Bayesian Methods for Machine Learning

Radford M. Neal, University of Toronto

http://www.cs.utoronto.ca/~radford/

NIPS Tutorial, 13 December 2004

Tutorial Outline

Bayesian inference is based on using probability to represent all forms of uncertainty. In this tutorial, I will discuss:

1) How this is done, in general terms.

2) The details for a simple example — a hard linear classifier.

3) How Bayesian methods differ from other approaches.

4) Two big challenges — prior specification and computation.

5) Some particular models/priors:
   – Neural network and Gaussian process models for regression/classification.
   – Mixture models, finite and infinite, for density modeling and clustering.

6) Some computational techniques:
   – Markov chain Monte Carlo.
   – Variational approximations.

7) Some misguided "Bayesian" methods.

8) Some successes of Bayesian methodology.

The Bayesian Approach to Machine Learning (Or Anything)

1) We formulate our knowledge about the situation probabilistically:
   - We define a *model* that expresses qualitative aspects of our knowledge (eg, forms of distributions, independence assumptions). The model will have some unknown *parameters*.
   - We specify a *prior* probability distribution for these unknown parameters that expresses our beliefs about which values are more or less likely, before seeing the data.
2) We gather data.
3) We compute the *posterior* probability distribution for the parameters, given the observed data.
4) We use this posterior distribution to:
   - Reach scientific conclusions, properly accounting for uncertainty.
   - Make predictions by averaging over the posterior distribution.
   - Make decisions so as to minimize posterior expected loss.

Finding the Posterior Distribution

The *posterior distribution* for the model parameters given the observed data is found by combining the prior distribution with the likelihood for the parameters given the data.

This is done using *Bayes' Rule*:

$$P(\text{parameters} \,|\, \text{data}) \quad = \quad \frac{P(\text{parameters})\, P(\text{data} \,|\, \text{parameters})}{P(\text{data})}$$

The denominator is just the required normalizing constant, and can often be filled in at the end, if necessary. So as a proportionality, we can write

$$P(\text{parameters} \,|\, \text{data}) \quad \propto \quad P(\text{parameters})\, P(\text{data} \,|\, \text{parameters})$$

which can be written schematically as

$$\text{Posterior} \quad \propto \quad \text{Prior} \;\times\; \text{Likelihood}$$

We make predictions by integrating with respect to the posterior:

$$P(\text{new data} \,|\, \text{data}) \quad = \quad \int_{\text{parameters}} P(\text{new data} \,|\, \text{parameters})\, P(\text{parameters} \,|\, \text{data})$$

Representing the Prior and Posterior Distributions by Samples

The complex distributions we will often use as priors, or obtain as posteriors, may not be easily represented or understood using formulas.

A very general technique is to represent a distribution by a *sample* of many values drawn randomly from it. We can then:

– *Visualize* the distribution by viewing these sample values, or low-dimensional projections of them.

– Make *Monte Carlo* estimates for probabilities or expectations with respect to the distribution, by taking averages over these sample values.

Obtaining a sample from the prior is often easy. Obtaining a sample from the posterior is usually more difficult — but this is nevertheless the dominant approach to Bayesian computation.

Inference at a Higher Level: Comparing Models

So far, we've assumed we were able to start by making a definite choice of model. What if we're unsure which model is right?

We can compare models based on the *marginal likelihood* (aka, the *evidence*) for each model, which is the probability the model assigns to the observed data. This is the normalizing constant in Bayes' Rule that we previously ignored:

$$P(\text{data} \mid M_1) \;=\; \int_{\text{parameters}} P(\text{data} \,|\, \text{parameters}, M_1)\, P(\text{parameters} \,|\, M_1)$$

Here, M_1 represents the condition that model M_1 is the correct one (which previously we silently assumed). Similarly, we can compute $P(\text{data} \mid M_2)$, for some other model (which may have a different parameter space).

We might choose the model that gives higher probability to the data, or average predictions from both models with weights based on their marginal likelihood, multiplied by any prior preference we have for M_1 versus M_2.

A Simple Example — A Hard Linear Classifier

The problem:

We will be observing pairs $(x^{(i)}, y^{(i)})$, for $i = 1, \ldots, n$, where $x = (x_1, x_2)$ is a 2D "input" and y is a $-1/+1$ class indicator. We are interested in predicting y from x. We are not interested in predicting x, and this may not even make sense (eg, we may determine the $x^{(i)}$ ourselves).

Our informal beliefs:

We believe that there is a line somewhere in the input space that determines y perfectly — with -1 on one side, $+1$ on the other.

We think that this line could equally well have any orientation, and that it could equally well be positioned anywhere, as long as it is no more than a distance of three from the origin at its closest point.

We need to translate these informal beliefs into a *model* and a *prior*.

Formalizing the Model

Our model can be formalized by saying that

$$P(y^{(i)} = y \,|\, x^{(i)}, u, w) \quad = \quad \begin{cases} 1 & \text{if } y\, u\, (w^T x^{(i)} - 1) > 0 \\ 0 & \text{if } y\, u\, (w^T x^{(i)} - 1) < 0 \end{cases}$$

where $u \in \{-1, +1\}$ and $w = (w_1, w_2)$ are unknown *parameters* of the model. The value of w determines a line separating the classes, and u says which class is on which side. (Here, $w^T x$ is the scalar product of w and x.)

This model is rather dogmatic — eg, it says that y is **certain** to be $+1$ if $u = +1$ and $w^T x$ is greater than 1. A more realistic model would replace the probabilities of 0 and 1 above with ϵ and $1 - \epsilon$ to account for possible unusual items, or for misclassified items. ϵ might be another unknown parameter.

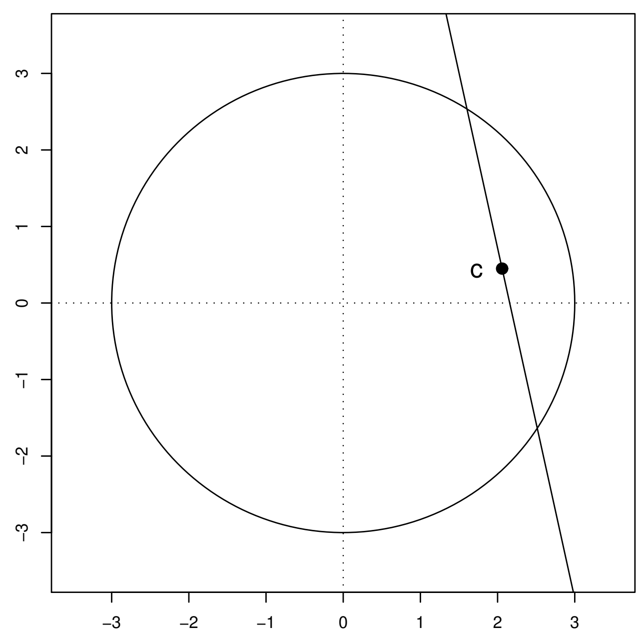
Formalizing the Prior

A line is completely determined by giving the point, c, on the line that is closest to the origin.

