A family of algorithms for approximate Bayesian inference

by

Thomas P Minka

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2001

Author .....
Department of Electrical Engineering and Computer Science
January 12, 2001

Certified by .....
Rosalind Picard
Associate Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A family of algorithms for approximate Bayesian inference

by

Thomas P Minka

Submitted to the Department of Electrical Engineering and Computer Science
on January 12, 2001, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

One of the major obstacles to using Bayesian methods for pattern recognition has been its computational expense. This thesis presents an approximation technique that can perform Bayesian inference faster and more accurately than previously possible. This method, "Expectation Propagation," unifies and generalizes two previous techniques: assumed-density filtering, an extension of the Kalman filter, and loopy belief propagation, an extension of belief propagation in Bayesian networks. The unification shows how both of these algorithms can be viewed as approximating the true posterior distribution with a simpler distribution, which is close in the sense of KL-divergence. Expectation Propagation exploits the best of both algorithms: the generality of assumed-density filtering and the accuracy of loopy belief propagation.

Loopy belief propagation, because it propagates exact belief states, is useful for limited types of belief networks, such as purely discrete networks. Expectation Propagation approximates the belief states with expectations, such as means and variances, giving it much wider scope. Expectation Propagation also extends belief propagation in the opposite direction—propagating richer belief states which incorporate correlations between variables.

This framework is demonstrated in a variety of statistical models using synthetic and real-world data. On Gaussian mixture problems, Expectation Propagation is found, for the same amount of computation, to be convincingly better than rival approximation techniques: Monte Carlo, Laplace's method, and variational Bayes. For pattern recognition, Expectation Propagation provides an algorithm for training Bayes Point Machine classifiers that is faster and more accurate than any previously known. The resulting classifiers outperform Support Vector Machines on several standard datasets, in addition to having a comparable training time. Expectation Propagation can also be used to choose an appropriate feature set for classification, via Bayesian model selection.

Thesis Supervisor: Rosalind Picard
Title: Associate Professor of Media Arts and Sciences

Acknowledgments

I wish to thank Rosalind Picard for giving me the freedom to satisfy my curiosity, not only in this project but many others before it. When I started as a graduate student six years ago, she said her job was to help me to get the most out of MIT. That she has done. For comments on drafts of this thesis, I thank Rosalind Picard, Yuan Qi, and my readers, Tommi Jaakkola and Trevor Darrell. This work was generously supported by members of the Things That Think and Digital Life Consortia at the MIT Media Lab.

I wish to thank Kris Popat for inducing me to go to graduate school in the first place and encouraging me to use probability theory for solving problems in machine learning. Thanks also to Andrew McCallum for hosting me at Justsystem Pittsburgh Research Center, where key groundwork for this thesis was performed.

It has been a pleasure to exchange ideas with everyone at the Vision and Modeling group at the MIT Media Lab, especially Martin Szummer, Flavia Sparacino, Ali Rahimi, and Yuan Qi. Thanks to Sumit Basu for providing his code for the billiard algorithm.

Contents

Chapter 1

Introduction

The dominant computational task in Bayesian inference is numerical integration. New methods for fast and accurate integration are therefore very important and can have great impact. This dissertation presents a new deterministic approximation framework, Expectation Propagation, which achieves higher accuracy than existing integration algorithms with similar computational cost.

The general scenario of Bayesian inference is that there is some observed data and some unobserved quantity of interest. Inferences about the unknown x are based on its posterior distribution given the observed D:

$$p(x|D) = \frac{p(x,D)}{p(D)} = \frac{p(x,D)}{\int_x p(x,D)dx} \tag{1.1}$$

For example, we may want to know the posterior mean and variance of the unknown, which are integrals over the posterior:

$$E[x|D] = \int_x xp(x|D)dx = \frac{\int_x xp(x,D)dx}{\int_x p(x,D)dx} \tag{1.2}$$

$$E[x^2|D] = \int_x x^2 p(x|D)dx = \frac{\int_x x^2 p(x,D)dx}{\int_x p(x,D)dx} \tag{1.3}$$

$$\mathrm{var}(x|D) = E[x^2|D] - E[x|D]^2 \tag{1.4}$$

In real situations, there are many unknowns, not just the one we are interested in. In Bayesian inference, these must be marginalized out of the joint distribution, which involves yet more integrals:

$$p(x|D) = \frac{\int_{y,z} p(x,y,z,D)}{\int_{x,y,z} p(x,y,z,D)} \tag{1.5}$$

Thus numerical integration goes hand in hand with practical Bayesian inference.

Numerical integration algorithms can be principally divided into deterministic vs. nondeterministic methods. Deterministic methods try to approximate the integrand with something whose integral is known exactly. They work from properties of the integrand like its maxima and curvature. Nondeterministic methods sample the integrand at random points to get a stochastic estimate of the integral. This approach is more general since it works for almost any integrand, but it also requires a great deal more computation than deterministic approximation.

The proposed method, Expectation Propagation, is a deterministic approximation method. It is an extension to *assumed-density filtering* (ADF), (Maybeck, 1982; Lauritzen, 1992; Bernardo & Giron, 1988; Stephens, 1997; Boyen & Koller, 1998b; Barber & Sollich, 1999; Opper & Winther, 1999; Frey et al., 2000) a one-pass, sequential method for computing an approximate posterior distribution. In ADF, observations are processed one by one, updating the posterior distribution which is then approximated before processing the next observation. For example, we might replace the exact one-step posterior with a Gaussian having the same mean and same variance (Maybeck, 1982; Lauritzen, 1992; Barber & Sollich, 1999; Opper & Winther, 1999). Or we might replace a posterior over many variables with one that renders the variables independent (Boyen & Koller, 1998b) or approximates them as Markovian (Frey et al., 2000). In each case, the approximate posterior is found by minimizing KL-divergence, which amounts to preserving a specific set of posterior *expectations*. Note that the statistical model in question need not be a time series model, and the processing order of observations need not correspond with time of arrival. The weakness of ADF stems from its sequential nature: information that is discarded early on may turn out to be important later. Even if ADF is augmented with a backward pass (Boyen & Koller, 1998a), this information cannot be recovered. ADF is also sensitive to observation ordering, which is undesirable in a batch context.

Expectation Propagation (EP) extends ADF to incorporate *iterative refinement* of the approximations, by making additional passes. The information from later observations refines the choices made earlier, so that the most important information is retained. When the refinement converges, the resulting posterior is independent of ordering and more accurate. Iterative refinement has previously been used in conjunction with sampling (Koller et al., 1999) and extended Kalman filtering (Shachter, 1990). Expectation Propagation differs by applying this idea to the *deterministic* approximation of *general* distributions, not just Gaussian distributions as in Shachter (1990). EP is more expensive than ADF by only a constant factor—the number of refinement passes (typically 4 or 5). As shown in chapter 3, the accuracy of EP is significantly better than ADF as well as rival approximation methods: Monte Carlo, Laplace's method, and variational Bayes.

Thinking in this framework has a number of benefits. For example, in belief networks with loops it is known that approximate marginal distributions can be obtained by iterating the belief propagation recursions, a process known as loopy belief propagation (Frey & MacKay, 1997; Murphy et al., 1999). As described in chapter 4, it turns out that this heuristic procedure is a special case of Expectation Propagation, where the approximate posterior is a completely disconnected network with no other constraints on functional form. In other words, loopy belief propagation is a direct generalization of the ADF algorithm of Boyen & Koller (1998b).

Expectation Propagation can therefore be seen as a way of generalizing loopy belief propagation—to less restrictive approximations that are not completely disconnected and to useful constraints on functional form such as multivariate Gaussian. Partially disconnected approximations are useful for improving accuracy, just as in variational methods (Jordan et al., 1999), while constraints on functional form are useful for reducing computation. Instead of propagating exact belief states, which may be intractable, EP only needs to propagate expectations relevant to the chosen approximating distribution (e.g. means and variances). Hence the name "Expectation Propagation."

Expectation Propagation also has connections to statistical physics techniques. Yedidia et al. (2000) have shown that belief propagation tries to minimize an approximate free energy on discrete networks. This is known as the TAP approach in statistical physics. Opper

& Winther (2000a) have generalized the TAP approach to networks with mixed continuous and discrete nodes via the cavity method, and applied it to Gaussian process classifiers (Opper & Winther, 2000c). Chapter 5 shows that Opper & Winther's algorithm, which has excellent performance, is a special case of Expectation Propagation where the posterior approximation is Gaussian. In this sense, Expectation Propagation as a generalization of belief propagation parallels the cavity method as a generalization of TAP.

Expectation Propagation, as a general framework for approximate Bayesian inference, can be applied to many real-world tasks. Chapter 5 demonstrates its use for the general task of discriminating objects into classes. Classification via Bayesian averaging has long been popular in the theoretical community, but difficult to implement in a computationally competitive way. An excellent example of this is the Bayes Point Machine (Rujan, 1997; Herbrich et al., 1999). With Expectation Propagation, the advantages of Bayesian averaging can be achieved at less expense than previously possible.

In short:

- Chapter 2 reviews the prior work in approximate Bayesian inference, excluding ADF.
- Chapter 3 reviews ADF and introduces Expectation Propagation. Both are illustrated on simple statistical models.
- Chapter 4 addresses belief networks, describing how loopy belief propagation is a special case of Expectation Propagation and how belief propagation may be extended.
- Chapter 5 applies Expectation Propagation to the Bayes Point Machine classifier.
- Chapter 6 summarizes the results and offers suggestions for future work on Expectation Propagation.

Chapter 2

Methods of numerical integration

This chapter describes prior work in approximate Bayesian inference. The outline is:

1. Numerical quadrature
2. Monte Carlo methods: importance sampling, Gibbs sampling
3. Laplace's method
4. Variational bound on the integral using Jensen's inequality
5. Variational bound on the integrand; variational Bayes
6. Statistical physics methods
7. Sequential filtering

The classical approach to numerical integration is quadrature (Davis & Rabinowitz, 1984). In this deterministic approach, the integrand is evaluated at several locations ('knots') and a function is constructed that interpolates these values. The interpolant is chosen from a simple family that can be integrated analytically, e.g. polynomials or splines. The integral of the interpolant approximates the desired integral, and with enough knots the approximation can be made arbitrarily accurate. By using a set of predefined knots $x_1, ..., x_n$, the interpolation and integration procedure can be compiled down to a simple summation of weighted function values:

$$I = \int_A f(\mathbf{x}) d\mathbf{x} \tag{2.1}$$

$$\hat{I} = \sum_{i=1}^{n} w_i f(\mathbf{x}_i) \tag{2.2}$$

where the weights w_i are known. This method is excellent with integrands that are simple, i.e. that are easy to interpolate. In one dimension, quadrature is nearly unbeatable. But in high dimensions it is infeasible, because of the vast number of knots required to get a good interpolant of a complex function. In Bayesian inference problems, the integrands are mostly zero except in small regions, i.e. they are *sparse*, which makes the situation even worse—most of the knots will be wasted. Newer deterministic techniques try to avoid these problems by exploiting more properties of the integrand than just its value at given points. Another approach is nondeterminism.

Nondeterministic, i.e. Monte Carlo, methods do not try to interpolate or otherwise approximate the integrand at all—they merely appeal to the law of large numbers. In the simplest approach, we sample knots uniformly in the region of integration A. The expected value $E[f(\mathbf{x})]$ under such a sampling must be the desired integral I divided by the area of A. Hence the estimate

$$\hat{I} = \frac{|A|}{n} \sum_{i=1}^{n} f(\mathbf{x}_i) \tag{2.3}$$

will converge to the true value, given enough samples, and this happens independent of dimensionality and independent of the complexity of f. Thus while Monte Carlo is rather inefficient in low dimensions, requiring thousands of knots when quadrature would only need around 10, it is often the only feasible method in high dimensions. In Bayesian inference problems, the sparsity of the integrand can be addressed by the technique of *importance sampling*. Instead of sampling uniformly in A, we sample from a proposal distribution $p(\mathbf{x})$ that matches the shape of $|f|$ as well as possible. Then the estimate

$$\hat{I} = \frac{1}{n} \sum_{i=1}^{n} \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)} \tag{2.4}$$

also converges to I, and much faster than (2.3) would. Importance sampling also has the advantage of working for infinite regions A. Various enhancements to the basic importance sampling procedure are possible (Ventura, 2000). With importance sampling, the difficulties of numerical integration are replaced by the difficulties of sampling from a complex distribution. Good proposal distributions may be hard to sample from. In this case, one can apply Markov Chain Monte Carlo methods (Neal, 1993; Liu, 1999), such as Metropolis sampling and Gibbs sampling. In these methods, we generate samples that are *approximately* from $p(\mathbf{x})$, and then apply (2.4) as before. For these methods, careful monitoring and restarting is required to ensure that the samples adequately represent $p(\mathbf{x})$. The Billiard algorithm for Bayes Point Machines, discussed in chapter 5, is a particularly clever Markov Chain Monte Carlo algorithm.

To learn about a function, we can compute its value at a large number of knots, but we could also compute a large number of derivatives at a single knot. This is the basic idea of Taylor expansion. By finite differences, a given number of knots translates into an equivalent number of derivatives, so theoretically this approach has no advantage over quadrature. But for Bayesian inference problems it has certain conceptual advantages. Since the integrand is sparse, it makes sense to focus on one area where the action is. Interpolation is also simpler since we just match derivatives. The most popular application of this idea in statistics is Laplace's method (Kass & Raftery, 1993), where we expand $\log(f)$ about its mode:

$$\log(f(\mathbf{x})) \approx \log(f(\hat{\mathbf{x}})) + \mathbf{g}^{\mathrm{T}}(\mathbf{x} - \hat{\mathbf{x}}) + \frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^{\mathrm{T}}\mathbf{A}(\mathbf{x} - \hat{\mathbf{x}}) \tag{2.5}$$

$$\mathbf{g} = \left(\frac{d \log(f(\mathbf{x}))}{d\mathbf{x}}\right)_{\mathbf{x}=\hat{\mathbf{x}}} \tag{2.6}$$

$$\mathbf{H} = \left.\frac{d^2 \log f(\mathbf{x})}{d\mathbf{x} d\mathbf{x}^{\mathrm{T}}}\right|_{\mathbf{x}=\hat{\mathbf{x}}} \tag{2.7}$$

Because $\hat{\mathbf{x}}$ is the mode, $\mathbf{g} = \mathbf{0}$ and we get

$$f(\mathbf{x}) \approx f(\hat{\mathbf{x}}) \exp(\frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^{\mathrm{T}}\mathbf{H}(\mathbf{x} - \hat{\mathbf{x}})) \tag{2.8}$$

$$I = \int_A f(\mathbf{x}) d\mathbf{x} \approx f(\hat{\mathbf{x}}) (2\pi)^{\mathrm{rows}(\mathbf{H})/2} |-\mathbf{H}|^{-1/2} \tag{2.9}$$

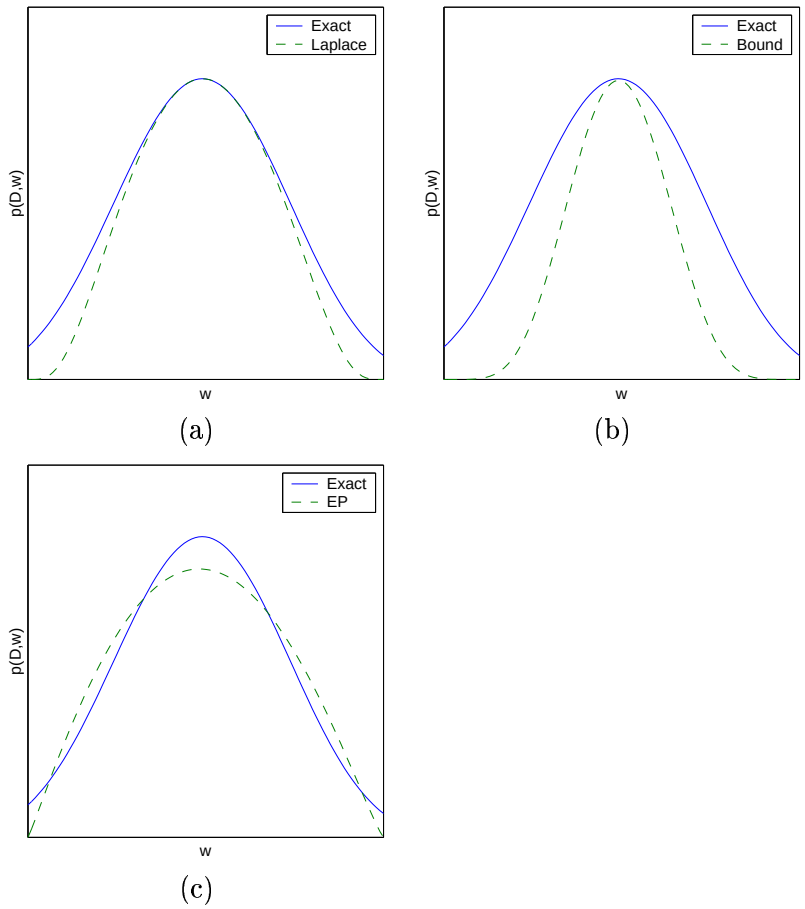


Figure 2-1: Deterministic methods for integration try to approximate the integrand. (a) Laplace's method uses a Gaussian that has the correct curvature at the mode. It produces approximations that are too local. (b) In one type of variational bound, the integrand is bounded everywhere. It produces approximations that are too inaccurate. (c) The proposed method, Expectation Propagation, tries to minimize KL-divergence, a global measure of deviation. It produces the most accurate integrals.

What Laplace's method does is approximate f by a scaled Gaussian density that matches the value, first derivative, and second derivatives of f at $\hat{\mathbf{x}}$. Figure 2-1 shows an example. The drawback of this method is that it is difficult to use higher order derivatives, resulting in an approximation that is limited in its accuracy and scope.

Variational bounding is a deterministic approach more global than Laplace's method. We start by introducing an arbitrary function $q(\mathbf{x})$:

$$I = \int_{\mathbf{x}} q(\mathbf{x}) \frac{f(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} \tag{2.10}$$

Jensen's inequality for convex functions says that (in the case of logarithm)

$$\log \int_{\mathbf{x}} q(\mathbf{x}) g(\mathbf{x}) d\mathbf{x} \geq \int_{\mathbf{x}} q(\mathbf{x}) \log g(\mathbf{x}) d\mathbf{x} \tag{2.11}$$

$$\text{if } \int_{\mathbf{x}} q(\mathbf{x}) d\mathbf{x} = 1 \tag{2.12}$$

Combining (2.10) and (2.11) gives the following bound on I:

$$I \geq \exp\left(\int_{\mathbf{x}} q(\mathbf{x}) \log \frac{f(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x}\right) \tag{2.13}$$

This of course requires that $f(\mathbf{x})$ is positive, a condition that is satisfied for most (but not all) integrals in Bayesian inference. We are free to choose $q(\mathbf{x})$ to get the tightest bound, which corresponds to maximizing the right hand side of (2.13). Note that this is equivalent to minimizing the (reversed) KL-divergence

$$D(q \mid\mid f) = \int_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{f(\mathbf{x})} d\mathbf{x} \tag{2.14}$$

over q subject to the constraint (2.12). If $q(\mathbf{x})$ is unconstrained, the maximum is achieved at $q(\mathbf{x}) = f(\mathbf{x})$ and the bound matches the original integral. To achieve a simplification in the integral, we must constrain $q(\mathbf{x})$ in some way. Typically, $q(\mathbf{x})$ is constrained to be Gaussian (Hinton & van Camp, 1993; Barber & Bishop, 1997; Seeger, 1999) but mixture distributions have also been suggested (Jaakkola & Jordan, 1999c). Having a lower bound is useful since it provides a firm guarantee about the true value of the integral—something none of the other methods can do. However it tends to be very computational and is feasible in a limited number of cases. Jensen's inequality takes advantage of the fact that $\log(f(\mathbf{x}))$ is often easy to integrate when $f(\mathbf{x})$ is not. But this is not always true. For example, in a mixture problem such as discussed in chapter 3, f is a product of sums, which does not simplify under a logarithm. And in chapter 5, the likelihood is a step function which reaches zero. The step function could be softened into a sigmoid to make the Jensen bound well-defined, but we could not expect a good fit to result.

A less accurate but simpler way to obtain a variational bound is to bound the integrand and then integrate the bound:

$$f(\mathbf{x}) \geq g(\mathbf{x}) \text{ for all } \mathbf{x} \tag{2.15}$$

$$I \geq \int_{\mathbf{x}} g(\mathbf{x}) d\mathbf{x} \tag{2.16}$$

Figure 2-1 shows an example. Unlike the previous variational method, this approach can be used in mixture problems. This approach was used explicitly by Jaakkola & Jordan (1999a; 1999b) and implicitly by Waterhouse et al. (1995); Attias (1999); Ghahramani & Beal (1999), under the name "variational Bayes." The implicit approach introduces hidden variables to define a bound. Start by writing $f(\mathbf{x})$ in terms of $h(\mathbf{x}, \mathbf{y})$:

$$f(\mathbf{x}) = \int_{\mathbf{y}} h(\mathbf{x}, \mathbf{y}) d\mathbf{y} \tag{2.17}$$

Apply the Jensen bound to get

$$I = \int_{\mathbf{x},\mathbf{y}} h(\mathbf{x}, \mathbf{y}) d\mathbf{y} d\mathbf{x} \tag{2.18}$$

$$\geq \exp\left(\int_{\mathbf{x},\mathbf{y}} q(\mathbf{x}, \mathbf{y}) \log \frac{h(\mathbf{x}, \mathbf{y})}{q(\mathbf{x}, \mathbf{y})} d\mathbf{y} d\mathbf{x}\right) \tag{2.19}$$

At this point, we constrain $q(\mathbf{x}, \mathbf{y})$ to factor into separate functions for $\mathbf{x}$ and for $\mathbf{y}$:

$$q(\mathbf{x}, \mathbf{y}) = q_x(\mathbf{x}) q_y(\mathbf{y}) \tag{2.20}$$

with no other constraints on functional form. The q_x and q_y functions are iteratively optimized to maximize the value of the bound. To see that this is equivalent to (2.16), note that for any q_y we can solve analytically for the optimal q_x, which is

$$q_x(\mathbf{x}) = \frac{g(\mathbf{x})}{\int_{\mathbf{x}} g(\mathbf{x}) d\mathbf{x}} \tag{2.21}$$

$$\text{where } g(\mathbf{x}) = \exp\left(\int_{\mathbf{y}} q_y(\mathbf{y}) \log \frac{h(\mathbf{x},\mathbf{y})}{q_y(\mathbf{y})} d\mathbf{y}\right) \tag{2.22}$$

When we substitute this q_x, the bound becomes

$$I \geq \int_{\mathbf{x}} g(\mathbf{x}) d\mathbf{x} \tag{2.23}$$

Regardless of which approach we use—implicit or explicit—we can optimize the bound via the EM algorithm. This is described in Minka (2000c).

Besides variational bounds, there are other integration techniques that are inspired by mean-field statistical physics. Most of these are extensions of TAP, such as the cavity method (Opper & Winther, 2000a), Bethe approximation (Yedidia, 2000), and Plefka expansion (Kappen & Wiegerinck, 2000). So far they have been applied to regular structures such as the Boltzmann machine and rarely to general probabilistic models, where it is less obvious how to apply them. An exception to this is the paper by Opper & Winther (2000c). The algorithm they derive is new and accurate, yet coincides with the framework proposed in this thesis. It will be interesting to see what other algorithms come out of this mean-field line of research.

For approximating an integral, we thus find ourselves in the following position. We have methods that work well for simple functions in low dimensions (quadrature) and complex functions in high dimensions (Monte Carlo). We have methods that are simple and fast but inaccurate (Laplace's method, variational Bayes). We have methods that apply in special cases (Jensen bound, TAP). What is missing is a general and accurate deterministic method in high dimensions, at least for simple functions. That is what this thesis provides.

The approach taken by this thesis continues the path started by the extended Kalman filter (EKF). The EKF is a sequential method developed for inference in dynamical systems with nonlinear dynamics. It is not a general method for integration. But there are several variations on the EKF (Maybeck, 1982) that have promise. One is sequential Monte Carlo, a family of nondeterministic techniques (Liu & Chen, 2000; Carpenter et al., 1999). Another is the assumed-density filter, a general deterministic method that approximately minimizes the KL-divergence $D(f \mid\mid g)$ between $f(\mathbf{x})$ and its approximation $g(\mathbf{x})$, as shown in figure 2-1. The next chapter discusses the assumed-density filter and its extension to Expectation Propagation.

Chapter 3

Expectation Propagation

This chapter describes recursive approximation techniques that try to minimize the KL-divergence between the true posterior and the approximation. Assumed-density filtering is a fast sequential method for this purpose. Expectation Propagation is introduced as an extension of assumed-density filtering to batch situations. It has higher accuracy than assumed-density filtering and other comparable methods for approximate inference.

3.1 Assumed-density filtering

This section reviews the idea of assumed-density filtering (ADF), to lay groundwork for Expectation Propagation. Assumed-density filtering is a general technique for computing approximate posteriors in Bayesian networks and other statistical models. ADF has been independently proposed in the statistics (Lauritzen, 1992; Bernardo & Giron, 1988; Stephens, 1997), artificial intelligence (Boyen & Koller, 1998b; Opper & Winther, 1999; Barber & Sollich, 1999; Frey et al., 2000), and control literature (Kushner & Budhiraja, 2000; Maybeck, 1982). "Assumed-density filtering" is the name used in control; other names include "online Bayesian learning," "moment matching," and "weak marginalization." ADF applies when we have postulated a joint distribution $p(D, \theta)$ where D has been observed and θ is hidden. We would like to know the posterior over θ, $p(\theta|D)$, as well as the probability of the observed data (or evidence for the model), $p(D)$. The former is useful for estimation while the latter is useful for model selection.

For example, suppose we have observations from a Gaussian distribution embedded in a sea of unrelated clutter, so that the observation density is a mixture of two Gaussians:

$$p(\mathbf{x}|\theta) = (1-w)\mathcal{N}(\mathbf{x};\theta,\mathbf{I}) + w\mathcal{N}(\mathbf{x};\mathbf{0},10\mathbf{I}) \tag{3.1}$$

$$\mathcal{N}(\mathbf{x};\mathbf{m},\mathbf{V}) = \frac{1}{|2\pi\mathbf{V}|^{1/2}}\exp(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^{\mathrm{T}}\mathbf{V}^{-1}(\mathbf{x}-\mathbf{m})) \tag{3.2}$$

The first component contains the parameter of interest, while the other component describes clutter. The constant w is the known ratio of clutter. Let the d-dimensional vector θ have a Gaussian prior distribution:

$$p(\theta) \sim \mathcal{N}(\mathbf{0}, 100\mathbf{I}_d) \tag{3.3}$$

The joint distribution of θ and n independent observations $D = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ is therefore:

$$p(D,\theta) = p(\theta)\prod_i p(\mathbf{x}_i|\theta) \tag{3.4}$$

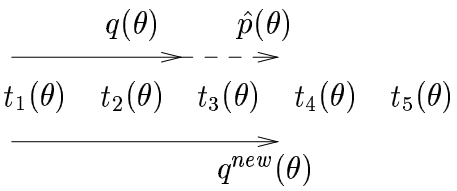


Figure 3-1: Assumed-density filtering computes an exact one-step posterior $\hat{p}(\theta)$ and then approximates it to get $q^{new}(\theta)$.

