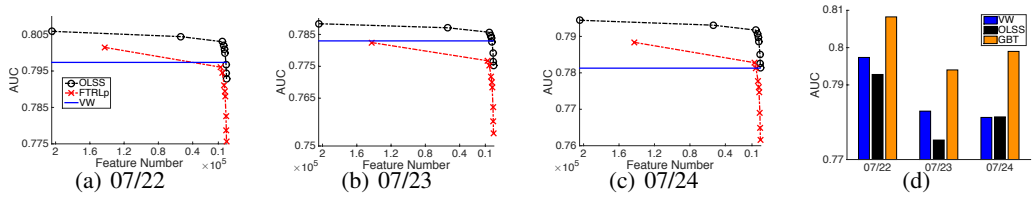
For tractable inference, we first approximate the prior term, $s_j \mathcal{N}(w_j|0, \tau_0) + (1 - s_j)\delta(w_j)$, with $\mathrm{Bern}(s_j|\rho_j)\mathcal{N}(w_j|\mu_{1j}, v_{1j})$. Then, we use two average-likelihood terms, $f_a^+(\mathbf{w}_I)$ and $f_a^-(\mathbf{w}_I)$, defined by $f_a^+(\mathbf{w}_I) = \prod_{j \in I} \mathcal{N}(w_j|\mu_{2j}^+, v_{2j}^+)$ and $f_a^-(\mathbf{w}_I) = \prod_{j \in I} \mathcal{N}(w_j|\mu_{2j}^-, v_{2j}^-)$, for the positive and negative samples respectively. We then have the approximate posterior to be $q(\mathbf{s}, \mathbf{w}) \propto \prod_{j=1}^d \mathrm{Bern}(s_j|\rho_0)\mathrm{Bern}(s_j|\rho_j)\mathcal{N}(w_j|\mu_{1j}, v_{1j}) \prod_{n=1}^N f_a^+(\mathbf{w}_{I_n})^{\mathbb{1}(y_n=1)} f_a^-(\mathbf{w}_{I_n})^{\mathbb{1}(y_n=-1)}$. Hence, $q(\mathbf{w}, \mathbf{s}) \propto \prod_{j=1}^d \mathrm{Bern}(s_j|\rho_0)\mathrm{Bern}(s_j|\rho_j)\mathcal{N}(w_j|\mu_{1j}, v_{1j})\mathcal{N}(w_j|\mu_{2j}^+, v_{2j}^+)^{n_j^+}\mathcal{N}(w_j|\mu_{2j}^-, v_{2j}^-)^{n_j^-}$, and is fully factorized over features, where n_j^+ and n_j^- are the appearance counts of feature j in positive and negative samples, respectively. Note that unlike the standard SEP using only one average-likelihood for all the samples, we consider the different sample types and for each type, we use a different average-likelihood. This has two advantages: first, the summarization of the data likelihoods can be more accurate; and in general we can cluster the data first, then for each cluster 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 n_j^+ and n_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 $\{n_j^-\}_j$ 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.

The algorithm, OLSS, sequentially processes data, each time a mini-batch. In each min-batch, we calculate the approximate likelihoods for each positive and negative samples in parallel, then update the corresponding average-likelihood terms for each feature j, i.e., $\mathcal{N}(w_j|\mu_{2j}^+, v_{2j}^+)$ and $\mathcal{N}(w_j|\mu_{2j}^-, v_{2j}^-)$, following the way mentioned in Section 2. After every a few mini-batches, we update the approximate prior terms, $\{\mathrm{Bern}(s_j|\alpha_j)\mathcal{N}(w_j|\mu_{1j}, v_{1j})\}_j$, 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($\mathcal{D}, \rho_0, \tau_0, M, T, \{n_j^+, n_j^-\}_j$)