To formalize our prior belief that the line separating classes could equally well be anywhere, as long as it is no more than a distance of three from the origin, we decide to use a uniform distribution for c over the circle with radius 3.

Given c, we can compute $w = c/||c||^2$, which makes $w^T x = 1$ for points on the line. (Here, $||c||^2$ is the squared norm, $c_1^2 + c_2^2$.)

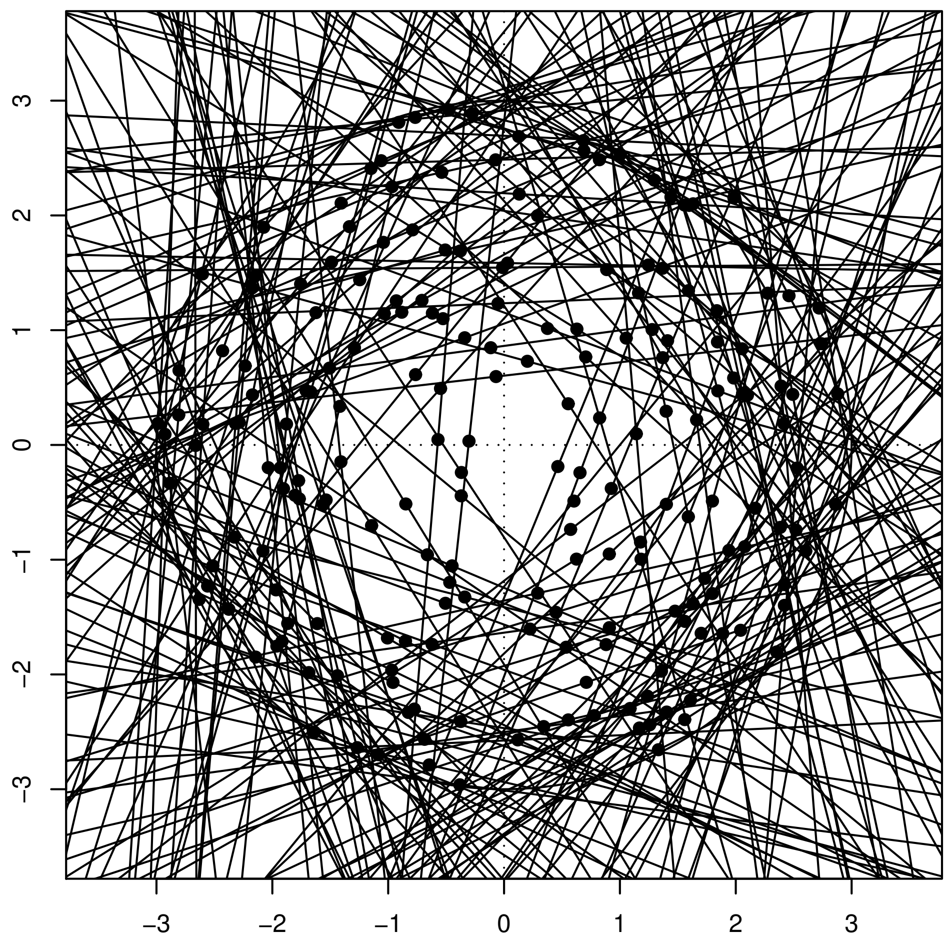
Here's an example:



We also say that u is equally likely to be $+1$ or -1, independently of w.

Looking at the Prior Distribution

We can check this prior distribution by looking at many lines sampled from it:

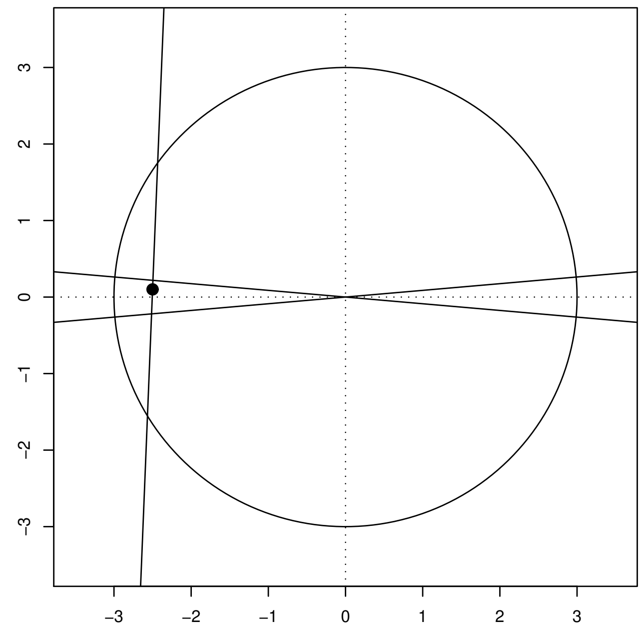


Something's wrong here. We meant for the lines to be uniformly distributed, but we see a sparse region near the origin.

Why This Prior Distribution is Wrong

Our first attempt at formalizing our prior beliefs didn't work. We can see why if we think about it.

Imagine moving a line that's within five degrees of vertical from left to right:

