Bayesian Conditional Random Fields

Yuan (Alan) Qi
MIT CSAIL
32 Vassar Street
Cambridge, MA 02139
alanqi@csail.mit.edu

Martin Szummer
Microsoft Research
Cambridge, CB3 0FB
United Kingdom
szummer@microsoft.com

Thomas P. Minka
Microsoft Research
Cambridge, CB3 0FB
United Kingdom
minka@microsoft.com

Abstract

We propose Bayesian Conditional Random Fields (BCRFs) for classifying interdependent and structured data, such as sequences, images or webs. BCRFs are a Bayesian approach to training and inference with conditional random fields, which were previously trained by maximizing likelihood (ML) (Lafferty et al., 2001). Our framework eliminates the problem of overfitting, and offers the full advantages of a Bayesian treatment. Unlike the ML approach, we estimate the posterior distribution of the model parameters during training, and average over this posterior during inference. We apply an extension of EP method, the power EP method, to incorporate the partition function. For algorithmic stability and accuracy, we flatten the approximation structures to avoid two-level approximations. We demonstrate the superior prediction accuracy of BCRFs over conditional random fields trained with ML or MAP on synthetic and real datasets.

1 Introduction

Traditional classification models assume that data items are independent. However, real world data is often interdependent and has complex structure. Suppose we want to classify web pages into different categories, e.g., homepages of students versus faculty. The category of a web page is often related to the categories of pages linked to it. Rather than classifying pages independently, we should model them jointly to incorporate such contextual cues.

Joint modeling of structured data can be performed by generative graphical models, such as Bayesian networks or Markov random fields. For example, hidden Markov models have been used in natural language applications to assign labels to words in a sequence, where labels depend both on words and other labels along a chain. However, generative models have fundamental limitations. Firstly, generative models require specification of the data generation process, i.e., how data can be sampled from the model. In many applications, this process is unknown or impractical to write down, and not of interest for the classification task. Secondly, generative models typically assume conditional independence of observations given the labels. This independence assumption limits their modeling power and restricts what features can be extracted for classifying the observations. In particular, this assumption rules out features capturing long-range correlations, multiple scales, or other context.

Conditional random fields (CRF) are a conditional approach for classifying structured data, proposed by Lafferty et al. (2001). CRFs model only the label distribution conditioned on the observations. Unlike generative models, they do not need to explain the observations or features, and thereby conserve model capacity and reduce effort. This also allows CRFs to use flexible features such as complex functions of multiple observations. The modeling power of CRFs has shown great benefit in several applications, such as natural language parsing (Sha & Pereira, 2003), information extraction (McCallum, 2003), and image modeling (Kumar & Hebert, 2004).

To summarize, CRFs provide a compelling model for structured data. Consequently, there has been an intense search for effective training and inference algorithms. The first approaches maximized conditional likelihood (ML), either by generalized iterative scaling or by quasi-Newton methods (Lafferty et al., 2001; Sha & Pereira, 2003). However, the ML criterion is prone to overfitting the data, especially since CRFs are often trained with very large numbers of correlated features. The maximum a posteriori (MAP) criterion can reduce overfitting, but provides no guidance on the choice of parameter prior. Furthermore, large margin criteria have been applied to regularize the model parameters and also to kernelize CRFs (Taskar et al., 2004; Lafferty et al., 2004). Nevertheless, training and inference for CRFs remains a challenge, and the problems of overfitting, feature and model selection have largely remained open.

In this paper, we propose Bayesian Conditional Random Fields (BCRF), a novel Bayesian approach to training and inference for conditional random fields. Applying the Bayesian framework brings principled solutions and tools for addressing overfitting, model selection and many other aspects of the problem. Unlike ML, MAP, or large-margin approaches, we train BCRFs by estimating the posterior distribution of the model parameters. Subsequently, we can average over the posterior distribution for BCRF inference.

The complexity of the partition function in CRFs (the denominator of the likelihood function) necessitates approximations. Previous deterministic approximations including expectation propagation (EP) (Minka, 2001) or variational methods do not directly apply. In order to incorporate the partition function, we apply an extension of EP, the power EP method (Minka, 2004). Furthermore, we flatten the approximation structures for BCRFs to avoid two-level approximations. This significantly enhances the algorithmic stability and improves the estimation accuracy.

We first formally define CRFs, present the power EP method, and flatten the approximation structure for training. Then we propose an approximation method for model averaging, and finally show experimental results.

2 From conditional random fields to BCRFs

A conditional random field (CRF) models label variables according to an undirected graphical model conditioned on observed data (Lafferty et al., 2001). Let $\mathbf{x}$ be an "input" vector describing the observed data instance, and $\mathbf{t}$ be an "output" random vector over labels of the data components. We assume that all labels for the components belong to a finite label alphabet $\mathcal{T} = \{1, \ldots, T\}$. For example, the input $\mathbf{x}$ could be image features based on small patches, and $\mathbf{t}$ be labels denoting 'person, 'car' or 'other' patches. Formally, we have the following definition of CRFs (Lafferty et al., 2001):

Definition 2.1 *Let $G = (\mathcal{V}, \mathcal{E})$ be a graph such that $\mathbf{t}$ is indexed by the vertices of G. Then $(\mathbf{x}, \mathbf{t})$ is a conditional random field (CRF) if, when conditioned on $\mathbf{x}$, the random variables t_i obey the Markov property with respect to the graph: $p(t_i|\mathbf{x}, \mathbf{t}_{\mathcal{V}-i}) = p(t_i|\mathbf{x}, \mathbf{t}_{\mathcal{N}_i})$ where $\mathcal{V}-i$ is the set of all nodes in G except the node i, $\mathcal{N}_i$ is the set of neighbors of the node i in G, and $\mathbf{t}_\Omega$ represents the random variables of the vertices in the set Ω.*

Unlike traditional generative random fields, CRFs only model the conditional distribution $p(\mathbf{t}|\mathbf{x})$ and do not explicitly model the marginal $p(\mathbf{x})$. Note that the labels $\{t_i\}$ are globally conditioned on the whole observation $\mathbf{x}$ in CRFs. Thus, we do not assume that the observed data $\mathbf{x}$ are conditionally independent as in a generative random field.

BCRFs are a Bayesian approach to training and inference with conditional random fields. In some sense, BCRFs can be viewed as an extension of conditional Bayesian linear classifiers, e.g., Bayes point machines (BPM) (Herbrich et al., 1999; Minka, 2001), which are used to classify independent data points.

