BOWL: Bayesian Optimization for Weight Learning in Probabilistic Soft Logic

Sriram Srinivasan
UC Santa Cruz
ssriniv9@ucsc.edu

Golnoosh Farnadi
Mila, Université de Montréal
farnadig@mila.quebec

Lise Getoor
UC Santa Cruz
getoor@ucsc.edu

Abstract

Probabilistic soft logic (PSL) is a statistical relational learning framework that represents complex relational models with weighted first-order logical rules. The weights of the rules in PSL indicate their importance in the model and influence the effectiveness of the model on a given task. Existing weight learning approaches often attempt to learn a set of weights that maximizes some function of data likelihood. However, this does not always translate to optimal performance on a desired domain metric, such as accuracy or F1 score. In this paper, we introduce a new weight learning approach called *Bayesian optimization for weight learning (BOWL)* based on Gaussian process regression that directly optimizes weights on a chosen domain performance metric. The key to the success of our approach is a novel projection that captures the semantic distance between the possible weight configurations. Our experimental results show that our proposed approach outperforms likelihood-based approaches and yields up to a 10% improvement across a variety of performance metrics. Further, we performed experiments to measure the scalability and robustness of our approach on various realworld datasets.

1 Introduction

Statistical relational learning (SRL) frameworks (Richardson and Domingos 2006; De Raedt and Kersting 2011; Getoor and Taskar 2007) combine the power of graphical models with probabilistic programming to produce accurate models on complex relational data. Probabilistic soft logic (PSL) (Bach et al. 2017) is a recently developed SRL framework that makes use of weighted first-order logical rules to generate a special kind of Markov random field (MRF) called a hinge-loss Markov random field (HL-MRF). Some of the key distinguishing properties of PSL are, unlike other SRL frameworks, random variables take continuous values between 0 and 1 and the potential functions are defined as hinge losses (further details are given in Section 2.1). These properties enable MAP inference in PSL to be cast as a convex optimization problem which makes inference in PSL scalable. PSL has achieved state-of-the-art results in various domains such as recommender systems (Kouki et al. 2017), bioinformatics (Sridhar, Fakhraei, and Getoor 2016), natural language processing (Deng and Wiebe 2015), product search (Srinivasan et al. 2019), and social network analysis (Farnadi et al. 2017).

Learning the weights of the rules is one of the key challenges for templated rule languages such as PSL, since the weighted rules interact in complex ways and cannot be optimized independently. Because of their templated nature, the weights, i.e., the parameters of the model, are used in multiple places in the instantiated graphical model, and the context varies depending on the other rules that have been instantiated. In addition, the corresponding probability distribution is not easy to compute; specifically, computing the normalization constant is often intractable.

Typically, the weights of the rules are learned through maximizing some form of likelihood function (Bach et al. 2013; Lowd and Domingos 2007; Singla and Domingos 2005; Kok and Domingos 2005; Chou et al. 2016; Sarkhel et al. 2016; Das et al. 2016; Farabi, Sarkhel, and Venugopal 2018). This is a well-motivated approach if the downstream objective makes use of the probability density function directly. However, the objective is to often improve an external domain metric such as accuracy, F1 for classification, or ROC for ranking. Several approaches address this issue by augmenting the metric into a loss function and solving a max-margin problem (Huynh and Mooney 2009; 2010; Bach et al. 2013). However, this does not directly optimize the desired metric but instead optimizes a surrogate loss. Such approaches do not easily extend to new metrics as they require deriving new losses, which may be non-convex and hard to optimize.

In this paper, we introduce a Bayesian optimization framework for finding the best weight configuration in PSL. The key advantage of our approach is that it directly optimizes the chosen domain performance metric and, unlike other approaches, does not require rederivation of the loss function for each metric. Our proposed approach, Bayesian optimization for weight learning (BOWL), is based on Gaussian process regression (GPR) (Rasmussen and Williams 2005) in a Bayesian optimization (BO) (Mockus 1977) framework. BO is an effective approach for optimization of black-box functions (Lizotte et al. 2007; Martinez-Cantin et al. 2009; Srinivas et al. 2012; Brochu, Brochu, and de Freitas 2010) and GPR is a non-parametric Bayesian approach that is often used to approximate arbitrary functions. GPRs have been used

extensively for hyperparameter tuning in machine learning (Snoek, Larochelle, and Adams 2012).

Our proposed weight learning approach takes into account both the nuances of the weights in PSL and the impact of model instantiation (referred to as *grounding*). To the best of our knowledge, there is no prior work that uses a BO framework to perform weight learning in SRL frameworks. Even though we define our weight learning method for PSL, it can be extended to other SRL frameworks such as Markov logic networks (Richardson and Domingos 2006).

Our contributions in this paper are as follows: 1) we devise a new approach to weight learning in PSL called BOWL which optimizes a user-defined performance metric; 2) we introduce two variants, BOWL-original space (BOWL-OS) and BOWL-scaled space (BOWL-SS), and show that BOWL-SS yields a more effective exploration of weight space; 3) we prove correctness of our approach; 4) we show that BOWL outperforms likelihood-based approaches on multiple datasets by up to 10%; and 5) through a series of experiments, we show that BOWL is scalable and robust.

2 Background

In this section, we first briefly review probabilistic soft logic (PSL) (Bach et al. 2017) and three commonly used PSL weight learning approaches. Next, we present required background on black-box optimization and Gaussian process regression (GPR) which serve as the foundation for our proposed approach.

2.1 Probabilistic soft logic

PSL is a probabilistic programming language that uses a set of weighted first-order logical rules to define a probabilistic graphical model called a *Hinge-loss Markov Random Field (HL-MRF)*. We begin by defining a HL-MRF and then show how the weighted first-order logical rules, together with data, instantiate a HL-MRF.

Definition 1 (Hinge-loss Markov random field)**.** *Let* $\mathbf{y} = \{y_1, y_2, ..., y_n\}$ *be* n *random variables,* $\mathbf{x} = \{x_1, x_2, ..., x_m\}$ *be* m *observed variables or evidence, and* $\phi = \{\phi_1, \phi_2, ..., \phi_\iota\}$ *be* ι *hinge-loss potentials such that,* $\phi_i(\mathbf{x}, \mathbf{y}) = \max(\ell_i(\mathbf{x}, \mathbf{y}), 0)^{d_i}$*, where* ℓ_i *is a linear function and* $d_i \in \{1, 2\}$ *provides a choice of two different loss functions,* $d_i = 1$ *(i.e., linear) and* $d_i = 2$ *(i.e, quadratic). For weights* $\mathbf{w} \in \{w_1, w_2, ..., w_\iota\}$ *a hinge-loss energy function can be defined as:*

$$\mathbf{E}(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{\iota} w_i \phi_i(\mathbf{x}, \mathbf{y}) \quad \textit{s.t.,}\ \mathbf{0} \leq \mathbf{y} \leq \mathbf{1}\ ;\ \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \tag{1}$$

and the HL-MRF is defined as:

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{1}{Z(\mathbf{y})} exp(-E(\mathbf{y}|\mathbf{x})) \tag{2}$$

where $Z(\mathbf{y}) = \int_{\mathbf{y}} exp(-E(\mathbf{y}|\mathbf{x}))$.