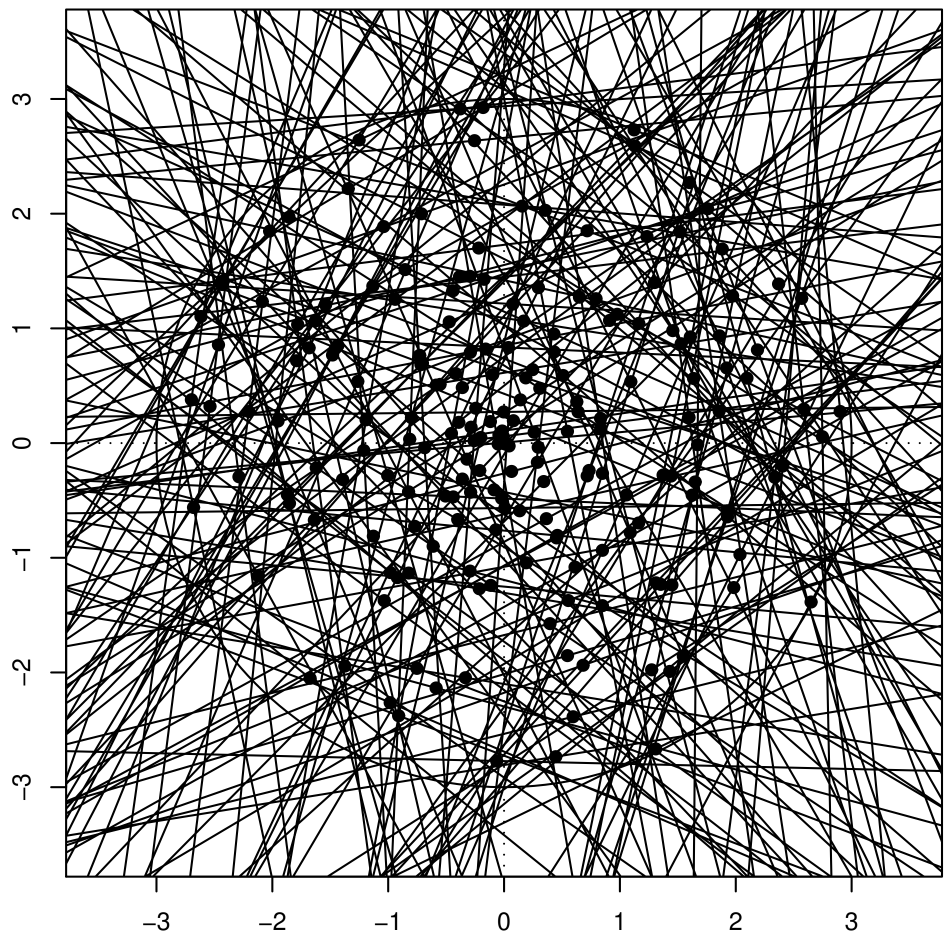


To stay within five degrees of vertical, the closest point to the origin has to be within the wedge shown. This becomes less and less likely as the origin is approached. We don't get the same probability of a near-vertical line for all horizontal positions.

Fixing the Prior Distribution

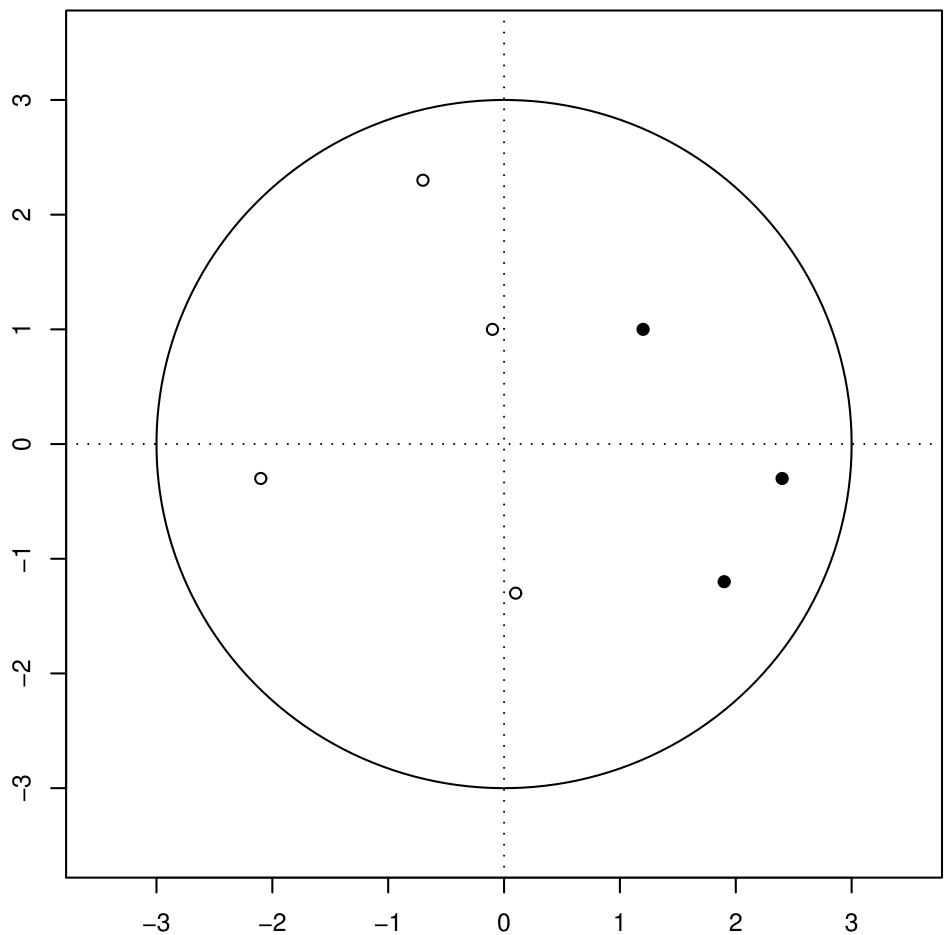
We can fix the prior by letting the closest point on the line to the origin be $c = ru$, with r uniformly distributed over $(0, 3)$ and u uniformly distributed over the unit circle.

Now a sample drawn from the prior looks the way we want it to:



Some Data Points

Now that we have defined our model and prior, let's get some data:



The black points are in class $+1$, the white points in class -1.

Posterior Distribution for the Hard Linear Classifier

For the hard linear classifier, the likelihood is either 0 or 1:

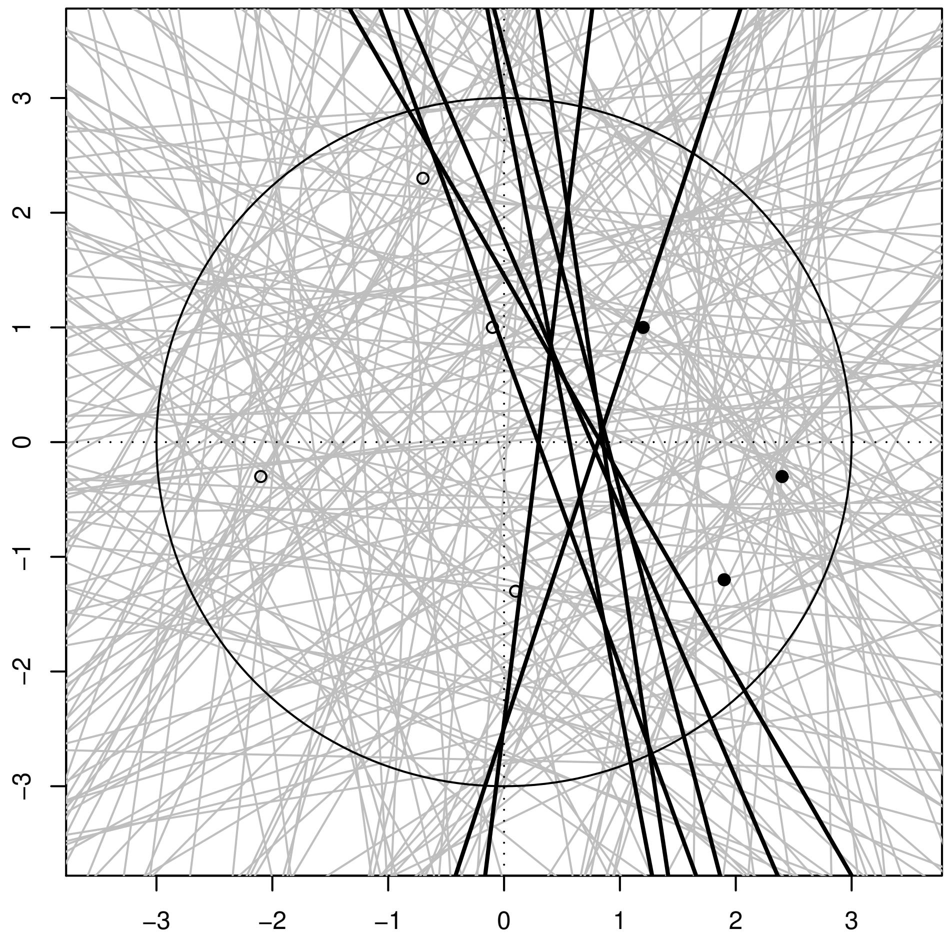
$$\begin{aligned} P(y^{(1)}, \ldots, y^{(n)} \,|\, x^{(1)}, \ldots, x^{(n)}, u, w) &= \prod_{i=1}^{n} P(y^{(i)} \,|\, x^{(i)}, u, w) \\ &= \begin{cases} 1 & \text{if } y^{(i)}\, u\, (w^T x^{(i)} - 1) > 0, \ \text{ for } i = 1, \ldots, n \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

The posterior distribution for u and w is therefore the same as their prior distribution, except that parameter values incompatible with the data are eliminated.