According to the Hammersley-Clifford theorem, a CRF defines the conditional distribution of the labels $\mathbf{t}$ given the observations $\mathbf{x}$ to be proportional to a product of potential functions on cliques of the graph G. For simplicity, we consider only pairwise clique potentials such that

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \frac{1}{Z(\mathbf{w})} \prod_{\{i,j\}\in\mathcal{E}} g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w}) \qquad (1)$$

where

$$Z(\mathbf{w}) = \sum_{\mathbf{t}} \prod_{\{i,j\}\in\mathcal{E}} g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w}) \qquad (2)$$

is a normalizing factor known as the partition function, $g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w})$ are pairwise potentials, and $\mathbf{w}$ are the model parameters. Note that the partition function is a complicated function of the model parameter $\mathbf{w}$. This makes Bayesian training much harder for CRFs than for Bayesian linear classifiers, since the normalizer of a Bayesian linear classifier is a constant.

In standard conditional random fields, the pairwise potentials are defined as

$$g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w}) = \exp(\mathbf{w}_{t_i,t_j}^{\mathrm{T}} \boldsymbol{\phi}_{i,j}(\mathbf{x}, t_i, t_j)) \qquad (3)$$

where $\boldsymbol{\phi}_{i,j}(\mathbf{x}, t_i, t_j)$ are features extracted for the edge between vertices i and j of the conditional random field, and $\mathbf{w}_{t_i,t_j}$ are elements corresponding to labels $\{t_i, t_j\}$ in $\mathbf{w}$, where $\mathbf{w} = [\mathbf{w}_{1,1}^{\mathrm{T}}, \mathbf{w}_{1,2}^{\mathrm{T}}, \ldots, \mathbf{w}_{T,T}^{\mathrm{T}}]^{\mathrm{T}}$. There are no restrictions on the relation between features.

Instead of using an exponential potential function, we prefer to use the probit function $\Psi(\cdot)$ (the cumulative distribution function of a Gaussian with mean 0 and variance 1). This function is bounded, unlike the exponential, and permits efficient Bayesian training. Furthermore, to incorporate robustness against labeling errors, we allow a small probability ϵ of a label being incorrect, thus bounding the potential away from 0. Specifically, our robust potentials are:

$$g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w}) = (1-\epsilon)\Psi(\mathbf{w}_{t_i,t_j}^{\mathrm{T}} \boldsymbol{\phi}_{i,j}(\mathbf{x}, t_i, t_j)) + \epsilon(1 - \Psi(\mathbf{w}_{t_i,t_j}^{\mathrm{T}} \boldsymbol{\phi}_{i,j}(\mathbf{x}, t_i, t_j))). \qquad (4)$$

Given the data likelihood and a Gaussian prior $p_0(\mathbf{w}) \sim \mathcal{N}(\mathbf{w}; \mathbf{0}, \mathrm{diag}(\boldsymbol{\alpha}))$, the posterior of the parameters is

$$p(\mathbf{w}|\mathbf{t}, \mathbf{x}) \propto \frac{1}{Z(\mathbf{w})} p_0(\mathbf{w}) \prod_{\{i,j\}\in\mathcal{E}} g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w}) \qquad (5)$$

Expectation Propagation exploits the fact that the posterior is a product of simple terms. If we approximate each of these terms well, we can get a good approximation of the posterior. Mathematically, EP approximates $p(\mathbf{w}|\mathbf{t},\mathbf{x})$ as

$$q(\mathbf{w}) = p_0(\mathbf{w})\frac{1}{\tilde{Z}(\mathbf{w})}\prod_{\{i,j\}\in\mathcal{E}}\tilde{g}_{i,j}(\mathbf{w}) \tag{6}$$

$$= \frac{1}{\tilde{Z}(\mathbf{w})}\tilde{R}(\mathbf{w}) \tag{7}$$

where $\tilde{R}(\mathbf{w}) = p_0(\mathbf{w})\prod_{\{i,j\}\in\mathcal{E}}\tilde{g}_{i,j}(\mathbf{w})$ is the numerator in $q(\mathbf{w})$. The approximation terms $\tilde{g}_{i,j}(\mathbf{w})$ and $\frac{1}{\tilde{Z}(\mathbf{w})}$ have the form of a Gaussian, so that the approximate posterior $q(\mathbf{w})$ is a Gaussian, i.e., $q(\mathbf{w}) \sim \mathcal{N}(\mathbf{m}_w, \mathbf{\Sigma}_w)$. We can approximate the pairwise potential functions $g_{i,j}(t_i, t_j, \mathbf{x}; \mathbf{w})$ by $\tilde{g}_{i,j}(\mathbf{w})$ in the numerator $\tilde{R}(\mathbf{w})$, just as in Bayesian linear classifiers (Minka, 2001). The main difficulty here is how to approximate the denominator $Z(\mathbf{w})$ by $\tilde{Z}(\mathbf{w})$ and incorporate it into $q(\mathbf{w})$.

3 EP and Power EP

This section reviews EP and presents power EP algorithms for BCRFs. Power EP is an extension of EP to make the computations more tractable. It was first used by Minka and Lafferty (2002), in the case of positive powers. However, power EP also works with negative powers, and this is one of the key insights that makes BCRF training tractable.

Given a distribution p written as a product of terms, and an approximating family q as in the previous section, Expectation Propagation tries to make q "close" to p in the sense of the Kullback Leibler divergence $KL(p||q) = \int p(\mathbf{w})\log(p(\mathbf{w})/q(\mathbf{w}))d\mathbf{w}$. This is done by minimizing the divergence with respect to each term individually, holding the other terms fixed. We repeatedly cycle through all the terms until a fixed point is reached.