Random shuffle samples in $\mathcal{D}$.
Initialize for each feature j: $\rho_j = 0.5, \mu_{1j} = \mu_{2j}^+ = \mu_{2j}^- = 0, v_{1j} = v_{2j}^+ = v_{2j}^- = 10^6$.
repeat
  Collect a mini-batch of samples B_i with size M, where B_i^+ are B_i^- denote the positive and negative samples, and b_{ij}^+ and b_{ij}^- denote the appearance counts of feature j in B_i^+ and B_i^-.
  Calculate the approximate likelihood for each sample in B_i: $\{\mathcal{N}(w_j|\mu_{jt}, v_{jt})\}_{j,t \in B_i}$.
  Update the Gaussian terms for the average-likelihoods:
  ${v_{2j}^+}^{-1} \leftarrow \frac{b_{ji}^+}{n_j^+} \sum_{t \in B_i^+} v_{jt}^{-1} + \frac{n_j^+ - b_{ji}^+}{n_j^+} {v_{2j}^+}^{-1}, \frac{\mu_{2j}^+}{v_{2j}^+} \leftarrow \frac{b_{ji}^+}{n_j^+} \sum_{t \in B_i^+} \frac{\mu_{jt}}{v_{jt}} + \frac{n_j^+ - b_{ji}^+}{n_j^+} \frac{\mu_{2j}^+}{v_{2j}^+}$,
  ${v_{2j}^-}^{-1} \leftarrow \frac{b_{ji}^-}{n_j^-} \sum_{t \in B_i^-} v_{jt}^{-1} + \frac{n_j^- - b_{ji}^-}{n_j^-} {v_{2j}^-}^{-1}, \frac{\mu_{2j}^-}{v_{2j}^-} \leftarrow \frac{b_{ji}^-}{n_j^-} \sum_{t \in B_i^-} \frac{\mu_{jt}}{v_{jt}} + \frac{n_j^- - b_{ji}^-}{n_j^-} \frac{\mu_{2j}^-}{v_{2j}^-}$.
  If T mini-batches have been processed, update $\{\rho_j, \mu_{1j}, v_{1j}\}_j$ for the approximate prior terms.
until all samples in $\mathcal{D}$ is processed.
return $q(\mathbf{w}, \mathbf{s}) = \prod_j \mathcal{N}(w_j|\mu_j, v_j)\mathrm{Bern}(s_j|\alpha_j)$, where $v_j = \big(v_{1j}^{-1} + n_j^+ {v_{2j}^+}^{-1} + n_j^- {v_{2j}^-}^{-1}\big)^{-1}$, $\mu_j = v_j\big(\frac{\mu_{1j}}{v_{1j}} + n_j^+ \frac{\mu_{2j}^+}{v_{2j}^+} + n_j^- \frac{\mu_{2j}^-}{v_{2j}^-}\big)$, $\alpha_j = \sigma\big(\sigma^{-1}(\rho_0) + \sigma^{-1}(\rho_j)\big)$ ($\sigma(\cdot)$ is the logitic function).



(a) 07/22 (b) 07/23 (c) 07/24 (d)

Figure 1: Prediction accuracy *v.s.* the number of features (a-c), and prediction of GBT trained on 504 features selected by OLSS (d). Note that VW in (a-c) uses all the features and does not perform feature selection.

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 τ_0 to 1.0, M to 100 and T to 1. We varied ρ_0—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, λ_1 and λ_2, where $\{\alpha, \beta\}$ are used to control the per-coordinate learning rate, and $\{\lambda_1, \lambda_2\}$ are the strengths for L_1 and L_2 regularization; to choose the best parameters, we fixed $L_1 = 1.0$ and $L_2 = 1.0$ and fine tuned $\{\alpha, \beta\}$ in a validation dataset sampled from the log in 07/23/2016. The best settings are $\alpha = \beta = 0.1$. Then we fixed λ_2 to 1.0, adjusted λ_1 and examined the sparsity and the predictive accuracy (in our application, different choices of λ_2 have little effect to the predictive performance) .

First, we examined how much sparsity OLSS can yield when varying ρ_0. From Table 1, we can see that big ρ_0 encouraged a large number of features to be selected; when we decreased ρ_0, 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 ρ_0.

ρ_0	0.8	0.5	0.4	0.3	0.1	10^{-3}	10^{-5}	10^{-7}
feature number	204,080	53,827	5,591	3,810	2,174	1,004	663	504
ratio (%)	99.9%	26.3%	2.7%	1.9%	1.1%	0.5%	0.3%	0.2%

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 $\rho_0 = 10^{-7}$). 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

[1] Deepak Agarwal, Bo Long, Jonathan Traupman, Doris Xin, and Liang Zhang. Laser: A scalable response prediction platform for online advertising. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 173–182. ACM, 2014.

[2] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[3] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[4] Edward I George and Robert E McCulloch. Approaches for Bayesian variable selection. *Statistica Sinica*, pages 339–373, 1997.

[5] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[6] John Langford. Vowpal wabbit, 2013. *URL http://hunch. net/vw*.

[7] Yingzhen Li, José Miguel Hernández-Lobato, and Richard E Turner. Stochastic expectation propagation. In *Advances in Neural Information Processing Systems*, pages 2323–2331, 2015.

[8] H Brendan McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *AISTATS*, pages 525–533, 2011.

[9] Thomas P Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence (UAI)*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[10] Matthias Seeger. Expectation propagation for exponential families. Technical report.

[11] Ilya Trofimov, Anna Kornetova, and Valery Topinskiy. Using boosted trees for click-through rate prediction for sponsored search. In *Proceedings of the Sixth International Workshop on Data Mining for Online Advertising and Internet Economy*, page 2. ACM, 2012.