After renormalizing so that posterior probabilities integrate to one, the parameter values compatible with the data will have higher probability than they did in the prior.

Obtaining a Sample from the Posterior Distribution

To obtain a sample of values from the posterior, we can sample w values from the prior, but retain only those that are compatible with the data (for some u). Here's what we get using a sample of size 200:



The eight bold lines are a random sample from the posterior distribution.

Making a Prediction for a Test Case

The Bayesian predictive probability that in a test case with inputs x^*, the class, y^*, will be $+1$ is found by integrating/summing over the parameters w and u:

$$P(y^* = +1 \,|\, x^*, (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}))$$
$$= \int \sum_{u=\pm 1} P(y^* = +1 \,|\, x^*, u, w)\, P(u, w \,|\, x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\, dw$$

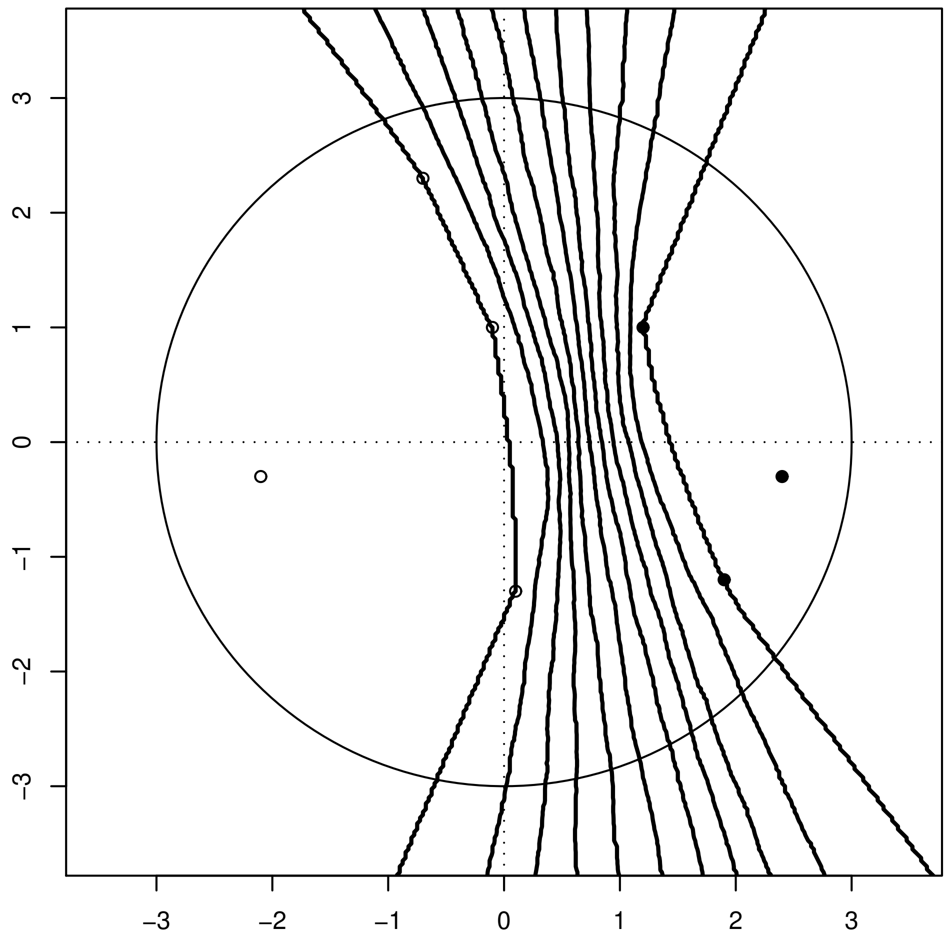
Using a sample of K values from the posterior, $(u^{(1)}, w^{(1)}), \ldots, (u^{(K)}, w^{(K)})$, we can approximate this as follows:

$$P(y^* = +1 \,|\, x^*, (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})) \;\approx\; \frac{1}{K} \sum_{j=1}^{K} P(y^* = +1 \,|\, x^*, u^{(j)}, w^{(j)})$$

For this model, $P(y^* = +1 \,|\, x^*, u^{(j)}, w^{(j)})$ is either 0 or 1, depending on the sign of $u^{(j)}\,({w^{(j)}}^T x^* - 1)$. The average above is just the fraction of lines drawn from the posterior that would put the test point in class $+1$.

A Plot of the Predictive Probabilities

Here is a contour plot over the input space of the approximate predictive probability of class +1, based on a sample of size 10000 from the prior, which resulted in a sample of size 450 from the posterior:



The contour lines go from 0 on the left to 1 on the right, in steps of 0.1.

The Marginal Likelihood

The sample of 10000 values from the prior also lets us estimate the marginal likelihood for this model, given the seven observed data points.

We consider the $x^{(i)}$ to be fixed (not random), so the marginal likelihood is just the probability of all the $y^{(i)}$ having their observed values. This probability is one for a line that classifies all the points correctly, and zero for any other line.

We can therefore estimate the marginal likelihood by the fraction of lines drawn from the prior that are compatible with the data: $450/10000 = 0.045$.

We could use this to compare this model with some other, such as a model that said the classes were separated by quadratic rather than linear curves.

However... the marginal likelihood is **very sensitive** to the prior used. If we used a prior for the separating line that was uniform over a bigger region, say allowing the closest point to the origin to be up to a distance of 10 away, the marginal likelihood would be smaller. Computing marginal likelihoods makes sense only if you have given careful thought to the prior.

Final Thoughts on This Example

- We see that correctly translating informal knowledge into a prior distribution isn't always trivial.
- However, a prior can be *tested*, by checking how well it corresponds to our prior beliefs. Prior distributions are **not** "arbitrary".
- More elaborate priors might sometimes be appropriate. For example, we might use a prior that favoured lines that are almost horizontal or vertical, if we believe that probably one of the two inputs is mostly irrelevant.
- For a data set with seven points, only about 4.5% of the parameter values we drew from the prior made it into the posterior sample. This technique isn't going to work for realistic problems. We need better ways of sampling from the posterior distribution.
- Several papers (eg, Ruján 1997, Herbrich, Graepel, Campbell 2001) have discussed models like this one. Sampling from the posterior can be done using a "billiard" technique. Some of this work emphasizes approximating the Bayesian predictive probabilities using a single "best" separating line. In my opinion, such eagerness to abandon the correct answer is probably not the most productive line of research.