For the g_k terms, where k indexes edges, the algorithm first computes $q^{\backslash k}(\mathbf{w})$, which represents the "rest of the distribution." Then it minimizes KL-divergence over $\tilde{g}_k$, holding $q^{\backslash k}$ fixed. This process can be written succinctly as follows:

$$q^{\backslash k}(\mathbf{w}) \propto q(\mathbf{w})/\tilde{g}_k(\mathbf{w}) \tag{8}$$

$$\tilde{g}_k(\mathbf{w})^{\text{new}} = \operatorname{argmin} KL(g_k(\mathbf{w})q^{\backslash k}(\mathbf{w}) \;||\; \tilde{g}_k(\mathbf{w})q^{\backslash k}(\mathbf{w})) \tag{9}$$

$$= \text{proj}\Big[g_k(\mathbf{w})q^{\backslash k}(\mathbf{w})\Big]/q^{\backslash k}(\mathbf{w}) \tag{10}$$

$$q(\mathbf{w})^{\text{new}} = q^{\backslash k}(\mathbf{w})\tilde{g}_k(\mathbf{w})^{\text{new}} \tag{11}$$

Here proj is a "moment matching" operator: it finds the Gaussian having the same moments as its argument, thus minimizing KL. Algorithmically, (8) means "divide the Gaussians to get a new Gaussian, and call it $q^{\backslash k}(\mathbf{w})$." Similarly, (9) means "construct a Gaussian whose moments match $g_k(\mathbf{w})q^{\backslash k}(\mathbf{w})$ and divide it by $q^{\backslash k}(\mathbf{w})$, to get a new Gaussian which replaces $\tilde{g}_k(\mathbf{w})$." This is the basic EP algorithm.

Power EP introduces a power n_k into EP and modifies the updates as follows:

$$q^{\backslash k}(\mathbf{w}) \propto q(\mathbf{w})/\tilde{g}_k(\mathbf{w})^{1/n_k} \tag{12}$$

$$\tilde{g}_k(\mathbf{w})^{\text{new}} = \Big(\text{proj}\Big[g_k(\mathbf{w})^{1/n_k}q^{\backslash k}(\mathbf{w})\Big]/q^{\backslash k}(\mathbf{w})\Big)^{n_k} \tag{13}$$

$$q(\mathbf{w})^{\text{new}} = q(\mathbf{w})\frac{\tilde{g}_k(\mathbf{w})^{\text{new}}}{\tilde{g}_k(\mathbf{w})} \tag{14}$$

As shown by Minka (2004), this update seeks to minimize a different measure of divergence, the α-divergence, where $\alpha = 2/n_k - 1$. Note that α can be any real number. By picking the power n_k appropriately, the updates can be greatly simplified. (This result is originally due to Wiegerinck and Heskes (2002). They discussed an algorithm called "fractional belief propagation" which is a special case of power EP, and all of their results also apply to power EP.)

We will use this update for the denominator term, so that g_k becomes $1/Z$, $\tilde{g}_k$ becomes $1/\tilde{Z}$, and pick $n_k = -1$:

$$q^{\backslash z}(\mathbf{w}) \propto q(\mathbf{w})/\tilde{Z}(\mathbf{w}) = \tilde{R}(\mathbf{w})/\tilde{Z}(\mathbf{w})^2 \tag{15}$$

$$\tilde{Z}(\mathbf{w})^{\text{new}} = \text{proj}\Big[Z(\mathbf{w})q^{\backslash z}(\mathbf{w})\Big]/q^{\backslash z}(\mathbf{w}) \tag{16}$$

$$q(\mathbf{w})^{\text{new}} = q(\mathbf{w})\frac{\tilde{Z}(\mathbf{w})}{\tilde{Z}(\mathbf{w})^{\text{new}}} \tag{17}$$

In this way, we only need the moments of $Z(\mathbf{w})$, not $1/Z(\mathbf{w})$.

4 Approximating the partition function

In the moment matching step, we need to approximate the moments of $Z(\mathbf{w})q^{\backslash z}(\mathbf{w})$. Murray and Ghahramani (2004) have proposed approximate MCMC methods to approximate the partition function of an undirected graph. This section presents an alternative method.

For clarity, let us rewrite the partition function as follows:

$$Z(\mathbf{w}) = \sum_{\mathbf{t}} Z(\mathbf{w}, \mathbf{t}) \tag{18}$$

$$Z(\mathbf{w}, \mathbf{t}) = \prod_{k\in\mathcal{E}} g_k(t_i, t_j, \mathbf{x}; \mathbf{w}) \tag{19}$$

where $k = \{i, j\}$ indexes edges. To compute the moments of $\mathbf{w}$, we can use EP recursively, to approximate $Z(\mathbf{w}, \mathbf{t})q^{\backslash z}(\mathbf{w})$ as a function of $\mathbf{w}$ and $\mathbf{t}$. The approximation will have a factorized form:

$$q(\mathbf{w})q(\mathbf{t}) = \tilde{Z}(\mathbf{w})\tilde{Z}(\mathbf{t})q^{\backslash z}(\mathbf{w}) \tag{20}$$

$$\tilde{Z}(\mathbf{w})\tilde{Z}(\mathbf{t}) = \prod_{k\in\mathcal{E}} \tilde{f}_k(\mathbf{w})\tilde{f}_k(t_i)\tilde{f}_k(t_j) \tag{21}$$

where $\tilde{f}_k(\mathbf{w})\tilde{f}_k(t_i)\tilde{f}_k(t_j)$ approximates $g_k(t_i, t_j, \mathbf{x}; \mathbf{w})$ in the denominator. Note that this $q(\mathbf{w})$ is the same as the overall $q(\mathbf{w})$ at the convergence.

Because the approximation is factorized, the computation will have the flavor of loopy belief propagation. The initial $\tilde{f}_k$ will be 1, making the initial $q(\mathbf{w}) = q^{\backslash z}(\mathbf{w})$ and $q(\mathbf{t}) = 1$. The EP update for $\tilde{f}_k$ is:

$$q^{\backslash k}(\mathbf{w}) \propto q(\mathbf{w})/\tilde{f}_k(\mathbf{w}) \tag{22}$$

$$q^{\backslash k}(t_i) \propto q(t_i)/\tilde{f}_k(t_i) \qquad \text{(similarly for } j\text{)} \tag{23}$$

$$f_k(\mathbf{w}) = \sum_{t_i,t_j} g_k(t_i, t_j, \mathbf{x}; \mathbf{w}) q^{\backslash k}(t_i) q^{\backslash k}(t_j) \tag{24}$$

$$\tilde{f}_k(\mathbf{w})^{\text{new}} = \text{proj}\Big[q^{\backslash k}(\mathbf{w}) f_k(\mathbf{w})\Big] / q^{\backslash k}(\mathbf{w}) \tag{25}$$

$$\tilde{f}_k(t_i)^{\text{new}} = \sum_{t_j} \int_{\mathbf{w}} g_k(t_i, t_j, \mathbf{x}; \mathbf{w}) q^{\backslash k}(\mathbf{w}) q^{\backslash k}(t_j) d\mathbf{w} \tag{26}$$

$$q(\mathbf{w})^{\text{new}} = q^{\backslash k}(\mathbf{w}) \tilde{f}_k(\mathbf{w})^{\text{new}} \tag{27}$$

$$q(t_i)^{\text{new}} = q^{\backslash k}(t_i) \tilde{f}_k(t_i)^{\text{new}} \tag{28}$$

These updates are iterated for all k, until a fixed point is reached. A straightforward implementation of the above updates costs $O(d^3)$ time, where d is the dimension of the parameter vector $\mathbf{w}$, since they involve inverting the covariance matrix of $\mathbf{w}$. However, as shown in the next section, it is possible to do them with low-rank matrix updates, in $O(d^2)$ time.