To apply ADF, we first factor the joint distribution $p(D, \theta)$ into a product of simple terms:

$$p(D, \theta) = \prod_i t_i(\theta) \tag{3.5}$$

There are many ways to do this. As a rule of thumb, fewer terms are better, since it entails fewer approximations. However, we need each term to be simple enough to propagate expectations through. In the mixture example, we can use the factoring into $n + 1$ terms implied by (3.4):

$$t_0(\theta) = p(\theta) \tag{3.6}$$
$$t_i(\theta) = p(\mathbf{x}_i|\theta) \tag{3.7}$$

For a general Bayesian network, we would use the factoring into conditional probability tables: $\prod_{\text{nodes } Y} p(Y|\text{pa}(Y))$, where $\text{pa}(Y)$ is the parents of node Y in the graph.

The second step is to choose a parametric approximating distribution. It is essential that the distribution be in the exponential family, so that only a fixed number of expectations (the sufficient statistics) need to be propagated. Usually the nature and domain of θ constrains the distribution enough that there is no choice left to make. In the clutter problem, a spherical Gaussian distribution is appropriate. The approximate posterior is therefore

$$q(\theta) \sim \mathcal{N}(\mathbf{m}_\theta, v_\theta \mathbf{I}) \tag{3.8}$$

Finally, we sequence through and incorporate the terms t_i into the approximate posterior. At each step we move from an old $q(\theta)$ to a new $q(\theta)$, as shown in figure 3-1. (For notational simplicity, we drop the dependence of $q(\theta)$ on i.) Initialize with $q(\theta) = 1$. Incorporating the prior term is trivial, with no approximation needed. To incorporate the next term $t_i(\theta)$, take the exact posterior

$$\hat{p}(\theta) = \frac{t_i(\theta) q(\theta)}{\int_\theta t_i(\theta) q(\theta) d\theta} \tag{3.9}$$

and minimize the KL-divergence $D(\hat{p}(\theta)||q^{new}(\theta))$ subject to the constraint that $q^{new}(\theta)$ is a Gaussian distribution. Zeroing the gradient with respect to $(\mathbf{m}_\theta, v_\theta)$ gives the conditions

$$\mathbf{m}_\theta^{new} = \int_\theta \hat{p}(\theta) \theta \, d\theta \tag{3.10}$$
$$v_\theta^{new} d + (\mathbf{m}_\theta^{new})^{\mathrm{T}} (\mathbf{m}_\theta^{new}) = \int_\theta \hat{p}(\theta) \theta^{\mathrm{T}} \theta \, d\theta \tag{3.11}$$

or in other words, expectation constraints:

$$E_{q^{new}}[\theta] = E_{\hat{p}}[\theta] \tag{3.12}$$
$$E_{q^{new}}[\theta^{\mathrm{T}} \theta] = E_{\hat{p}}[\theta^{\mathrm{T}} \theta] \tag{3.13}$$

We see that the spherical Gaussian distribution is characterized by the expectations $(E[\theta], E[\theta^{\mathrm{T}}\theta])$. For other members of the exponential family, we will get constraints on different expectations, as illustrated in section 3.3.

To compute these expectations, it is helpful to exploit the following relations:

$$
\begin{aligned}
Z(\mathbf{m}_\theta, v_\theta) &= \int_\theta t(\theta) q(\theta)\ d\theta && (3.14)\\
&= \int_\theta \frac{t(\theta)}{(2\pi v_\theta)^{d/2}} \exp(-\frac{1}{2v_\theta}(\theta - \mathbf{m}_\theta)^{\mathrm{T}}(\theta - \mathbf{m}_\theta))\ d\theta && (3.15)\\
\nabla_m \log Z(\mathbf{m}_\theta, v_\theta) &= \frac{1}{Z}\int_\theta \frac{(\theta - \mathbf{m}_\theta)}{v_\theta} \frac{t(\theta)}{(2\pi v_\theta)^{d/2}} \exp(-\frac{1}{2v_\theta}(\theta - \mathbf{m}_\theta)^{\mathrm{T}}(\theta - \mathbf{m}_\theta))\ d\theta && (3.16)\\
&= \frac{E_{\hat{p}}[\theta]}{v_\theta} - \frac{\mathbf{m}_\theta}{v_\theta} && (3.17)\\
E_{\hat{p}}[\theta] &= \mathbf{m}_\theta + v_\theta \nabla_m \log Z(\mathbf{m}_\theta, v_\theta) && (3.18)\\
E_{\hat{p}}[\theta^{\mathrm{T}}\theta] - E_{\hat{p}}[\theta]^{\mathrm{T}} E_{\hat{p}}[\theta] &= v_\theta d - v_\theta^2(\nabla_m^{\mathrm{T}} \nabla_m - 2\nabla_v \log Z(\mathbf{m}_\theta, v_\theta)) && (3.19)
\end{aligned}
$$

These relations are a property of the Gaussian distribution and hold for any $t(\theta)$. In the clutter problem, we have

$$
\begin{aligned}
Z(\mathbf{m}_\theta, v_\theta) &= (1-w)\mathcal{N}(\mathbf{x}; \mathbf{m}_\theta, (v_\theta + 1)\mathbf{I}) + w\mathcal{N}(\mathbf{x}; \mathbf{0}, 10\mathbf{I}) && (3.20)\\
r &= \frac{(1-w)\mathcal{N}(\mathbf{x}; \mathbf{m}_\theta, (v_\theta + 1)\mathbf{I})}{(1-w)\mathcal{N}(\mathbf{x}; \mathbf{m}_\theta, (v_\theta + 1)\mathbf{I}) + w\mathcal{N}(\mathbf{x}; \mathbf{0}, 10\mathbf{I})} && (3.21)\\
\nabla_m \log Z(\mathbf{m}_\theta, v_\theta) &= r\frac{\mathbf{x} - \mathbf{m}_\theta}{v_\theta + 1} && (3.22)\\
\nabla_v \log Z(\mathbf{m}_\theta, v_\theta) &= -\frac{rd}{2(v_\theta + 1)} + \frac{r(\mathbf{x} - \mathbf{m}_\theta)^{\mathrm{T}}(\mathbf{x} - \mathbf{m}_\theta)}{2(v_\theta + 1)^2} && (3.23)\\
\nabla_m^{\mathrm{T}} \nabla_m - 2\nabla_v \log Z(\mathbf{m}_\theta, v_\theta) &= \frac{rd}{v_\theta + 1} - r(1-r)\frac{(\mathbf{x} - \mathbf{m}_\theta)^{\mathrm{T}}(\mathbf{x} - \mathbf{m}_\theta)}{(v_\theta + 1)^2} && (3.24)
\end{aligned}
$$

An estimate of the probability of the data, $p(D)$, is a trivial byproduct of ADF. Simply accumulate the normalization factors $Z_i(\mathbf{m}_\theta, v_\theta)$ produced by each update, to get an overall normalization for the posterior. This normalizer estimates $p(D)$ because $p(D, \theta) = p(\theta|D)p(D)$.

The final ADF algorithm is:

1. Initialize $\mathbf{m}_\theta = \mathbf{0}$, $v_\theta = 100$ (the prior). Initialize $s = 1$ (the scale factor).

2. For each data point $\mathbf{x}_i$, update $(\mathbf{m}_\theta, v_\theta, s)$ according to

$$
\begin{aligned}
\mathbf{m}_\theta^{new} &= \mathbf{m}_\theta + v_\theta r_i \frac{\mathbf{x}_i - \mathbf{m}_\theta}{v_\theta + 1} && (3.25)\\
v_\theta^{new} &= v_\theta - r_i \frac{v_\theta^2}{v_\theta + 1} + r_i(1 - r_i)\frac{v_\theta^2(\mathbf{x}_i - \mathbf{m}_\theta)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{m}_\theta)}{d(v_\theta + 1)^2} && (3.26)\\
s^{new} &= s \times Z_i(\mathbf{m}_\theta, v_\theta) && (3.27)
\end{aligned}
$$

This algorithm can be understood in an intuitive way: for each data point we compute its probability r of not being clutter, make a soft update to our estimate of θ ($\mathbf{m}_\theta$), and change our confidence in the estimate (v_θ). However, it is clear that this algorithm will depend on

the order in which data is processed, because the clutter probability depends on the current estimate of θ.

This algorithm was derived using a spherical Gaussian approximating distribution. If we use a richer set of distributions like diagonal or full-covariance Gaussians, we can get a better result, since more expectations will be preserved. Unlike fitting models to data, there is no overfitting problem when fitting approximations to posteriors. The only penalty is computational cost.

Figure 3-2 shows the output of this algorithm using three random orderings of the same data. Synthetic data was generated using $w = 0.5$ and $\theta = 2$. No theory is available for how ADF varies with ordering, but some empirical observations can be made. The error increases whenever similar data points are processed together. Processing the data in sorted order is especially bad. This is because the algorithm overcommits to one part of input space, only to realize that there is also data somewhere else. The approximate mean and especially the variance will suffer because of this. So one approach to improve ADF is to find an ordering that best reflects the natural variation in the data, if this can be quantified somehow. Another approach is to modify ADF to eliminate the dependence on ordering, which is described in the next section.

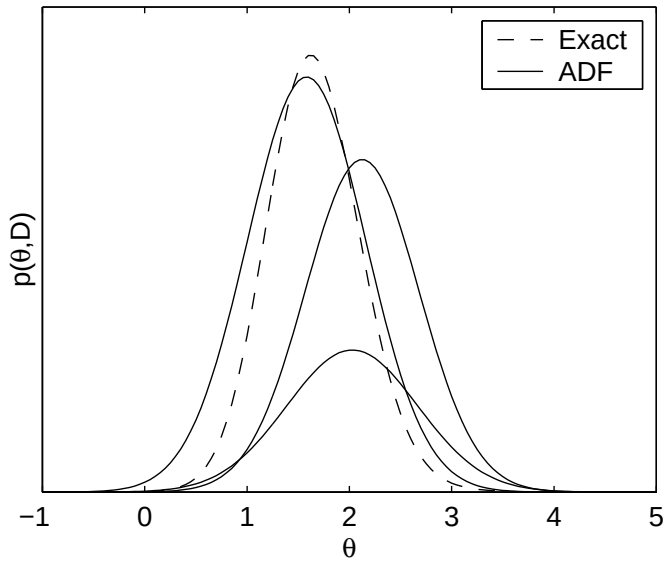


Figure 3-2: The approximate posterior resulting from ADF, using three different orderings of the same data. The posterior is scaled by the evidence estimate, $p(D)$.

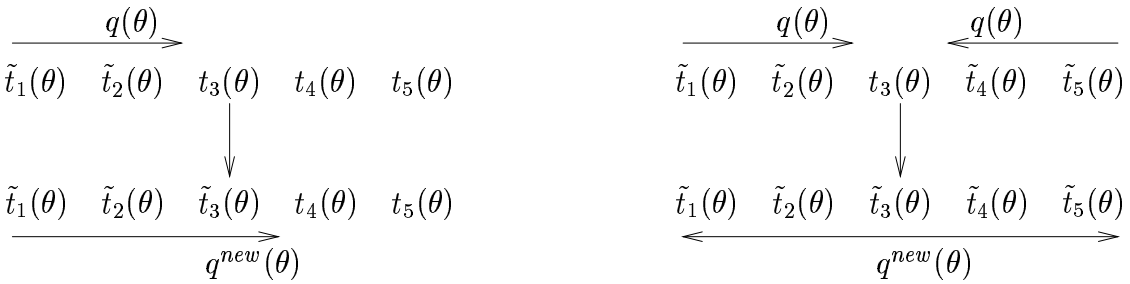


Figure 3-3: An alternative view of ADF as approximating each term and then computing $q^{new}(\theta)$ exactly. EP refines each term in the context of all other terms.

3.2 Expectation Propagation

This section describes the Expectation Propagation algorithm and demonstrates its use on the clutter problem. Expectation Propagation is based on a novel interpretation of assumed-density filtering. Normally we think of ADF as treating each observation term t_i exactly and then approximating the posterior that includes t_i. But we can also think of it as first approximating t_i with some $\tilde{t}_i$ and then using an exact posterior with $\tilde{t}_i$ (figure 3-3). This interpretation is always possible because we can define the approximate term $\tilde{t}_i$ to be the ratio of the new posterior to the old posterior times a constant:

$$\tilde{t}(\theta) = Z\frac{q^{new}(\theta)}{q(\theta)} \tag{3.28}$$

Multiplying this approximate term by $q(\theta)$ gives $q^{new}(\theta)$, as desired. An important property is that if the approximate posterior is Gaussian, i.e. an exponentiated quadratic in θ, then the equivalent term approximation, which is a ratio of these, will also be an exponentiated quadratic. If the approximate posterior is any distribution in the exponential family, then the term approximations will have the same functional form as that distribution.

The algorithm of the previous section can thus be interpreted as sequentially computing a Gaussian approximation $\tilde{t}_i(\theta)$ to every observation term $t_i(\theta)$, then combining these approximations analytically to get a Gaussian posterior on θ. Under this perspective, the approximations do not have any required order—the ordering only determined how we *made* the approximations. We are free to go back and refine the approximations, in any order. From this idea we get Expectation Propagation.

Consider how this new interpretation applies to the previous section. From the ratio of spherical Gaussians, we get the following term approximation:

$$Z_i(\mathbf{m}_\theta, v_\theta) = \int_\theta t_i(\theta)q(\theta)d\theta \tag{3.29}$$

$$\begin{aligned}\tilde{t}_i(\theta) &= Z_i(\mathbf{m}_\theta, v_\theta)\frac{q^{new}(\theta)}{q(\theta)} \\ &= Z_i(\mathbf{m}_\theta, v_\theta)\left(\frac{v_\theta}{v_\theta^{new}}\right)^{d/2}\exp(-\frac{1}{2v_\theta^{new}}(\theta-\mathbf{m}_\theta^{new})^{\mathrm{T}}(\theta-\mathbf{m}_\theta^{new})) \\ &\quad \exp(\frac{1}{2v_\theta}(\theta-\mathbf{m}_\theta)^{\mathrm{T}}(\theta-\mathbf{m}_\theta))\end{aligned} \tag{3.30, 3.31}$$

Now define $(\mathbf{m}_i, v_i)$ according to

$$
\begin{aligned}
v_i^{-1} &= (v_\theta^{new})^{-1} - v_\theta^{-1} && (3.32)\\
\mathbf{m}_i &= v_i (v_\theta^{new})^{-1} \mathbf{m}_\theta^{new} - v_i v_\theta^{-1} \mathbf{m}_\theta && (3.33)\\
&= \mathbf{m}_\theta + (v_i + v_\theta) v_\theta^{-1} (\mathbf{m}_\theta^{new} - \mathbf{m}_\theta) && (3.34)
\end{aligned}
$$

to get the simplification

$$
\begin{aligned}
\tilde{t}_i(\theta) &= Z_i(\mathbf{m}_\theta, v_\theta) \left(\frac{v_i + v_\theta}{v_i}\right)^{d/2} \exp(-\frac{1}{2v_i}(\theta - \mathbf{m}_i)^{\mathrm{T}}(\theta - \mathbf{m}_i)\\
&\quad \exp(\frac{1}{2(v_i + v_\theta)}(\mathbf{m}_i - \mathbf{m}_\theta)^{\mathrm{T}}(\mathbf{m}_i - \mathbf{m}_\theta)) && (3.35)\\
&= \frac{Z_i(\mathbf{m}_\theta, v_\theta)}{\mathcal{N}(\mathbf{m}_i; \mathbf{m}_\theta, (v_i + v_\theta)\mathbf{I})} \mathcal{N}(\theta; \mathbf{m}_i, v_i \mathbf{I}) && (3.36)
\end{aligned}
$$

By construction, multiplying $q(\theta)$ by this approximation gives exactly the ADF update, with the proper scale factor. The $\mathcal{N}$ notation is here used formally as shorthand, not to imply a proper density. The approximation is an exponentiated quadratic in θ, which has the form of a Gaussian with mean $\mathbf{m}_i$ and variance $v_i\mathbf{I}$, times a scale factor. What makes this approximation different from a Gaussian is that the variance v_i may be negative or infinite, which for $\tilde{t}_i$ is perfectly fine: negative variance corresponds to a function that curves upward and infinite variance corresponds to a constant.

Figure 3-4 illustrates $t_i(\theta)$ and $\tilde{t}_i(\theta)$ for the clutter problem. The exact term $t_i(\theta) = p(\mathbf{x}_i|\theta)$ is a Gaussian in θ, raised by a constant. The approximate term $\tilde{t}_i(\theta)$ is an exponentiated quadratic. The current posterior $q(\theta)$ determines where the approximation will be accurate. When $q(\theta)$ is narrow, i.e. v_θ is small, then $\tilde{t}_i(\theta)$ is accurate only in a narrow range of θ values, determined by $\mathbf{m}_\theta$. When v_θ is large, then $\tilde{t}_i(\theta)$ tries to approximate $t_i(\theta)$ more broadly, and $\mathbf{m}_\theta$ is less important.

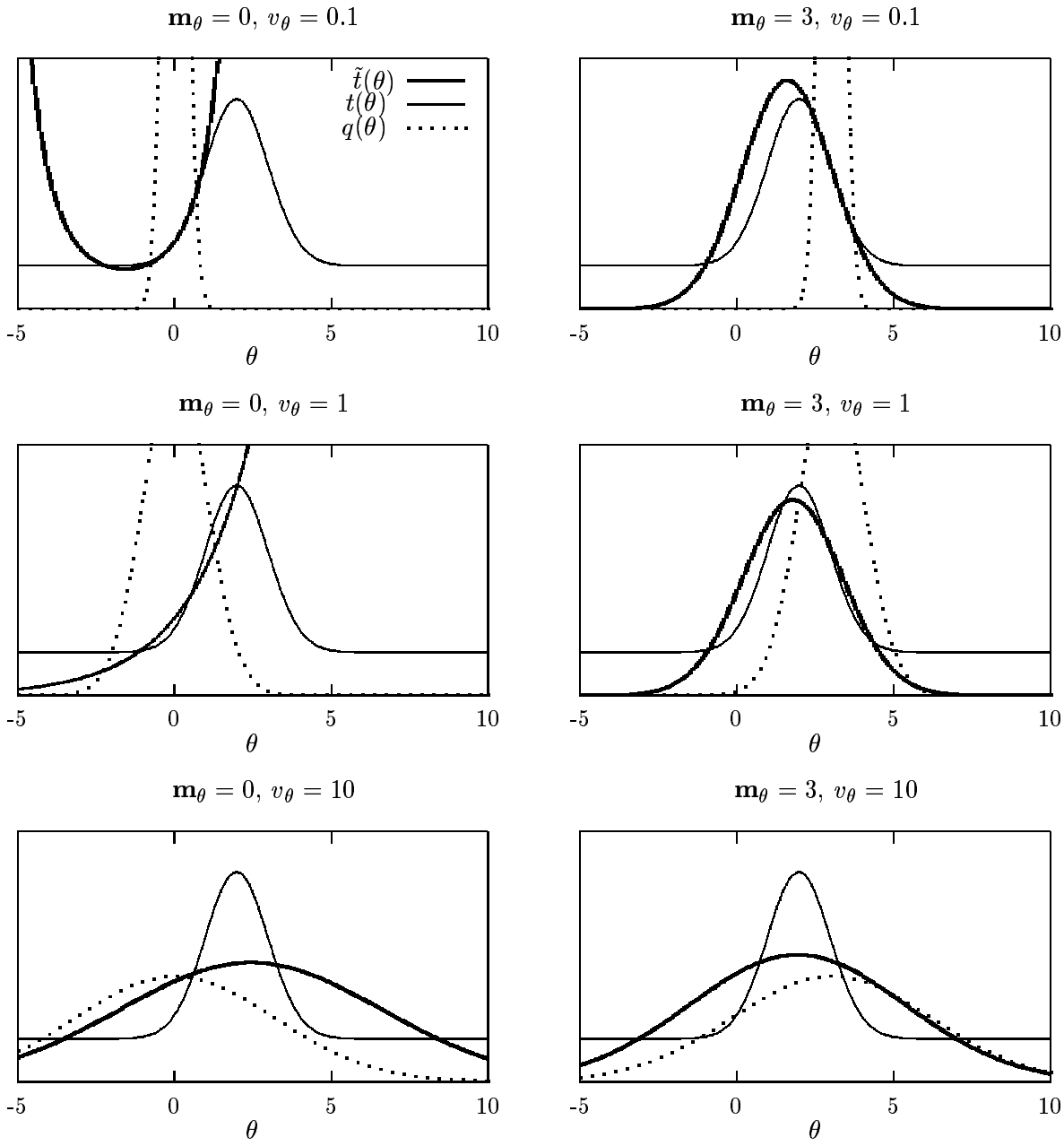


Figure 3-4: The approximate term $\tilde{t}(\theta)$ as a function of θ, plotted versus the exact term $t(\theta)$ and the current posterior $q(\theta)$. The current posterior controls where the approximation will be accurate.

So far we have just rewritten the ADF updates in a different way. To get EP, we refine the term variables $(\mathbf{m}_i, v_i)$ based on all of the other approximations. The general EP algorithm is:

1. Initialize the term approximations $\tilde{t}_i$
2. Compute the posterior for θ from the product of $\tilde{t}_i$:

$$q^{new}(\theta) = \frac{\prod_i \tilde{t}_i(\theta)}{\int \prod_i \tilde{t}_i(\theta) d\theta} \tag{3.37}$$

3. Until all $\tilde{t}_i$ converge:
   (a) Choose a $\tilde{t}_i$ to refine
   (b) Remove $\tilde{t}_i$ from the posterior to get an 'old' posterior $q(\theta)$, by dividing and normalizing:

$$q(\theta) \propto \frac{q^{new}(\theta)}{\tilde{t}_i(\theta)} \tag{3.38}$$

   (For notational simplicity, we drop the dependence of $q(\theta)$ on i. However $q^{new}(\theta)$ is not a function of i.)
   (c) Compute the posterior $q^{new}(\theta)$ and normalizing factor Z_i from $q(\theta)$ and $t_i(\theta)$ via ADF.
   (d) Set

$$\tilde{t}_i = Z_i \frac{q^{new}(\theta)}{q(\theta)} \tag{3.39}$$

4. Use the normalizing constant of $q^{new}(\theta)$ (from (3.37)) as an approximation to $p(D)$:

$$p(D) \approx \int \prod_i \tilde{t}_i(\theta) d\theta \tag{3.40}$$

At convergence, the result will be independent of processing order, as desired. In this algorithm, we have used division to remove $\tilde{t}_i$ from the posterior. Step 3b can also be performed without division, by accumulating all terms except for $\tilde{t}_i$:

$$q(\theta) \propto \prod_{j \neq i} \tilde{t}_j(\theta) \tag{3.41}$$

but division is usually more efficient.

3.2.1 The clutter problem

For the clutter problem of the previous section, the EP algorithm is:

1. The term approximations have the form

$$\tilde{t}_i(\theta) = s_i \exp(-\frac{1}{2v_i}(\theta - \mathbf{m}_i)^{\mathrm{T}}(\theta - \mathbf{m}_i)) \tag{3.42}$$

Initialize the prior term to itself:

$$v_0 = 100 \tag{3.43}$$
$$\mathbf{m}_0 = \mathbf{0} \tag{3.44}$$
$$s_0 = (2\pi v_0)^{-d/2} \tag{3.45}$$

Initialize the data terms to 1:

$$v_i = \infty \tag{3.46}$$
$$\mathbf{m}_i = \mathbf{0} \tag{3.47}$$
$$s_i = 1 \tag{3.48}$$

2. $\mathbf{m}_\theta^{new} = \mathbf{m}_0, v_\theta^{new} = v_0$

3. Until all $(\mathbf{m}_i, v_i, s_i)$ converge (changes are less than 10^{-4}):

   loop $i = 1, ..., n$:

   (a) Remove $\tilde{t}_i$ from the posterior to get an 'old' posterior:

$$v_\theta^{-1} = (v_\theta^{new})^{-1} - v_i^{-1} \tag{3.49}$$
$$\mathbf{m}_\theta = v_\theta(v_\theta^{new})^{-1}\mathbf{m}_\theta^{new} - v_\theta v_i^{-1}\mathbf{m}_i = \mathbf{m}_\theta^{new} + v_\theta v_i^{-1}(\mathbf{m}_\theta^{new} - \mathbf{m}_i) \tag{3.50}$$

   (b) Recompute $(\mathbf{m}_\theta^{new}, v_\theta^{new}, Z_i)$ from $(\mathbf{m}_\theta, v_\theta)$ (this is the same as ADF):

$$r_i = \frac{(1-w)\mathcal{N}(\mathbf{x}_i; \mathbf{m}_\theta, (v_\theta + 1)\mathbf{I})}{(1-w)\mathcal{N}(\mathbf{x}_i; \mathbf{m}_\theta, (v_\theta + 1)\mathbf{I}) + w\mathcal{N}(\mathbf{x}_i; \mathbf{0}, 10\mathbf{I})} \tag{3.51}$$
$$\mathbf{m}_\theta^{new} = \mathbf{m}_\theta + v_\theta r_i \frac{\mathbf{x}_i - \mathbf{m}_\theta}{v_\theta + 1} \tag{3.52}$$
$$v_\theta^{new} = v_\theta - r_i\frac{v_\theta^2}{v_\theta + 1} + r_i(1 - r_i)\frac{v_\theta^2(\mathbf{x}_i - \mathbf{m}_\theta)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{m}_\theta)}{d(v_\theta + 1)^2} \tag{3.53}$$
$$Z_i = (1-w)\mathcal{N}(\mathbf{x}_i; \mathbf{m}_\theta, (v_\theta + 1)\mathbf{I}) + w\mathcal{N}(\mathbf{x}_i; \mathbf{0}, 10\mathbf{I}) \tag{3.54}$$

   (c) Update $\tilde{t}_i$:

$$v_i^{-1} = (v_\theta^{new})^{-1} - v_\theta^{-1} \tag{3.55}$$
$$= \left(\frac{r_i}{v_\theta + 1} - r_i(1 - r_i)\frac{(\mathbf{x}_i - \mathbf{m}_\theta)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{m}_\theta)}{d(v_\theta + 1)^2}\right)^{-1} - v_\theta \tag{3.56}$$
$$\mathbf{m}_i = \mathbf{m}_\theta + (v_i + v_\theta)v_\theta^{-1}(\mathbf{m}_\theta^{new} - \mathbf{m}_\theta) \tag{3.57}$$
$$= \mathbf{m}_\theta + (v_i + v_\theta)r_i\frac{\mathbf{x}_i - \mathbf{m}_\theta}{v_\theta + 1} \tag{3.58}$$
$$s_i = \frac{Z_i}{(2\pi v_i)^{d/2}\mathcal{N}(\mathbf{m}_i; \mathbf{m}_\theta, (v_i + v_\theta)\mathbf{I})} \tag{3.59}$$

4. Compute the normalizing constant:

$$B = \frac{(\mathbf{m}_\theta^{new})^{\mathrm{T}} \mathbf{m}_\theta^{new}}{v_\theta^{new}} - \sum_i \frac{\mathbf{m}_i^{\mathrm{T}} \mathbf{m}_i}{v_i} \tag{3.60}$$

$$p(D) \approx \int \prod_i \tilde{t}_i(\theta) d\theta \tag{3.61}$$

$$= (2\pi v_\theta^{new})^{d/2} \exp(B/2) \prod_{i=0}^{n} s_i \tag{3.62}$$

Because the term approximations start at 1, the result after one pass through the data is identical to ADF. On later passes, the refinement may sometimes fail due to a negative value for v_θ. This happens when many of the v_i are negative and we wish to refine a term with positive v_i. In the subtraction step (3.49), we subtract a positive value from v_θ^{new} and are left with something negative. From this Z_i does not exist and the algorithm fails.