Distinctive Features of the Bayesian Approach

Probability is used not only to describe "physical" randomness, such as errors in labeling, but also uncertainty regarding the true values of the parameters. These prior and posterior probabilities represent **degrees of belief**, before and after seeing the data.

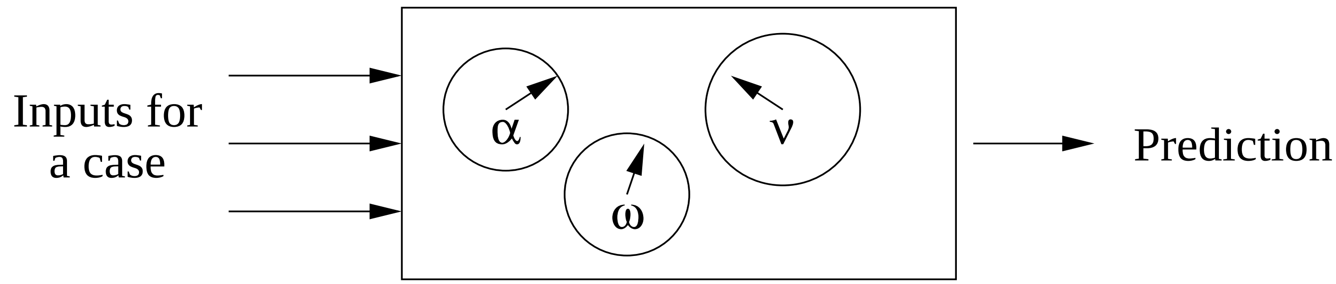
The Bayesian approach takes **modeling** seriously. A Bayesian model includes a suitable prior distribution for model parameters. If the model/prior are chosen without regard for the actual situation, there is *no justification* for believing the results of Bayesian inference.

The model and prior are chosen based on our knowledge of the problem. These choices are **not**, in theory, affected by the amount of data collected, or by the question we are interested in answering. We **do not**, for example, restrict the complexity of the model just because we have only a small amount of data.

Pragmatic compromises are inevitable in practice — eg, no model and prior perfectly express to our knowledge of the situation. The Bayesian approach relies on reducing such flaws to a level where we think they won't seriously affect the results. If this isn't possible, it may be better to use some other approach.

Contrast With the "Learning Machine" Approach

One view of machine learning pictures a "learning machine", which takes in inputs for a training/test case at one end, and outputs a prediction at the other:



The machine has various "knobs", whose settings change how a prediction is made from the inputs. Learning is seen as a procedure for twiddling the knobs in the hopes of making better predictions on test cases — for instance, we might use the knob settings that minimize prediction error on training cases.

This approach differs profoundly from the Bayesian view:

– The choice of learning machine is essentially *arbitrary* — unlike a model, the machine has no meaningful semantics, that we could compare with our beliefs.

– The "knobs" on the machine *do not* correspond to the parameters of a Bayesian model — Bayesian predictions, found by averaging, usually cannot be reproduced using *any* single value of the model parameters.

Contrast With "Learning Theory"

An aim of "learning theory" is to prove that certain learning machines "generalize" well. One can sometimes prove that if you adjust the knobs on the learning machine to minimize training error, then apply it to test cases, the training error rates and test error rates are unlikely to be far apart:

$$P(|\text{test error rate} - \text{training error rate}| > \epsilon) \quad < \quad \delta$$

where δ and ϵ have certain small values, which depend on the training set size.

Such a result would be of little interest, if it weren't usually interpreted as guaranteeing that, for instance:

$$P(|\text{test error rate} - 0.02| > \epsilon \ \mid \ \text{training error rate} = 0.02) \quad < \quad \delta$$

This is a fallacy, however — no valid probabilistic statement about test error rates conditional on the observed error rate on training cases is possible without assuming some prior distribution over possible situations. This fallacy is analogous to the common misinterpretation of a frequentist p-value as the probability that the null hypothesis is true, or of a frequentist confidence interval as an interval that likely contains the true value.

What About "Bias" and "Variance"?

Another approach to analysing learning methods (especially for predicting real-valued quantities) looks at the following two indicators of how well a method predicts some quantity:

Bias: how much predictions depart from the truth on average.

Variance: the average squared difference of predictions from their average.

The average squared error for the method can be decomposed as the sum of the squared bias and the variance. This leads to a strategy: choose a method that minimizes this sum, possibly trading off increased bias for reduced variance, or vice versa, by adjusting complexity, or introducing some form of "regularization".

There are two problems with this strategy:

- The bias and variance depend on the true situation, which is unknown.
- There is no reason to think that trying nevertheless to minimize squared bias plus variance produces a unique answer.

Assessments of bias and variance play no role in the Bayesian approach.

The Challenge of Specifying Models and Priors

The first challenge in making the Bayesian approach work is to choose a suitable model and prior. This can be especially difficult for the complex, high-dimensional problems that are traditional in machine learning.

> **A suitable model** should encompass all the possibilities that are thought to be at all likely. Unrealistically limited forms of functions (eg, linear) or distributions (eg, normal) should be avoided.

> **A suitable prior** should avoid giving zero or tiny probability to real possibilities, but should also avoid spreading out the probability over all possibilities, however unrealistic. To avoid a prior that is too spread out, dependencies between parameters may need to be modeled.

Unfortunately, the effort in doing a good job can easily get out of hand. One strategy is to introduce *latent variables* into the model, and *hyperparameters* into the prior. Both of these are devices for modeling dependencies in a tractable way.

In practice, an iterative approach may be needed — we formulate our model and prior, try it out on data, and see by examining *diagnostics* that we've made a mistake. We go back and revise our model or prior, trying to avoid having our choice be unduly influenced by the data (since the data would then count twice).

Infinite Models

Many real phenomena are of essentially unlimited complexity:

> Suppose we model consumer behaviour by categorizing consumers into various "types" (mixture components). There is no reason to think that there are only (say) five types of consumer. Surely there are an unlimited number of types, though some may be rare.
>
> Suppose we model the growth rate of trees as a function of climate, soil type, genetic characteristics, disease suppression measures taken, etc. There is no reason to think any simple functional form (eg, linear, low-order polynomial) will capture the many ways these factors interact to determine tree growth.

How can we build a model that accommodates such complexity? One approach:

- Define models that can have any finite amount of complexity (eg, a finite number of mixture components, or of hidden units).
- Define priors for these models that make sense.
- See if the limit as the complexity goes to infinity is sensible.

If the limit makes sense, we can use a model that is as large as we can handle computationally. And sometimes, we can figure out how to actually implement the infinite model on a finite computer!

The Computational Challenge

The other big challenge in making Bayesian modeling work is computing the posterior distribution. There are four main approaches:

Analytical integration: Works when "conjugate" prior distributions can be used, which combine nicely with the likelihood — usually too much to hope for.