A PSL model defines a HL-MRF using a set of weighted first-order logical rules. PSL instantiates the hinge-loss energy function by grounding logical rules with data D. This process specifies the dependencies between variables and evidence in hinge-loss potentials ϕ_i. PSL uses continuous random variables in $[0, 1]$ and defines potentials using convex functions that are relaxations of Boolean logical connectives. For example, $a \rightarrow b$ corresponds to the hinge potential $\max(a - b, 0)$, and $a \wedge b$ corresponds to $\max(a + b - 1, 0)$ (see Bach et. al. for full details).

To better understand HL-MRFs, PSL and the process of grounding, consider a simple collective labeling problem:

Example 1. *Assume we have a set of users* $\mathbf{U}$ *and a label which can be either* $true$ *or* $false$ *associated with each user. Labels are observed for some users (*$\mathbf{U}_o$*) and unobserved for the rest (*$\mathbf{U}_u$*). The task is to infer labels for* $\mathbf{U}_u$*. The input data includes a social network of the supplied users* $\mathbf{U}$*,* $Friend(U, V)$*. Suppose we also have a local predictor that makes label predictions for users,* $LocalPredictor(U)$*. Below is a simple PSL model for collectively inferring labels:*

$$w_1 : \textit{LocalPredictor}(U) \rightarrow \textit{Label}(U)$$
$$w_2 : \textit{Label}(U) \wedge \textit{Friend}(U, V) \rightarrow \textit{Label}(V)$$

where w_1 *and* w_2 *are non-negative weights for the rules. The above model is then grounded with users* $\mathbf{U}$ *to generate an HL-MRF. Each ground rule (such as* w_1 : *LocalPredictor(*"Bob"*)* $\rightarrow$ *Label(*"Bob"*)) generates a hinge-loss potential* ϕ_i*. Further, each ground predicate generated by grounding users* $\mathbf{U}_u$ *with the predicate* $Label$ *produces the unobserved random variables* $\mathbf{y}$ *and rest of the ground predicates generate observed random variables* $\mathbf{x}$*. Fig. 1 shows the resulting graphical model when instantiated over a small social network of* 100 *individuals.*

Inference in PSL is performed by finding a maximum aposteriori estimate (MAP) of the random variables $\mathbf{y}$ given evidence $\mathbf{x}$. This is performed by maximizing the density function or minimizing the energy function in Equation 1. MAP inference is expressed as:

$$\underset{\mathbf{y}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmin}}\, \mathbf{E}(\mathbf{y}|\mathbf{x}) \tag{3}$$

A key advantage of using PSL is that the inference objective is convex. This enables the use of efficient convex optimization procedures, such as alternating direction method of multipliers (ADMM) (Boyd et al. 2011). Hence, given known weights, inference in PSL can be performed at scale enabling predictions on large realworld datasets. Unfortunately, the task of learning the rule weights from training data is not as efficient, although, as we will see, having tractable MAP inference is still helpful.

There are three primary approaches used to perform weight learning in PSL as discussed in (Bach et al. 2013):

Maximum Likelihood Estimation (MLE) An approach for weight learning in PSL that maximizes the log-likelihood function with respect to the weights of the rules based on the training data. Since all the potentials generated by a rule share the same weight, Equation 1 can be written as

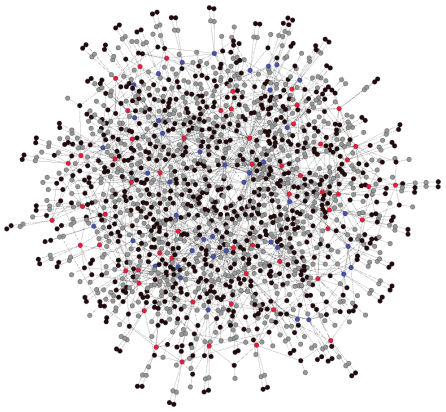


Figure 1: Factor graph produced by grounding the example PSL model with synthetic data for 100 users. The blue nodes are users whose label is true, the red nodes are users whose label is false, grey nodes are the rest of the grounded atoms and the black nodes are hinge-loss potentials. The red nodes represent users and blue nodes are potentials. Here we see that the resulting factor graph, even for this simple case, is large, complex, and highly connected.

$\sum_{i=1}^{r}[w_i \sum_{j=1}^{g_i} \phi_{i,j}(\mathbf{x},\mathbf{y})]$ where r is the number of template rules, w_i represents the weight of the i^{th} rule, g_i is the number of groundings generated by the i^{th} rule, and $\phi_{i,j}$ is the j^{th} potential generated by the i^{th} rule. The partial derivative of the log of the likelihood function given in Equation 2 with respect to w_q, for $q \in \{1,\ldots,r\}$ is:

$$\frac{\partial logP(\mathbf{y}|\mathbf{x})}{\partial w_q} = \mathbb{E}_{\mathbf{w}}\Big[\Phi_q(\mathbf{x},\mathbf{y})\Big] - \Phi_q(\mathbf{x},\mathbf{y}) \tag{4}$$

where $\mathbf{w} = \{w_1,...,w_r\}$ and $\Phi_q = \sum_{j=1}^{g_q} \phi_{qj}(\mathbf{x},\mathbf{y})$. It is infeasible to compute the expectation, hence to make the learning tractable, a MAP approximation is used that replaces the expectation in the gradient with the corresponding values in the MAP state. This approach is a structured variant of the voted perceptron algorithm (Collins 2002).

Maximum Pseudolikelihood Estimation (MPLE) An alternative approach to weight learning that maximizes the pseudolikelihood function, which is given by:

$$P^*(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P^*(y_i|MB(y_i),\mathbf{x}) \tag{5}$$

where n is the number of random variables and $MB(y_i)$ is the Markov blanket of y_i. Equation 5 is maximized using a gradient ascent based approach and the derivative of the log-pseudolikelihood function with respect to w_q is given by:

$$\frac{\partial logP^*(\mathbf{y}|\mathbf{x})}{\partial w_q} = \sum_{i=1}^{n} \mathbb{E}_{y_i|MB}\Big[\sum_{j\in g_q : i\in\phi_{qj}} \phi_{qj}(\mathbf{x},\mathbf{y})\Big] - \Phi_q(\mathbf{x},\mathbf{y}) \tag{6}$$

where $i \in \phi_{qj}$ implies that variable y_i participates in the potential ϕ_{qj}. Using a Monte Carlo approach this derivative can be computed in linear time in the size of $\mathbf{y}$.