Another problem is that the term approximations may oscillate without ever converging. This tends to occur in conjunction with negative v_i's, which suggests a simple work-around: force all $v_i > 0$ by relaxing some of the expectation constraints. Whenever a v_i would become negative, make it large (10^8) instead and set $v_\theta^{new} = v_\theta$ (because $v_\theta^{new} = (v_\theta^{-1} + v_i^{-1})^{-1}$). This trick does provide convergence, but leads to inaccurate posteriors since we are effectively ignoring some of the data. When EP does converge with negative v_i's, the result is always better than having forced $v_i > 0$. So a better solution would be to use a refinement algorithm with better convergence properties—see the discussion at the end of section 4.1.

3.2.2 Results and comparisons

This section evaluates ADF and EP on the clutter problem and compares them to four other algorithms for approximate inference: Laplace's method, variational Bayes, importance sampling (specifically likelihood-weighted sampling), and Gibbs sampling.

In Laplace's method, we first run EM to get a MAP estimate of θ. Then we construct a Gaussian approximation to the posterior by using the curvature of the posterior at $\hat{\theta}$. This curvature is (Minka, 2000c)

$$\mathbf{H} = \nabla_{\theta\theta^{\mathrm{T}}} \log p(\theta|D) = -\frac{1}{100}\mathbf{I} - \sum_i r_i \mathbf{I} + \sum_i r_i(1-r_i)(\mathbf{x}_i - \hat{\theta})(\mathbf{x}_i - \hat{\theta})^{\mathrm{T}} \tag{3.63}$$

$$r_i = \frac{(1-w)\mathcal{N}(\mathbf{x}; \hat{\theta}, \mathbf{I})}{(1-w)\mathcal{N}(\mathbf{x}; \hat{\theta}, \mathbf{I}) + w\mathcal{N}(\mathbf{x}; \mathbf{0}, 10\mathbf{I})} \tag{3.64}$$

The Gaussian approximation and normalizing constant are

$$p(\theta|D) \approx \mathcal{N}(\hat{\theta}, -\mathbf{H}^{-1}) \tag{3.65}$$

$$p(D) \approx p(D, \hat{\theta})(2\pi)^{d/2} |-\mathbf{H}|^{-1/2} \tag{3.66}$$

For variational Bayes, we lower bound the joint distribution by bounding each data term (Minka, 2000c):

$$p(\mathbf{x}_i|\theta) \geq \left(\frac{(1-w)\mathcal{N}(\mathbf{x}; \theta, \mathbf{I})}{q_{i1}}\right)^{q_{i1}} \left(\frac{w\mathcal{N}(\mathbf{x}; \mathbf{0}, 10\mathbf{I})}{q_{i2}}\right)^{q_{i2}} \tag{3.67}$$

The variational parameters q_{ij} are optimized to give the tightest bound. Given the bounds, the posterior for θ is Gaussian:

$$
\begin{aligned}
K_j &= \sum_i q_{ij} && (3.68)\\
\bar{\mathbf{x}}_j &= \frac{1}{K_j}\sum_i q_{ij}\mathbf{x}_i && (3.69)\\
S_j &= \sum_i q_{ij}(\mathbf{x}_i - \bar{\mathbf{x}}_j)(\mathbf{x}_i - \bar{\mathbf{x}}_j)^{\mathrm{T}} && (3.70)\\
\mathbf{V}_\theta &= \left(K_1\mathbf{I} + \frac{\mathbf{I}}{100}\right)^{-1} && (3.71)\\
\mathbf{m}_\theta &= \mathbf{V}_\theta K_1 \bar{\mathbf{x}}_1 && (3.72)\\
p(\theta|D) &\approx \mathcal{N}(\mathbf{m}_\theta, \mathbf{V}_\theta) && (3.73)
\end{aligned}
$$

$$
\begin{aligned}
p(D) \geq\ & \frac{1}{(2\pi)^{d(K_1-1)/2}K_1^{d/2}}\mathcal{N}(\bar{\mathbf{x}}_1; \mathbf{0}, \mathbf{I}/K_1 + 100)\exp(-\frac{1}{2}\mathrm{tr}(\mathbf{S}_1)) && (3.74)\\
& \frac{1}{(2\pi 10)^{d(K_2-1)/2}K_2^{d/2}}\mathcal{N}(\bar{\mathbf{x}}_2; \mathbf{0}, 10\mathbf{I}/K_2)\exp(-\frac{1}{2}\mathrm{tr}(\mathbf{S}_2/10)) && (3.75)\\
& \prod_i \left(\frac{w}{q_{i1}}\right)^{q_{i1}} \left(\frac{1-w}{q_{i2}}\right)^{q_{i2}} && (3.76)
\end{aligned}
$$

For importance sampling, or more specifically likelihood-weighted sampling, we draw samples $\{\theta_1, ..., \theta_S\}$ from $p(\theta)$ and approximate the integrals via

$$
\begin{aligned}
p(D) &\approx \frac{1}{S}\sum_{i=1}^{S} p(D|\theta_i) && (3.77)\\
E[\theta|D] &\approx \frac{\sum_{i=1}^{S}\theta_i p(D|\theta_i)}{\sum_{i=1}^{S} p(D|\theta_i)} && (3.78)
\end{aligned}
$$

For Gibbs sampling, we introduce hidden variables c_i which indicate whether $\mathbf{x}_i$ is clutter or not. Given a choice for θ, we sample c_i from a Bernoulli distribution with mean r_i. Then given c_i, we form an exact Gaussian posterior for θ and sample a new θ. The average of the θ's that arise from this process is our estimate of the posterior mean.

The three deterministic algorithms, EP, Laplace, and VB all obtain approximate posteriors close to the true one. The sampling algorithms only estimate specific integrals. To compare the algorithms quantitatively, we compute the absolute difference between the estimated and exact evidence and the estimated and exact posterior mean. Figure 3-5 shows the results on a typical run with data size $n = 20$ and $n = 200$. It plots the accuracy vs. cost of the algorithms on the two integrals. Accuracy is measured by absolute difference from the true integral. Cost is measured by the number of floating point operations (FLOPS) in Matlab, via Matlab's `flops` function. This is better than using CPU time because FLOPS ignores interpretation overhead.

The exact shape of these curves, especially the EP curve, should not be taken too seriously since the convergence properties of EP, including whether it converges at all, can be substantially affected by the particular dataset and data ordering. The Laplace curve is generated by applying (3.66) to intermediate values of $\hat{\theta}$, even though the gradient may not be zero.

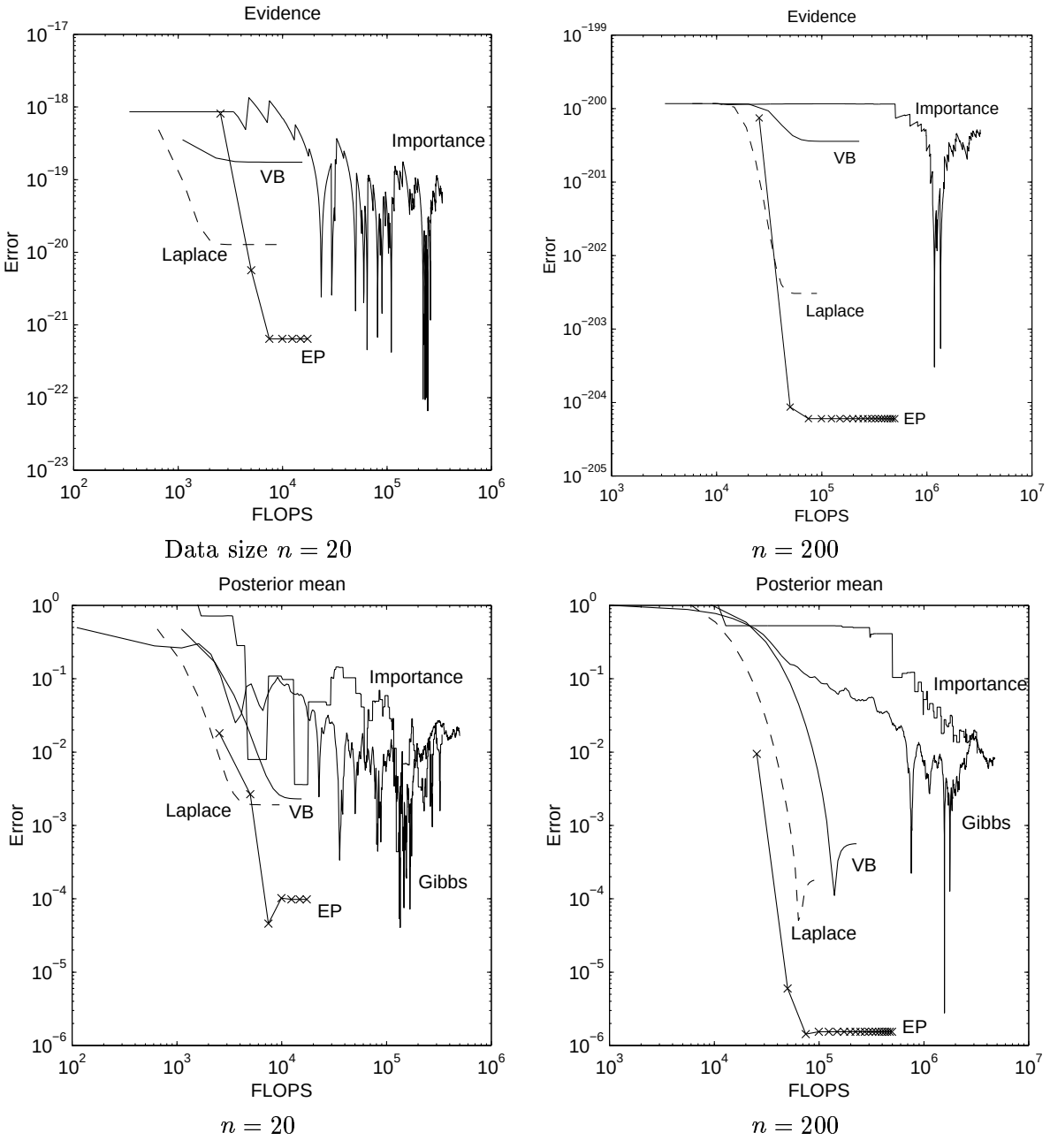


Figure 3-5: Cost vs. accuracy curves for expectation propagation (EP), Laplace's method, variational Bayes (VB), importance sampling, and Gibbs sampling on the clutter problem with $w = 0.5$ and $\theta = 2$. Each 'x' is one iteration of EP. ADF is the first 'x'.

ADF is equivalent to the first iteration of EP (the first 'x' on the curve). It performs significantly worse than variational Bayes and Laplace's method, which are offline algorithms. Yet by refining the ADF approximations, we move from last place to first. The worth of an algorithm is not always what it seems.

It is interesting to see how the different properties of the algorithms manifest themselves. EP, Laplace, and VB all try to approximate the posterior with a Gaussian, and their performance depends on how well this assumption holds. With more data, the posterior becomes more Gaussian so they all improve. Sampling methods, by contrast, assume very little about the posterior and consequently cannot exploit the fact that it is becoming more Gaussian.

However, this disadvantage turns into an advantage when the posterior has a complex shape. Figure 3-6 shows a run with $n = 20$ where the true posterior has three distinct modes. EP crashes on this data, due to negative variances. Restricted EP, which forces all $v_i > 0$, does converge but to a poor result that captures only a single mode. Laplace's method and variational Bayes are even worse. For all three algorithms, the error in the posterior mean increases with more iterations, as they focus on a single mode. Sampling methods, by contrast, are only slightly challenged by this posterior.

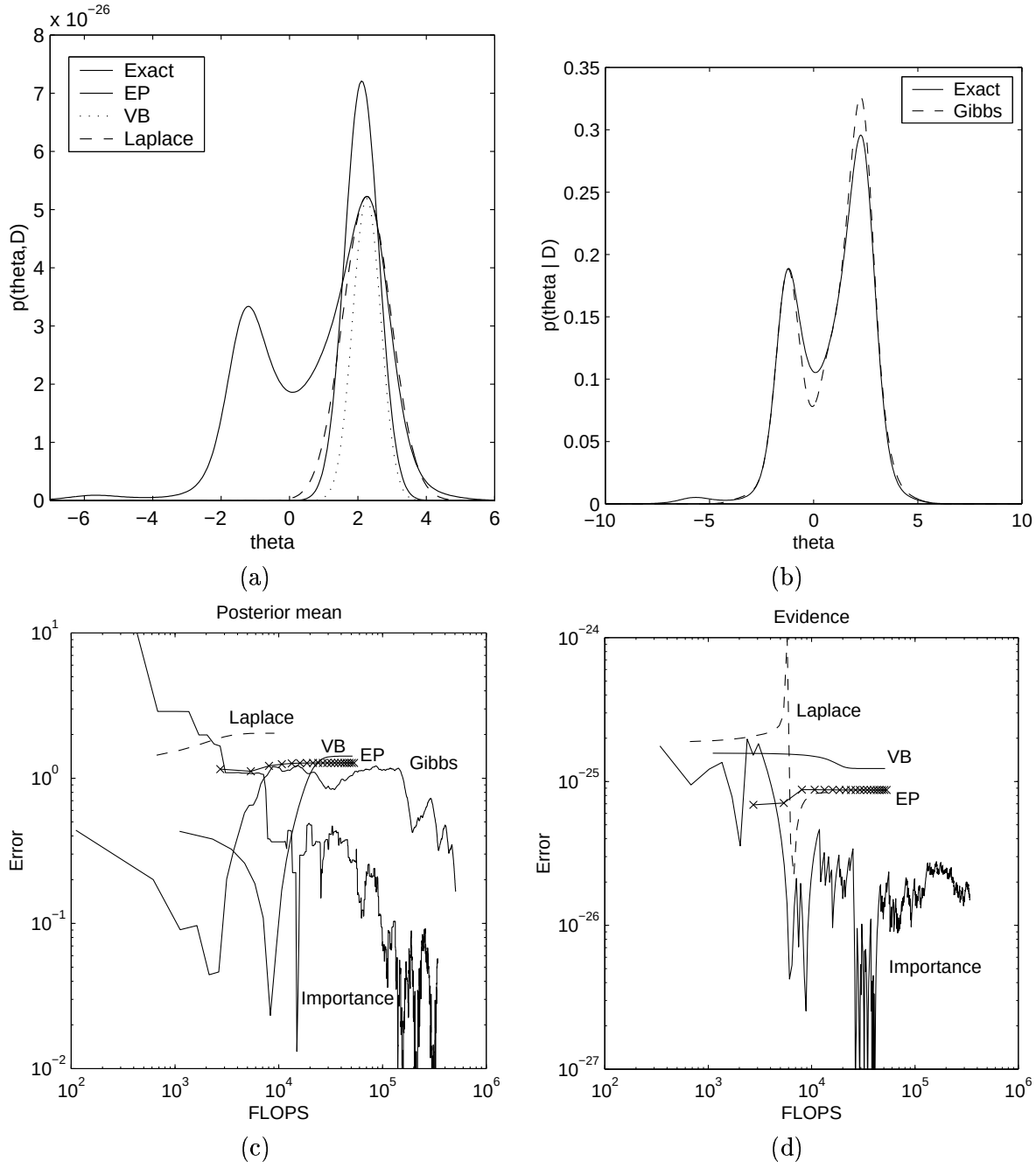


Figure 3-6: A complex posterior in the clutter problem. (a) Exact posterior vs. approximations obtained by Restricted EP, Laplace's method, and variational Bayes. (b) Exact posterior vs. approximation obtained by Gibbs sampling and complete conditional density averaging (Gelfand & Smith, 1990). (c,d) Cost vs. accuracy. EP only converged when restricted to positive v_i.

3.3 Another example: Mixture weights

This section demonstrates how the structure of ADF and EP are preserved with a different approximating distribution—a Dirichlet distribution. Let the probabilistic model be a mixture of K known densities, $p_1(x), ..., p_K(x)$, with unknown mixing weights $\mathbf{w}$:

$$p(x|\mathbf{w}) = \sum_k w_k p_k(x) \tag{3.79}$$

$$\sum_k w_k = 1 \tag{3.80}$$

$$p(\mathbf{w}) = 1 \tag{3.81}$$

$$p(D, \mathbf{w}) = p(\mathbf{w}) \prod_i p(x_i|\mathbf{w}) \tag{3.82}$$

We want to compute the posterior $p(\mathbf{w}|D)$ and the evidence $p(D)$. ADF was applied to this problem by Bernardo & Giron (1988) and Stephens (1997).

Break the joint distribution $p(D, \mathbf{w})$ into $n+1$ terms as given by (3.82). The parameter vector $\mathbf{w}$ must lie in the simplex $\sum_k w_k = 1$, so a Dirichlet distribution is an appropriate approximation to the posterior. The approximate posterior will have the form

$$q(\mathbf{w}) = \frac{\Gamma(\sum_k a_k)}{\prod_k \Gamma(a_k)} \prod_k w_k^{a_k - 1} \tag{3.83}$$

3.3.1 ADF

For ADF, we sequence through and incorporate the terms t_i into the approximate posterior. The prior is a Dirichlet distribution, so we initialize with that. To incorporate a data term $t_i(\mathbf{w})$, take the exact posterior

$$\hat{p}(\mathbf{w}) = \frac{t_i(\mathbf{w}) q(\mathbf{w})}{\int_{\mathbf{w}} t_i(\mathbf{w}) q(\mathbf{w}) d\mathbf{w}} \tag{3.84}$$

and minimize the KL-divergence $D(\hat{p}(\mathbf{w}) || q^{new}(\mathbf{w}))$ subject to the constraint that $q^{new}(\mathbf{w})$ is a Dirichlet distribution. Zeroing the gradient with respect to a_k^{new} gives the conditions $(k = 1, ..., K)$

$$\Psi(a_k^{new}) - \Psi(\sum_j a_j^{new}) = \int_{\mathbf{w}} \hat{p}(\mathbf{w}) \log(w_k) d\mathbf{w} \tag{3.85}$$

$$\text{where } \Psi(a) = \frac{d \log \Gamma(a)}{da} \tag{3.86}$$

As usual, these are expectation constraints:

$$E_{q^{new}}[\log(w_k)] = E_{\hat{p}}[\log(w_k)] \qquad k = 1, ..., K \tag{3.87}$$

The Dirichlet distribution is characterized by the expectations $E[\log(w_k)]$.

How do we solve for a^{new}? Given the values $E_{\hat{p}}[\log(w_k)]$, Newton's method is very effective, and EM-like algorithms are also possible (Minka, 2000b). To compute the expectations, use integration by parts to find

$$Z(\mathbf{a}) = \int_{\mathbf{w}} t(\mathbf{w}) q(\mathbf{w}) d\mathbf{w} \tag{3.88}$$

$$E_{\hat{p}}[\log(w_k)] = \Psi(a_k) - \Psi(\sum_j a_j) + \nabla_{a_k} \log Z(\mathbf{a}) \tag{3.89}$$

This is a property of the Dirichlet distribution, valid for any $t(\mathbf{w})$. In the mixture weight problem, we have

$$t(\mathbf{w}) = \sum_k p_k(x) w_k \tag{3.90}$$

$$Z(\mathbf{a}) = \sum_k p_k(x) E_q[w_k] = \frac{\sum_k p_k(x) a_k}{\sum_k a_k} \tag{3.91}$$

$$\nabla_{a_k} \log Z(\mathbf{a}) = \left(\frac{p_k(x)}{\sum_j a_j} - \frac{\sum_j p_j(x) a_j}{(\sum_j a_j)^2} \right) / Z(\mathbf{a}) \tag{3.92}$$

$$= \frac{p_k(x)}{\sum_j p_j(x) a_j} - \frac{1}{\sum_j a_j} \tag{3.93}$$

So the final ADF algorithm is

1. Initialize $\mathbf{a} = \mathbf{1}$ (the prior). Initialize $s = 1$ (the scale factor).
2. For each data point x_i in turn, solve the equations ($k = 1, ..., K$)

$$\Psi(a_k^{new}) - \Psi(\sum_j a_j^{new}) = \Psi(a_k) - \Psi(\sum_j a_j) + \frac{p_k(x_i)}{\sum_j p_j(x_i) a_j} - \frac{1}{\sum_j a_j} \tag{3.94}$$

   to get $\mathbf{a}^{new}$. Update s via

$$s^{new} = s \times Z_i(\mathbf{a}) = s \frac{\sum_j p_j(x_i) a_j}{\sum_j a_j} \tag{3.95}$$

Bernardo & Giron (1988) and Stephens (1997) show that this algorithm performs better than simpler rules like Directed Decision and Probabilistic Teacher. With a little extra work we can get an EP algorithm that is even better.

3.3.2 EP

Divide the new posterior by the old to get a term approximation parameterized by $\mathbf{b}$:

$$\tilde{t}(\theta) = Z \frac{q^{new}(\theta)}{q(\theta)} \tag{3.96}$$

$$= Z \frac{\Gamma(\sum_k a_k^{new})}{\prod_k \Gamma(a_k^{new})} \frac{\prod_k \Gamma(a_k)}{\Gamma(\sum_k a_k)} \prod_k w_k^{b_k} \tag{3.97}$$

$$\text{where } b_k = a_k^{new} - a_k \tag{3.98}$$

The EP algorithm is:

1. The term approximations have the form

$$\tilde{t}_i(\theta) = s_i \prod_k w_k^{b_{ik}} \tag{3.99}$$

   Initialize the prior term to itself:

$$\mathbf{b}_0 = \mathbf{0} \tag{3.100}$$

$$s_0 = 1 \tag{3.101}$$

   Initialize the data terms to 1:

$$\mathbf{b}_i = \mathbf{0} \tag{3.102}$$

$$s_i = 1 \tag{3.103}$$

2. $a_k^{new} = 1 + \sum_{i=0}^{n} b_{ik} = 1$

3. Until all $(\mathbf{b}_i, s_i)$ converge:

   loop $i = 1, ..., n$:

   (a) Remove $\tilde{t}_i$ from the posterior to get an 'old' posterior:

$$\mathbf{a} = \mathbf{a}^{new} - \mathbf{b}_i \tag{3.104}$$

   (b) Recompute $\mathbf{a}^{new}$ from $\mathbf{a}$ by solving $(k = 1, ..., K)$

$$\Psi(a_k^{new}) - \Psi(\sum_j a_j^{new}) = \Psi(a_k) - \Psi(\sum_j a_j) + \frac{p_k(x_i)}{\sum_j p_j(x_i) a_j} - \frac{1}{\sum_j a_j} \tag{3.105}$$

   (c) Update $\tilde{t}_i$:

$$\mathbf{b}_i = \mathbf{a}^{new} - \mathbf{a} \tag{3.106}$$

$$s_i = Z_i(\mathbf{a}) \frac{\Gamma(\sum_k a_k^{new})}{\prod_k \Gamma(a_k^{new})} \frac{\prod_k \Gamma(a_k)}{\Gamma(\sum_k a_k)} \tag{3.107}$$

4. Compute the normalizing constant:

$$p(D) \approx \frac{\prod_k \Gamma(a_k^{new})}{\Gamma(\sum_k a_k^{new})} \prod_{i=0}^{n} s_i \tag{3.108}$$

3.3.3 A simpler method

The complexity of this algorithm results from preserving the expectations $E[\log(w_k)]$. Cowell et al. (1996) suggest instead preserving only the first two moments of the posterior distribution, i.e. the expectations $(E[w_k], \sum_k E[w_k^2])$. Preserving these expectations will not minimize KL-divergence, but allows a closed-form update. Their proposed ADF update is:

$$m_k = E_{\hat{p}}[w_k] \tag{3.109}$$

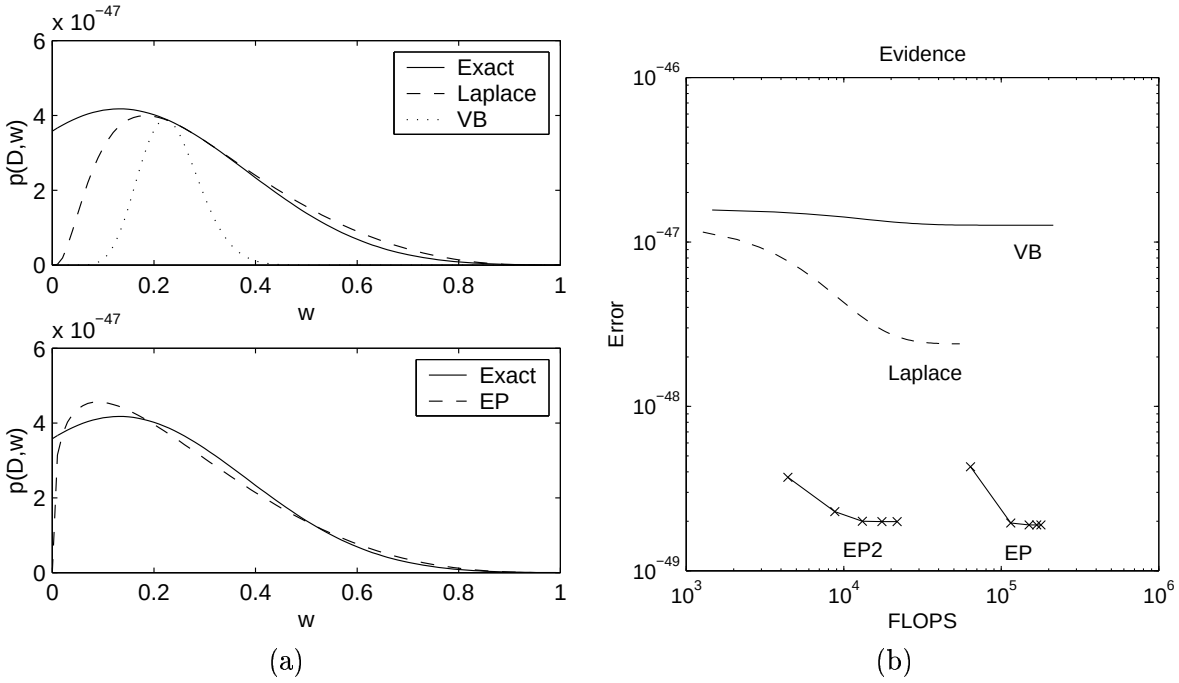
$$= \frac{a_k}{Z_i(\mathbf{a}) \sum_j a_j} \frac{p_k(x_i) + \sum_j p_j(x_i) a_j}{1 + \sum_j a_j} \tag{3.110}$$

$$m_k^{(2)} = E_{\hat{p}}[w_k^2] \tag{3.111}$$

$$= \frac{(a_k + 1) a_k}{Z_i(\mathbf{a})(1 + \sum_j a_j)(\sum_j a_j)} \frac{2p_k(x_i) + \sum_j p_j(x_i) a_j}{2 + \sum_j a_j} \tag{3.112}$$

$$a_k^{new} = \left(\frac{\sum_k (m_k - m_k^{(2)})}{\sum_k (m_k^{(2)} - m_k^2)} \right) m_k \tag{3.113}$$

This update can be incorporated into EP by replacing (3.105). The benefit of this change is determined empirically in the next section.