5 Efficient low-rank matrix computation for moments

First, let us define $\boldsymbol{\phi}_k(m, n, \mathbf{x})$ as shorthand of $\boldsymbol{\phi}_k(t_i = m, t_j = n, \mathbf{x})$, where $\boldsymbol{\phi}_k(t_i, t_j, \mathbf{x})$ are feature vectors extracted at edge $k = \{i, j\}$ with labels t_i and t_j on nodes i and j, respectively. Then we have

$$\mathbf{A}_k = \begin{pmatrix} \boldsymbol{\phi}_k(1,1,\mathbf{x}) & 0 & \cdots & 0 \\ 0 & \boldsymbol{\phi}_k(1,2,\mathbf{x}) & 0 & \cdots \\ 0 & \cdots & 0 & \boldsymbol{\phi}_k(T,T,\mathbf{x}) \end{pmatrix}$$

$$\mathbf{y} = \mathbf{A}_k^{\mathrm{T}} \mathbf{w} \tag{29}$$

$$f_k(\mathbf{y}) = \sum_{t_i,t_j} \Psi(y_{t_i,t_j}) q^{\backslash k}(t_i) q^{\backslash k}(t_j) \tag{30}$$

where $y_{t_i,t_j} = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}_k(t_i, t_j, \mathbf{x})$. Clearly, we can rewrite $q(\mathbf{w})$ as $f_k(\mathbf{y}) q^{\backslash k}(\mathbf{w})$. Since the dimensionality of $\mathbf{y}$ is usually a lot smaller than that of $\mathbf{w}$, the exact term $f_k(\mathbf{y})$ only constrains the distribution $q(\mathbf{w})$ in a smaller subspace. Therefore, it is sensible to use low-rank matrix computation to obtain $\mathbf{m}_w$ and $\mathbf{V}_w$, the mean and variance of $q(\mathbf{w})$.

The derivation is omitted because of the space limitation. The details can be found in Qi (2004). Here, we simply give the updates:

$$\mathbf{m}_w = \mathbf{m}_w^{\backslash k} + \mathbf{V}_w^{\backslash k} \mathbf{A}_k \mathbf{c} \tag{31}$$

$$\mathbf{V}_w = \mathbf{V}_w^{\backslash k} - \mathbf{V}_w^{\backslash k} \mathbf{A}_k \mathbf{D} \mathbf{A}_k^{\mathrm{T}} \mathbf{V}_w^{\backslash k} \tag{32}$$

$$\mathbf{c} = (\mathbf{V}_y^{\backslash k})^{-1} (\mathbf{m}_y - \mathbf{m}_y^{\backslash k}) \tag{33}$$

$$\mathbf{D} = (\mathbf{V}_y^{\backslash k})^{-1} - (\mathbf{V}_y^{\backslash k})^{-1} (\mathbf{G}_y - \mathbf{m}_y \mathbf{m}_y^{\mathrm{T}}) (\mathbf{V}_y^{\backslash k})^{-1} \tag{34}$$

where

$$Z = \sum_{t_i,t_j} q^{\backslash k}(t_i, t_j) \int \Psi(y_{t_i,t_j}) \mathcal{N}(\mathbf{y} | \mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k}) \mathrm{d}\mathbf{y} \tag{35}$$

$$= \sum_{t_i,t_j} q^{\backslash k}(t_i) q^{\backslash k}(t_j) Z_{t_i,t_j} \tag{36}$$

$$\mathbf{m}_y = \frac{\int f_k(\mathbf{y}) \mathbf{y} \mathcal{N}(\mathbf{y} | \mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})}{Z} \tag{37}$$

$$= \frac{\sum_{t_i,t_j} Z_{t_i,t_j} \mathbf{m}_{y_{t_i,t_j}} q^{\backslash k}(t_i) q^{\backslash k}(t_j)}{Z} \tag{38}$$

$$\mathbf{G}_y = \frac{\int f_k(\mathbf{y}) \mathbf{y}\mathbf{y}^{\mathrm{T}} \mathcal{N}(\mathbf{y} | \mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k})}{Z} \tag{39}$$

$$= \frac{\sum_{t_i,t_j} Z_{t_i,t_j} \mathbf{G}_{y_{t_i,t_j}} q^{\backslash k}(t_i) q^{\backslash k}(t_j)}{Z} \tag{40}$$

where

$$z_{t_i,t_j} = \frac{\mathbf{e}_k \mathbf{m}_y^{\backslash k}}{\sqrt{\mathbf{e}_k \mathbf{V}_y^{\backslash k} \mathbf{e}_k^{\mathrm{T}} + 1}} \tag{41}$$

$$\rho_{t_i,t_j} = \frac{1}{\sqrt{\mathbf{e}_k \mathbf{V}_y^{\backslash k} \mathbf{e}_k^{\mathrm{T}} + 1}} \frac{(1 - 2\epsilon \mathcal{N}(z_{t_i,t_j} | 0, 1))}{\epsilon + (1 - 2\epsilon) \Psi(z_{t_i,t_j})} \tag{42}$$

$$Z_{t_i,t_j} = \int \Psi(y_{i,j}) \mathcal{N}(\mathbf{y} | \mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k}) \mathrm{d}\mathbf{y} \tag{43}$$

$$= \int \Psi(\mathbf{e}_k \mathbf{y}) \mathcal{N}(\mathbf{y} | \mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k}) \mathrm{d}\mathbf{y} \tag{44}$$