Gaussian approximation: Works well when there's a lot of data, compared to the model complexity — the posterior distribution in then close to Gaussian, and can be handled by finding its mode, and the second derivatives at the mode.

Monte Carlo integration: Once we have a sample of parameter values from the posterior distribution, most things are possible. But how to get a sample?

- *Simple Monte Carlo* — sample directly from the posterior. Seldom possible.
- *Importance sampling* — sample from an approximation to the posterior, then reweight to compensate for the difference. Maybe OK in moderate dimensions.
- *Markov chain Monte Carlo (MCMC)* — simulate a Markov chain that eventually converges to the posterior distribution. Can be applied to a remarkable variety of problems. Currently the dominant approach.

Variational approximation: A cleverer way to approximate the posterior. May sometimes be faster than MCMC, but it's not as general, and not exact.

Multilayer Perceptron Neural Networks

The multilayer perceptron (MLP) network has long been popular for regression and classification problems. Such a network computes its outputs, $o_1, \ldots, o_q$, from its inputs, $x_1, \ldots, x_p$, using a layer of N *hidden units*:

$$\begin{aligned} o_k(x) &= b_k + \sum_{j=1}^{N} v_{jk} h_j(x) \\ h_j(x) &= \tanh\left(a_j + \sum_{i=1}^{p} u_{ij} x_i\right) \end{aligned}$$

The tanh function is sigmoidal, with value ranging from -1 at $-\infty$ to $+1$ at $+\infty$.

These networks can approximate any function arbitrarily well, with enough hidden units, and appropriate parameters, or *weights*, $w = \{a_j, u_{ij}, b_j, v_{jk}\}$.

There are many variations: more layers of hidden units, direct input-to-output connections, other activation functions.

Models From Neural Networks

We can use a multilayer perceptron network to build a non-linear regression or classification model.

For regression, we use a network with one output, $o(x)$, and let

$$y \,|\, x, w, \sigma \quad \sim \quad \text{Gaussian}(o(x), \sigma^2)$$

where $\sim$ means "has the distribution", and the right side above represents a Gaussian distribution with the given mean and variance. Note that $o(x)$ depends on w. σ is an additional model parameter.

For binary classification, we again use one output, and let

$$P(y = 1 \,|\, x, w) \quad = \quad \frac{1}{1 + \exp(-o(x))}$$

For classification with q classes, we use as many outputs as classes, and let

$$P(y = k \,|\, x, w) \quad = \quad \frac{\exp(-o_k(x))}{\sum_{j=1}^{q} \exp(-o_j(x))}$$

The Meaning of a Neural Network Model & Prior

A network with only a few hidden units is not a reasonable model — seldom is the real function we need a sum of tanh functions.

But a network with many hidden units is a reasonable "non-parametric" model, since it can approximate any function arbitrarily closely.

So we should use a network with many hidden units. But how does a prior over network weights translate into a prior over the functions computed by the network, and hence the distributions for y given x?

Suppose we give independent Gaussian priors to all the network parameters:

$$\begin{aligned} a_k &\sim \text{Gaussian}(0, \sigma_a^2) \\ u_{ij} &\sim \text{Gaussian}(0, \sigma_u^2) \\ b_k &\sim \text{Gaussian}(0, \sigma_b^2) \\ v_{jk} &\sim \text{Gaussian}(0, \sigma_v^2) \end{aligned}$$

This produces a prior distribution for the outputs of the network at any selection of input points — i.e. for $o(x^{(1)})$, $o(x^{(2)})$, ... It is this distribution that really matters.

The Limit of the Prior Over Functions

Consider the prior over functions as the number of hidden units, N, goes to infinity.

Look at one component, k, of the function evaluated at a single point, $x^{(1)}$:

$$o_k(x_1) \quad = \quad b_k + \sum_{j=1}^{N} v_{jk} h_j(x^{(1)})$$

The first term above is Gaussian.

By the Central Limit Theorem, the second term will become Gaussian as $N \to \infty$, as long as each term has finite variance. Since $h_j(x^{(1)})$ is bounded, this must be the case.

Hence $o_k(x_1)$ becomes Gaussian for large N. Its distribution will reach a limit if we make σ_v scale as $N^{-1/2}$.

Similarly, the joint distribution of the function at any number of input points converges to a multivariate Gaussian — i.e. we have a Gaussian process prior over functions.

Some Properties of the Gaussian Network Prior

- The hidden-to-output weights go to zero as the number of hidden units goes to infinity. So hidden units are individually of no importance.

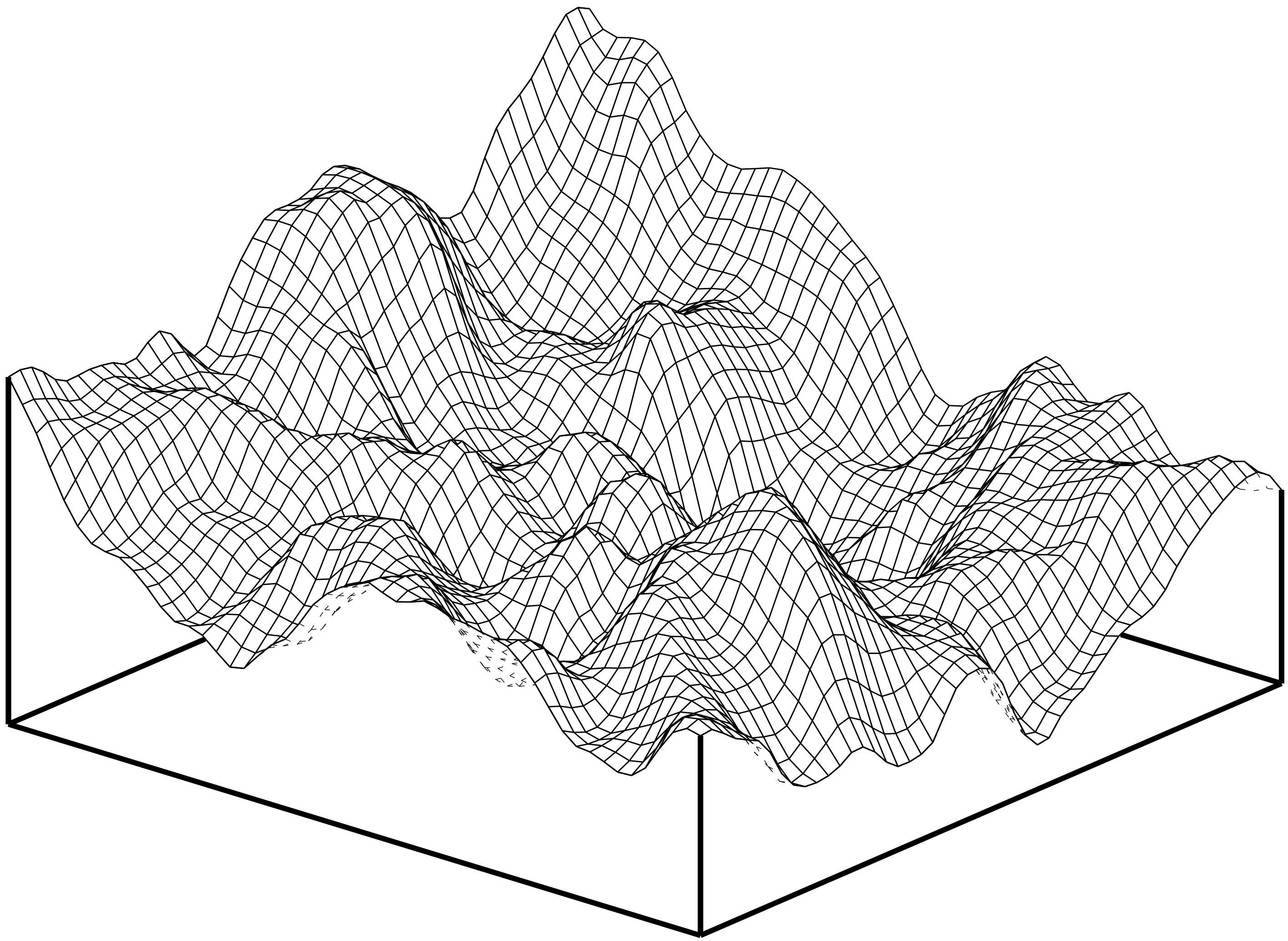
- With a smooth hidden unit activation function, the functions are smooth, with
$$\mathrm{Corr}\left[o(x^{(1)}),\, o(x^{(2)})\right] \;\approx\; 1 - |x^{(1)} - x^{(2)}|^2$$