(a) (b)

Figure 3-7: (a) Exact posterior vs. approximations, scaled by evidence $p(D)$. The variational Bayes approximation is a lower bound while the others are not. The approximation from EP2 is very similar to EP. (b) Cost vs. accuracy in approximating $p(D)$. ADF is the first 'x' on the EP curve. EP2 uses the fast update of Cowell et al.

3.3.4 Results and comparisons

EP is compared against Laplace's method and variational Bayes. For Laplace's method, we use a softmax parameterization of w (MacKay, 1998), which makes the approximate posterior a logistic-normal distribution (the result of passing a Gaussian random variable through the logistic function) (Aitchison & Shen, 1980). For variational Bayes, we use a Jensen bound as before, to get a Dirichlet approximation (Minka, 2000c).

Figure 3-7 shows a typical example with $\mathbf{w} = [0.5, 0.5]$, $n = 50$. The component densities were two closely spaced Gaussians:

$$p_1(x) \sim \mathcal{N}(0, 3) \tag{3.114}$$

$$p_2(x) \sim \mathcal{N}(1, 3) \tag{3.115}$$

The accuracy of the three methods at estimating $p(D)$ can be seen immediately from the different posterior approximations. EP, using either the exact update or the fast update, is the most accurate. Even ADF, the first iteration of EP, is very accurate. The success of the fast update illustrates that it is sometimes good to consider criteria other than KL-divergence.

Chapter 4

Disconnected approximations of belief networks

This chapter describes how Expectation Propagation can be used to approximate a belief network by a simpler network with fewer edges. Such networks are called *disconnected approximations*.

4.1 Belief propagation

Belief propagation is normally considered an algorithm for exact marginalization in tree-structured graphs. But it can also be applied to graphs with loops, to get approximate marginals (Frey & MacKay, 1997; Murphy et al., 1999). In this section, it is shown that belief propagation applied in this heuristic way is actually a special case of Expectation Propagation. Loopy belief propagation arises when the approximating network in EP is required to be completely disconnected (all variables independent). This is interesting because loopy belief propagation is normally considered distinct from techniques that compute approximate models (but see Rusmevichientong & Roy (1999) for a similar interpretation). The equivalence also suggests how to improve both algorithms.

First consider the ADF case, where a completely disconnected approximation was studied by Boyen & Koller (1998b). Let the hidden variables be $x_1, ..., x_K$ and collect the observed variables into $D = \{y_1, ..., y_N\}$. A completely disconnected distribution for $\mathbf{x}$ has

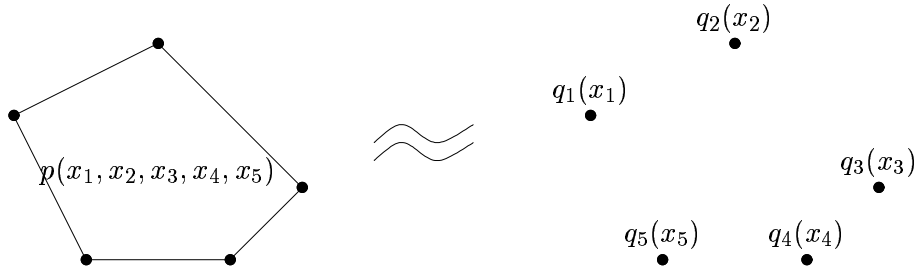


Figure 4-1: Loopy belief propagation corresponds to approximating a complex Bayesian network by a disconnected network

the form (figure 4-1)

$$q(\mathbf{x}) = \prod_{k=1}^{K} q_k(x_k) \tag{4.1}$$

To incorporate a term $t_i(\mathbf{x})$, take the exact posterior

$$\hat{p}(\mathbf{x}) = \frac{t_i(\mathbf{x})q(\mathbf{x})}{\sum_{\mathbf{x}} t_i(\mathbf{x})q(\mathbf{x})} \tag{4.2}$$

and minimize the KL-divergence $D(\hat{p}(\mathbf{x})||q^{new}(\mathbf{x}))$ subject to the constraint that $q^{new}(\mathbf{x})$ is completely disconnected. Zeroing the gradient with respect to q_k^{new} gives the conditions

$$q_k^{new}(x_k) \quad = \quad \hat{p}(x_k) = \sum_{\mathbf{x}\backslash x_k} \hat{p}(\mathbf{x}) \qquad k = 1, ..., K \tag{4.3}$$

i.e. the marginals of q^{new} and $\hat{p}$ must match. As usual, these are expectation constraints:

$$E_{q^{new}}[\delta(x_k - v)] \quad = \quad E_{\hat{p}}[\delta(x_k - v)] \qquad k = 1, ..., K, \text{ all values of } v \tag{4.4}$$

From this we arrive at the algorithm of Boyen & Koller (1998b):

1. Initialize $q_k(x_k) = 1$ and $s = 1$
2. For each term $t_i(\mathbf{x})$ in turn, set q_k^{new} to the kth marginal of $\hat{p}$:

$$q_k^{new}(x_k) \quad = \quad \sum_{\mathbf{x}\backslash x_k} \hat{p}(\mathbf{x}) \tag{4.5}$$

$$= \quad \frac{1}{Z_i} \sum_{\mathbf{x}\backslash x_k} t_i(\mathbf{x})q(\mathbf{x}) \tag{4.6}$$

$$\text{where } Z_i \quad = \quad \sum_{\mathbf{x}} t_i(\mathbf{x})q(\mathbf{x}) \tag{4.7}$$

   Update s via

$$s^{new} = s \times Z_i \tag{4.8}$$

While it appears that we have circumvented the exponential family, in practice we must still use it. The marginals of $\hat{p}$ are feasibly computed and stored only if $\hat{p}$ is in the exponential family, which means $q(\mathbf{x})$ and $t_i(\mathbf{x})$ must also be in the family. The most common use of belief propagation is for discrete $\mathbf{x}$ where $\hat{p}$ is a multinomial distribution.

Now we turn this into an EP algorithm. From the ratio q^{new}/q, we see that the approximate terms $\tilde{t}_i(\mathbf{x})$ are completely disconnected. The EP algorithm is thus:

1. The term approximations have the form

$$\tilde{t}_i(\mathbf{x}) = \prod_k \tilde{t}_{ik}(x_k) \tag{4.9}$$

   Initialize $\tilde{t}_i(\mathbf{x}) = 1$.

2. Compute the posterior for $\mathbf{x}$ from the product of $\tilde{t}_i$:

$$q_k^{new}(x_k) \propto \prod_i \tilde{t}_{ik}(x_k) \tag{4.10}$$

3. Until all $\tilde{t}_i$ converge:

   (a) Choose a $\tilde{t}_i$ to refine

   (b) Remove $\tilde{t}_i$ from the posterior. For all k:

$$q_k(x_k) \propto \frac{q_k^{new}(x_k)}{\tilde{t}_{ik}(x_k)} \tag{4.11}$$

$$\propto \prod_{j \neq i} \tilde{t}_{jk}(x_k) \tag{4.12}$$

   (c) Recompute $q^{new}(\mathbf{x})$ from $q(\mathbf{x})$ as in ADF.

$$q_k^{new}(x_k) = \frac{1}{Z_i} \sum_{\mathbf{x} \setminus x_k} t_i(\mathbf{x}) q(\mathbf{x}) \tag{4.13}$$

$$Z_i = \sum_{\mathbf{x}} t_i(\mathbf{x}) q(\mathbf{x}) \tag{4.14}$$

   (d) Set

$$\tilde{t}_{ik}(x_k) = Z_i \frac{q_k^{new}(x_k)}{q_k(x_k)} = \frac{\sum_{\mathbf{x} \setminus x_k} t_i(\mathbf{x}) q(\mathbf{x})}{q_k(x_k)} \tag{4.15}$$

$$= \sum_{\mathbf{x} \setminus x_k} t_i(\mathbf{x}) \prod_{j \neq k} q_j(x_j) \tag{4.16}$$

4. Compute the normalizing constant:

$$p(D) \approx \prod_k \left(\sum_{x_k} \prod_i \tilde{t}_{ik}(x_k) \right) \tag{4.17}$$

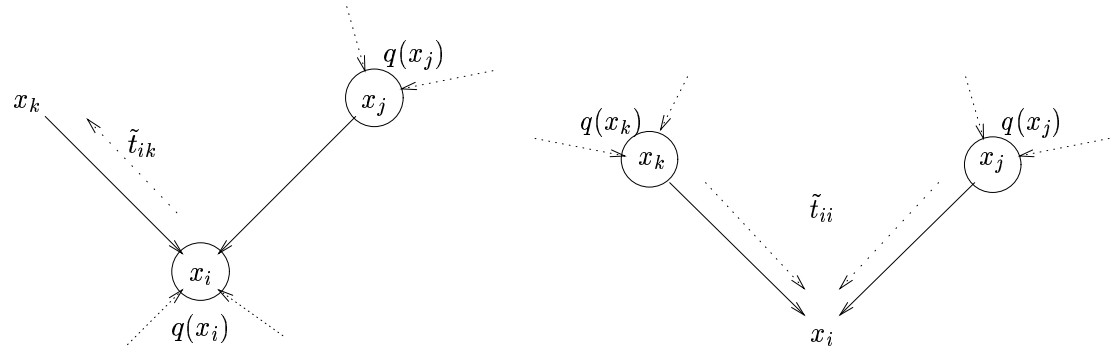
To make this equivalent to belief propagation (see e.g. Murphy et al. (1999)), the original terms t_i should correspond to the conditional probability tables of a directed belief network. That is, we should break the joint distribution $p(D, \mathbf{x})$ into

$$p(D, \mathbf{x}) = \prod_k p(x_k | \mathrm{pa}(x_k)) \prod_j p(y_j | \mathrm{pa}(y_j)) \tag{4.18}$$

where $\mathrm{pa}(X)$ is the set of parents of node X. The network has observed nodes y_j and hidden nodes x_k. The parents of an observed node might be hidden, and vice versa. The quantities in EP now have the following interpretations:

- $q_k^{new}(x_k)$ is the belief state of node x_k, i.e. the product of all messages into x_k.
- The 'old' posterior $q_k(x_k)$ for a particular term i is a partial belief state, i.e. the product of messages into x_k except for those originating from term i.
- When $i \neq k$, the function $\tilde{t}_{ik}(x_k)$ is the message that node i (either hidden or observed) sends to its parent x_k in belief propagation. For example, suppose node i is hidden and $t_i(\mathbf{x}) = p(x_i | \mathrm{pa}(x_i))$. The other parents send $q_j(x_j)$, and the child combines them with $q_i(x_i)$ to get (figure 4-2(a))

$$\tilde{t}_{ik}(y) = \sum_{\mathbf{x} \setminus x_k} p(x_i | \mathrm{pa}(x_i)) q_i(x_i) \prod_{\text{other parents } j} q_j(x_j) \tag{4.19}$$



$\tilde{t}_{ik}(x_k) = \sum_{x_i,x_j} p(x_i|x_k,x_j)q_i(x_i)q_j(x_j)$ (a)

$\tilde{t}_{ii}(x_i) = \sum_{x_k,x_j} p(x_i|x_k,x_j)q_k(x_k)q_j(x_j)$ (b)

Figure 4-2: The messages in belief propagation correspond to term approximations in EP

- When node i is hidden, the function $\tilde{t}_{ii}(x_i)$ is a combination of messages sent to node i from its parents in belief propagation. Each parent sends $q_j(x_j)$, and the child combines them according to (figure 4-2(b))

$$\tilde{t}_{ii}(x_i) = \sum_{\mathrm{pa}(x_i)} p(x_i|\mathrm{pa}(x_i)) \prod_{\text{parents } j} q_j(x_j) \tag{4.20}$$

Unlike the traditional derivation of belief propagation in terms of λ and π messages, this derivation is symmetric with respect to parents and children. In fact, it is the form used in in factor graphs (Kschischang et al., 2000). All of the nodes that participate in a conditional probability table $p(X|\mathrm{pa}(X))$ send messages to each other based on their partial belief states.

This algorithm applies generally to any decomposition into terms $t_i(\mathbf{x})$, not just that given by (4.18). For example, equivalence also holds for undirected networks, if the terms t_i are the clique potentials—compare to the equations in Yedidia et al. (2000). See section 4.2 for an example with an undirected network.

The remaining difference between belief propagation and EP is that belief propagation does not specify the order in which messages are sent. In the EP algorithm above, each node must exchange messages with all of its parents at each step. However, we can relax this condition by considering a simple variation of EP, where we do not minimize KL-divergence completely at each step, but only partially. Specifically, we only update one of the $\tilde{t}_{ik}$ functions each time. With this variation, we can pass messages in any order. This reordering will not change the result at convergence.

This equivalence shows that there is a close connection between Boyen & Koller's algorithm and loopy belief propagation. Boyen & Koller's algorithm for dynamic Bayesian networks is simply one pass of loopy belief propagation where each term t_i is the product of multiple conditional probability tables—all of the tables for a given timeslice. This equivalence was also noticed by Murphy & Weiss (2000), though described in terms of modifications to the graph.

The equivalence also tells us two things about EP:

- Even though EP is derived as an approximation, in some cases the final expectations may be exact. We know this because the marginals resulting from belief propagation

are exact on tree-structured networks. But for other EP algorithms we do not yet know how to predict when this will happen.

- EP may not converge for some models and/or approximating densities. We know this because belief propagation does not always converge on graphs with loops.

Unlike the clutter problem of section 3.2.1, there is no obvious constraint like positive variance that we can use to ensure convergence. The convergence of EP can be helped in two ways. One way is to improve the quality of the approximation. For example, Yedidia et al. (2000) have shown that clustering nodes in a belief network can encourage belief propagation to converge (as well as improve its accuracy). Clustering corresponds to an approximating density that is partially disconnected, such as $q(\mathbf{x}) = q(x_1, x_2)q(x_3, x_4)$. As described in section 4.2.1, we can also cluster some of the terms $t_i(\mathbf{x})$ to get fewer approximations. Or we can use a Markov chain approximation, as described in section 4.2.2.

A second way to improve convergence is to devise a better iteration scheme. Analogous to belief propagation, EP can be derived by minimizing a free energy-like objective. The expression is similar to that of Yedidia et al. (2000) but involving expectation constraints. Different EP iterations can be found by applying different optimization techniques to this objective. (The same applies to belief propagation.) This technique will be described more fully in a separate paper.

4.2 Extensions to belief propagation

This section shows how the flexibility of EP allows us to extend loopy belief propagation for higher accuracy. In ADF/EP there are two decisions to make: how to factor the joint distribution $p(\mathbf{x}, D)$ into terms $t_i(\mathbf{x})$ and what approximating family $q(\mathbf{x})$ to apply. As a running example, consider the undirected network in figure 4-3. The joint distribution of all nodes is proportional to a product of pairwise potentials, one for each edge:

$$p(x_1, x_2, x_3, x_4, x_5, x_6) \propto h(x_1, x_2)h(x_2, x_3)h(x_3, x_4)h(x_1, x_4)h(x_4, x_5)h(x_5, x_6)h(x_3, x_6) \tag{4.21}$$

We want to approximate this distribution with a simpler one.

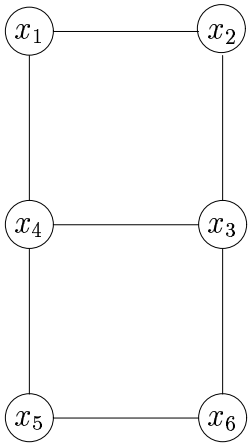


Figure 4-3: An undirected belief network

4.2.1 Grouping terms

Belief propagation arises when the EP terms $t_i(\mathbf{x})$ correspond to the edges of the graph, that is $t_1(\mathbf{x}) = h(x_1, x_2)$, $t_2(\mathbf{x}) = h(x_2, x_3)$, etc. Instead of breaking the joint into seven factors, we could break it into only *two* factors, say

$$t_1(x_1, x_2, x_3, x_4) = h(x_1, x_2)h(x_2, x_3)h(x_3, x_4)h(x_1, x_4) \tag{4.22}$$
$$t_2(x_3, x_4, x_5, x_6) = h(x_4, x_5)h(x_5, x_6)h(x_3, x_6) \tag{4.23}$$

as illustrated in figure 4-4. Let the approximating family $q(\mathbf{x})$ be completely disconnected as before.

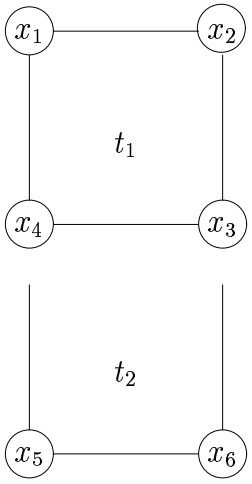


Figure 4-4: Belief propagation can be improved by factoring the joint distribution into subgraphs instead of individual edges

It is sufficient to consider the ADF updates. Start with a uniform distribution on $\mathbf{x}$ (assuming all variables are binary):

$$q(\mathbf{x}) = \frac{1}{2^6} \tag{4.24}$$

The first step of ADF will be

$$Z = \sum_{\mathbf{x}} t_1(x_1, x_2, x_3, x_4)q(\mathbf{x}) \tag{4.25}$$
$$\hat{p}(\mathbf{x}) = \frac{1}{Z} t_1(x_1, x_2, x_3, x_4)q(\mathbf{x}) \tag{4.26}$$
$$q_1^{new}(x_1) = \hat{p}(x_1) = \frac{1}{Z} \sum_{\mathbf{x}\setminus x_1} t_1(x_1, x_2, x_3, x_4)q(\mathbf{x}) \tag{4.27}$$
$$q_2^{new}(x_2) = \frac{1}{Z} \sum_{\mathbf{x}\setminus x_2} t_1(x_1, x_2, x_3, x_4)q(\mathbf{x}) \tag{4.28}$$
$$\vdots \tag{4.29}$$
$$q_5^{new}(x_5) = q_5(x_5) \qquad \text{(unchanged)} \tag{4.30}$$
$$q_6^{new}(x_6) = q_6(x_6) \qquad \text{(unchanged)} \tag{4.31}$$

The second step of ADF will be

$$
\begin{aligned}
Z &= \sum_{\mathbf{x}} t_2(x_3, x_4, x_5, x_6) q(\mathbf{x}) && (4.32)\\
\hat{p}(\mathbf{x}) &= \frac{1}{Z} t_2(x_3, x_4, x_5, x_6) q(\mathbf{x}) && (4.33)\\
q_1^{new}(x_1) &= q_1(x_1) \qquad \text{(unchanged)} && (4.34)\\
q_2^{new}(x_2) &= q_2(x_2) \qquad \text{(unchanged)} && (4.35)\\
q_3^{new}(x_3) &= \frac{1}{Z} \sum_{\mathbf{x} \setminus x_3} t_2(x_3, x_4, x_5, x_6) q(\mathbf{x}) && (4.36)\\
&\vdots && (4.37)
\end{aligned}
$$

In the usual way, these ADF updates can be turned into an EP algorithm, which will be more accurate than belief propagation. However, it will involve significantly more work, since each message requires marginalizing three variables (the number of variables in each term, minus one) instead of marginalizing just one variable.

4.2.2 Partially disconnected approximations

Besides the factoring of the joint into individual edges, belief propagation arises when the approximating network is completely disconnected: $q(\mathbf{x}) = q_1(x_1)q_2(x_2)\cdots$. Instead of going to this extreme, we can use a Markov chain approximation, as shown in figure 4-5:

$$q(\mathbf{x}) = q(x_1)q(x_2|x_1)q(x_3|x_2)q(x_4|x_3)q(x_5|x_4)q(x_6|x_5) \tag{4.38}$$

Let the terms $t_i(\mathbf{x})$ be the individual edges, as in belief propagation. Partially disconnected approximations have previously been used in ADF by Frey et al. (2000) and in the context of variational bounds by Ghahramani & Jordan (1997), Barber & Wiegerinck (1998), and references therein.

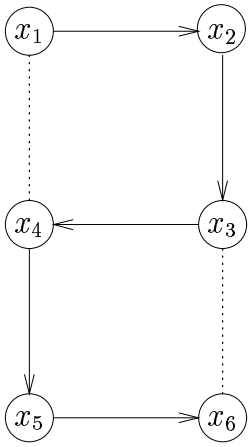


Figure 4-5: Belief propagation can also be improved by using a Markov chain approximation instead of a completely disconnected approximation

Note that we could equally well parameterize the Markov chain as

$$q(\mathbf{x}) = q(x_1|x_2)q(x_2|x_3)q(x_3)q(x_4|x_3)q(x_5|x_4)q(x_6|x_5) \tag{4.39}$$

Before, x_1 was the root; now x_3 is the root. Given a distribution in one parameterization, it is straightforward to convert to another parameterization. This will be called *changing the root.*

In ADF, we incorporate edges $h(x_i, x_j)$ from the original graph one at a time. We take the exact posterior

$$\hat{p}(\mathbf{x}) = \frac{t_i(\mathbf{x})q(\mathbf{x})}{\sum_{\mathbf{x}} t_i(\mathbf{x})q(\mathbf{x})} \tag{4.40}$$

and minimize the KL-divergence $D(\hat{p}(\mathbf{x})||q^{new}(\mathbf{x}))$ subject to the constraint that $q^{new}(\mathbf{x})$ has the form (4.38). Zeroing the gradient with respect to q^{new} gives the conditions

$$\begin{aligned} q^{new}(x_1) &= \hat{p}(x_1) && (4.41)\\ q^{new}(x_2|x_1) &= \hat{p}(x_2|x_1) && (4.42)\\ q^{new}(x_3|x_2) &= \hat{p}(x_3|x_2) && (4.43)\\ &\vdots \end{aligned}$$

These directly generalize the marginal constraints (4.3) for belief propagation.

The approximation we have chosen preserves all edges in the graph except two. The preserved edges can be incorporated into $q(\mathbf{x})$ without approximation:

$$q(\mathbf{x}) \propto h(x_1, x_2)h(x_2, x_3)h(x_3, x_4)h(x_4, x_5)h(x_5, x_6) \tag{4.44}$$

Now we process the edge $h(x_1, x_4)$. The marginals of the exact posterior are

$$\begin{aligned} Z &= \sum_{\mathbf{x}} h(x_1, x_4)q(\mathbf{x}) && (4.45)\\ &= \sum_{x_1,x_2,x_3,x_4} h(x_1, x_4)q(x_1)q(x_2|x_1)q(x_3|x_2)q(x_4|x_3) && (4.46)\\ \hat{p}(x_1) &= \frac{1}{Z}\sum_{\mathbf{x}\setminus x_1} h(x_1, x_4)q(\mathbf{x}) && (4.47)\\ &= \frac{1}{Z}\sum_{x_2,x_3,x_4} h(x_1, x_4)q(x_1)q(x_2|x_1)q(x_3|x_2)q(x_4|x_3) && (4.48)\\ \hat{p}(x_2, x_1) &= \frac{1}{Z}\sum_{\mathbf{x}\setminus(x_1,x_2)} h(x_1, x_4)q(\mathbf{x}) && (4.49)\\ &= \frac{1}{Z}\sum_{x_3,x_4} h(x_1, x_4)q(x_1)q(x_2|x_1)q(x_3|x_2)q(x_4|x_3) && (4.50)\\ \hat{p}(x_2|x_1) &= \frac{\hat{p}(x_2, x_1)}{\hat{p}(x_1)} && (4.51)\\ &\vdots \quad \text{(same for } x_3 \text{ and } x_4\text{)}\\ \hat{p}(x_5|x_4) &= q(x_5|x_4) && (4.52)\\ \hat{p}(x_6|x_5) &= q(x_6|x_5) && (4.53) \end{aligned}$$

There is a simple pattern going on here. Adding the edge (x_1, x_4) creates a loop in $\hat{p}$, involving the nodes x_1, ..., x_4, which include the root. The conditional distributions for nodes outside the loop do not change. The marginal for a node inside the loop only involves summing over the rest of the loop, and similarly for pairwise marginals.

This leads us to a general update rule:

1. To incorporate an edge $h(x_a, x_b)$, first change the root to x_a.

2. Let $\mathbf{x}_L$ be the set of nodes on the loop created by adding (x_a, x_b).

$$Z = \sum_{\mathbf{x}_L} h(x_a, x_b) q(\mathbf{x}_L) \tag{4.54}$$

   For nodes x_i on the loop:

$$\hat{p}(x_i) = \frac{1}{Z} \sum_{\mathbf{x}_L \setminus x_i} h(x_a, x_b) q(\mathbf{x}_L) \tag{4.55}$$

   For edges (x_i, x_j) on the loop:

$$\hat{p}(x_i, x_j) = \frac{1}{Z} \sum_{\mathbf{x}_L \setminus (x_i, x_j)} h(x_a, x_b) q(\mathbf{x}_L) \tag{4.56}$$

$$q^{new}(x_i|x_j) = \hat{p}(x_i, x_j)/\hat{p}(x_j) \tag{4.57}$$

   For a large loop, these sums can be computed efficiently via forward-backward recursions.

3. For edges (x_i, x_j) not on the loop:

$$q^{new}(x_i|x_j) = q(x_i|x_j) \tag{4.58}$$

Returning to the example, after processing $h(x_1, x_4)$ the pairwise marginal for (x_3, x_4) under $q(\mathbf{x})$ will be

$$q(x_3, x_4) \propto \sum_{x_1, x_2} h(x_1, x_4) h(x_1, x_2) h(x_2, x_3) h(x_3, x_4) \tag{4.59}$$

This is a step forward from section 4.2.1, where only the marginals $q(x_3)$ and $q(x_4)$ were retained between terms. After processing the second edge, $h(x_3, x_6)$, the pairwise marginal will be

$$q^{new}(x_3, x_4) \propto q(x_3, x_4) \sum_{x_5, x_6} h(x_4, x_5) h(x_5, x_6) h(x_3, x_6) \tag{4.60}$$

$$= \left(\sum_{x_1, x_2} h(x_1, x_4) h(x_1, x_2) h(x_2, x_3) h(x_3, x_4) \right) \left(\sum_{x_5, x_6} h(x_4, x_5) h(x_5, x_6) h(x_3, x_6) \right) \tag{4.61}$$

This is an interesting result: it is the exact marginal for (x_3, x_4) in the original graph! The same is true for $q^{new}(x_4, x_5)$ and $q^{new}(x_5, x_6)$. This is after processing both edges with ADF. If we use EP to re-process the original edge, then the exact $q^{new}(x_3, x_4)$ will be used to recompute $q^{new}(x_1, x_2)$ and $q^{new}(x_2, x_3)$, which will also be exact. Thus by moving from a completely disconnected approximation to a Markov chain, we move from having approximate marginals to having exact marginals throughout the entire chain. Obviously this is a special property of the graph and the approximation we have chosen, and we would not expect it in general. But it demonstrates how effective partially disconnected approximations can be.