Large-Margin Estimation (LME) A different approach to weight learning that focuses on maximizing the MAP state rather than producing accurate probabilistic models. This approach uses the intuition that the ground-truth state $\mathbf{y}$ should have energy lower than any alternate state $\tilde{\mathbf{y}}$ by a large margin defined by a loss function L. The objective function to find the optimal set of weights is given by:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2 + C\xi \tag{7}$$
$$s.t: \mathbf{w}^T(\Phi(\tilde{\mathbf{y}},\mathbf{x}) - \Phi(\mathbf{y},\mathbf{x})) \leq -L(\mathbf{y},\tilde{\mathbf{y}}) + \xi$$

where L is a loss function such as the L1 distance between $\mathbf{y}$ and $\tilde{\mathbf{y}}$, and ξ is a slack variable. Equation 7 is then solved by performing a large-margin estimation based on a cutting-plane approach for structural support vector machines (Joachims, Finley, and Yu 2009).

2.2 Black-box optimization

Black-box optimization is a well studied technique, especially in the context of hyperparameter tuning using BO (Shahriari et al. 2016).

Definition 2 (Black-box optimization)**.** *Given a black-box function $\gamma(\tilde{\mathbf{x}}) : \mathbb{R}^d \rightarrow \mathbb{R}$, where d is the input dimension, the task of finding the optimal value for $\gamma(\tilde{\mathbf{x}})$ in a predefined amount of time is called* black-box optimization*.*

The goal of black-box optimization using BO is to find the best possible value for the function $\gamma(\tilde{\mathbf{x}})$ in a sequential setup within a predefined number of epochs. Various strategies have been proposed to choose the next point to evaluate given the previous evaluations (Srinivas et al. 2010; Kushner 1964; Mockus 1977; Thompson 1933). Each strategy is encoded through an acquisition function α. The objective of these strategies is to minimize the number of epochs required to find the best solution. A simple black-box Bayesian approach iteratively obtains a point to explore from the acquisition function α using the prior distribution; then the function γ is evaluated to obtain a new outcome at that point which is then used to update the posterior. Gaussian process regression (GPR) is a non-parametric Bayesian approach which is effective in performing black-box optimization in a BO framework.

2.3 Gaussian Process Regression

A Gaussian process (GP) is fully characterized by its mean function μ_0 and either a positive definite covariance matrix $\mathbf{K}$ or a kernel function k. Consider a finite set of s inputs $\tilde{\mathbf{X}} = \tilde{\mathbf{x}}_{1:s}$ and a random variable $g_i = \gamma(\tilde{\mathbf{x}}_i)$ representing the function γ evaluated at $\tilde{x_i}$ and let $\tilde{y_i}$ be the noisy output of the function. In GP, we assume that $\mathbf{g} = g_{1:s}$ is jointly Gaussian and $\tilde{y_i}$ given $\mathbf{g}$ is Gaussian. The generative model is of the form: $\mathbf{g}|\tilde{\mathbf{X}} \sim \mathcal{N}(\mathbf{m},\mathbf{K})$, and $\tilde{\mathbf{y}}|\mathbf{g} \sim \mathcal{N}(\mathbf{g},\sigma^2\mathbf{I})$, where $m_i = \mu_0(\tilde{\mathbf{x}}_i)$, $\mathbf{K}$ is an $(s \times s)$ positive definite matrix such that $K_{i,j} = k(\tilde{\mathbf{x}}_i,\tilde{\mathbf{x}}_j)$. Since the distributions are Gaussian and using the kernalization trick (Rasmussen and Williams 2005), the posterior mean and variance given a set

of observed data can be written as:

$$\mu_s(\tilde{\mathbf{x}}_{s+1}) = \mu_0(\tilde{\mathbf{x}}_{s+1}) + k(\tilde{\mathbf{x}}_{s+1})^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}(y - m)$$
$$\sigma_s(\tilde{\mathbf{x}}_{s+1}) = k(\tilde{\mathbf{x}}_{s+1}, \tilde{\mathbf{x}}_{s+1}) - k(\tilde{\mathbf{x}}_{s+1})^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}k(\tilde{\mathbf{x}}_{s+1})$$

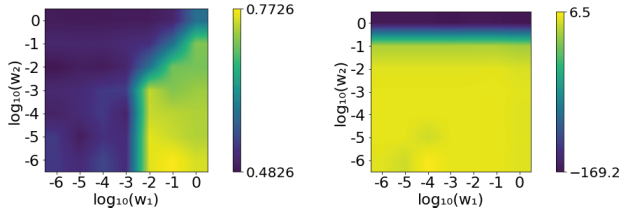
where $k(\tilde{\mathbf{x}}_{s+1})$ is a kernel function applied to the inputs with observed function evaluations and the new input; i.e., it represents the covariance between observed inputs and any unobserved input. Using the above expressions, the mean and variance for any point can be computed. There is a suite of kernel functions available in the literature (Rasmussen and Williams 2005). Note that the kernel function should be chosen based on the problem domain and it is often the key to finding the best approximation of the true function.

3 Bayesian Optimization for Weight Learning

As mentioned earlier, commonly used approaches for rule weight learning in PSL are based on maximizing a likelihood function. In this section, we first give a motivating example that highlights the issues with likelihood-based approaches. Next, we introduce our proposed algorithm, BOWL (Bayesian Optimization for Weight Learning), which uses GPR to perform weight learning[1]. We describe components of our approach, including the search space and acquisition function. Finally, we provide justification for our assumptions and prove the correctness of our approach.

3.1 Motivating Example

Consider our simple PSL program, Example 1, applied to a toy dataset with 100 users. Fig. 2 shows the performance of the model as we vary the rule weights logarithmically from 10^{-6} to 1.0. Fig. 2 (a) shows AUROC and Fig. 2 (b) shows the log-likelihood of the model. Lighter shades (yellow) represent a high value and darker shades (dark blue) represent a low value. We observe that the AUROC is maximized when the first rule's weight is 0.1 and the second rule's weight is 10^{-6}. However, the likelihood is not maximized at these weights. For this model and dataset, we observe that the likelihood is not well correlated with the AUROC.



(a) AUROC (b) log-likelihood

Figure 2: Heat map for accuracy and log-likelihood for the model in Example 1. The lighter color indicates higher values and higher values are desired for both metrics.

[1]Note that, in our method we loosely refer to this specific way of using GPR in BO framework as Bayesian optimization.