$$= \epsilon + (1 - 2\epsilon) \Psi(z_{t_i,t_j}) \tag{45}$$

$$\mathbf{m}_{y_{t_i,t_j}} = \frac{\int \Psi(\mathbf{e}_k \mathbf{y}) \mathbf{y} \mathcal{N}(\mathbf{y} | \mathbf{m}_y^{\backslash k}, \mathbf{V}_y^{\backslash k}) \mathrm{d}\mathbf{y}}{Z_{t_i,t_j}} \tag{46}$$

$$= \mathbf{m}_y^{\backslash k} + \mathbf{V}_y^{\backslash k} \rho_{t_i,t_j} \mathbf{e}_k^{\mathrm{T}} \tag{47}$$

$$\mathbf{G}_{y_{t_i,t_j}} = \mathbf{V}_y^{\backslash k} - \mathbf{V}_y^{\backslash k} \mathbf{e}_k^{\mathrm{T}} \Big(\frac{\rho_{t_i,t_j} (\mathbf{e}_k \mathbf{m}_{y_{t_i,t_j}} + \rho_{t_i,t_j})}{\mathbf{e}_k \mathbf{V}_y^{\backslash k} \mathbf{e}_k' + 1} \Big) \mathbf{e}_k \mathbf{V}_y^{\backslash k} \tag{48}$$

where $\mathbf{e}_k$ is a vector with all elements being zeros except its k^{th} element being one.

We update $q(t_i)$ and $q(t_j)$ as follows:

$$q(t_i, t_j) = \frac{Z_{t_i,t_j} q^{\backslash k}(t_i) q^{\backslash k}(t_j)}{Z} \tag{49}$$

$$q(t_i) = \sum_{t_j} q(t_i, t_j), \quad q(t_j) = \sum_{t_i} q(t_i, t_j) \tag{50}$$

6 Flattening the approximation structure

In practice, we found that the approximation method, presented in Sections 3 and 4, led to non-positive covariance matrices in training. In this section, we examine the reason for this problem and propose a method to fix it.

The approximation method has two levels of approximations, which are visualized at the top of Figure 1. At the upper level of the top graph, the approximation method iteratively refines the approximate posterior $q(\mathbf{w})$ based on the term approximation $\tilde{Z}(\mathbf{w})$; at the lower level, it iteratively refines $\tilde{Z}(\mathbf{w})$ by smaller approximation terms $\{\tilde{f}_k(\mathbf{w})\}$.

A naive implementation of the two-level approximation will always initialize the approximation terms $\{\tilde{f}_k(\mathbf{w})\}$ as 1, such that the iterations at the lower level start from scratch every time. Thus, removing the denominator $\tilde{Z}(\mathbf{w})$ in the upper level amounts to removing all the previous approximation terms $\{\tilde{f}_k(\mathbf{w})\}$ in the lower level. The naive implementation requires the "leave-one-out" approximation $q^{\setminus z}(\mathbf{w}) \propto \frac{q(\mathbf{w})}{\tilde{Z}(\mathbf{w})}$ to have a positive definite covariance matrix. Since this requirement is hard to meet, the training procedure often skips the whole denominator $Z(\mathbf{w})$ in the upper level. This skipping would dramatically decrease the approximation accuracy in practice.

A better idea is to keep the values of the approximation terms and initialize the approximation terms using the values obtained from the previous iterations. By doing so, we do not require the covariance of $q^{\setminus z}(\mathbf{w})$ to be positive definite anymore. Instead, we need that of $q^{\setminus k}(\mathbf{w})$ in equation (22) to be positive definite, which is easier to satisfy. Now, when the iterations in the lower level start, $q^{\setminus k}(\mathbf{w})$ usually has a positive definite covariance. However, after a few iterations, the covariance of $q^{\setminus k}(\mathbf{w})$ often becomes non-positive definite again. The underlying reason is that the partition function $Z(\mathbf{w})$ is a complicated function, which is difficult to be accurately approximated by EP.

To address the problem, we flatten the two-level approximation structure by expanding $\tilde{Z}(\mathbf{w})$ in the upper level. Now we focus on $q(\mathbf{w})$, which is of our interest in training, without directly approximating the difficult partition function $Z(\mathbf{w})$ in the intermediate step. The flattened structure is shown at the bottom of the Figure 1. Specifically, the approximate posterior $q(\mathbf{w})$ has the following form:

$$q(\mathbf{w}) \propto p_0(\mathbf{w}) \prod_{k \in \mathcal{E}} \tilde{g}_k(\mathbf{w}) \frac{1}{\prod_{k \in \mathcal{E}} \tilde{f}_k(\mathbf{w})} \tag{51}$$

Equation (51) uses the approximation term $\tilde{f}_k(\mathbf{w})$ for each edge rather than using $\tilde{Z}(\mathbf{w})$. It is also possible to interpret the flattened structure from the perspective of the two-level approximation structure. That is, each time we partially update $\tilde{Z}(\mathbf{w})$ based on only one small term approximation $\tilde{f}_k(\mathbf{w})$, and then refine $q(\mathbf{w})$ before updating $\tilde{Z}(\mathbf{w})$ again based on another small term approximation.

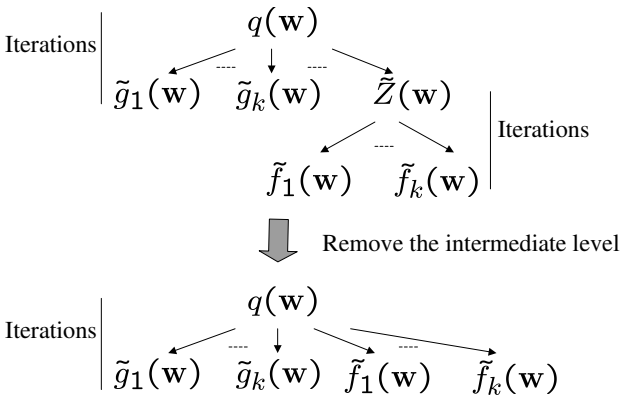

Figure 1: Flattening the approximation structure. The upper graph shows the two-level approximation structure of the methods described in the previous sections. The lower graph shows the flattened single-level approximation structure.