4.2.3 Combining both extensions

The extensions described in the last two sections can be combined in the same algorithm. Consider the two-term grouping

$$t_1(x_1, x_2, x_3, x_4) = h(x_1, x_2)h(x_2, x_3)h(x_3, x_4)h(x_1, x_4) \quad (4.62)$$

$$t_2(x_3, x_4, x_5, x_6) = h(x_4, x_5)h(x_5, x_6)h(x_3, x_6) \quad (4.63)$$

along with the clustered approximation

$$q(\mathbf{x}) = q(x_1, x_2)q(x_3, x_4)q(x_5, x_6) \quad (4.64)$$

as shown in figure 4-6.

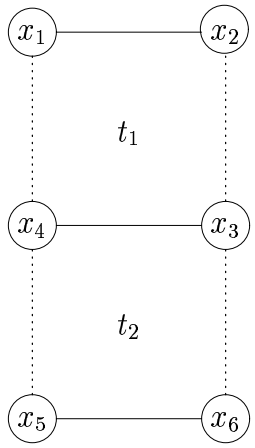


Figure 4-6: We can group edges in addition to using a partially disconnected approximation

Starting from a uniform $q(\mathbf{x})$, the first step of ADF will be

$$Z = \sum_{\mathbf{x}} t_1(x_1, x_2, x_3, x_4)q(\mathbf{x}) \quad (4.65)$$

$$= \sum_{x_1,x_2,x_3,x_4} t_1(x_1, x_2, x_3, x_4)q(x_1, x_2)q(x_3, x_4) \quad (4.66)$$

$$\hat{p}(\mathbf{x}) = \frac{1}{Z} t_1(x_1, x_2, x_3, x_4)q(\mathbf{x}) \quad (4.67)$$

$$q^{new}(x_1, x_2) = \frac{q(x_1, x_2)}{Z} \sum_{x_3,x_4} t_1(x_1, x_2, x_3, x_4)q(x_3, x_4) \quad (4.68)$$

$$q^{new}(x_3, x_4) = \frac{q(x_3, x_4)}{Z} \sum_{x_1,x_2} t_1(x_1, x_2, x_3, x_4)q(x_1, x_2) \quad (4.69)$$

$$q^{new}(x_5, x_6) = q(x_5, x_6) \quad (4.70)$$

The second step will be

$$Z = \sum_{\mathbf{x}} t_2(x_3, x_4, x_5, x_6)q(\mathbf{x}) \quad (4.71)$$

$$= \sum_{x_3,x_4,x_5,x_6} t_2(x_3, x_4, x_5, x_6)q(x_3, x_4)q(x_5, x_6) \quad (4.72)$$

$$\hat{p}(\mathbf{x}) = \frac{1}{Z} t_2(x_3, x_4, x_5, x_6) q(\mathbf{x}) \tag{4.73}$$

$$q^{new}(x_1, x_2) = q(x_1, x_2) \tag{4.74}$$

$$q^{new}(x_3, x_4) = \frac{q(x_3, x_4)}{Z} \sum_{x_5, x_6} t_2(x_3, x_4, x_5, x_6) q(x_5, x_6) \tag{4.75}$$

$$q^{new}(x_5, x_6) = \frac{q(x_5, x_6)}{Z} \sum_{x_3, x_4} t_2(x_3, x_4, x_5, x_6) q(x_3, x_4) \tag{4.76}$$

Once again, $q^{new}(x_3, x_4)$ is the exact pairwise marginal in the original graph. So is $q^{new}(x_5, x_6)$, and after one refinement pass $q^{new}(x_1, x_2)$ will also be exact. In this case, it is clear why the results are exact: the algorithm is equivalent to running belief propagation on a clustered graph where the pairs (x_1, x_2), (x_3, x_4), and (x_5, x_6) are merged into super-nodes.

Interestingly, if we did not group terms, and used individual edges, the algorithm would be identical to belief propagation on the remaining edges, and we would have gained little from making the clustered approximation. This is because adding an edge to q does not create a loop in $\hat{p}$. EP approximates each edge by a ratio q^{new}/q. If the edge joins two disconnected clusters in q, then there is no way for EP to represent the correlation induced by the edge.

4.2.4 Future work

This section showed that improvements to belief propagation were possible if we grouped terms, used a partially disconnected approximation, or both. For a general graph, it would be useful to know what combination of grouping and disconnection gives the highest accuracy for the least cost, i.e. which is most *efficient*. Determining the cost of a given grouping/disconnection scheme is straightforward, but estimating the accuracy is not. For example, as described in the last subsection, it is not a good idea to break the graph into disconnected subgraphs, *unless* terms are grouped appropriately. Similar complexities arise when choosing the right approximating structure for variational methods (Jordan et al., 1999). This is a useful area for future work.

Chapter 5

Classification using the Bayes Point

This chapter applies Expectation Propagation to inference in the Bayes Point Machine. The Bayes Point Machine is a Bayesian approach to linear classification that competes with the popular but non-Bayesian Support Vector Machine (SVM). Bayesian averaging of linear classifiers has been proven both theoretically and empirically optimal in terms of generalization performance (Watkin, 1993; Bouten et al., 1995; Buhot et al., 1997). But an efficient algorithm has remained elusive. The Bayes Point Machine approximates the Bayesian average by choosing one 'average' classifier, the Bayes Point (Watkin, 1993; Rujan, 1997; Herbrich et al., 1999). Computing the Bayes Point is a simpler, but still very difficult task.

This chapter shows that Expectation Propagation, using a full-covariance Gaussian approximation to the posterior, provides an accurate estimate of the Bayes Point—an approximation to the full Bayesian average. The algorithm turns out to be identical to what Opper & Winther (2000c) derived by statistical physics methods. However, EP uses a different optimization scheme that is faster and does not require a stepsize parameter. EP also provides an estimate of the evidence $p(D)$, which is useful for feature selection but not provided by Opper & Winther.

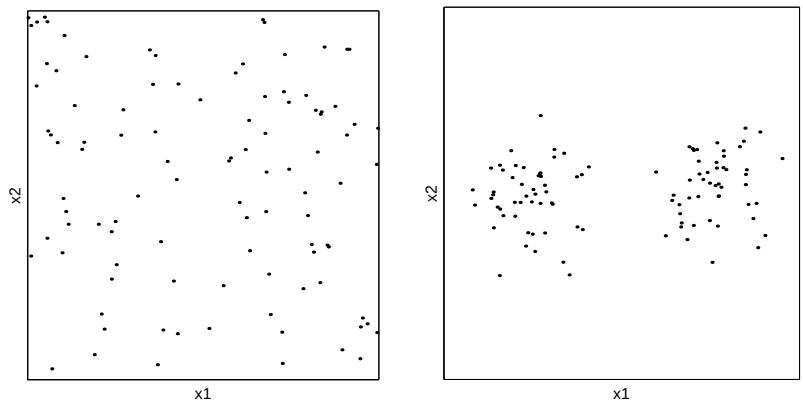
5.1 The Bayes Point Machine

The Bayes Point Machine (BPM) is a Bayesian approach to linear classification. A linear classifier classifies a point $\mathbf{x}$ according to $y = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$ for some parameter vector $\mathbf{w}$ (the two classes are $y = \pm 1$). Given a training set $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, the likelihood for $\mathbf{w}$ can be written

$$p(D|\mathbf{w}) = \prod_i p(y_i|\mathbf{x}_i, \mathbf{w}) = \prod_i \Theta(y_i \mathbf{w}^{\mathrm{T}}\mathbf{x}_i) \tag{5.1}$$

$$\Theta(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases} \tag{5.2}$$

This is a slight abuse of notation since $p(D|\mathbf{w})$ is only a distribution for y, not $\mathbf{x}$. This likelihood is 1 if $\mathbf{w}$ is a perfect separator and 0 otherwise. A simple way to achieve linear separation is to progressively enlarge the set of features, e.g. by computing squares and third powers, until the data is separable in the new feature space. This has the effect of producing a nonlinear decision boundary in the original measurement space. Feature expansion can be implemented efficiently using the 'kernel trick', where we rewrite the algorithm in terms



(a) (b)

Figure 5-1: Two different classification data sets, with label information removed. The Bayes Point Machine assumes that, without labels, we cannot infer where the boundary lies. In (b), one might be inclined to think that the horizontal axis determines the decision. But the horizontal axis could easily be something irrelevant, like the time of day the datum was collected. Only domain knowledge can determine the validity of the assumption.

of inner products and perform the feature expansion implicitly via the inner product. This technique will be described more fully later.

Because it is conditional on $\mathbf{x}_i$, this model assumes that the distribution of the $\mathbf{x}$'s is unrelated to the true decision boundary. That is, if we were given the $\mathbf{x}$'s without any labels, we would have no information about where the boundary lies. The validity of this assumption must come from domain knowledge—it cannot be determined from the data itself (see figure 5-1). The Support Vector Machine (SVM), for example, does not seem to make this assumption; Tong & Koller (2000) show that the SVM follows from assuming that the $\mathbf{x}$'s follow a Gaussian mixture distribution in each class. In addition, some workers have used the SVM with unlabeled data (Bennett & Demiriz, 1998), while the BPM, by design, cannot. The advantage of assuming independence in the BPM is that we avoid stronger assumptions about the nature of the dependence, and thus obtain a more general algorithm. Experiments show it is quite robust (section 5.6).

A refinement to the basic model is to admit the possibility of errors in labeling or measurement. A labeling error rate of ϵ can be modeled using (Opper & Winther, 2000b)

$$p(y|\mathbf{x},\mathbf{w}) = \epsilon(1-\Theta(y\mathbf{w}^{\mathrm{T}}\mathbf{x})) + (1-\epsilon)\Theta(y\mathbf{w}^{\mathrm{T}}\mathbf{x}) \quad (5.3)$$

$$= \epsilon + (1-2\epsilon)\Theta(y\mathbf{w}^{\mathrm{T}}\mathbf{x}) \quad (5.4)$$

Under this model, the likelihood $p(D|\mathbf{w})$ for a given $\mathbf{w}$ will be $\epsilon^r(1-\epsilon)^{n-r}$, where r is the number of errors on the training set. Only the number of errors is important, not where they are. This way of modeling errors can automatically reject outliers, because it doesn't care how far an error is from the boundary. Other approaches based on adding 'slack' to the boundary only allow errors near the boundary and are sensitive to outliers. These approaches can be simulated by using a smooth likelihood function such as

$$p(y|\mathbf{x},\mathbf{w}) = \phi(y\mathbf{w}^{\mathrm{T}}\mathbf{x}) \quad (5.5)$$

where ϕ is the cumulative distribution function for a Gaussian. The tails of this function fall like $\exp(-x^2)$, so it provides 'quadratic slack'. The logistic function could also be

used; it falls like $\exp(-x)$, providing 'linear slack'. By changing Θ to one of these smooth functions, the labeling error model (5.3) can be combined with slack, with no computational difficulties. However, this chapter focuses on (5.3) without slack.

To complete our Bayesian model, we need to specify a prior on $\mathbf{w}$. Since the magnitude of $\mathbf{w}$ is irrelevant for classification, some authors have restricted $\mathbf{w}$ to the unit sphere in d dimensions, and have given $\mathbf{w}$ a uniform distribution on the sphere (Rujan, 1997). This distribution is nice because it is fair to all decision boundaries, but it is not very convenient to work with. A more general non-informative prior is an average over spheres. Let $r = \|\mathbf{w}\|$ be the magnitude of $\mathbf{w}$. Given r, let $\mathbf{w}$ be uniformly distributed on the sphere of radius r. This prior is non-informative for any r. It is also non-informative if r is unknown with distribution $p(r)$. For example, $p(r)$ could be uniform from 0 to 1, allowing $\mathbf{w}$ to live in the unit ball instead of on the unit sphere. By choosing $p(r)$ appropriately, we can also give $\mathbf{w}$ a spherical Gaussian distribution:

$$p(\mathbf{w}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{5.6}$$

It is a property of the spherical Gaussian distribution that, conditional on any r, $\mathbf{w}$ is uniformly distributed on the sphere of radius r, so no particular decision boundary is favored by this prior.

Given this model, the optimal way to classify a new data point $\mathbf{x}$ is to use the predictive distribution for y given the training data:

$$p(y|\mathbf{x}, D) = \int_{\mathbf{w}} p(y|\mathbf{x}, \mathbf{w}) p(\mathbf{w}|D) d\mathbf{w} \tag{5.7}$$

However, this requires solving an integral each time. A practical substitute is to use a single value of $\mathbf{w}$ which approximates the predictive distribution as well as possible. Watkin (1993) and Rujan (1997) have defined the 'Bayes point' as the single best $\mathbf{w}$. However, this point is difficult to find. Thus, following Rujan (1997), we use the posterior mean for $\mathbf{w}$, $E[\mathbf{w}|D]$. Following convention, we will be sloppy and also call this the 'Bayes point'. Our strategy will be to make a Gaussian approximation to the posterior and use its mean as the estimated Bayes point. Of course, if we really wanted to, we could also use EP to solve the integral (5.7) for each test point. The normalizing constant of the posterior is also useful because, as an estimate of $p(D)$, it allows us to choose among different models (see section 5.7).

5.2 Training via ADF

Start with the ADF updates. ADF was previously applied to the Bayes Point Machine by Csato et al. (1999). Divide the joint distribution $p(D, \mathbf{w})$ into n terms, one for each data point. Since y and $\mathbf{x}$ always appear multiplied together, the formulas will drop y and assume that $\mathbf{x}_i$ is already scaled by y_i. Let the approximate posterior have the form

$$q(\mathbf{w}) \sim \mathcal{N}(\mathbf{m}_w, \mathbf{V}_w) \tag{5.8}$$

From minimizing the KL-divergence we get the expectation constraints

$$E_{q^{new}}[\mathbf{w}] = E_{\hat{p}}[\mathbf{w}] \tag{5.9}$$

$$E_{q^{new}}[\mathbf{w}\mathbf{w}^{\mathrm{T}}] = E_{\hat{p}}[\mathbf{w}\mathbf{w}^{\mathrm{T}}] \tag{5.10}$$

To compute these expectations, use the following relations (obtained from integration-by-parts):

$$
\begin{aligned}
Z(\mathbf{m}_w, \mathbf{V}_w) &= \int_{\mathbf{w}} t(\mathbf{w}) q(\mathbf{w}) \; d\mathbf{w} && (5.11)\\
E_{\hat{p}}[\mathbf{w}] &= \mathbf{m}_w + \mathbf{V}_w \nabla_m \log Z(\mathbf{m}_w, \mathbf{V}_w) && (5.12)\\
E_{\hat{p}}[\mathbf{w}\mathbf{w}^{\mathrm{T}}] - E_{\hat{p}}[\mathbf{w}] E_{\hat{p}}[\mathbf{w}]^{\mathrm{T}} &= \mathbf{V}_w - \mathbf{V}_w (\nabla_m \nabla_m^{\mathrm{T}} - 2\nabla_v \log Z(\mathbf{m}_w, \mathbf{V}_w)) \mathbf{V}_w && (5.13)
\end{aligned}
$$

These hold for any $t(\mathbf{w})$. For the Bayes Point Machine, we have (combining $y\mathbf{x}$ into $\mathbf{x}$):

$$
\begin{aligned}
t(\mathbf{w}) &= \epsilon + (1 - 2\epsilon)\Theta(\mathbf{w}^{\mathrm{T}}\mathbf{x}) && (5.14)\\
\phi(z) &= \int_{-\infty}^{z} \mathcal{N}(z; 0, 1) dz && (5.15)\\
z &= \frac{\mathbf{m}_w^{\mathrm{T}}\mathbf{x}}{\sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{V}_w\mathbf{x}}} && (5.16)\\
Z(\mathbf{m}_w, \mathbf{V}_w) &= \epsilon + (1 - 2\epsilon)\phi(z) && (5.17)\\
\alpha &= \frac{1}{\sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{V}_w\mathbf{x}}} \frac{(1 - 2\epsilon)\mathcal{N}(z; 0, 1)}{\epsilon + (1 - 2\epsilon)\phi(z)} && (5.18)\\
\nabla_m \log Z(\mathbf{m}_w, \mathbf{V}_w) &= \alpha\mathbf{x} && (5.19)\\
\nabla_v \log Z(\mathbf{m}_w, \mathbf{V}_w) &= -\frac{1}{2} \frac{\alpha \mathbf{m}_w^{\mathrm{T}}\mathbf{x}}{\mathbf{x}^{\mathrm{T}}\mathbf{V}_w\mathbf{x}} \mathbf{x}\mathbf{x}^{\mathrm{T}} && (5.20)\\
\nabla_m \nabla_m^{\mathrm{T}} - 2\nabla_v \log Z(\mathbf{m}_w, \mathbf{V}_w) &= \alpha^2 \mathbf{x}\mathbf{x}^{\mathrm{T}} + \frac{\alpha \mathbf{m}_w^{\mathrm{T}}\mathbf{x}}{\mathbf{x}^{\mathrm{T}}\mathbf{V}_w\mathbf{x}} \mathbf{x}\mathbf{x}^{\mathrm{T}} && (5.21)\\
&= \frac{\alpha(\mathbf{m}_w + \mathbf{V}_w\alpha\mathbf{x})^{\mathrm{T}}\mathbf{x}}{\mathbf{x}^{\mathrm{T}}\mathbf{V}_w\mathbf{x}} \mathbf{x}\mathbf{x}^{\mathrm{T}} && (5.22)
\end{aligned}
$$

The ADF algorithm is thus:

1. Initialize $\mathbf{m}_w = \mathbf{0}$, $\mathbf{V}_w = \mathbf{I}$ (the prior). Initialize $s = 1$ (the scale factor).
2. For each data point $\mathbf{x}_i$, update $(\mathbf{m}_w, \mathbf{V}_w, s)$ according to

$$
\begin{aligned}
\mathbf{m}_w^{new} &= \mathbf{m}_w + \mathbf{V}_w \alpha_i \mathbf{x}_i && (5.23)\\
\mathbf{V}_w^{new} &= \mathbf{V}_w - (\mathbf{V}_w\mathbf{x}_i) \left(\frac{\alpha_i \mathbf{x}_i^{\mathrm{T}} \mathbf{m}_w^{new}}{\mathbf{x}_i^{\mathrm{T}}\mathbf{V}_w\mathbf{x}_i} \right) (\mathbf{V}_w\mathbf{x}_i)^{\mathrm{T}} && (5.24)\\
s^{new} &= s \times Z_i(\mathbf{m}_w, \mathbf{V}_w) && (5.25)
\end{aligned}
$$

The final $\mathbf{m}_w$ is the estimate of $\mathbf{w}$ used for classification. Note that this algorithm does not require matrix inversion. It is the same algorithm obtained by Csato et al. (1999), except they expressed it in terms of kernel inner products, which makes it more complex.

5.3 Training via EP

Now we turn the ADF algorithm into an EP algorithm. By dividing two consecutive Gaussian posteriors, we find that the term approximations are also Gaussian, parameterized by $\mathbf{m}_i$ and $\mathbf{V}_i$:

$$
\tilde{t}(\mathbf{w}) = Z \frac{q^{new}(\mathbf{w})}{q(\mathbf{w})} \qquad (5.26)
$$

$$= \frac{Z}{\mathcal{N}(\mathbf{m}_i; \mathbf{m}_w; \mathbf{V}_i + \mathbf{V}_w)} \mathcal{N}(\mathbf{w}; \mathbf{m}_i, \mathbf{V}_i) \tag{5.27}$$

$$\text{where } \mathbf{V}_i^{-1} = (\mathbf{V}_w^{new})^{-1} - \mathbf{V}_w^{-1} \tag{5.28}$$

$$\mathbf{m}_i = \mathbf{V}_i (\mathbf{V}_w^{new})^{-1} \mathbf{m}_w^{new} - \mathbf{V}_i \mathbf{V}_w^{-1} \mathbf{m}_w \tag{5.29}$$

$$= \mathbf{m}_w + (\mathbf{V}_i + \mathbf{V}_w) \mathbf{V}_w^{-1} (\mathbf{m}_w^{new} - \mathbf{m}_w) \tag{5.30}$$

To simplify this, combine it with the ADF updates (5.23,5.24) to get $(\mathbf{m}_i, \mathbf{V}_i)$ directly from $(\mathbf{m}_w, \mathbf{V}_w)$:

$$\mathbf{V}_i = \left(\frac{\alpha_i \mathbf{x}_i^{\mathrm{T}} \mathbf{m}_w^{new}}{\mathbf{x}_i^{\mathrm{T}} \mathbf{V}_w \mathbf{x}_i} \mathbf{x}_i \mathbf{x}_i^{\mathrm{T}} \right)^{-1} - \mathbf{V}_w \tag{5.31}$$

$$\mathbf{m}_i = \mathbf{m}_w + (\mathbf{V}_i + \mathbf{V}_w) \alpha_i \mathbf{x}_i \tag{5.32}$$

The matrix $\frac{\alpha_i \mathbf{x}_i^{\mathrm{T}} \mathbf{m}_w^{new}}{\mathbf{x}_i^{\mathrm{T}} \mathbf{V}_w \mathbf{x}_i} \mathbf{x}_i \mathbf{x}_i^{\mathrm{T}}$ has one nonzero eigenvalue, which means $\mathbf{V}_i$ will have one finite eigenvalue, in the direction of $\mathbf{x}_i$. This makes sense because term i only constrains the projection of $\mathbf{w}$ along $\mathbf{x}_i$. This special structure allows us to represent $\mathbf{V}_i$ with a scalar, v_i:

$$\mathbf{V}_i^{-1} = v_i^{-1} \mathbf{x}_i \mathbf{x}_i^{\mathrm{T}} \tag{5.33}$$

$$\mathbf{x}_i^{\mathrm{T}} \mathbf{V}_i \mathbf{x}_i = v_i \tag{5.34}$$

From (5.31), we know that

$$\mathbf{x}_i^{\mathrm{T}} \mathbf{V}_i \mathbf{x}_i = \frac{\mathbf{x}_i^{\mathrm{T}} \mathbf{V}_w \mathbf{x}_i}{\alpha_i \mathbf{x}_i^{\mathrm{T}} \mathbf{m}_w^{new}} - \mathbf{x}_i^{\mathrm{T}} \mathbf{V}_w \mathbf{x}_i \tag{5.35}$$

which means the update for v_i is

$$v_i = \mathbf{x}_i^{\mathrm{T}} \mathbf{V}_w \mathbf{x}_i \left(\frac{1}{\alpha_i \mathbf{x}_i^{\mathrm{T}} \mathbf{m}_w^{new}} - 1 \right) \tag{5.36}$$

Another consequence of this special structure is that instead of the full vector $\mathbf{m}_i$, we only need the projection $\mathbf{m}_i^{\mathrm{T}} \mathbf{x}_i$, which can be stored as a scalar m_i.

Now we can write an efficient EP algorithm:

1. Except for the prior, the term approximations have the form

$$\tilde{t}_i(\mathbf{w}) = s_i \exp(-\frac{1}{2v_i}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i - m_i)^2) \quad (5.37)$$

Initialize them to 1:

$$\begin{aligned} v_i &= \infty \quad (5.38) \\ m_i &= 0 \quad (5.39) \\ s_i &= 1 \quad (5.40) \end{aligned}$$

2. $\mathbf{m}_w^{new} = \mathbf{0}, \mathbf{V}_w^{new} = \mathbf{I}$ (the prior)

3. Until all (m_i, v_i, s_i) converge (changes are less than 10^{-4}):

   loop $i = 1, ..., n$:

   (a) Remove $\tilde{t}_i$ from the posterior to get an 'old' posterior:

$$\begin{aligned} \mathbf{V}_w &= ((\mathbf{V}_w^{new})^{-1} - v_i^{-1}\mathbf{x}_i\mathbf{x}_i^{\mathrm{T}})^{-1} && (5.41) \\ &= \mathbf{V}_w^{new} + (\mathbf{V}_w^{new}\mathbf{x}_i)(v_i - \mathbf{x}_i^{\mathrm{T}}\mathbf{V}_w^{new}\mathbf{x}_i)^{-1}(\mathbf{V}_w^{new}\mathbf{x}_i)^{\mathrm{T}} && (5.42) \\ \mathbf{m}_w &= \mathbf{m}_w^{new} + \mathbf{V}_w\mathbf{V}_i^{-1}(\mathbf{m}_w^{new} - \mathbf{m}_i) && (5.43) \\ &= \mathbf{m}_w^{new} + (\mathbf{V}_w\mathbf{x}_i)v_i^{-1}(\mathbf{x}_i^{\mathrm{T}}\mathbf{m}_w^{new} - m_i) && (5.44) \end{aligned}$$

   These expressions involving $\mathbf{V}_w$ can be computed efficiently:

$$\begin{aligned} \mathbf{V}_w\mathbf{x}_i &= (\mathbf{V}_w^{new}\mathbf{x}_i)\frac{v_i}{v_i - \mathbf{x}_i^{\mathrm{T}}\mathbf{V}_w^{new}\mathbf{x}_i} && (5.45) \\ \mathbf{x}_i^{\mathrm{T}}\mathbf{V}_w\mathbf{x}_i &= \left(\frac{1}{\mathbf{x}_i^{\mathrm{T}}\mathbf{V}_w^{new}\mathbf{x}_i} - \frac{1}{v_i}\right)^{-1} && (5.46) \end{aligned}$$

   (b) Recompute $(\mathbf{m}_w^{new}, \mathbf{V}_w^{new}, Z_i)$ from $(\mathbf{m}_w, \mathbf{V}_w)$, as in ADF.