- If the hidden units use a step function, the functions are locally Brownian, with
$$\mathrm{Corr}\left[o(x^{(1)}),\, o(x^{(2)})\right] \;\approx\; 1 - |x^{(1)} - x^{(2)}|$$

- The functions computed by different output units are independent.

- Even with more hidden layers, we still get a Gaussian process prior (perhaps with a different covariance function).

A Function From the Gaussian Network Prior



The network had two inputs, one output, and 10000 tanh hidden units.

Priors Based on Other Stable Distributions

If $X_1, \ldots, X_N$ are i.i.d. from a symmetric stable distribution of index $\alpha \in (0, 2]$,

$$(X_1 + \cdots + X_N)\, N^{-1/\alpha}$$

has the same stable distribution. The same is true as $N \to \infty$ if the X_i are in the "domain of attraction" of the stable distribution.

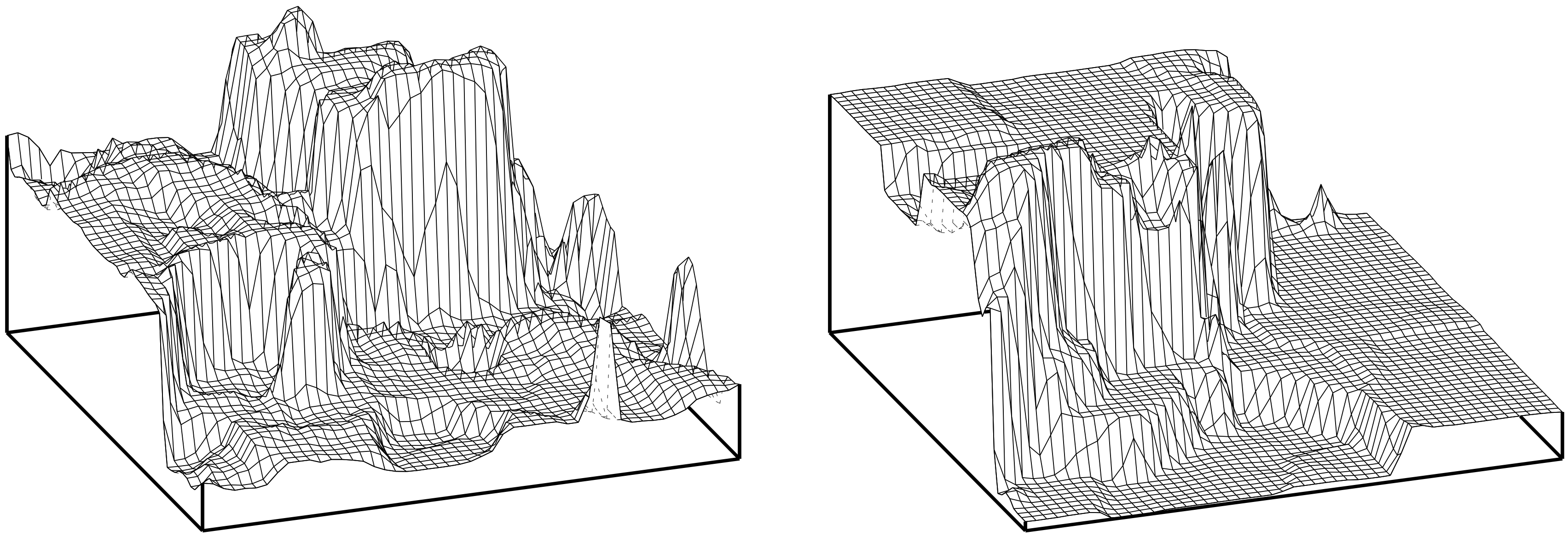
If we give the hidden-to-output weights a prior in the domain of attraction of the stable distribution of index α, and scale the width of the prior as $N^{-1/\alpha}$, we should get a well-defined $N \to \infty$ limit.

Example: Use a Cauchy prior for the v_{jk} with width parameter scaling as N^{-1}.

Properties of Non-Gaussian Stable Priors

- The hidden-to-output weights do not go to zero, but asymptotically come from a Poisson process. This is an alternative way of defining the prior.
- The functions computed by different output units can be made dependent without being correlated, by making the weights from the same hidden unit be dependent.
- Networks with more than one hidden layer can produce interesting new effects.
- Analysis seems more difficult than for Gaussian priors.