With the flattened structure, the deletion steps for removing $\tilde{g}_k(\mathbf{w})$ and $\tilde{f}_k(\mathbf{w})$ remain the same as before. The moment matching steps are the same too. We only need to assign negative power one to the approximation terms in the denominators. Specifically, we have the following updates:

$$\mathbf{h}_k = \left(\mathbf{D}^{-1} - \mathbf{A}_k^{\mathrm{T}} \mathbf{V}_w^{\setminus k} \mathbf{A}_k\right)^{-1} \tag{52}$$

$$\boldsymbol{\mu}_k = \mathbf{c} + \mathbf{h}_k \mathbf{A}_k^{\mathrm{T}} \mathbf{m}_w \tag{53}$$

$$\boldsymbol{\xi}_k = 2\mathbf{h}_k^{old} - \mathbf{h}_k \quad \boldsymbol{\eta}_k = 2\boldsymbol{\mu}_k^{old} - \boldsymbol{\mu}_k \tag{54}$$

$$\mathbf{V}_w = \mathbf{V}_w^{\setminus k} - \mathbf{V}_w^{\setminus k} \mathbf{A}_k (\boldsymbol{\xi}_k^{-1} + \mathbf{A}_k^{\mathrm{T}} \mathbf{V}_w^{\setminus k} \mathbf{A}_k)^{-1} \mathbf{A}_k^{\mathrm{T}} \mathbf{V}_w^{\setminus k} \tag{55}$$

$$\mathbf{m}_w = \mathbf{m}_w^{\setminus k} - \mathbf{V}_w^{\setminus k} \mathbf{A}_k (\boldsymbol{\xi}_k^{-1} + \mathbf{A}_k^{\mathrm{T}} \mathbf{V}_w^{\setminus k} \mathbf{A}_k)^{-1} \mathbf{A}_k^{\mathrm{T}} \mathbf{m}_w^{\setminus k} + \mathbf{V}_w \mathbf{A}_k \boldsymbol{\eta}_k \tag{56}$$

where $\mathbf{c}$ and $\mathbf{D}$ are defined in Equations (33) and (34). Note that the above computation takes $O(d^2)$ time, while a simple implementation of the two-level approximation would take $O(d^3)$ time due to the inverse of the covariance matrix of $\tilde{Z}(\mathbf{w})$.

As a result of structure flattening, we have the following advantages over the previous two approaches. First, in our experiments, training can converge easily. Instead of iteratively approximating $Z(\mathbf{w})$ in the lower level based all the small terms as before, a partial update based on only one small term makes training much more stable. Second, we can obtain a better "leave-one-out" posterior $q^{\setminus k}(\mathbf{w})$, since we update $q(\mathbf{w})$ more frequently than in the previous two cases. A more refined $q(\mathbf{w})$ leads to a better $q^{\setminus k}(\mathbf{w})$, which in turn guides the KL mininmization to find a better new $q(\mathbf{w})$. Finally, the flattened approximation structure allows us to process approximation terms in any order. We found empirically that, compared to using a random order, it is better to process the denominator term $\{\tilde{f}_k(\mathbf{w})\}$ right after processing the numerator term $\{\tilde{g}_k(\mathbf{w})\}$, which is associ-

ated with the same edge as $\{\tilde{f}_k(\mathbf{w})\}$.

Using the flattened structure and pairing the processing of the corresponding numerator and denominator terms, we can train BCRFs robustly. For example, on the tasks of analyzing synthetic datasets in the experimental section, training with the two-level structure breaks down by skipping $\{\tilde{Z}(\mathbf{w})\}$ or $\{\tilde{f}_k(\mathbf{w})\}$ and fails to converge. In contrast, training with the flattened structure converges successfully and leads to a test error around 10%.

7 Inference by approximate model averaging

Unlike traditional classification problems, where we have a scalar output for each input, a BCRF jointly labels all the hidden vertices in an undirected graph. The trained BCRF infer the lables by model averaging, which makes full use of the training data by employing not only the estimated mean of the parameters, but also the estimated uncertainty (variance).

Given a new graph $\mathbf{x}^\star$, a BCRF trained on $(\mathbf{x}, \mathbf{t})$ can approximate the predictive distribution as follows:

$$p(\mathbf{t}^\star|\mathbf{x}^\star, \mathbf{t}, \mathbf{x}) = \int p(\mathbf{t}^\star|\mathbf{x}^\star, \mathbf{w}) p(\mathbf{w}|\mathbf{t}, \mathbf{x}) d\mathbf{w} \tag{57}$$

$$\approx \int p(\mathbf{t}^\star|\mathbf{x}^\star, \mathbf{w}) q(\mathbf{w}) d\mathbf{w} \tag{58}$$

$$= \int \frac{q(\mathbf{w})}{Z(\mathbf{w})} \prod_{\{i,j\}\in\mathcal{E}} g_{i,j}(t_i^\star, t_j^\star, \mathbf{x}^\star; \mathbf{w}) d\mathbf{w}$$

where $q(\mathbf{w})$ is the approximation of the true posterior $p(\mathbf{w}|\mathbf{t}, \mathbf{x})$. Since the exact integration for model averaging is intractable, we need to approximate this integral.

We can approximate the predictive posterior term by term as in EP. But typical EP updates will involve the updates over both the parameters and the labels. That is much more expensive than using a point estimate for inference, which invloves only updates of the labels. To reduce the computational complexity, we propose the following approach. It is based on the simple fact that without any label, the test data point does not offer information about the parameters in the conditional model, since we do not couple BCRFs with semi-supervised learning. Since $q(\mathbf{w})$ is unchanged, we can only update $q(\mathbf{t}^\star)$ when incorporating one term $g_{i,j}(t_i^\star, t_j^\star, \mathbf{x}; \mathbf{w})$. Specifically, given the posterior $q(\mathbf{w}) \sim \mathcal{N}(\mathbf{m}_w, \mathbf{V}_w)$, we use the factorized approximation $q(\mathbf{t}^\star) = \prod_i q(t_i^\star)$ and update $q(t_i^\star)$ and $q(t_j^\star)$ as follows:

$$z_{t_i^\star, t_j^\star} = \frac{\boldsymbol{\phi}_k^{\mathrm{T}} \mathbf{m}_w^{\backslash k}}{\sqrt{\boldsymbol{\phi}_k^{\mathrm{T}} \mathbf{V}_w^{\backslash k} \boldsymbol{\phi}_k + 1}} \tag{59}$$

$$Z_{t_i^\star, t_j^\star} = \epsilon + (1 - 2\epsilon)\Psi(z_{t_i^\star, t_j^\star}) \tag{60}$$

$$q(t_i^\star, t_j^\star) = \frac{Z_{t_i^\star, t_j^\star} q^{\backslash k}(t_i^\star) q^{\backslash k}(t_j^\star)}{Z} \tag{61}$$

$$q(t_i^\star) = \sum_{t_j} q(t_i^\star, t_j^\star) \tag{62}$$

$$q(t_j^\star) = \sum_{t_i} q(t_i^\star, t_j^\star) \tag{63}$$

where $\boldsymbol{\phi}_k$ is the feature vector extracted at the k^{th} edge. Note that the deletion and inclusion steps for $q(\mathbf{t}^\star)$ are similar to those in BCRF training.