   (c) Update $\tilde{t}_i$:

$$\begin{aligned} v_i &= \mathbf{x}_i^{\mathrm{T}}\mathbf{V}_w\mathbf{x}_i\left(\frac{1}{\alpha_i\mathbf{x}_i^{\mathrm{T}}\mathbf{m}_w^{new}} - 1\right) && (5.47) \\ m_i &= \mathbf{x}_i^{\mathrm{T}}\mathbf{m}_w + (v_i + \mathbf{x}_i^{\mathrm{T}}\mathbf{V}_w\mathbf{x}_i)\alpha_i && (5.48) \\ s_i &= Z_i\frac{|\mathbf{V}_i + \mathbf{V}_w|^{1/2}}{|\mathbf{V}_i|^{1/2}}\exp(\frac{1}{2}(\mathbf{m}_i - \mathbf{m}_w)^{\mathrm{T}}(\mathbf{V}_i + \mathbf{V}_w)^{-1}(\mathbf{m}_i - \mathbf{m}_w)) && (5.49) \\ &= Z_i\sqrt{1 + v_i^{-1}\mathbf{x}_i^{\mathrm{T}}\mathbf{V}_w\mathbf{x}_i}\exp(\frac{1}{2}\frac{\mathbf{x}_i^{\mathrm{T}}\mathbf{V}_w\mathbf{x}_i}{\mathbf{x}_i^{\mathrm{T}}\mathbf{m}_w^{new}}\alpha_i) && (5.50) \end{aligned}$$

4. Compute the normalizing constant:

$$\begin{aligned} B &= (\mathbf{m}_w^{new})^{\mathrm{T}}(\mathbf{V}_w^{new})^{-1}(\mathbf{m}_w^{new}) - \sum_i \frac{m_i^2}{v_i} && (5.51) \\ p(D) &\approx |\mathbf{V}_w^{new}|^{1/2}\exp(B/2)\prod_{i=1}^{n} s_i && (5.52) \end{aligned}$$

This algorithm processes each data point in $O(d^2)$ time. Assuming the number of iterations is constant, which seems to be true in practice, computing the Bayes point therefore takes $O(nd^2)$ time. Computing the normalizing constant at the end requires $O(d^3)$ time.

5.4 EP with kernels

To use the kernel trick, we need to rewrite EP in terms of inner products. Following the notation of Opper & Winther (2000c), define

$$\begin{aligned}
\lambda_i &= \mathbf{x}_i^{\mathrm{T}} \mathbf{V}_w^{\backslash i} \mathbf{x}_i \qquad (\mathbf{V}_w^{\backslash i} \text{ is the } \mathbf{V}_w \text{ when term } i \text{ is left out}) && (5.53)\\
C_{ij} &= \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j && (5.54)\\
\Lambda &= \mathrm{diag}(v_1, ..., v_n) && (5.55)\\
h_i &= \mathbf{x}_i^{\mathrm{T}} \mathbf{m}_w^{new} && (5.56)\\
h_i^{\backslash i} &= \mathbf{x}_i^{\mathrm{T}} \mathbf{m}_w^{\backslash i} && (5.57)
\end{aligned}$$

From the definition of $q^{new}(\mathbf{w})$ as the product of approximate terms $\tilde{t}_i(\mathbf{w})$, we know that

$$\begin{aligned}
\mathbf{V}_w^{new} &= \left(\mathbf{I} + \sum_i \frac{\mathbf{x}_i \mathbf{x}_i^{\mathrm{T}}}{v_i}\right)^{-1} = \left(\mathbf{I} + \mathbf{X}\Lambda^{-1}\mathbf{X}^{\mathrm{T}}\right)^{-1} && (5.58)\\
\mathbf{m}_w^{new} &= \mathbf{V}_w^{new} \sum_j \frac{m_j \mathbf{x}_j}{v_j} && (5.59)\\
\mathbf{x}_i^{\mathrm{T}} \mathbf{m}_w^{new} &= \sum_j (\mathbf{x}_i^{\mathrm{T}} \mathbf{V}_w^{new} \mathbf{x}_j) \frac{m_j}{v_j} && (5.60)
\end{aligned}$$

From the matrix inversion lemma we therefore have

$$\begin{aligned}
(\mathbf{C} + \Lambda)^{-1} &= \Lambda^{-1} - \Lambda^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{V}_w^{new}\mathbf{X}\Lambda^{-1} && (5.61)\\
\mathbf{X}^{\mathrm{T}}\mathbf{V}_w^{new}\mathbf{X} &= \Lambda - \Lambda(\mathbf{C} + \Lambda)^{-1}\Lambda && (5.62)\\
&= (\mathbf{C}^{-1} + \Lambda^{-1})^{-1} && (5.63)
\end{aligned}$$

In this way, we can compute $\mathbf{x}_i^{\mathrm{T}} \mathbf{V}_w^{new} \mathbf{x}_j$ for any (i, j) using only $\mathbf{C}$ and Λ. The EP algorithm becomes:

1. Initialize $v_i = \infty, m_i = 0, s_i = 1$.
2. Initialize $h_i = 0, \lambda_i = C_{ii}$.
3. Until all (m_i, v_i, s_i) converge:

   loop $i = 1, ..., n$:

   (a) Remove $\tilde{t}_i$ from the posterior to get an 'old' posterior:

$$h_i^{\backslash i} = h_i + \lambda_i v_i^{-1}(h_i - m_i) \qquad (5.64)$$

   (b) Recompute (part of) the new posterior, as in ADF:

$$\begin{aligned}
z &= \frac{h_i^{\backslash i}}{\sqrt{\lambda_i}} && (5.65)\\
Z_i &= \epsilon + (1 - 2\epsilon)\phi(z) && (5.66)\\
\alpha_i &= \frac{1}{\sqrt{\lambda_i}} \frac{(1 - 2\epsilon)\mathcal{N}(z; 0, 1)}{\epsilon + (1 - 2\epsilon)\phi(z)} && (5.67)\\
h_i &= h_i^{\backslash i} + \lambda_i \alpha_i && (5.68)
\end{aligned}$$

(c) Set

$$v_i = \lambda_i \left(\frac{1}{\alpha_i h_i} - 1 \right) \tag{5.69}$$

$$m_i = h_i^{\setminus i} + (v_i + \lambda_i)\alpha_i = h_i + v_i \alpha_i \tag{5.70}$$

$$s_i = Z_i \sqrt{1 + v_i^{-1} \lambda_i} \exp(\frac{\lambda_i \alpha_i}{2 h_i}) \tag{5.71}$$

(d) Now that Λ is updated, finish recomputing the new posterior:

$$\mathbf{A} = (\mathbf{C}^{-1} + \Lambda^{-1})^{-1} \tag{5.72}$$

For all i:

$$h_i = \sum_j A_{ij} \frac{m_j}{v_j} \qquad \text{(from (5.60))} \tag{5.73}$$

$$\lambda_i = \left(\frac{1}{A_{ii}} - \frac{1}{v_i} \right)^{-1} \qquad \text{(from (5.46))} \tag{5.74}$$

4. Compute the normalizing constant:

$$B = \sum_{ij} A_{ij} \frac{m_i m_j}{v_i v_j} - \sum_i \frac{m_i^2}{v_i} \tag{5.75}$$

$$p(D) \approx \frac{|\Lambda|^{1/2}}{|\mathbf{C} + \Lambda|^{1/2}} \exp(B/2) \prod_{i=1}^{n} s_i \tag{5.76}$$

The determinant expression in (5.76) follows from (5.58):

$$|\mathbf{V}_w^{new}| = \left| \mathbf{I} + \mathbf{X} \Lambda^{-1} \mathbf{X}^{\mathrm{T}} \right|^{-1} = \left| \mathbf{I} + \mathbf{X}^{\mathrm{T}} \mathbf{X} \Lambda^{-1} \right|^{-1} \tag{5.77}$$

$$= \left| \mathbf{I} + \mathbf{C} \Lambda^{-1} \right|^{-1} = \frac{|\Lambda|}{|\mathbf{C} + \Lambda|} \tag{5.78}$$

In step (5.72), it would appear that we need to invert a matrix, taking $O(n^3)$ time, for each data point. However, since only one v_i changes each time, $\mathbf{A}$ can be updated incrementally in $O(n^2)$ time. Initialize $\mathbf{A} = \mathbf{C}$ and then update with

$$\mathbf{A}^{new} = \left(\mathbf{C}^{-1} + \Lambda^{-1} + \Delta \Lambda^{-1} \right)^{-1} \tag{5.79}$$

$$= \mathbf{A} - \frac{\mathbf{a}_i \mathbf{a}_i^{\mathrm{T}}}{\delta + a_{ii}} \tag{5.80}$$

$$\text{where } \delta = \left(\frac{1}{v_i^{new}} - \frac{1}{v_i^{old}} \right)^{-1} \tag{5.81}$$

Assuming a constant number of iterations, the algorithm thus runs in $O(n^3)$ time, plus the time to compute the inner product matrix $\mathbf{C}$. This is a great improvement over $O(nd^2)$ if the dimensionality is very large, e.g. if the feature set is artificially expanded.

To show the equivalence with Opper & Winther's algorithm, we make the following observations:

- At convergence, (5.70) will hold for all i, so that

$$\sum_j \frac{m_j \mathbf{x}_j}{v_j} = \sum_j \frac{h_j \mathbf{x}_j}{v_j} + \sum_j \alpha_j \mathbf{x}_j \tag{5.82}$$

$$\mathbf{m}_w^{new} + \sum_j \frac{m_j \mathbf{x}_j}{v_j} = \left(\mathbf{I} + \sum_j \frac{\mathbf{x}_j \mathbf{x}_j^{\mathrm{T}}}{v_j} \right) \mathbf{m}_w^{new} + \sum_j \alpha_j \mathbf{x}_j \tag{5.83}$$

$$\mathbf{m}_w^{new} = \sum_j \alpha_j \mathbf{x}_j \tag{5.84}$$

$$h_i = \sum_j \alpha_j (\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j) = \sum_j C_{ij} \alpha_j \tag{5.85}$$

  This is the equation Opper & Winther use instead of (5.73).

- The update (5.74) for λ_i can be written

$$\lambda_i = \left(\frac{1}{v_i - v_i^2 \left[(\mathbf{C} + \Lambda)^{-1} \right]_{ii}} - \frac{1}{v_i} \right)^{-1} \tag{5.86}$$

$$= (v_i - v_i^2 \left[(\mathbf{C} + \Lambda)^{-1} \right]_{ii}) (v_i \left[(\mathbf{C} + \Lambda)^{-1} \right]_{ii})^{-1} \tag{5.87}$$

$$= \frac{1}{\left[(\mathbf{C} + \Lambda)^{-1} \right]_{ii}} - v_i \tag{5.88}$$

  This is the update Opper & Winther use instead of (5.74).

- EP computes a normalizing constant for the posterior, while Opper & Winther do not.

- All other updates are identical to those in Opper & Winther (2000c), though carried out in a different order. Reordering the updates does not matter as long as both algorithms converge.

The input to the algorithm is simply the matrix $\mathbf{C}$, which can be computed using an arbitrary inner product function $C(\mathbf{x}_i, \mathbf{x}_j)$ instead of $\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$. However, in that case we need to keep y_i and $\mathbf{x}_i$ separate:

$$C_{ij} = y_i y_j C(\mathbf{x}_i, \mathbf{x}_j) \tag{5.89}$$

Using a nonlinear inner product function is an efficient way to use an expanded feature space, since we don't have to represent each expanded data point. We only have to compute the inner product between two expanded data points, which is a scalar function of the original data.

The output of the algorithm is the α_i, which implicitly represent $\mathbf{m}_w^{new}$ through (5.84). With these we classify a new data point according to

$$f(\mathbf{x}) = \mathrm{sign}(\mathbf{x}^{\mathrm{T}} \mathbf{m}_w^{new}) = \mathrm{sign}(\sum_i \alpha_i y_i C(\mathbf{x}, \mathbf{x}_i)) \tag{5.90}$$

5.5 Results on synthetic data

Figure 5-2(a) demonstrates the Bayes point classifier vs. the SVM classifier on 3 training points. Besides the two dimensions shown here, each point had a third dimension set at

1. This provides a 'bias' coefficient w_3 so that the decision boundary doesn't have to pass through $(0,0)$. Each classifier is set to zero label noise ($\epsilon = 0$ and zero slack). The Bayes point classifier approximates a vote between all linear separators, ranging from an angle of $0°$ to $135°$. The Bayes point chooses an angle in the middle of this range. SVM chooses the decision boundary to maximize 'margin', the distance to the nearest data point. This makes it ignore the topmost data point. In fact, the Bayes point and SVM would coincide if that point were removed. Adding the point reduces the set of linear separators and Bayesian inference must take this into account.

Figure 5-3 plots the situation in parameter space. For viewing convenience, we restrict $\mathbf{w}$ to the unit sphere. The exact Bayes point was computed by likelihood-weighted sampling, i.e. sampling a random $\mathbf{w}$ on the sphere and discarding those which are not separators. EP provides an accurate estimate of the posterior mean $\mathbf{m}_w$ and posterior covariance $\mathbf{V}_w$. Compare:

$$\text{Exact } \mathbf{V}_w = \begin{bmatrix} 0.366 & 0.0551 & -0.125 \\ 0.0551 & 0.533 & -0.073 \\ -0.125 & -0.073 & 0.164 \end{bmatrix} \qquad \text{EP } \mathbf{V}_w = \begin{bmatrix} 0.358 & 0.0589 & -0.118 \\ 0.0589 & 0.542 & -0.0809 \\ -0.118 & -0.0809 & 0.163 \end{bmatrix}$$

Figure 5-2(b) plots cost vs. error for EP versus three other algorithms for estimating the Bayes point: the billiard algorithm of Herbrich et al. (1999), the TAP algorithm of Opper & Winther (2000c), and the mean-field (MF) algorithm of Opper & Winther (2000c). The error is measured by Euclidean distance to the exact solution found by importance sampling. The error in using the SVM solution is also plotted for reference. Its unusually long running time is due to Matlab's `quadprog` solver. TAP and MF were slower to converge than EP, even with a large initial step size of 0.5. As expected, EP and TAP converge to the same solution.

The billiard algorithm is a Monte Carlo method that bounces a ball inside the region defined by hyperplane constraints. It must be initialized with a linear separator and the SVM was used for that purpose. Since there are many other ways one could initialize the billiard, which may be much cheaper, the initialization step was not counted against the billiard's FLOP count (otherwise the billiard curve would be right of the SVM point). The error axis must also be interpreted carefully. While it is tempting to use the lower envelope of the curve as the measure of error, it is actually more accurate to use the upper envelope, since this is all that the algorithm can consistently achieve as samples accumulate. Nevertheless it is clear that EP is much faster and more accurate.

Laplace's method and variational Bayes do not work at all on this problem. Laplace fails because the derivatives of the likelihood are not informative: they are either zero or infinity. Variational Bayes fails for $\epsilon = 0$ because no Gaussian can lower bound the likelihood (Gaussians never reach zero). Even with $\epsilon > 0$, a Gaussian bound must be quite narrow so we can't expect competitive performance.

Herbrich & Graepel (2000) point out that the SVM is sensitive to scaling an input vector, i.e. if we replace $\mathbf{x}_1$ by $2\mathbf{x}_1$ then the solution will change. They argue theoretically and empirically that data vectors should therefore be normalized to unit length before SVM training. This is quite a shock since no mention of this was made in the original SVM papers. If the data in figure 5-2, including the bias dimension, is normalized before training, then the SVM solution tilts slightly toward the BPM solution, supporting Herbrich & Graepel's suggestion. The Bayes Point Machine, by contrast, does not require any normalization, because the data likelihood (5.1) is correctly invariant to scaling an input vector. The solution found by EP is also invariant.

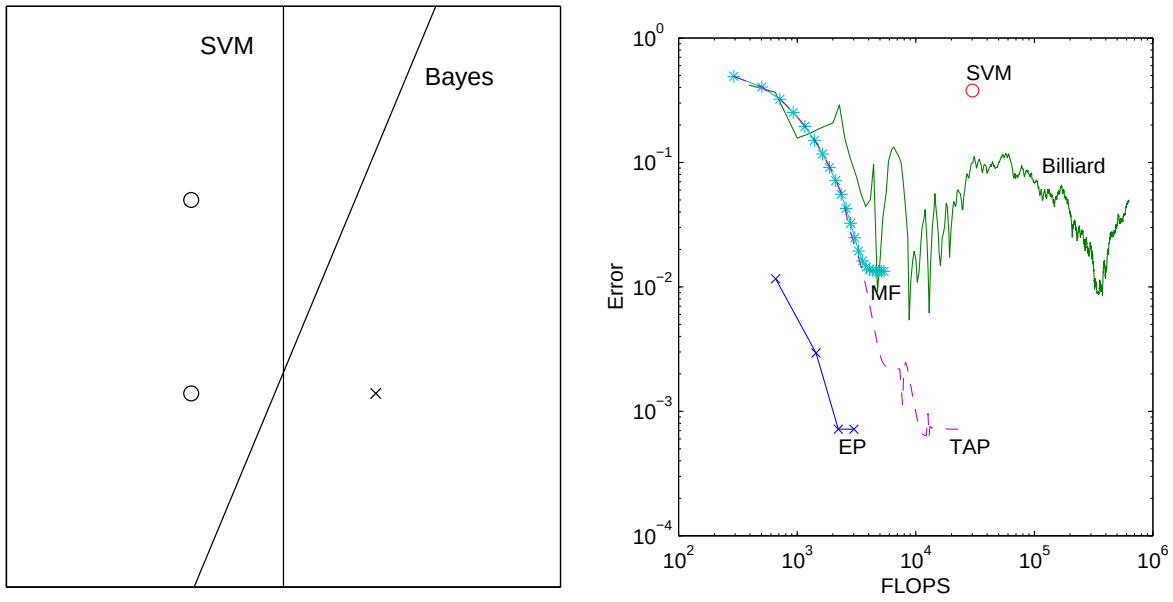


Figure 5-2: (left) Bayes point machine vs. Support Vector Machine on a simple data set. The Bayes point more closely approximates a vote between all linear separators of the data. (right) Cost vs. error in estimating the posterior mean. ADF is the first 'x' on the EP curve.

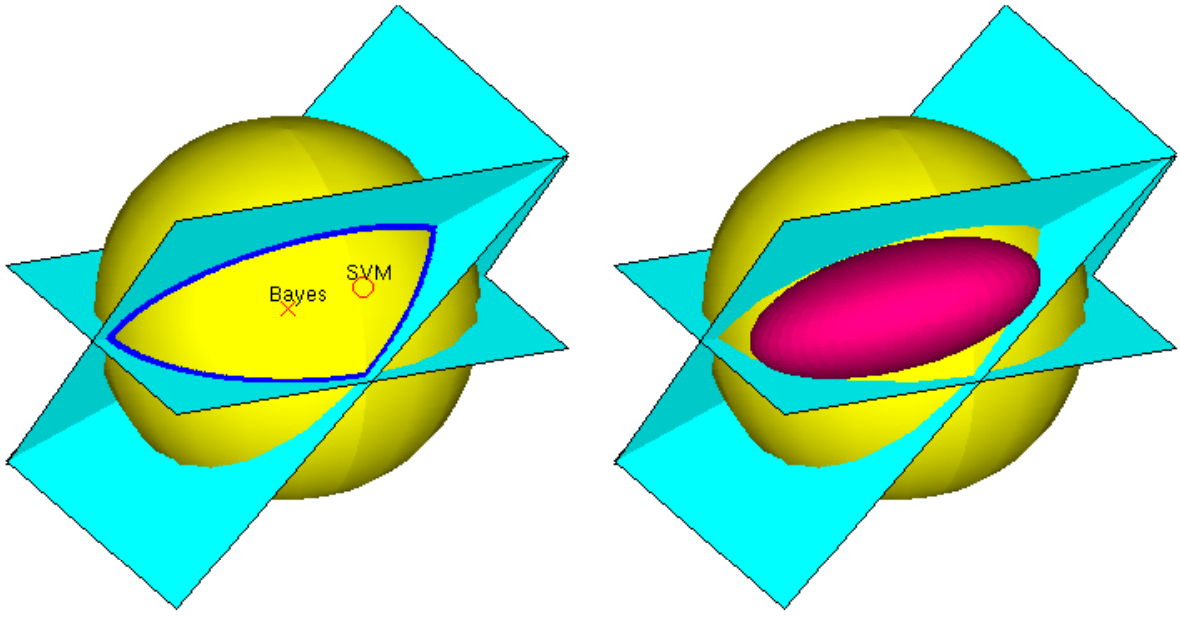


Figure 5-3: (left) The same problem viewed in parameter space (the unit sphere). Each data point imposes a linear constraint, giving an odd-shaped region on the sphere. The Bayes point is the centroid of the region. The EP estimate is indistinguishable from the exact Bayes point on this plot. (right) The Gaussian posterior approximation obtained by EP, rendered as a one standard deviation isoprobability ellipsoid.

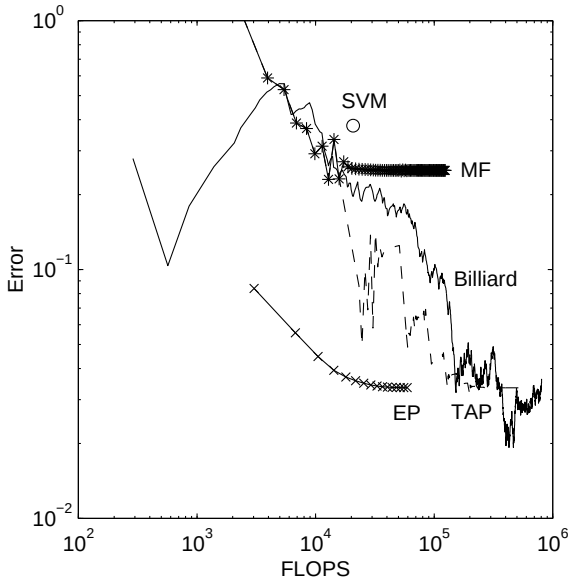


Figure 5-4: Cost vs. error for the 3-point problem with one point replicated 10 times

The linear classification problem is special in that some points can have large influence on the solution while adjacent points have zero influence. This is because the constraints imposed by the latter points may be redundant given the constraints already imposed by other points. This property allows the support vector machine to focus on a small number of points, the support vectors, when finding its solution. Unfortunately, this property can cause havoc with EP. EP uses Gaussians to smooth out the hard constraints and cannot always recognize when a point is redundant and should be ignored. The result is that clusters of points can degrade the EP estimate. Figure 5-4 shows what happens to figure 5-2(b) when the point at $(1, 0)$ is replicated 10 times. As expected, the cluster of points counts more than it should, making the EP boundary move slightly away from it. The accuracy of EP, TAP, and MF degrades significantly, while SVM and Billiard are unchanged. On the bright side, the error cannot be made arbitrarily high—using more than 10 replications, or replicating the other points, does not degrade the accuracy any further. One work-around for this behavior would be to reduce the dataset in advance, e.g. to just the support vectors. This idea has not been tested yet.

Figure 5-5 compares the EP solution to the exact posterior mean when noisy labels are allowed ($\epsilon = 0.1$). Theoretically, this integral is harder since we have to consider the entire parameter space, not just perfect separators. EP can handle it with no additional cost. The solution comes out vertical because it is a compromise between the small set of perfect separators at 135° and the larger set of boundaries between 0° and 135° which have one error. (There is also a set of boundaries past 135° which also have one error.) On this dataset, a similar result can be achieved with the SVM if we set the slack parameter $C = 2$, though this would not necessarily be true on another dataset. Winther (1998) has given extensive evaluations of EP with $\epsilon = 0$ and $\epsilon > 0$ on synthetic datasets, showing that it achieves the theoretical performance of Bayesian averaging. Those experiments are not repeated here.

Figure 5-6 demonstrates EP with a nonlinear kernel, contrasting it to the SVM solution

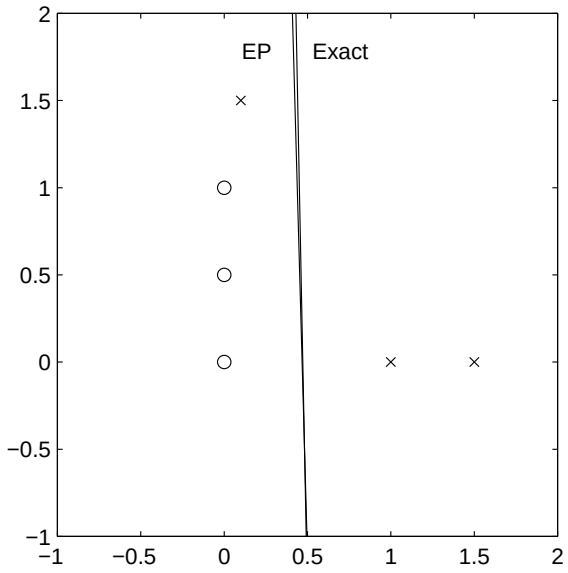


Figure 5-5: The EP solution has good agreement with the exact posterior mean when noisy labels are allowed ($\epsilon = 0.1$)

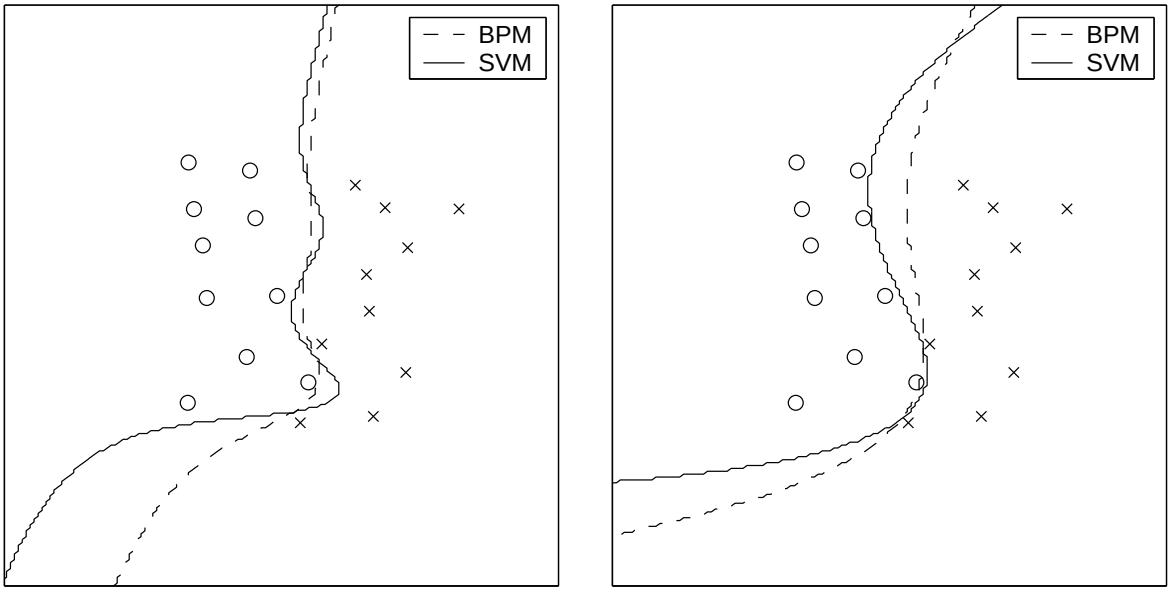
with the same kernel. The kernel was Gaussian:

$$C(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)) \tag{5.91}$$

with width parameter $\sigma = 0.2$ and $\sigma = 0.5$. The feature expansion equivalent to this kernel has an infinite number of features. The SVM is quite sensitive to the choice of σ as well as the idiosyncrasies of the dataset. This is related to the sparsity of the SVM solution: when the boundary depends on only a few data points, i.e. a few Gaussian kernels, small changes in those kernels have a larger effect on the boundary. Similar results obtain if we use a polynomial kernel:

$$C(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + 1)^p \tag{5.92}$$

For large p, the boundaries are almost identical to those for a wide Gaussian kernel.



narrow Gaussian kernel wide Gaussian kernel

Figure 5-6: Classification boundaries resulting from Support Vector Machine training vs. Bayes Point Machine training with EP. Both used a Gaussian kernel with the same width parameter. (left) Narrow width. The SVM tends to put extra bumps and kinks in the boundary. (right) Wider width. The SVM chooses an unusual solution in order to maximize margin in the expanded space. The BPM boundary is non-sparse (all $\alpha_i > 0$) but smoother. It is hardly affected by changing the kernel width. Billiard gives results similar to EP.