3.2 Problem definition

Consider a PSL model with r template rules where each rule $i \in \{1 \dots r\}$ is associated with a weight $w_i \in [0, 1]$. Grounding all the rules with data D yields a set of m observed random variables $\mathbf{x} = \{x_1, \dots, x_m\}$, n unobserved random variables $\mathbf{y} = \{y_1, \dots, y_n\}$, and ι potentials $\phi = \{\phi_1, \phi_2, ..., \phi_\iota\}$. The unobserved random variables $\mathbf{y}$ are inferred by optimizing Equation 3. Further, all unknown random variables are associated with corresponding ground truth $\mathbf{y}^* = \{y_1^*, \dots, y_n^*\}$ used to compute evaluation metrics. Let $\mathbf{w} = \{w_1, \dots, w_r\}$ be the vector representing the set of rule weights, i.e., the weight configuration. Next, let $\omega(\mathbf{y}, \mathbf{y}^*) : (\mathbf{y}, \mathbf{y}^*) \to \mathbb{R}$ be a problem-specific performance metric (e.g., accuracy, AUROC, or F-measure) and let $\gamma(\mathbf{w}) : \mathbf{w} \to \omega(\mathbf{y}, \mathbf{y}^*)$ be the same function ω parameterized by $\mathbf{w}$ that maps weights to the metric. Then the objective of weight learning can be expressed as finding the set of weights that maximize the function γ which represents the true metric function ω, i.e., $\text{argmax}_{\mathbf{w}}\, \gamma(\mathbf{w})$. The objective of GPR is to find an approximate function $g \approx \gamma$ by sampling t weight configurations from a set of possible weight configurations $\mathbf{W}$.

3.3 BOWL

A high-level sketch for BOWL is as follows: first, a weight configuration $\mathbf{w} \in \mathbf{W}$ is chosen using an acquisition function α. Next, inference is performed using the current weight configuration $\mathbf{w}$, and $\gamma(\mathbf{w})$ is computed. Then, GPR is updated with $\mathbf{w}$ and $\gamma(\mathbf{w})$. Finally, after t iterations the weight configuration that resulted in highest value for γ is returned. There are two primary components of BOWL that need to be defined: the kernel function used in GPR and the acquisition function α. These two components will determine the effectiveness of BOWL. Note that we restrict the weights to be $w_i \in [0, 1]$, whereas weights in PSL take values $w_i \in \mathbb{R}^+$. Later, in Section 4, we show that this restriction does not limit the capabilities of the model.

3.4 BOWL-OS

In order to use GPR and choose a kernel function, we must make an assumption about the function γ. Here, we assume that the function γ is smooth. This assumption is true if the problem is well-defined and the metric function ω being optimized is a smooth function, such as *mean square error* (MSE). For now, we make this assumption (justified further in Section 4), and choose the *squared exponential kernel* as the kernel for the GP:

$$k(\mathbf{w}_i, \mathbf{w}_j) = \tilde{\sigma} \cdot exp^{\{\frac{\delta_{i,j}}{2\rho^2}\}} \tag{8}$$

where $\tilde{\sigma}$ is the amplitude, ρ is the *characteristic length-scale*, and δ is the distance between any two weight configurations. ρ and σ are the kernel hyperparameters. The scaling factor ρ affects the smoothness of the approximation (a large value implies more smooth) and the number of iterations required to explore the space. We choose ρ such that a reasonable exploration of the space is possible in t iterations. The value of $\tilde{\sigma}$ is chosen based on the range of the metric being learned.

The distance function δ is crucial in determining the covariance between two weight configurations. Ideally, if the distance between two weight configurations is zero then the output of the function γ should be the same. And, as the distance between the two weight configurations increases, the correlation between the output of the γ function should go to zero. We refer to the weights in the $[0,1]^r$ space as the *original space* (OS), and define the distance function δ as follows:

$$\delta_{i,j} = ||\mathbf{w}_i - \mathbf{w}_j||_2^2 \tag{9}$$

We refer to the GPR which uses the above distance function as BOWL-OS .

3.5 Correlated Configurations

In PSL, the rule weights correspond to their relative importance, and multiple weight configurations with the same relative importance result in the same solution for $\mathbf{y}$. This in turn produces the same value for the γ function. This means that if we are not careful, the behavior of the function γ may not correlate with the distance function defined in Equation 9, i.e., two weight configurations $\mathbf{w}_1$ and $\mathbf{w}_2$ might be far apart, $\delta_{1,2} \gg 0$, but perfectly correlate, $\gamma(\mathbf{w}_1) = \gamma(\mathbf{w}_2)$. This leads to inefficient exploration of the space and will likely result in a poor approximation of the function. To illustrate this issue consider the following example:

Example 2. *Consider a model with two rules* $\mathbf{w} = \{w_1, w_2\}$. *Let us assume three possible weight configurations for this problem:* $\mathbf{w}_1 = \{0.1, 0.1\}$, $\mathbf{w}_2 = \{1.0, 1.0\}$, *and* $\mathbf{w}_3 = \{0.1, 0.0001\}$. *Assuming that the number of groundings for both rules are the same, the weights of the rules in* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *indicate that both rules are equally important, while in* $\mathbf{w}_3$ *the first rule is* 1000 *times more important than the second rule. This results in the function* γ *producing the same output for* $\mathbf{w}_1$ *and* $\mathbf{w}_2$*, and potentially a different value for* $\mathbf{w}_3$. *Based on this, the weight configuration* $\mathbf{w}_3$ *should be significantly different from the weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$*, while* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *should be similar. Unfortunately, the distances measured using Equation 9,* $\delta_{1,2} = 1.27$, $\delta_{1,3} = 0.09$, *and* $\delta_{2,3} = 1.34$, *do not behave in this manner. The distance* $\delta_{1,2}$ *is much larger than distance* $\delta_{1,3}$. *Therefore,* BOWL-OS *would infer that the function value of* $\gamma(\mathbf{w}_1)$ *is more correlated with* $\gamma(\mathbf{w}_3)$ *than* $\gamma(\mathbf{w}_2)$. *However, as argued above, we want the opposite behavior. This discrepancy can lead to a poor approximation of the function* γ.

3.6 BOWL-SS

To match the actual correlation between the weight configurations and their corresponding distance, we define a new configuration space, the *scaled space* (SS). The SS is a projection of weights onto a relative space. We use the ratio of weights between the rules to define the relative importance of weights in the configuration. This projection results in distances that correspond to the actual correlation between the weight configurations in PSL. Formally, we define SS and the distance measured in SS as:

Definition 3. *Given a set of weights* $\mathbf{w} = \{w_1, \ldots, w_r\} \in (0,1]^r$, *the SS* $\mathcal{E}$ *is a projection defined on* $\mathbf{w}$ *such that* $\mathcal{E}(\mathbf{w}) \in \mathbb{R}^{(r-1)}$ *is given by:*

$$\mathcal{E}(\mathbf{w}) = \{\forall_{i=2}^{r}(ln(w_i) - ln(w_1))\} \tag{10}$$

and the distance Δ *between weights is defined as:*

$$\Delta_{i,j} = ||\mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)||_2^2 \tag{11}$$

In SS $\mathcal{E}$, a distance of $\Delta_{i,j} = 0$, implies that the two weight configurations yield the same solution for the random variables $\mathbf{y}$ at the time of inference.