8 Experimental results

This section compares BCRFs with CRFs trained by maximum likelihood (ML) and maximum a posteriori (MAP) methods on several synthetic datasets and a document labeling task, demonstrating BCRFs' superior test performance. MAP-trained CRFs include CRFs with probit potential functions (4) and CRFs with exponential potential functions (3). We used probit models as potential functions by setting $\epsilon = 0$ in equation (4). In BCRF training, we used a small step size to avoid divergence (see details in Qi (2004)). For comparsion, the errors were counted on all vertices in the test graphs.

8.1 Synthetic CRFs classification

All the synthetic datasets were sampled from CRFs with probit potential functions (4). On these synthetic datasets, we compared the test performance of BCRFs and MAP-trained probit CRFs for different sizes of training sets and different graphical structures.

The labels of the vertices in synthetic graphs are all binary. The parameter vector $\mathbf{w}$ has 24 elements. The feature vectors $\{\boldsymbol{\phi}_{i,j}\}$ are randomly sampled from one of four Gaussians. We can easily control the discriminability of the data by changing the variance of the Gaussians. Based on the model parameter vector and the sampled feature vectors, we can compute the joint probability of the labels as in equation (1) and randomly sample the labels. For BCRFs, we used a step size of 0.8 for training. For MAP-trained CRFs, we used quasi-Newton methods with the BFGS approximation of Hessians (Sha & Pereira, 2003).

8.1.1 Different training sizes for loopy CRFs

Each graph has 3 vertices in a loop. In each trial, 10 loops were sampled for training and 1000 loops for testing. The procedure was repeated for 10 trials. A Gaussian prior with mean $\mathbf{0}$ and diagonal variance $5\mathbf{I}$ was used for both BCRF and MAP CRF training. For ML- and MAP-trained CRFs, we applied the junction tree algorithm for inference. For BCRFs, we used approximate model averaging for inference. We repeated the same experiments by increasing the number of training graphs from 10 to 30 to 100.

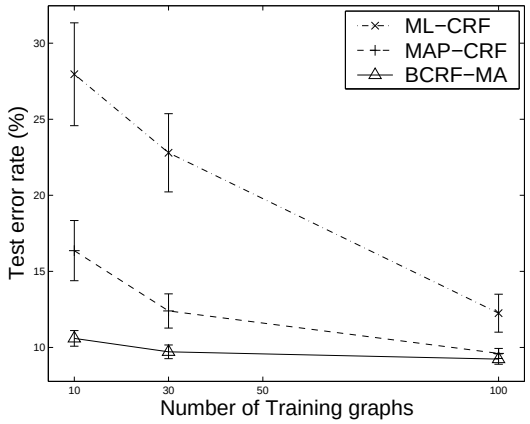

Figure 2: Test error rates for MAP-trained CRFs and BCRFs on synthetic datasets with different numbers of training loops. The results are averaged over 10 runs. Each run has 1000 test loops. Non-overlapping of error bars, the standard errors scaled by 1.64, indicates 95% significance of the performance difference.

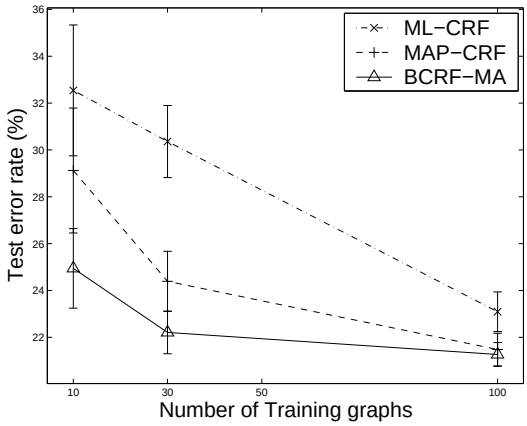

Figure 3: Test error rates for MAP-trained CRFs and BCRFs on synthetic datasets with different numbers of training chains. The results are averaged over 10 runs. Each run has 1000 test chains. Though the error bars overlap a little bit, BCRFs still outperform ML- and MAP-trained CRFs at 95% significance according to t-tests.

The results are visualized in Figure 2. According to t-tests, which have stronger test power than the error bars in Figure 2, BCRFs outperform ML- and MAP-trained CRFs in all cases at 98% statistical significance level. When more training graphs are available, MAP-trained CRFs and BCRFs perform increasingly similarly, though still statistically differently. The reason is that the posterior is narrower than in the case of fewer training graphs, such that the posterior mode is closer to the posterior mean.

8.1.2 Different training sizes for chain-structured CRFs

We then changed the graphs to be chain-structured. Specifically, each graph has 3 vertices in a chain. In each trial, 10 chains were sampled for training and 1000 chains for testing. The procedure was repeated for 10 trials. A Gaussian prior with mean $\mathbf{0}$ and diagonal variance $5\mathbf{I}$ was used for both BCRF and MAP CRF training. Then we repeated the same experiments by increasing the number of training graphs from 10 to 30 to 100. The results are visualized in Figure 3. Again, BCRFs outperform ML- and MAP-trained CRFs with high statistical significance.

8.2 FAQ labeling

We compared BCRFs with MAP-trained probit and exponential CRFs on the frequently asked questions (FAQ) dataset, introduced by McCallum et al. (2000). The dataset consists of 47 files, belonging to 7 Usenet newsgroup FAQs. Each files has multiple lines, which can be the header (H), a question (Q), an answer (A), or the tail (T). Since identifying the header and the tail is relatively easy, we simplify the task to label only the lines that are questions or answers. To save time, we truncated all the FAQ files such that no file has more than 500 lines. On average, the truncated files have 479 lines. The dataset was randomly split 10 times into 19 training and 28 test files. Each file was modeled by a chain-structured CRF, whose vertices correspond to lines in the files. The feature vector for each edge of a CRF is simply the concatenation of feature vectors extracted at the two neighboring vertices.