5.6 Results on real data

For higher dimensional problems, it is more difficult to compute the exact Bayes point and make cost vs. accuracy comparisons. So instead we'll measure cost vs. test performance on benchmark problems.

Figure 5-7 compares EP, Billiard, and SVM on discriminating handwritten digit '3' from '5'. Each data point $\mathbf{x}_i$ is an 8×8 binary image (64 dimensions). The dataset was randomly split 40 times into a (small) training set of 70 points and a test set of 430 points. All algorithms were linear and set for zero noise. Billiard was run for 2000 iterations, which is far more computation than EP or SVM used. Nevertheless, EP is best.

Figures 5-8 and 5-9 show the same type of comparison on four datasets from the UCI repository (Blake & Merz, 1998). Each dataset was randomly split 40 times into a training set and test set, in the ratio 60%:40%. In each trial, the features were normalized to have zero mean and unit variance in the training set. The classifiers used zero label noise and a Gaussian kernel with $\sigma = 3$. Billiard was run for 500 iterations. The `thyroid` dataset was made into a binary classification problem by merging the different classes into normal vs. abnormal. Except for `sonar`, EP and Billiard beat the SVM a majority of the time, and in all cases EP performed similarly to Billiard. However, the running time for Billiard is significantly higher, since it must be initialized at the SVM solution. Figure 5-10 shows cost vs. test error curves on some typical runs. The SVM using `quadprog` is particularly slow on `thyroid`. The unusual success of the SVM on `sonar` was also reported by Herbrich et al. (1999), and may be related to the different assumptions made by the SVM (see section 5.1).

To give an idea of the actual running time in Matlab, figure 5-11 plots the training time vs. training set size for 40 trials. As expected, the time scales as n^3. Extrapolating from this, it would take an hour for 1,000 data points and 6 weeks for 10,000 data points. Finding a way to speed up EP, similar to the way that quadratic programming can be sped up for the SVM, will be essential for large data sets.

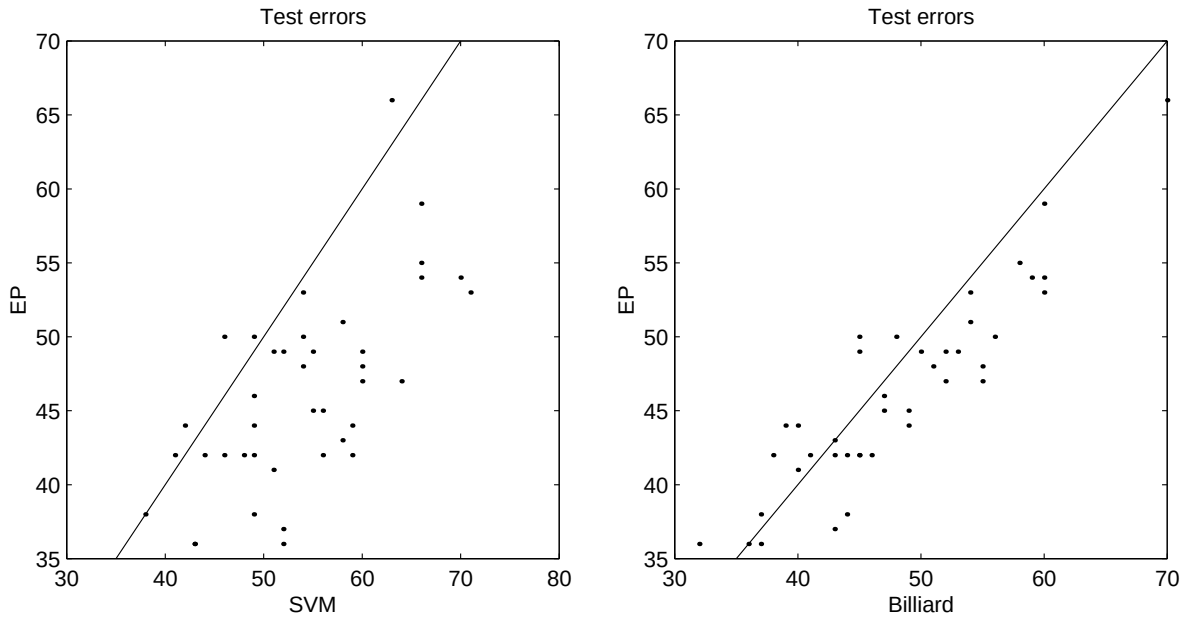


Figure 5-7: Test error of SVM vs. EP (left) and Billiard vs. EP (right) on digit classification, over 40 random train/test splits. Each point is (test error for SVM/Billiard, test error for EP). Points under the line correspond to trials where EP had a lower test error. EP beat SVM 34 times and beat Billiard 28 times, despite being the fastest of all three algorithms of this dataset.

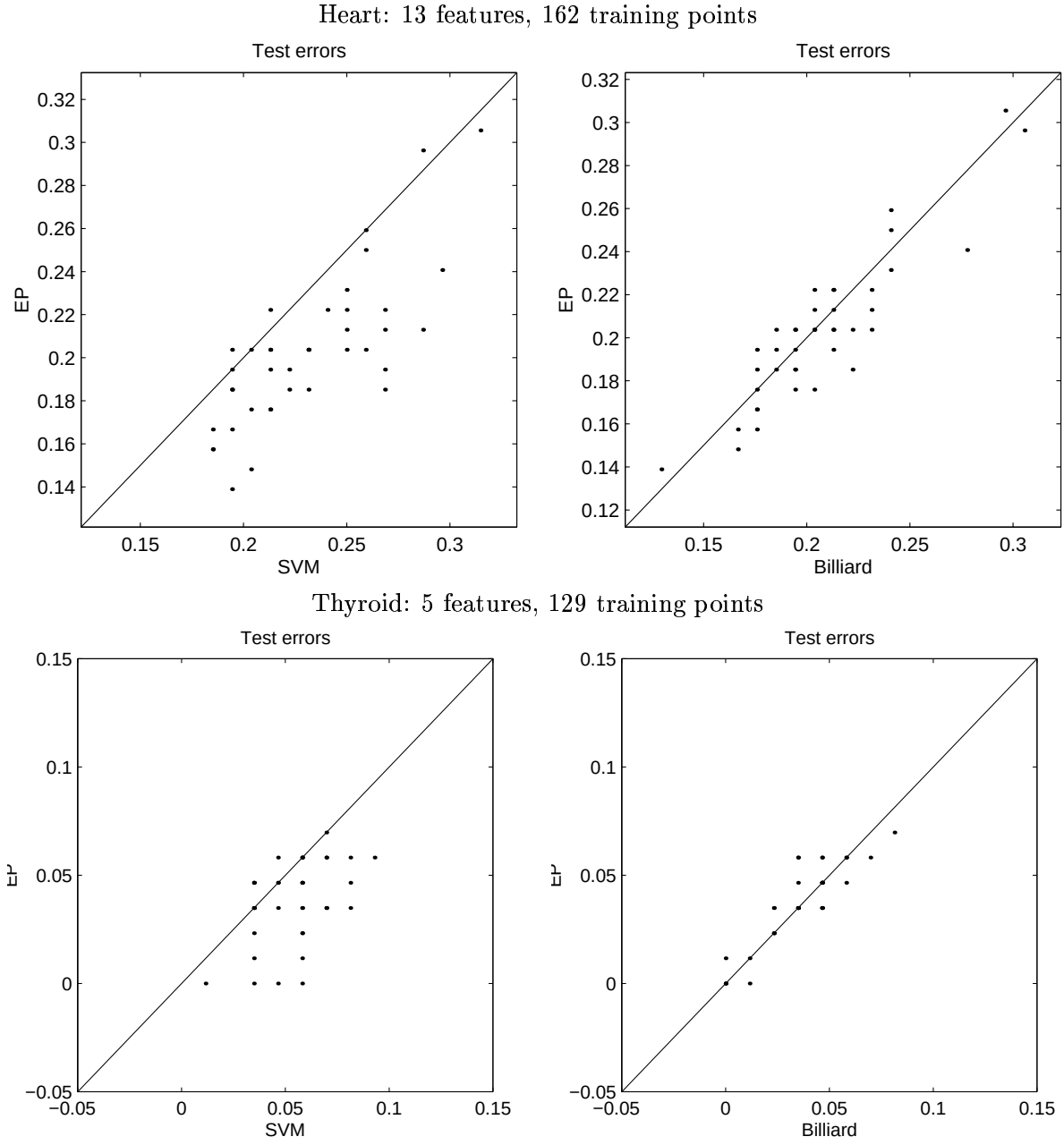


Figure 5-8: Test error rates over 40 train/test splits. Each point is (test error for SVM/Billiard, test error for EP). Points under the line correspond to trials where EP had a lower test error. For running times, see figure 5-10.

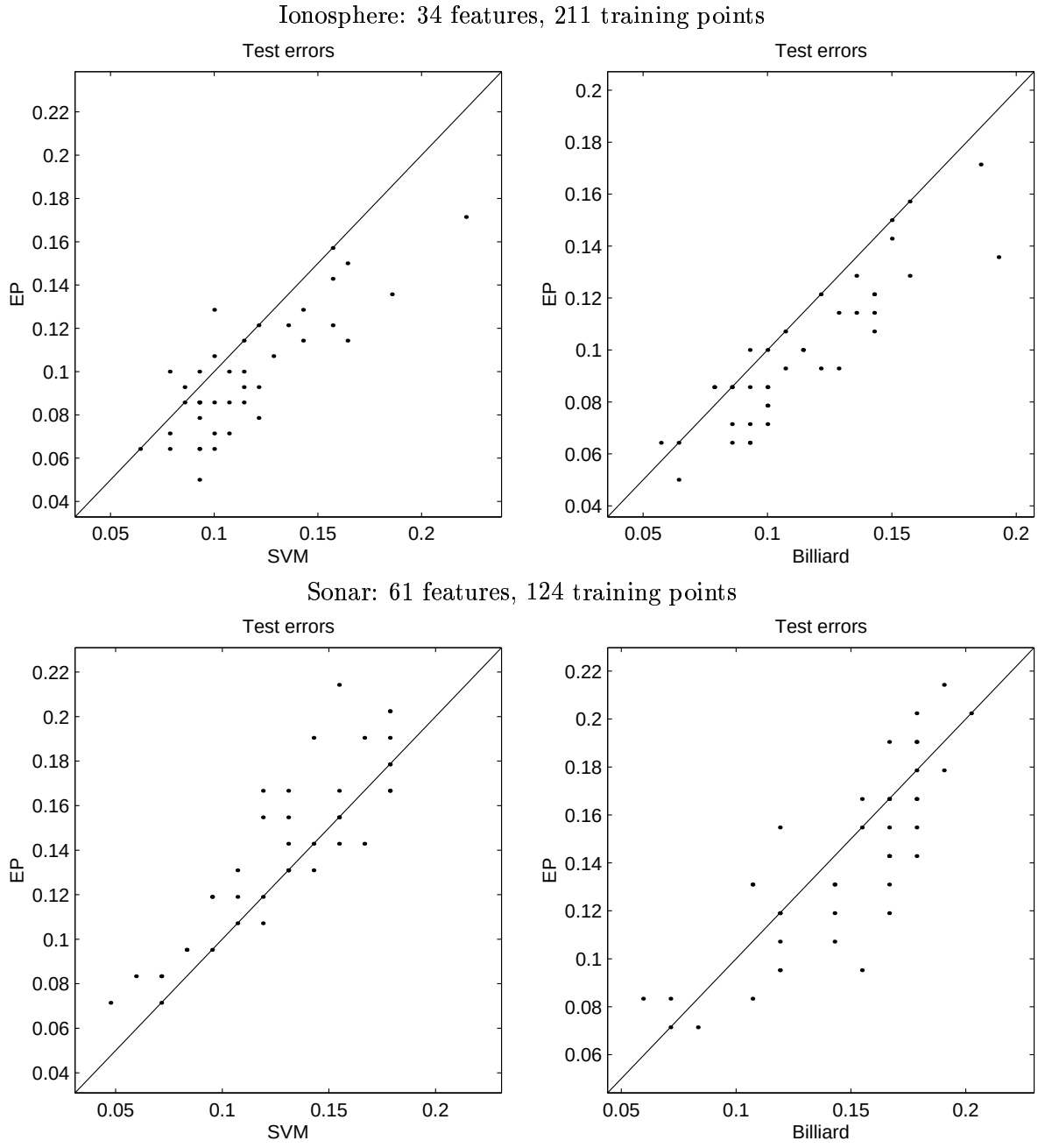


Figure 5-9: Test error rates over 40 train/test splits. Each point is (test error for SVM/Billiard, test error for EP) for one trial. Points under the line correspond to trials where EP had a lower test error. For running times, see figure 5-10.

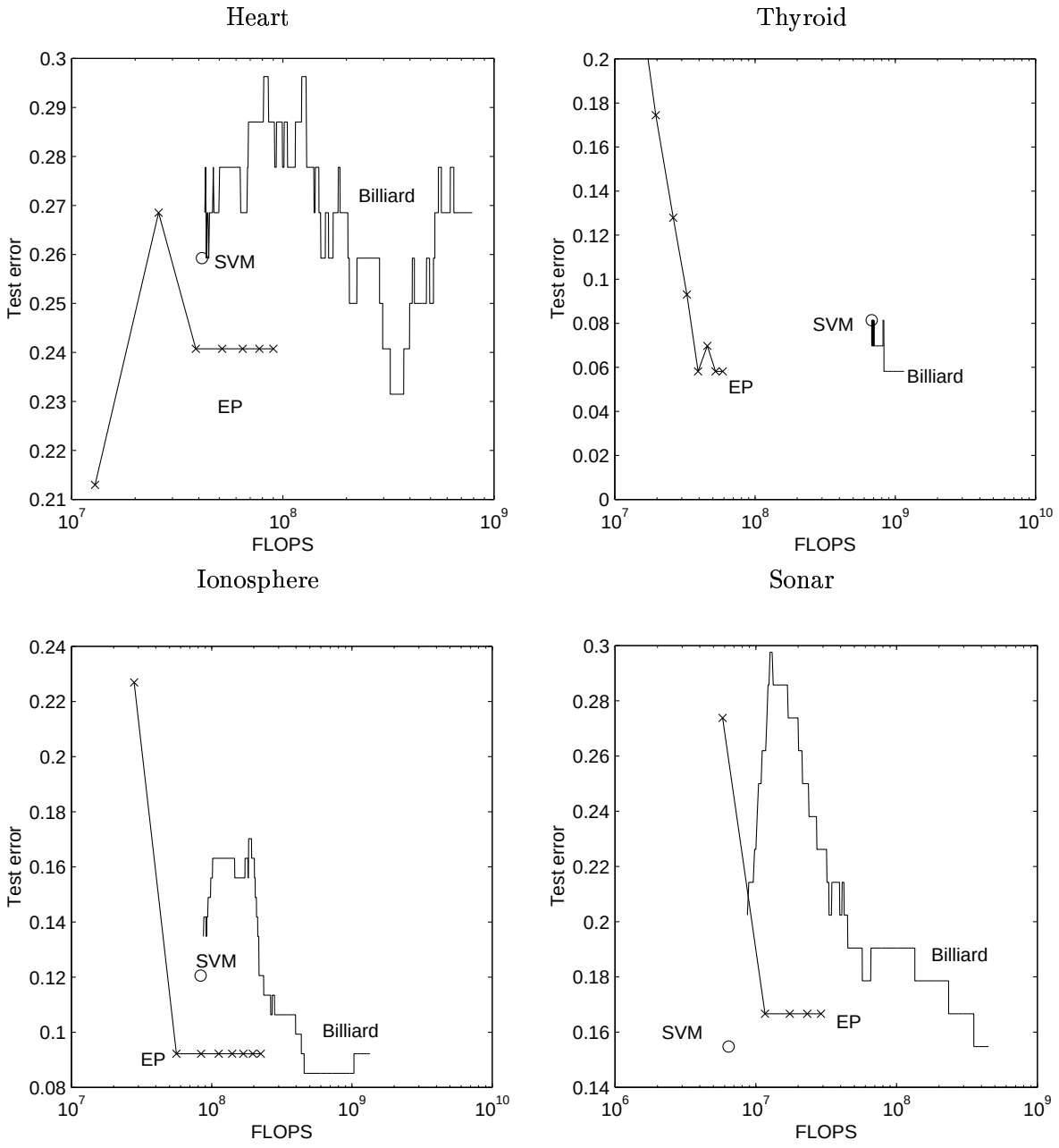


Figure 5-10: Cost vs. test error on a typical trial, for each dataset

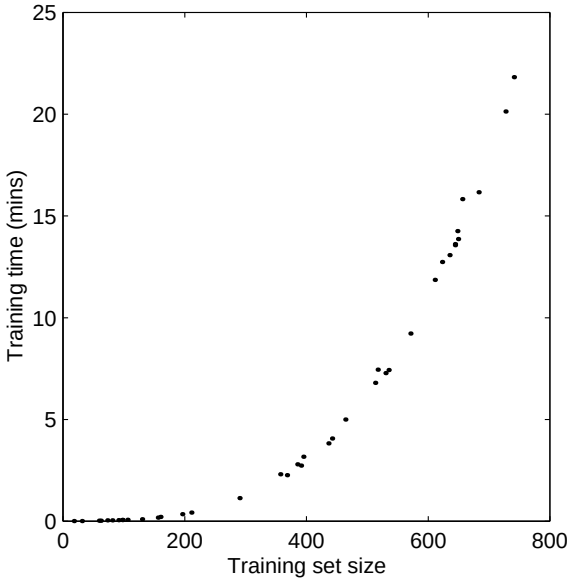


Figure 5-11: Actual training time in Matlab for EP with kernels, for 40 different dataset sizes. It nearly perfectly follows n^3 growth. The time to compute the inner product matrix is not included.

5.7 Model selection

This section reports results on Bayesian model selection using the estimated evidence $p(D)$. This is a unique feature of EP because the billiard algorithm cannot provide an estimate of $p(D)$. By maximizing the evidence, we can select the appropriate feature set, feature kernel, or noise parameter ϵ. For background on Bayesian model selection, see MacKay (1995); Kass & Raftery (1993); Minka (2000a). A popular approach to model selection for the SVM is to minimize $\frac{R^2}{M^2}$, where R is the radius of the smallest ball containing the training set (measured in the mapped feature space) and M is the margin (Cristianini et al., 1998). Another approach is to minimize the span bound given by thresholding the α's (Chapelle & Vapnik, 1999; Chapelle et al., 2000):

$$S = \sum_{\text{support vectors } i} \Theta(\alpha_i - \left[\mathbf{C}_{\text{sv}}^{-1}\right]_{ii}) \tag{5.93}$$

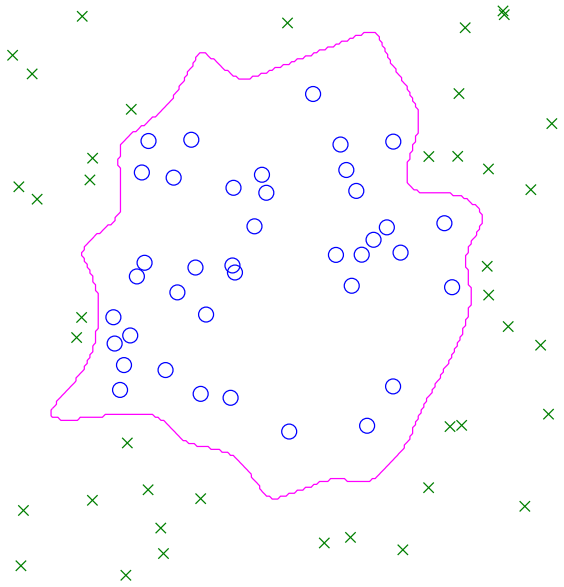
These three criteria, evidence, margin, and span, can lead to quite different results. Figure 5-12 shows a synthetic training set classified according to distance from the origin, i.e. the true decision boundary is a circle. Three different decision boundaries result from training an SVM with three different kernels (the decision boundaries from EP are nearly identical). All of these boundaries achieve zero error so we need to invoke some other rule besides training error to select among them. Over all Gaussian kernels, the margin criterion is minimized by taking $\sigma = 0.1$. This produces a decision boundary with spurious kinks and bumps. The span bound is minimized by an even narrower kernel: $\sigma = 0.011$, where the bound is 0. By contrast, the best Gaussian kernel according to evidence has $\sigma = 0.9$, providing a much smoother and more realistic boundary. The quadratic kernel, which is the closest to the true boundary, has even greater evidence yet less margin.

In the noise-free case, the Bayesian evidence $p(D)$ has a natural interpretation. It is the fraction of perfect separators out of all representable classifiers. If a particular set of features has a large percentage of perfect separators, then we intuitively should have more faith in that feature set. This is what Bayesian model selection does, and in choosing the Bayes point we follow the same basic argument. Maximizing the margin does not necessarily behave this way.

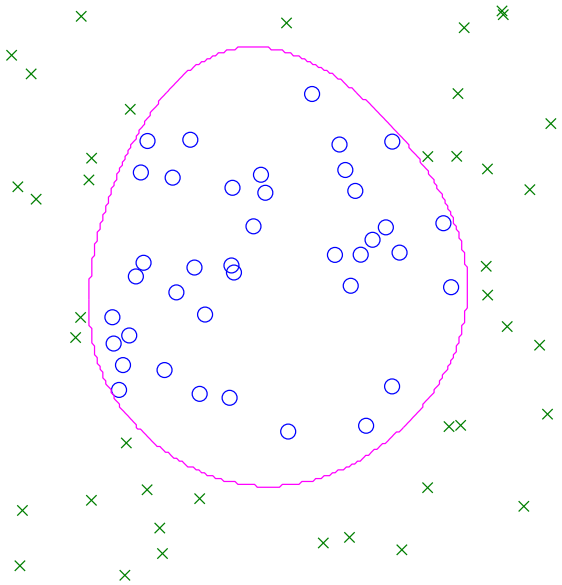
The next example repeats the feature selection experiment of Weston et al. (2000); Chapelle et al. (2000). Synthetic data with six features was created by first sampling $y = \pm 1$ with equal probability and then setting

$$x_1 = \mathcal{N}(y, 1) \tag{5.94}$$
$$x_2 = \mathcal{N}(2y, 1) \tag{5.95}$$
$$x_3 = \mathcal{N}(3y, 1) \tag{5.96}$$
$$x_4 = \mathcal{N}(0, 1) \tag{5.97}$$
$$x_5 = \mathcal{N}(0, 1) \tag{5.98}$$
$$x_6 = \mathcal{N}(0, 1) \tag{5.99}$$

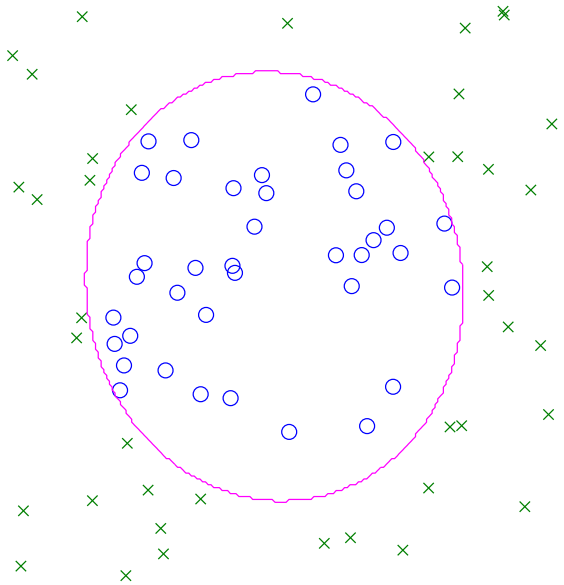
With probability 0.3, features x_1–x_3 were swapped with features x_4–x_6. Thus all six features are relevant to some degree. To these six relevant features are appended 14 irrelevant features with distribution $\mathcal{N}(0, 20)$. The test is to see whether we can reject the 14 irrelevant features. In Chapelle et al. (2000), the algorithm was forced to keep only two features. Here we use a more rigorous test: features are added one by one, starting with the relevant ones,



Gaussian $\sigma = 0.1$



Gaussian $\sigma = 0.9$



Quadratic

Kernel	R^2/M^2	span	$\log(p(D))$
$\sigma = 0.1$	13	5	-40
$\sigma = 0.9$	284	5	-18
quadratic	1300	6	-17

Figure 5-12: A dataset classified with three different kernels. The margin criterion R^2/M^2 prefers the $\sigma = 0.1$ solution, while the Bayesian evidence $p(D)$ prefers the quadratic solution. The span bound prefers a Gaussian with tiny $\sigma = 0.011$ (not shown). The true boundary is a circle.