Theorem 1. *Given two weight configurations* $\mathbf{w}_1$ *and* $\mathbf{w}_2$, *if* $\mathcal{E}(\mathbf{w}_1) = \mathcal{E}(\mathbf{w}_2)$ *(i.e.,* $\Delta_{1,2} = 0$*) then the solution obtained for* $\mathbf{y}$ *by minimizing Equation 1 with both the weight configurations are the same.*

Proof. Let $\mathbf{w}_1 = \{w_{1,1}, \ldots, w_{1,r}\}$, $\mathbf{w}_2 = \{w_{2,1}, \ldots, w_{2,r}\}$ and $\mathcal{E}(\mathbf{w}_1) = \mathcal{E}(\mathbf{w}_2)$. As the two weight configurations are the same in SS, the equality can be written as:

$$ln(\mathbf{w}_1) - ln(w_{1,1}) = ln(\mathbf{w}_2) - ln(w_{2,1})$$
$$\mathbf{w}_1 = \frac{w_{1,1}}{w_{2,1}}\mathbf{w}_2$$

Since $w_{1,1} \in (0,1]$ and $w_{2,1} \in (0,1]$ are constants, the resulting optimization problems are equivalent:

$$\begin{aligned}\operatorname*{argmin}_y \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_1) &= \operatorname*{argmin}_y \frac{w_{1,1}}{w_{2,1}}\mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_2) \\ &= \operatorname*{argmin}_y \mathbf{E}(\mathbf{y}|\mathbf{x}, \mathbf{w}_2)\end{aligned}$$

Therefore, if the distance between two weight configurations is 0 in SS, then the solutions of their corresponding PSL program by optimizing Equation 3 are the same. □

Theorem 1 proves the soundness of the scaled space. Note that in SS we use the weight of the first rule to compute the projection. This choice is arbitrary and can be switched to any rule without affecting the space.

Example 2. *(Continued) Consider our earlier example. The weights and the distances of our running example in the SS* $\mathcal{E}$ *using Equation 10 and 11 are:* $\mathcal{E}(\mathbf{w}_1) = \{0\}$, $\mathcal{E}(\mathbf{w}_2) = \{0\}$, $\mathcal{E}(\mathbf{w}_3) = \{6.907\}$, $\Delta_{1,2} = 0$, $\Delta_{1,3} = 47.7$, *and* $\Delta_{2,3} = 47.7$.

A drawback of SS is that it does not support a weight of zero for any rule in the model. This means that all rules in the configuration must participate in the model. However, in practice, we mitigate this by simply assuming a very small lower bound (e.g., $10^{-\varrho}$, where $\varrho \in Z^+$, $\varrho \gg 0$) and sample weights uniformly from the log space, i.e., $\mathbf{W} \sim exp\{Unif([ln(10^{-m}), ln(1.0)]^r)\}$.

3.7 The Effect of Varied Number of Groundings

Our discussion so far has made a very important simplifying assumption, that the number of groundings for each rule in the model is the same. However, the number of groundings produced by a rule has an impact on the inference of the random variables. The weight associated with each rule is

repeated for each ground instance of that rule. This leads to the weight of each rule having varied influence on the minimization of the energy function. For instance, if a model has two equally weighted rules, but one rule produces 10 times more groundings than the other, then that rule implicitly becomes 10 times more important in the model.

We modify BOWL to accommodate the number of groundings of the rules in the model. Consider a model with r rules and let $\beta = \{\beta_1, \ldots, \beta_r\}$ be the number of groundings for each of the r rules. We define a grounding factor κ for each rule. For rule z, the grounding factor $\kappa_z = \frac{\beta_z}{\max(\beta)}$, where $\boldsymbol{\kappa} = \{\kappa_1, \ldots, \kappa_r\}$ is the vector of grounding factors. Therefore, the true weight associated with the z^{th} rule is $\kappa_z \cdot w_z$ and the grounding adjusted weight configuration can be represented as an element-wise dot product between $\boldsymbol{\kappa}$ and $\mathbf{w}$, i.e., $\tilde{\mathbf{w}} = \boldsymbol{\kappa} \cdot \mathbf{w}$. The distance between two weight configurations i and j in OS can be re-written as $||\tilde{\mathbf{w}}_i - \tilde{\mathbf{w}}_j||_2^2$. Similarly the distance in SS can be re-written as $||\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j)||_2^2$. However, the scaling factor $\boldsymbol{\kappa}$ does not affect the distance in SS as $\boldsymbol{\kappa}$ is constant for both weight configurations and cancels when computing the distance.

Theorem 2. *Given, two weight configurations $\mathbf{w}_i$ and $\mathbf{w}_j$, a set of grounding factors of $\boldsymbol{\kappa}$, and grounding adjusted weight configurations $\tilde{\mathbf{w}}_i = \boldsymbol{\kappa} \cdot \mathbf{w}_i$ and $\tilde{\mathbf{w}}_j = \boldsymbol{\kappa} \cdot \mathbf{w}_j$, the distance measured between both $(\mathbf{w}_i, \mathbf{w}_j)$ and $(\tilde{\mathbf{w}}_i, \tilde{\mathbf{w}}_j)$ in SS are equal, i.e. $||\mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)||_2^2 = ||\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j)||_2^2$.*

Proof. To prove the above theorem we consider the difference between the weight configurations $\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j)$:

$$\begin{aligned}\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j) &= (ln(\boldsymbol{\kappa} \cdot \mathbf{w}_i) - ln(\kappa_1 \cdot w_{i,1})) - \\ &\quad (ln(\boldsymbol{\kappa} \cdot \mathbf{w}_j) - ln(\kappa_1 \cdot w_{j,1})) \\ &= (ln(\mathbf{w}_i) - ln(w_{i,1})) - \\ &\quad (ln(\mathbf{w}_j) - ln(w_{j,1})) \\ &= \mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)\end{aligned}$$

Since $\mathcal{E}(\tilde{\mathbf{w}}_i) - \mathcal{E}(\tilde{\mathbf{w}}_j) = \mathcal{E}(\mathbf{w}_i) - \mathcal{E}(\mathbf{w}_j)$, the distances are also equal. □

Theorem 2 shows that the distance measured between two weight configurations in SS is robust while considering the size of their groundings.

3.8 Acquisition Function

Another crucial component of our algorithm is the acquisition function α. The function α determines the next weight configuration on which to evaluate the function γ, i.e., $\mathbf{w}_{next} = \text{argmax}_{\mathbf{w} \in \mathbf{W}}\, \alpha(\mathbf{w})$. Since our approach approximates the function γ with g, we would like to choose points that allow us to learn the approximation g while also maximizing the metric γ. To achieve this, we consider four well studied acquisition functions in the context of BO.