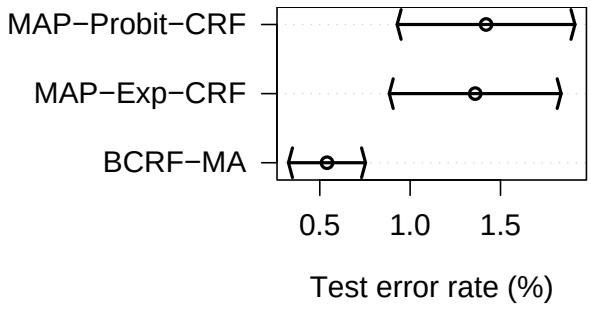

Figure 4: Test error rates of different algorithms on FAQ dataset. The results are averaged over 10 random splits. Non-overlapping of the error bars, the standard errors multiplied by 1.64, indicates that BCRFs outperform MAP-trained CRFs with probit and exponential potentials at 95% statistical significance level.

The test performance is visualized in Figure 4. According to t-tests, BCRFs outperform ML- and MAP-trained CRFs with probit or exponential potentials on the truncated FAQ dataset at 98% statistical significance level.

8.3 Comparing computational complexity

In general, the computational cost of ML and Bayesian training depends on many factors. On the one hand, for ML and MAP training, the cost of the BFGS algorithm is $O(d \max\{d, |\mathcal{E}|\})$ per iteration, where d is the length of $\mathbf{w}$, and $|\mathcal{E}|$ is the total number of edges in training graphs. The cost of BCRF training is $O(|\mathcal{E}|d^2)$ per iteration. Therefore BCRF training is about as $\min\{d, |\mathcal{E}|\}$ times expensive as ML and MAP training per iteration. On the other hand, BCRF training generally takes much fewer iterations to converge than BFGS training. Moreover, in BFGS training there is an embedded inference problem to obtain the needed statistics for optimization. This inference problem can be relatively expensive and cause a big hidden constant in $O(d \max\{d, |\mathcal{E}|\})$, the BFGS cost per iteration. On synthetic data where the number of edges is limited, BCRF training is at least as efficient as BFGS training. For example, given the 10 training sets in Section 8.1.2, each of which has 30 chain-structured CRFs, BCRF and BFGS training on a Pentium 4 3.1GHz computer used 8.81 and 21.16 seconds on the average, respectively. On real-world data, many factors play together to determine the efficiency of a training method. On a random split of the FAQ dataset where the total number of edges in graphs is more than 8000, it took about 9 and 2 hours (443 and 130 iterations) for BFGS to train probit and exponential CRFs, while it took BCRF training about 6 hours (30 iterations).

9 Conclusions

This paper has presented BCRFs, a new approach to training and inference on conditional random fields. In training, BCRFs approximate the posterior distribution of the parameters using a variant of the power EP method. Also, BCRFs flatten approximation structures to increase the algorithmic stability, efficiency, and prediction accuracy. In testing, BCRFs use approximate model averaging. On synthetic data and FAQ files, we compared BCRFs with ML- and MAP-trained CRFs. In almost all the experiments, BCRFs outperformed ML- and MAP-trained CRFs significantly.

Compared to ML- and MAP-trained CRFs, BCRFs can approximate model averaging over the posterior distribution of the parameters, instead of using a MAP or ML point estimate of the parameter vector for inference. Furthermore, BCRF hyperparameters can be optimized in a principled way, such as by maximizing the evidence, with parameters integrated out. EP returns an estimate of the evidence as a by-product. Similarly, we can use the method by Qi et al. (2004) to do feature selection with BCRFs and to obtain sparse kernelized BCRFs. More importantly, the techniques developed for BCRFs have promise for Bayesian learning in Markov networks.

Acknowledgment

Y. Qi was supported by the Things That Think consortium at the MIT Media laboratory before he moved to CSAIL. Thanks to Andrew McCallum for providing the features used in the FAQ dataset and Fei Sha for offering the maximum likelihood training code for CRFs with exponential potentials.

References

Herbrich, R., Graepel, T., & Campbell, C. (1999). Bayes point machine: Estimating the Bayes point in kernel space. *IJCAI Workshop SVMs* (pp. 23–27).

Kumar, S., & Hebert, M. (2004). Discriminative fields for modeling spatial dependencies in natural images. *Advances in Neural Information Processing Systems 16*.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. 18th International Conf. on Machine Learning* (pp. 282–289). Morgan Kaufmann, San Francisco, CA.

Lafferty, J., Zhu, X., & Liu, Y. (2004). Kernel conditional random fields: Representation and clique selection. *Proc. International Conf. on Machine Learning*.

McCallum, A. (2003). Efficiently inducing features of conditional random fields. *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence*.

McCallum, A., Freitag, D., & Pereira, F. (2000). Maximum entropy Markov models for information extraction and segmentation. *Proc. 17th International Conf. on Machine Learning* (pp. 591–598). Morgan Kaufmann, San Francisco, CA.

Minka, T. P. (2001). Expectation propagation for approximate Bayesian inference. *Uncertainty in AI'01*. `http://www.stat.cmu.edu/~minka/papers/ep/`.

Minka, T. P. (2004). Power EP. `http://www.stat.cmu.edu/~minka/`.

Minka, T. P., & Lafferty, J. (2002). Expectation propagation for the generative aspect model. *Proc UAI*.

Murray, I., & Ghahramani, Z. (2004). Bayesian learning in undirected graphical models: Approximate MCMC algorithms. *UAI*.

Qi, Y. (2004). *Extending expectation propagation for graphical models*. Doctoral dissertation, MIT.

Qi, Y., Minka, T. P., Picard, R. W., & Ghahramani, Z. (2004). Predictive automatic relevance determination by expectation propagation. *Proceedings of Twenty-first International Conference on Machine Learning*.

Sha, F., & Pereira, F. (2003). *Parsing with conditional random fields* (Technical Report MS-CIS-02-35). University of Pennsylvania.

Taskar, B., Guestrin, C., & Koller, D. (2004). Max-margin Markov networks. In S. Thrun, L. Saul and B. Schölkopf (Eds.), *Advances in neural information processing systems 16*.

Wiegerinck, W., & Heskes, T. (2002). Fractional belief propagation. *NIPS 15*.