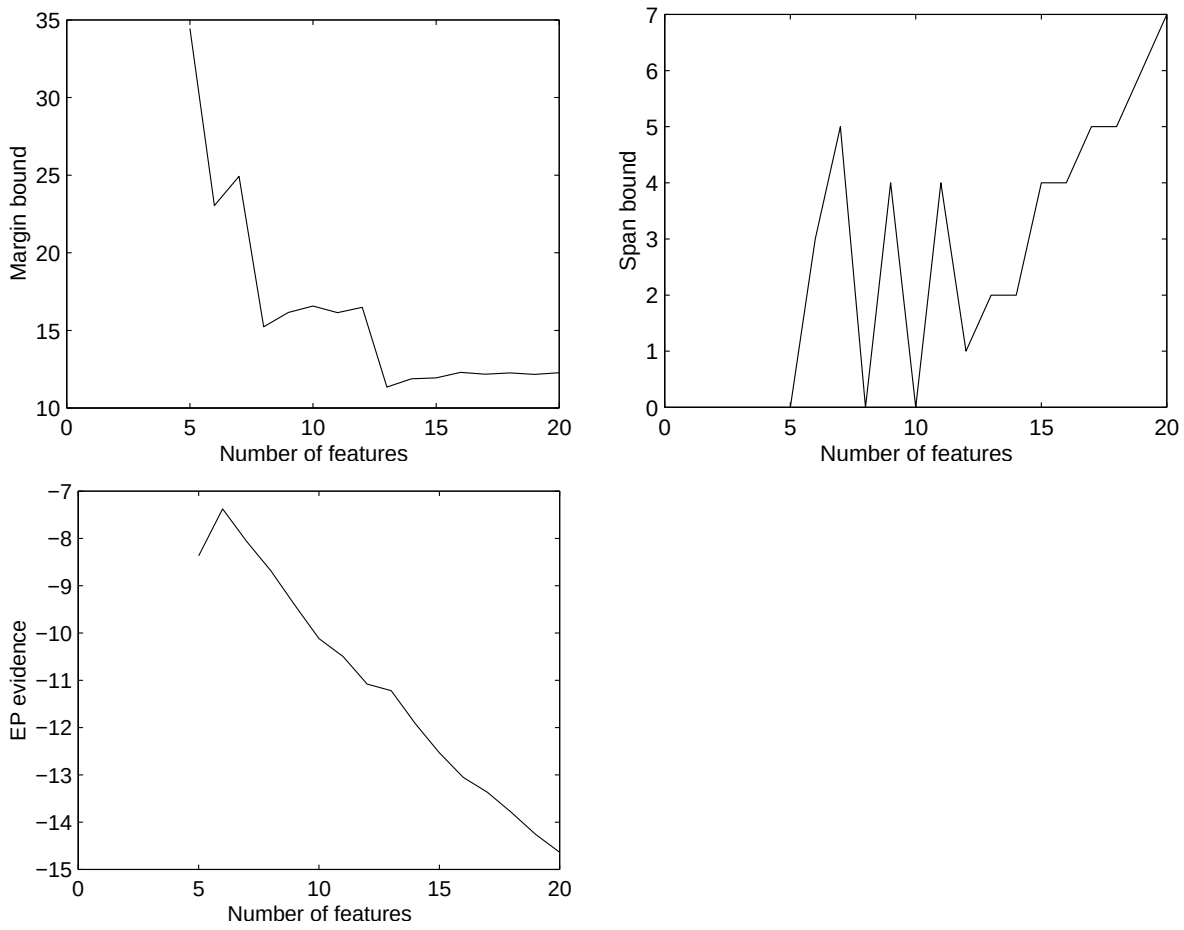


Figure 5-13: Feature selection curves for the margin bound, span bound, and Bayesian evidence. The Bayesian evidence has a clear peak at six, the correct answer. The other criteria, which are to be minimized, do not perform as well. The margin bound has only a local minimum at six, and the span bound only a local minimum at five.

and the algorithm must decide when to stop. Figure 5-13 shows the curves for the margin bound, span bound, and Bayesian evidence. Both the SVM and BPM used a linear kernel with zero slack. The Bayesian evidence is clearly a better measure of relevance.

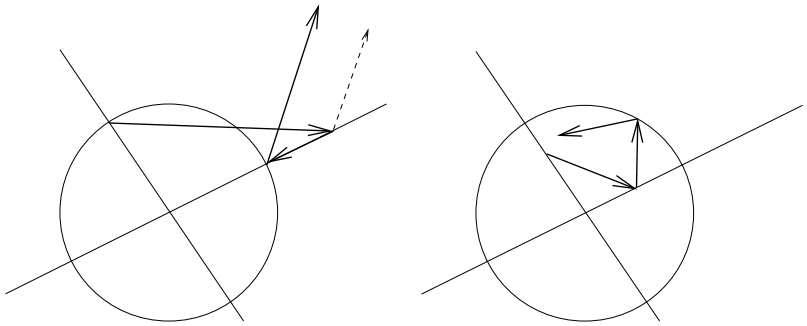


Figure 5-14: (a) Conventional billiard algorithms allow the ball to escape. They project the collision point onto the surface of the sphere, but preserve the rebound velocity (dashed line). (b) The new algorithm works inside the sphere, using the sphere's surface as an additional wall.

5.8 A better billiard algorithm

A major drawback of current billiard algorithms is that they do not really operate on the sphere. The billiard ball always follows a straight line trajectory until it hits a wall, at which point it is projected back onto the sphere (figure 5-14(a)). Unfortunately, this means the ball can sometimes leave the sphere without ever hitting a wall. On the 3-point problem in figure 5-3, this happens on 50% of the bounces. Rujan (1997) recovers from this by restarting the billiard at its initial point, with a random velocity. Unfortunately, this introduces severe bias in the estimate, by overcounting the initial point. Herbrich et al. (1999) have a better solution: leave the ball where it is, but choose a new random velocity. This has less bias, but doesn't resemble a billiard anymore.

The problem of escaping billiard balls can be avoided entirely if we simply redefine the pool. Remember that the unit sphere was an arbitrary choice; we can use any prior distribution that assigns equal probability to all $\mathbf{w}$ vectors of a given length. So let's use the *inside* of the unit sphere, as shown in figure 5-14(b). The sphere's surface is an additional wall that prevents escape. The new algorithm is:

1. Determine the next collision. From position $\mathbf{w}$ and velocity $\mathbf{v}$, compute the flight time to each wall and take the minimum. The distance from wall $\mathbf{x}_i$ is

$$d_i = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{x}_i}{\sqrt{\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i}} \tag{5.100}$$

and the velocity normal to $\mathbf{x}_i$ is

$$v_i = \frac{\mathbf{v}^{\mathrm{T}}\mathbf{x}_i}{\sqrt{\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i}} \tag{5.101}$$

so the flight time is

$$\tau_i = \begin{cases} -\frac{d_i}{v_i} & \text{if } v_i < 0 \\ \infty & \text{otherwise} \end{cases} \tag{5.102}$$

The flight time to the surface of the sphere satisfies

$$(\mathbf{w} + \mathbf{v}\tau)^{\mathrm{T}}(\mathbf{w} + \mathbf{v}\tau) \quad = \quad 1 \tag{5.103}$$

$$\tau = \frac{\sqrt{(\mathbf{w}^{\mathrm{T}}\mathbf{v})^2 + \mathbf{v}^{\mathrm{T}}\mathbf{v}(1 - \mathbf{w}^{\mathrm{T}}\mathbf{w})} - \mathbf{w}^{\mathrm{T}}\mathbf{v}}{\mathbf{v}^{\mathrm{T}}\mathbf{v}} \tag{5.104}$$

which is always positive and finite.

2. Update the position of the ball, the total distance traveled (s), and the center of mass estimate ($\mathbf{m}$).

$$\mathbf{w}^{new} = \mathbf{w} + \mathbf{v}\tau_{min} \tag{5.105}$$
$$z = \|\mathbf{w}^{new} - \mathbf{w}\| \tag{5.106}$$
$$\mathbf{m}^{new} = \frac{s}{s+z}\mathbf{m} + \frac{z}{s+z}\frac{\mathbf{w}^{new} + \mathbf{w}}{2} \tag{5.107}$$
$$s^{new} = s + z \tag{5.108}$$

3. Update the velocity. If $\mathbf{x}_i$ was hit,

$$\mathbf{v}^{new} = \mathbf{v} - 2v_i \frac{\mathbf{x}_i}{\sqrt{\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i}} \tag{5.109}$$

If the surface of the sphere was hit,

$$\mathbf{v}^{new} = \mathbf{v} - 2\mathbf{w}^{new}(\mathbf{w}^{new})^{\mathrm{T}}\mathbf{v} \tag{5.110}$$

The length of $\mathbf{v}$ is unchanged by these updates, so if we initialize $\mathbf{v}^{\mathrm{T}}\mathbf{v} = 1$, it will stay that way throughout the algorithm.

The algorithm can also be made use a kernel inner product, as in Herbrich et al. (1999). With this algorithm, we can run one billiard trajectory, without ever restarting. However, that is not a good idea, because the pool is not perfectly ergodic. By periodically randomizing the velocity of the ball, we can achieve a more ergodic sampling. Periodic randomization is important in all Markov chain Monte Carlo schemes, including Gibbs sampling. For the billiard, randomization at every 100 bounces seems to work well.

Figure 5-15 compares the three algorithms on the 3-point problem of section 5.5. Each was run for 100,000 iterations, using a comparable number of flops. Because of its bias, Rujan's algorithm is never more accurate than 10^{-1} in Euclidean distance to the true value. Herbrich's algorithm does better but also seems to level off at 10^{-2}. The new algorithm, with randomization every 100 bounces, converges nicely. These characteristics are typical and hold across many different runs on this dataset.

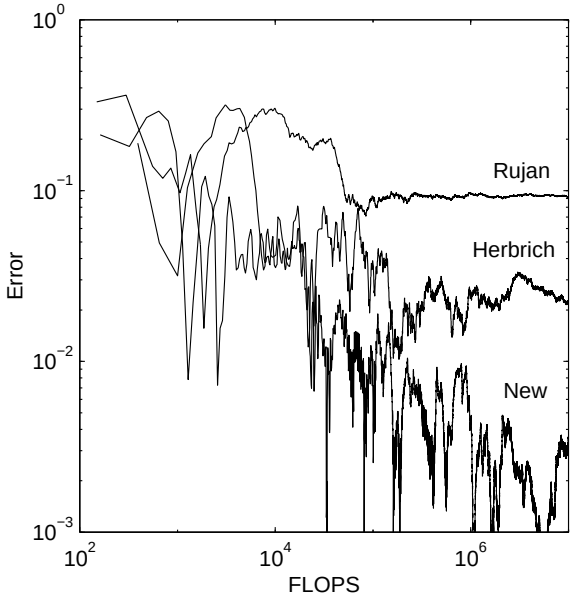


Figure 5-15: Cost vs. error of conventional billiard algorithms vs. the new algorithm.

Chapter 6

Summary and future work

This thesis has developed a family of algorithms for approximate inference, based on extending assumed-density filtering (ADF) to use iterative refinement. This extension removes all order dependence from ADF and increases its accuracy. The family includes loopy belief propagation in Bayesian networks, and extends it to allow approximate messages (for reduced computation) as well as more elaborate messages (for increased accuracy). For inference in the Bayes Point Machine, it includes the algorithm of Opper & Winther (2000c) and provides an alternative to the costly billiard algorithm.

The accuracy of the Expectation Propagation family was demonstrated on a variety of examples, with most emphasis on the Bayes Point Machine:

Section 3.2.2 For the clutter problem in one dimension, EP was (in well-behaved cases) 10 times more accurate than its nearest competitor, Laplace's method, both in estimating the posterior mean and the normalizing constant $p(D)$. In other cases, when the true posterior was strongly multimodal, EP did not converge at all, while Restricted EP and other deterministic methods lagged behind Monte Carlo. The computational cost of EP is slightly higher than Laplace's method but much less than Monte Carlo.

Section 3.3.4 For marginalizing the mixing weight in a mixture density of two Gaussians, EP was again 10 times more accurate than Laplace's method, its nearest competitor. With modifications inspired by Cowell et al. (1996), EP is also faster than Laplace's method. In this problem, the posterior is always unimodal (in fact log-convex), which may explain why EP always converged.

Section 5.5 On a toy dataset where the Bayes Point Machine is very different from the Support Vector Machine, EP delivers high accuracy at a fraction of the cost of its competitor, the billiard algorithm. The accuracy of EP degrades when points are clustered together, but even then it still beats the billiard algorithm with respect to cost. On separable problems, the posterior is unimodal and EP seems to always converge.

Section 5.5 EP is also accurate in situations that the billiard algorithm cannot handle, namely when there is label noise. This was demonstrated on a toy dataset; more detailed experiments, showing that EP achieves theoretical bounds, are given by Winther (1998), since his algorithm is equivalent to EP.

Section 5.6 On benchmark datasets with around 150 points and 5–60 features, EP achieves test set error rates consistent with the billiard algorithm. Beating the SVM on 4 out

of 5 datasets, EP realizes in practice the theoretical gains expected with a Bayesian approach.

Many opportunities for future work are available, both within the framework of EP as well as beyond it. First, how effective is EP for other statistical models? For example, is it useful for inference in coupled hidden Markov models, sigmoid belief networks (Barber & Sollich, 1999), and dynamic trees (Storkey, 2000)? Is EP effective for Bayesian parameter estimation of classification models beyond linear classifiers, e.g. hidden Markov models? Is any deterministic method effective for nonparametric models such as Dirichlet processes (Rasmussen, 1999)?

Second, can we anticipate how EP will perform? Can EP provide an estimate of its error? (The experiments in this paper always used the ADF initialization. Are the fixed points of EP unique, or does initialization matter? What causes EP to diverge? Can it be made to always converge? Can EP be made to give reasonable approximations to multimodal posteriors?

Can EP be made to run faster, especially for the Bayes Point Machine? This thesis compared EP to the original quadratic programming implementation of the Support Vector Machine. But much faster, specialized algorithms for the SVM are now available due to intense research. Can we exploit special properties of the Bayes Point Machine to speed up EP? For example, can we represent the $\mathbf{A}$ matrix in a sparse fashion? Can we identify the terms that need to be refined? Having an error estimate for EP could facilitate this.

Is it optimal to minimize KL-divergence at each step, or should we use some other criterion? The modified update inspired by Cowell et al. (1996) shows that we may obtain computational speedups with other criteria. But improvements in accuracy may also be possible, if we replace the greedy KL criterion with something that incorporates lookahead. For example, if we know the posterior has heavy tails and we are approximating it with a Gaussian, then we may choose to use a Gaussian with inflated variance, to better account for the tails.

The entire difference between EP and ADF is that EP makes better choices for the approximate terms $\tilde{t}_i$. But both algorithms base their choices on the 'old' posterior $q(x)$, which is assumed to be accurate. If the true posterior is multimodal, then $q(x)$ cannot possibly be accurate. Could we propagate additional information about $q(x)$ to use in choosing $\tilde{t}_i$?

Can iterative refinement be applied to other filtering algorithms? For example, some variants of ADF do not use the exponential family. Cowell et al. (1996) approximate a mixture posterior with a smaller mixture after processing each data point; see also the "generalized pseudo-Bayes" algorithms in Murphy (1998). Frey et al. (2000) approximate the discrete posterior with an arbitrary tree-structured distribution. Conventional EP is not practical with these kinds of approximations because the equivalent terms $\tilde{t}_i(x) = q^{new}(x)/q(x)$ do not simplify.

Yedidia et al. (2000) give a generalization of belief propagation that is different from the generalization afforded by EP. Their algorithm propagates exact marginals and pairwise marginals, which gives it high accuracy but also limits its practicality. Can it be extended to allow approximate messages as in EP? Is there a common parent of their algorithm and EP?

Bayesian inference is a fertile area for developing numerical integration algorithms, especially deterministic ones, because of the rich problem structure and prior knowledge about the functions being integrated. There is also relatively little research on it, compared to

optimization for example. Many researchers are swayed enough by this imbalance to use inferior techniques, such as maximum likelihood estimation, because they only involve optimization. As argued in this thesis, it doesn't have to be that way. And I anticipate that there are even better integration algorithms out there, waiting for someone to discover them.

Bibliography

Aitchison, J., & Shen, S. M. (1980). Logistic-normal distributions: Some properties and uses. *Biometrika, 67*, 261–272.

Attias, H. (1999). Inferring parameters and structure of latent variable models by variational Bayes. *Uncertainty in Artificial Intelligence*. `http://www.gatsby.ucl.ac.uk/~hagai/papers.html`.

Barber, D., & Bishop, C. (1997). Ensemble learning for multi-layer networks. *NIPS 10*. MIT Press. `http://www.mbfys.kun.nl/~davidb/papers/kl_mlp_nips10.html`.

Barber, D., & Sollich, P. (1999). Gaussian fields for approximate inference in layered sigmoid belief networks. *NIPS 12*.

Barber, D., & Wiegerinck, W. (1998). Tractable variational structures for approximating graphical models. *NIPS 11*. `ftp://ftp.mbfys.kun.nl/snn/pub/reports/Barber.nips98.ps.Z`.

Bennett, K., & Demiriz, A. (1998). Semi-supervised support vector machines. *NIPS 11*. `http://www.rpi.edu/~bennek/s3vm.ps`.

Bernardo, J. M., & Giron, F. J. (1988). A Bayesian analysis of simple mixture problems. *Bayesian Statistics 3* (pp. 67–78).

Blake, C. L., & Merz, C. J. (1998). UCI repository of machine learning databases. `http://www.ics.uci.edu/ mlearn/MLRepository.html`. Irvine, CA: University of California, Department of Information and Computer Science.

Bouten, M., Schietse, J., & den Broeck, C. V. (1995). Gradient descent learning in perceptrons - a review of its possibilities. *Physical Review E, 52*, 1958–1967.

Boyen, X., & Koller, D. (1998a). Approximate learning of dynamic models. *NIPS 11*. `http://robotics.Stanford.EDU/~koller/papers/nips98.html`.

Boyen, X., & Koller, D. (1998b). Tractable inference for complex stochastic processes. *Uncertainty in AI*. `http://robotics.Stanford.EDU/~koller/papers/uai98.html`.

Buhot, A., Moreno, J., & Gordon, M. (1997). Finite size scaling of the Bayesian perceptron. *Physical Review E, 55*, 7434–7440.

Carpenter, J., Clifford, P., & Fearnhead, P. (1999). Building robust simulation-based filters for evolving data sets (Technical Report). Department of Statistics, Oxford University. `http://citeseer.nj.nec.com/carpenter99building.html`.

Chapelle, O., Vapnik, V., Bousquet, O., & Mukherjee, S. (2000). Choosing kernel parameters for support vector machines. http://www.ens-lyon.fr/~ochapell/.

Chapelle, O., & Vapnik, V. N. (1999). Model selection for support vector machines. *NIPS 12*. http://www.ens-lyon.fr/~ochapell/.

Cowell, R. G., Dawid, A. P., & Sebastiani, P. (1996). A comparison of sequential learning methods for incomplete data. *Bayesian Statistics 5*. http://www.ucl.ac.uk/Stats/research/psfiles/135.zip.

Cristianini, N., Campbell, C., & Shawe-Taylor, J. (1998). Dynamically adapting kernels in support vector machines. *NIPS 11*.

Csato, L., Fokue, E., Opper, M., Schottky, B., & Winther, O. (1999). Efficient approaches to Gaussian process classification. *NIPS 12*. http://www.ncrg.aston.ac.uk/cgi-bin/tr_avail.pl?trnumber=NCRG/99/025.

Davis, P. J., & Rabinowitz, P. (1984). *Methods of numerical integration*. Academic Press. 2nd edition.

Frey, B. J., & MacKay, D. J. (1997). A revolution: Belief propagation in graphs with cycles. *NIPS 10*. http://www.cs.toronto.edu/~mackay/rev.ps.gz.

Frey, B. J., Patrascu, R., Jaakkola, T., & Moran, J. (2000). Sequentially fitting inclusive trees for inference in noisy-OR networks. *NIPS 13*.

Gelfand, A. E., & Smith, A. F. M. (1990). Sampling-based approaches to calculating marginal densities. *J American Stat Assoc*, *85*, 398–409.

Ghahramani, Z., & Beal, M. J. (1999). Variational inference for Bayesian mixtures of factor analysers. *NIPS 12*. http://www.gatsby.ucl.ac.uk/~zoubin/papers/nips99.ps.gz.

Ghahramani, Z., & Jordan, M. I. (1997). Factorial hidden Markov models. *Machine Learning*, *29*, 245–273. http://citeseer.nj.nec.com/13664.html.

Herbrich, R., & Graepel, T. (2000). A PAC-Bayesian margin bound for linear classifiers: Why SVMs work. *NIPS 13*. http://www.research.microsoft.com/users/rherb/abstract-HerGrae00b.htm.

Herbrich, R., Graepel, T., & Campbell, C. (1999). Bayes point machines: Estimating the Bayes point in kernel space. *IJCAI Workshop Support Vector Machines* (pp. 23–27). http://stat.cs.tu-berlin.de/~ralfh/abstract-HerGraeCamp99a.html.

Hinton, G., & van Camp, D. (1993). Keeping neural networks simple by minimizing the description length of the weights. *Sixth ACM conference on Computational Learning Theory*. http://citeseer.nj.nec.com/hinton93keeping.html.

Jaakkola, T. S., & Jordan, M. I. (1999a). Variational probabilistic inference and the QMR-DT network. *Journal of Artificial Intelligence Research*, *10*, 291–322. http://www.cs.berkeley.edu/~jordan/papers/varqmr.ps.Z.

Jaakkola, T. S., & Jordan, M. I. (1999b). Bayesian parameter estimation via variational methods. *Statistics and Computing, to appear.* http://www.cs.berkeley.edu/~jordan/papers/variational-bayes.ps.Z.

Jaakkola, T. S., & Jordan, M. I. (1999c). Improving the mean field approximation via the use of mixture distributions. In *Learning in graphical models.* MIT Press. http://www.cs.berkeley.edu/~jordan/papers/mixture-mean-field.ps.Z.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning, 37*, 183–233. http://citeseer.nj.nec.com/jordan98introduction.html.

Kappen, H., & Wiegerinck, W. (2000). Second order approximations for probability models. *NIPS 13.* ftp://ftp.mbfys.kun.nl/snn/pub/reports/Kappen_NIPS2000.ps.

Kass, R. E., & Raftery, A. E. (1993). Bayes factors and model uncertainty (Technical Report 254). University of Washington. http://www.stat.washington.edu/tech.reports/tr254.ps.

Koller, D., Lerner, U., & Angelov, D. (1999). A general algorithm for approximate inference and its application to hybrid Bayes nets. *Uncertainty in AI* (pp. 324–333). http://robotics.Stanford.EDU/~koller/papers/uai99kla.html.

Kschischang, F. R., Frey, B. J., & Loeliger, H.-A. (2000). Factor graphs and the sum-product algorithm. *IEEE Trans Info Theory, to appear.* http://www.comm.toronto.edu/frank/factor/.

Kushner, H. J., & Budhiraja, A. S. (2000). A nonlinear filtering algorithm based on an approximation of the conditional distribution. *IEEE Trans Automatic Control, 45*, 580–585.

Lauritzen, S. L. (1992). Propagation of probabilities, means and variances in mixed graphical association models. *J American Statistical Association, 87*, 1098–1108.

Liu, J. S. (1999). Markov chain Monte Carlo and related topics (Technical Report). Department of Statistics, Stanford University. http://www-stat.stanford.edu/~jliu/TechRept/99.html.

Liu, J. S., & Chen, R. (2000). Mixture Kalman filters. *J Royal Statistical Society B, to appear.* http://www-stat.stanford.edu/~jliu/TechRept/00.html.

MacKay, D. J. C. (1995). Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems, 6*, 469–505. http://wol.ra.phy.cam.ac.uk/mackay/abstracts/network.html.

MacKay, D. J. C. (1998). Choice of basis for Laplace approximation. *Machine Learning, 33.* http://wol.ra.phy.cam.ac.uk/mackay/abstracts/laplace.html.

Maybeck, P. S. (1982). *Stochastic models, estimation and control*, chapter 12.7. Academic Press.

Minka, T. P. (2000a). Automatic choice of dimensionality for PCA. *NIPS 13*. ftp://whitechapel.media.mit.edu/pub/tech-reports/TR-514-ABSTRACT.html.

Minka, T. P. (2000b). Estimating a Dirichlet distribution. http://vismod.www.media.mit.edu/~tpminka/papers/dirichlet.html.

Minka, T. P. (2000c). Variational Bayes for mixture models: Reversing EM. http://vismod.www.media.mit.edu/~tpminka/papers/rem.html.

Murphy, K. (1998). Learning switching Kalman filter models (Technical Report CSD-98-990). Compaq Cambridge Research Lab. http://www.cs.berkeley.edu/~murphyk/Papers/skf.ps.gz.

Murphy, K., & Weiss, Y. (2000). The Factored Frontier algorithm for approximate inference in DBNs (Technical Report). UC Berkeley. http://www.cs.berkeley.edu/~murphyk/Papers/ff.ps.gz.

Murphy, K., Weiss, Y., & Jordan, M. (1999). Loopy-belief propagation for approximate inference: An empirical study. *Uncertainty in AI*. http://www.cs.berkeley.edu/~murphyk/Papers/loopy.ps.gz.

Neal, R. M. (1993). Probabilistic inference using Markov chain Monte Carlo methods (Technical Report CRG-TR-93-1). Dept. of Computer Science, University of Toronto. http://www.cs.toronto.edu/~radford/review.abstract.html.

Opper, M., & Winther, O. (1999). A Bayesian approach to on-line learning. *On-Line Learning in Neural Networks*. Cambridge University Press. http://www.ncrg.aston.ac.uk/cgi-bin/tr_avail.pl?trnumber=NCRG/99/029.

Opper, M., & Winther, O. (2000a). Adaptive TAP equations. *Advanced Mean Field Methods - Theory and Practice*. MIT Press. http://nimis.thep.lu.se/tf2/staff/winther/opper.adaptive.ps.Z.

Opper, M., & Winther, O. (2000b). Gaussian Processes and SVM: Mean field results and leave-one-out. *Advances in Large Margin Classifiers*. MIT Press. http://nimis.thep.lu.se/tf2/staff/winther/opper.largemargin.ps.Z.

Opper, M., & Winther, O. (2000c). Gaussian processes for classification: Mean field algorithms. *Neural Computation*. http://nimis.thep.lu.se/tf2/staff/winther/opper.neuralcomp.ps.Z.

Rasmussen, C. E. (1999). The infinite Gaussian mixture model. *NIPS 12*. http://bayes.imm.dtu.dk/pub/inf.mix.nips.99.abs.html.

Rujan, P. (1997). Perceptron learning by playing billiards. *Neural Computation*, *9*, 99–122. http://saturn.neuro.uni-oldenburg.de/~rujan/.

Rusmevichientong, P., & Roy, B. V. (1999). An analysis of turbo decoding with gaussian densities. *NIPS 12*.

Seeger, M. (1999). Bayesian model selection for support vector machines, Gaussian processes and other kernel classifiers. *NIPS 12*. http://www.dai.ed.ac.uk/homes/seeger/papers/nips-paper.ps.gz.

Shachter, R. (1990). A linear approximation method for probabilistic inference. *Uncertainty in AI.*

Stephens, M. (1997). *Bayesian methods for mixtures of normal distributions.* Doctoral dissertation, Oxford University.
`http://www.stats.ox.ac.uk/~stephens/papers.html`.

Storkey, A. (2000). Dynamic trees: A structured variational method giving efficient propagation rules. *UAI'00.* `http://anc.ed.ac.uk/~amos/uai143.ps`.

Tong, S., & Koller, D. (2000). Restricted Bayes optimal classifiers. *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI).*
`http://robotics.stanford.edu/~koller/papers/aaai00tk.html`.

Ventura, V. (2000). Double importance sampling (Technical Report). Dept of Statistics, Carnegie Mellon University.
`http://www.stat.cmu.edu/cmu-stats/tr/tr694/tr694.html`.

Waterhouse, S., MacKay, D., & Robinson, T. (1995). Bayesian methods for mixtures of experts. *NIPS 8.*

Watkin, T. (1993). Optimal learning with a neural network. *Europhysics Letters*, *21*, 871–876.

Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., & Vapnik, V. (2000). Feature selection for SVMs. *NIPS 13.* `http://www.ens-lyon.fr/~ochapell/`.

Winther, O. (1998). *Bayesian mean field algorithms for neural networks and Gaussian processes.* Doctoral dissertation, University of Copenhagen.
`http://nimis.thep.lu.se/tf2/staff/winther/thesis.ps.Z`.

Yedidia, J. S. (2000). An idiosyncratic journey beyond mean field theory (Technical Report TR2000-27). MERL. `http://www.merl.com/reports/TR2000-27/index.html`.

Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2000). Generalized belief propagation (Technical Report TR2000-26). MERL.
`http://www.merl.com/reports/TR2000-26/index.html`.