Upper confidence bound (UCB) (Srinivas et al. 2010): is an optimistic policy with provable cumulative regret bounds. The acquisition function can be written as: $\alpha(\mathbf{W}) = \mu(\mathbf{w}) + \psi \cdot \sigma(\mathbf{w})$, where μ and σ are the mean and variance predicted by the GP and $\psi \geq 0$ is a hyperparameter set to achieve optimal regret bounds.

Thompson sampling (TS) (Thompson 1933): is an information-based policy that considers the posterior distribution over the weights $\mathbf{W}$. The acquisition function can be written as: $\alpha(\mathbf{W}) = \tilde{p}(\mathbf{w}); \quad \tilde{p}(\mathbf{w}) \sim \mathcal{N}(\mu(\mathbf{w}), \sigma(\mathbf{w}))$, where $\tilde{p}$ are samples obtained from the distribution computed at the point $\mathbf{w}$.

Probability of improvement (PI) (Kushner 1964): is an improvement-based policy that favors points that are likely to improve an incumbent target τ. The acquisition function can be written as: $\alpha(\mathbf{W}) = \mathbb{P}(\gamma(\mathbf{w}) > \tau) = \boldsymbol{\mathcal{F}}\Big(\frac{\mu(\mathbf{w}) - \tau}{\sigma(\mathbf{w})}\Big)$, where $\boldsymbol{\mathcal{F}}$ is the standard normal cumulative distribution function and τ is set adaptively to the current best observed value for γ.

Expected improvement (EI) (Mockus, Tiesis, and Zilinskas 1978): is an improvement-based policy similar to PI. But, instead of probability, it measures the expected amount of improvement. The acquisition function can be written as: $\alpha(\mathbf{W}) = \Big\{(\mu(\mathbf{w}) - \tau)\boldsymbol{\mathcal{F}}\Big(\frac{\mu(\mathbf{w}) - \tau}{\sigma(\mathbf{w})}\Big) + (\sigma(\mathbf{w}))\mathcal{F}\Big(\frac{\mu(\mathbf{w}) - \tau}{\sigma(\mathbf{w})}\Big)\Big\}$, where $\mathcal{F}$ is the probability density function of a standard normal distribution function.

4 Feasibility Analysis

When defining BOWL we made two assumptions. First, we restricted the weights to be in range $[0, 1]$ and second, we assumed the function γ to be smooth. Here, we justify both assumptions. For the first assumption, we show that any PSL program with weights in $\mathbb{R}^+$ can be mapped to weights in $[0, 1]$ without any change to the solution obtained by minimizing the function in Equation 1.

Theorem 3. *Consider any PSL program with r rules and weights $\mathbf{w} = \{w_i, \ldots, w_r\}, w_i \in \mathbb{R}^+$. There exists a mapping function $\zeta(\mathbf{w}) : \mathbb{R}^r \rightarrow [0, 1]^r$ which transforms the weights to $[0, 1]^r$ without modifying the solution of the inference problem $\mathbf{y}$ in PSL. The ζ function is given by:*

$$\zeta(\mathbf{w}) = \frac{\mathbf{w}}{\max(\mathbf{w})} \tag{12}$$

Proof. Similar to proof from Theorem 1. □

Since we make an assumption that the user-defined metric depends only on the random variables, the metric value obtained is unaffected.

With regard to the second assumption, we constrain ourselves to only those metrics $(\omega(\mathbf{y}, \mathbf{y}^*))$ that are smooth with respect to the random variables (such as MSE). Note, our assumption is that the function γ is smooth and γ is parametrized with $\mathbf{w}$ and not the random variables $\mathbf{y}$. Hence, it is non-trivial to prove smoothness in γ. With the above constraint on the possible metrics, we know that if a small change in $\mathbf{w}$ leads to a small change in $\mathbf{y}$, then the function γ is also smooth. We formally define smoothness of the function γ as follows:

Definition 4. *Given two sets of weights $\mathbf{w}_1$ and $\mathbf{w}_2$ for a PSL program with r rules and random variables $\mathbf{y}$, the function γ is considered to be smooth if $\Delta_{1,2} < \epsilon$ where $\epsilon \rightarrow 0$,*

then $||\mathbf{y}_1 - \mathbf{y}_2||_2 < \nu$ *where* $\nu \to 0$*,* $\mathbf{y}_1$ *and* $\mathbf{y}_2$ *are the random variables inferred using weights* $\mathbf{w}_1$ *and* $\mathbf{w}_2$ *respectively.*

This directly leads to the conditioning of the problem. If a problem is well-conditioned then our assumption about the smoothness of γ is precise. If the problem is ill-conditioned then this assumption fails to hold and the function learned in BOWL could be a poor approximation of γ.

Finally, in practice, it is inefficient to use GPR for high-dimensional problems (Wang et al. 2016); GPR works best when the number of dimensions is less than 50. This makes PSL weight learning an ideal use case for GPR, because typically PSL programs have just tens of rules and most often the number of rules does not exceed 50.

5 Empirical Evaluation

In this section, we evaluate BOWL on various realworld datasets. All our experiments were run on a machine with 16 cores and 64GB of memory. We investigate three research questions: [Q1] How does BOWL perform on realworld datasets compared to the existing methods? [Q2] Is BOWL-SS scalable? [Q3] Is BOWL-SS robust?

We selected five realworld datasets from different domains for which PSL models have promising results (Bach et al. 2017; Kouki et al. 2015).[2] Details are as follows:

Jester: contains 2,000 users and 100 jokes (Goldberg et al. 2001). The task is to predict user's preference to jokes.

LastFM: contains 1,892 users and 17,632 artists. The task is to recommend artists to users by predicting the ratings for user-artist pairs.

Citeseer: contains 2,708 scientific documents, seven categories, and 5,429 directed citations. The task is to assign a category to each document.

Cora: is similar to Citeseer dataset, but contains 3,312 documents, six categories and 4,591 directed citations.

Epinions: contains 2,000 users and 8,675 directed links which are positive and negative trust links between users. The task is to predict the trust relation.

5.1 Performance analysis

To address [Q1], we compare the performance of BOWL-SS, BOWL-OS, MLE, MPLE, and LME on several metrics. For each dataset, we use snowball sampling to create eight folds and perform cross validation. We perform a paired t-test (p-value ≤ 0.05) across methods. For BOWL-SS and BOWL-OS the maximum number of weight configurations to explore in order to approximate the user-defined metric function is set to $t = 50$. Although in our experiments, we observed that the best metric value is usually obtained at $t < 25$ (this is likely because the function we intend to learn has several flat regions). We also use a stopping criterion which terminates the exploration if the standard deviation at all sampled weight configurations is less than 0.5. MLE, MPLE, and LME were allowed to run for 100 iterations.

For each dataset, depending on the problem, we leverage the metric that has been used for the problem to measure its performance. Hence, we report *MSE* and *AUROC* for the Jester and LastFM datasets, categorical accuracy and *F1* for the Cora and the Citeseer datasets, and *AUROC* and *F1* for the Epinions dataset. We use UCB as our acquisition function with $\psi = 5$ to favor exploration. However, in Section 5.3 we show that similar performance can be obtained by using the other acquisition functions discussed in Section 3.8. The hyperparameters that we use for BOWL are: $\tilde{\sigma} = 1.0, \rho = 1$, and the mean function is a constant zero. We set the value of ρ to one after exploring different values in $[10^5, 10^{-5}]$, and we set $\tilde{\sigma}$ to 1.0 as our metrics are in the range $[-1, 1]$.

Table 1 shows the comparison between BOWL and other methods across the different datasets. In each row of the table, the best performing method and those which are not significantly different from the best performing method are shown in bold. We observe that BOWL-SS is the best performing method across all the datasets and metrics. For the Epinions dataset, we observe that there is no statistically significant difference in *F1* score between BOWL-SS, BOWL-OS, MLE, MPLE, and LME. However, it is interesting that for the same dataset when the evaluation metric is *AUROC*, MPLE produces significantly lower values. For the LastFM, Jester, and Citeseer datasets, we observe that BOWL-SS significantly outperforms MLE, MPLE, and LME. For the Cora dataset, MPLE and MLE performs similar to BOWL-SS, while LME produces poorer results when F1 score is under consideration. Finally, BOWL-OS performs similar to or better than MLE, MPLE, and LME. However, the BOWL-SS approach yields the best performing results across all datasets.

5.2 Scalability

In this section, we compare the runtimes of BOWL-SS, MLE, MPLE, and LME to measure the scalability of BOWL-SS and address [Q2]. The number of parameters to learn in PSL is equal to the number of rules in the model and the data size translates to the number of groundings generated by the model. In Fig. 3a, we show the number of groundings generated by each of the datasets.We also show the number of rules in each model. The Jester dataset produces the largest number of groundings ($\sim 1M$) using seven rules and the Epinions dataset produces the least number of groundings ($\sim 14K$) using the largest model (20 rules).

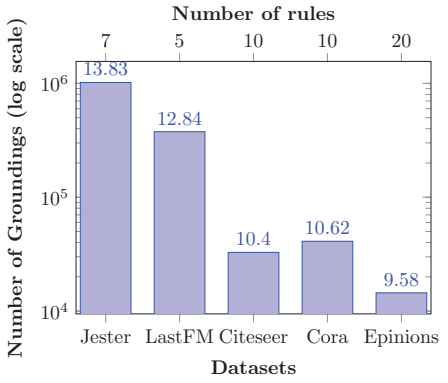
In Fig. 3b, we observe that the learning times of MLE, MPLE, and LME are not correlated to the size of the model, i.e., the number of rules in the model, but correlated to the number of groundings. MLE increases by about a factor of ~ 40 from Epinions to the Jester dataset and MPLE by a factor of ~ 60. For LME, runtime increases based on the number of groundings and complexity of finding the margin. Therefore, LME takes longer to finish on the LastFM dataset compared to the the Jester dataset. Next, we observe that the time taken to run BOWL-SS is almost the same for all five datasets with various sizes of rules and groundings. This is because the time taken to run BOWL depends mainly on the number of iterations it is allowed to run and the time it takes to solve MAP inference in PSL. Since the number of iterations is restricted to 50 for all models, this keeps the time almost the same for all datasets. Further, efficient infer-

[2] Models, code, and data: https://github.com/linqs/aaai-bowl.git

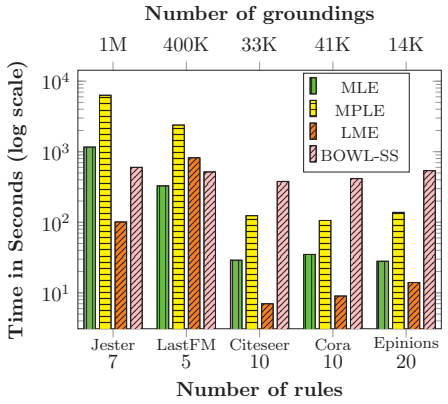
Table 1: Performance of methods across datasets; the best scoring methods (with $p < 0.05$) are shown in bold.

Method (Metric)	Jester (MSE)	Jester (AUROC)	LastFM (MSE)	LastFM (AUROC)	Citeeseer (Acc)	Citeeseer (F1)	Cora (Acc)	Cora (F1)	Epinions (AUROC)	Epinions (F1)
MLE	0.058	0.733	0.081	0.581	0.710	0.671	**0.832**	**0.869**	**0.815**	**0.960**
MPLE	0.060	0.737	0.083	0.568	0.729	0.754	**0.832**	**0.869**	0.744	**0.958**
LME	0.055	0.740	0.126	0.554	0.728	0.690	**0.831**	0.849	**0.826**	**0.960**
BOWL-OS	0.055	**0.767**	0.082	**0.599**	**0.740**	**0.796**	**0.832**	**0.869**	**0.812**	**0.962**
BOWL-SS	**0.053**	**0.767**	**0.078**	**0.599**	**0.743**	**0.798**	**0.833**	**0.868**	**0.825**	**0.960**

ence in PSL ensures small runtime increases for inference even on larger datasets. These results indicate that BOWL-SS can learn the best performing weights in the shortest time on large datasets.



(a) Groundings generated by different datasets.



(b) Time to learn vs. # of rules and groundings in datasets.

Figure 3: Analyzing scalability of BOWL-SS on number of rules and groundings. BOWL-SS is minimally affected by both the number of rules and groundings.

5.3 Robustness

To address [Q3], we run two sets of experiments: the first experiment is to test the effects of choosing an acquisition function on the performance of BOWL-SS and the second experiment is to check how robust BOWL-SS is w.r.t. different initialization. In Table 2 we compare the performance of BOWL-SS using four different acquisition functions (UCB, TS, PI, and EI) for all five datasets. We observe that using BOWL-SS with different acquisition functions yields similar results. This indicates that the performance of BOWL-SS is robust to these exploration strategies. For our second experiment, we perform weight learning with BOWL-SS using 100 random initializations and report the mean and standard deviation (std) of a metric per dataset in Table 2. Note that for this experiment we use UCB as our acquisition function. Further, we use only one (of the eight) folds per dataset as we intend to measure the variance introduced by different initializations. In Table 2, we also observe that the standard deviation is small for all datasets which indicates that BOWL-SS is robust to initialization.

Table 2: We observe that the performance of BOWL-SS is unaffected by both acquisition function and initialization.

Datasets	Different acquisition functions				Varied initializations	
	UCB	TS	PI	EI	Mean	Std
Jester (MSE)	0.053	0.052	0.053	0.053	0.052	0.001
LastFM (MSE)	0.078	0.078	0.078	0.078	0.079	0.001
Citeseer (F1)	0.797	0.797	0.797	0.798	0.804	0.001
Cora (F1)	0.868	0.869	0.866	0.869	0.876	0.002
Epinions (F1)	0.962	0.960	0.960	0.960	0.964	0.002

6 Conclusion and Future work

In this paper, we introduce BOWL, a BO approach to learn weights in PSL. BOWL yields improved performance across several metrics on a variety of different realworld datasets. There are many avenues for expanding our work. To perform weight learning using BOWL, the SRL model needs to be fully grounded. There are a variety of approaches for avoiding full grounding (Sarkhel, Singla, and Gogate 2015; Sarkhel et al. 2016) that would be interesting to integrate into our approach. Further, performance of GPs are highly dependent on the kernel function used. Therefore, an exploration of different kernels for BOWL could further improve the performance of our method.

7 Acknowledgements

This work was partially supported by the National Science Foundation grants CCF-1740850 and IIS-1703331, AFRL and the Defense Advanced Research Projects Agency. Golnoosh Farnadi is supported by postdoctoral scholarships from IVADO through the Canada First Research Excellence Fund (CFREF) grant.

References

Bach, S. H.; Huang, B.; London, B.; and Getoor, L. 2013. Hinge-loss Markov Random Fields: Convex Inference for Structured Prediction. In *UAI*.

Bach, S. H.; Broecheler, M.; Huang, B.; and Getoor, L. 2017. Hinge-Loss Markov Random Fields and Probabilistic Soft Logic. *JMLR* 18:109:1–109:67.

Boyd, S. P.; Parikh, N.; Chu, E.; Peleato, B.; and Eckstein, J. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*.

Brochu, E.; Brochu, T.; and de Freitas, N. 2010. A Bayesian Interactive Optimization Approach to Procedural Animation Design. In *SIGGRAPH*.

Chou, L.; Sarkhel, S.; Ruozzi, N.; and Gogate, V. 2016. On parameter tying by quantization. In *AAAI*.

Collins, M. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP*.

Das, M.; Wu, Y.; Khot, T.; Kersting, K.; and Natarajan, S. 2016. Scaling lifted probabilistic inference and learning via graph databases. In *SDM*.

De Raedt, L., and Kersting, K. 2011. *Statistical relational learning*. Springer US. 916–924.

Deng, L., and Wiebe, J. 2015. Joint prediction for entity/event-level sentiment analysis using probabilistic soft logic models. In *EMNLP*.

Farabi, K. M. A.; Sarkhel, S.; and Venugopal, D. 2018. Efficient weight learning in high-dimensional untied mlns. In *AISTATS*.

Farnadi, G.; Bach, S. H.; Moens, M.; Getoor, L.; and Cock, M. D. 2017. Soft quantification in statistical relational learning. *MLJ*.

Getoor, L., and Taskar, B. 2007. *Introduction to statistical relational learning*. The MIT Press.

Goldberg, K.; Roeder, T.; Gupta, D.; and Perkins, C. 2001. Eigentaste: A constant time collaborative filtering algorithm. *IR*.

Huynh, T. N., and Mooney, R. J. 2009. Max-Margin Weight Learning for Markov Logic Networks. In *KDD*.

Huynh, T. N., and Mooney, R. J. 2010. Online Max-margin Weight Learning with Markov Logic Networks. In *AAAI*.

Joachims, T.; Finley, T.; and Yu, C.-N. J. 2009. Cutting-plane training of structural svms. *MLJ* 77:27–59.

Kok, S., and Domingos, P. 2005. Learning the Structure of Markov Logic Networks. In *ICML*.

Kouki, P.; Fakhraei, S.; Foulds, J.; Eirinaki, M.; and Getoor, L. 2015. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. In *9th ACM Conference on Recommender Systems (RecSys)*. ACM.

Kouki, P.; Pujara, J.; Marcum, C.; Koehly, L. M.; and Getoor, L. 2017. Collective entity resolution in familial networks. In *ICDM*.

Kushner, H. J. 1964. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *JBE* 86(1):97–106.

Lizotte, D.; Wang, T.; Bowling, M.; and Schuurmans, D. 2007. Automatic gait optimization with gaussian process regression. In *IJCAI*.

Lowd, D., and Domingos, P. 2007. Efficient Weight Learning for Markov Logic Networks. In *KDD*.

Martinez-Cantin, R.; de Freitas, N.; Brochu, E.; Castellanos, J. A.; and Doucet, A. 2009. A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *AR* 27(2):93–103.

Mockus, J.; Tiesis, V.; and Zilinskas, A. 1978. The application of Bayesian methods for seeking the extremum. In *TGO*.

Mockus, J. 1977. On Bayesian Methods for Seeking the Extremum and their Application. In *IFIP Congress*.

Rasmussen, C. E., and Williams, C. K. I. 2005. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Richardson, M., and Domingos, P. M. 2006. Markov logic networks. *MLJ* 62(1-2):107–136.

Sarkhel, S.; Venugopal, D.; Pham, T. A.; Singla, P.; and Gogate, V. 2016. Scalable training of Markov logic networks using approximate counting. In *AAAI*.

Sarkhel, S.; Singla, P.; and Gogate, V. 2015. Fast lifted map inference via partitioning. In *NIPS*.

Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and de Freitas, N. 2016. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE* 104(1):148–175.

Singla, P., and Domingos, P. 2005. Discriminative Training of Markov Logic Networks. In *AAAI*.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS*.

Sridhar, D.; Fakhraei, S.; and Getoor, L. 2016. A probabilistic approach for collective similarity-based drug-drug interaction prediction. *Bioinformatics* 32(20):3175–3182.

Srinivas, N.; Krause, A.; Kakade, S.; and Seeger, M. 2010. Gaussian process optimization in the bandit setting: No regret and experimental design. In *ICML*.

Srinivas, N.; Krause, A.; Kakade, S. M.; and Seeger, M. W. 2012. Information-theoretic regret bounds for gaussian process optimization in the bandit setting. information theory. *IEEE Transactions on*.

Srinivasan, S.; Rao, N.; Subbian, K.; and Getoor, L. 2019. Identifying facet mismatches in search via micrographs. In *CIKM*.

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

Wang, Z.; Hutter, F.; Zoghi, M.; Matheson, D.; and De Freitas, N. 2016. Bayesian optimization in a billion dimensions via random embeddings. *JAIR* 55(1):361–387.