Functions From a Network With Two Hidden Layers

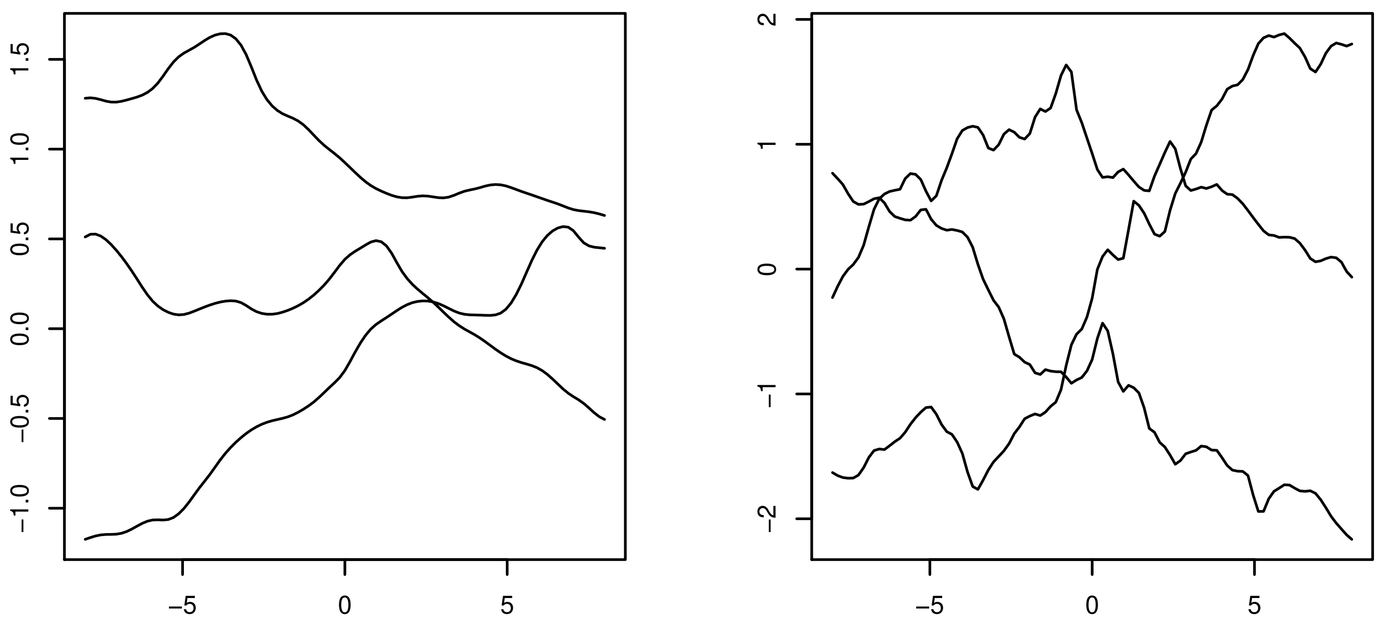


The networks had two inputs and one output. Gaussian priors were used for weights into the first hidden layer (with 500 tanh units) and second hidden layer (300 tanh units), but the prior on the weights from the second hidden layer to the output had a t-distribution with $\alpha = 0.6$. The two functions differ only in the random number seed used when generating weights from their priors.

Hyperparameters for Neural Network Models

The priors we give to weights, such as $u_{ij} \sim \text{Gaussian}(0, \sigma_u^2)$ for input-to-hidden weights, affect the nature of functions drawn from the network prior. Here are samples of three functions (of one input) drawn using Gaussian priors for networks of 1000 hidden units using two different σ_u^2:

$\sigma_u = 1$ $\sigma_u = 5$

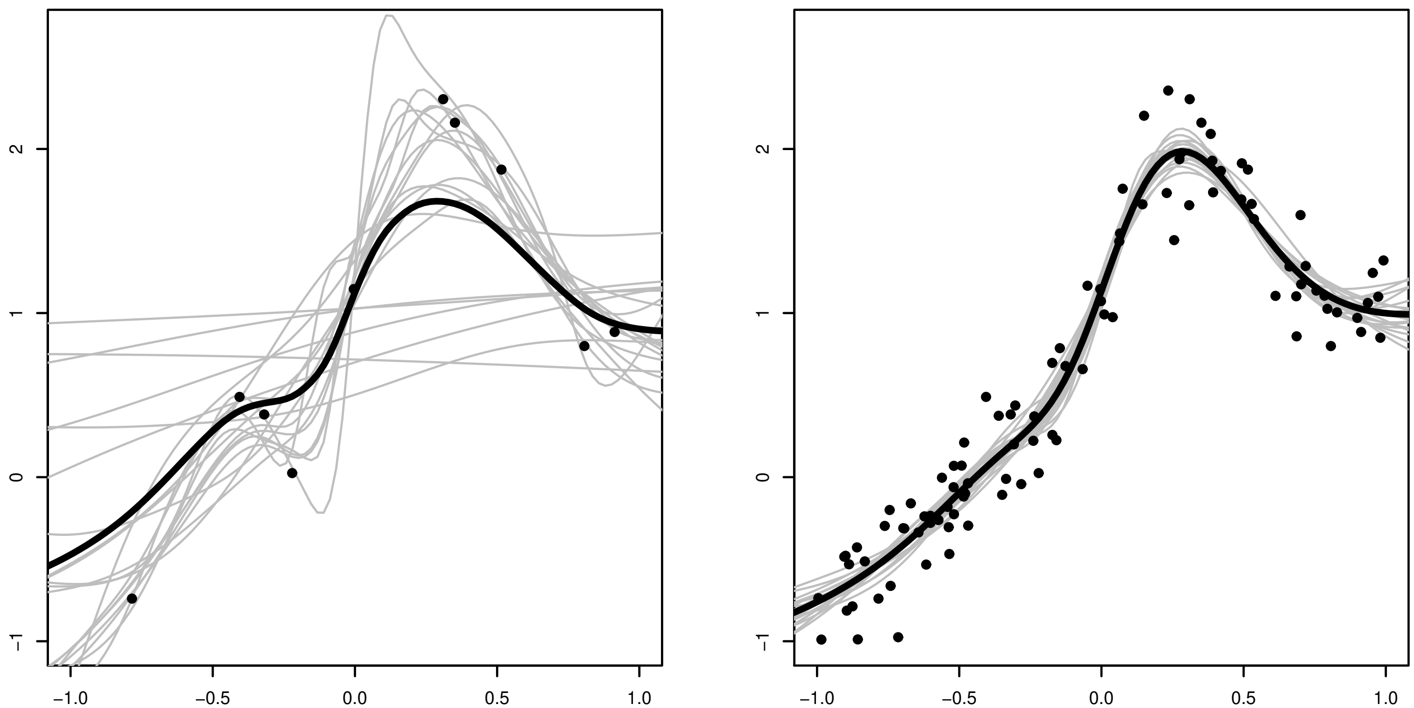


A larger σ_u produces "wigglier" functions. Usually, we won't know exactly how wiggly the function should be. So we make σ_u a variable *hyperparameter*, and give it a prior distribution that spans a few orders of magnitude. We usually make σ_a, σ_b, and σ_v hyperparameters as well.

Complexity of Functions Versus Complexity of Predictions

Here's an illustration of how a complex model — a neural network with 100 hidden units — can produce simple predictions when there is little data, despite being capable of representing more complex functions that fit the data better.

The data is from the function $x^3 + 2\exp(-6(x-0.3)^2)$, with Gaussian noise with standard deviation 0.2 added. The plots show show 20 functions from the posterior distributions given 10 data points (left) and 100 data points (right):



The bold line is the average of these functions, the Bayesian prediction for minimizing squared error, which is smoother than some individual functions.

Hierarchical Hyperpriors: Automatic Relevance Determination

More elaborate priors for hyperparameters can be used to allow for various possible high-level characteristics of the data. One very useful example is the *Automatic Relevance Determination* prior.

When we have many inputs, we are often uncertain how relevant each is to predicting the target variable. We can express this uncertainty by using a prior for input-to-hidden weights such as the following:

$$
\begin{aligned}
u_{ij} \mid \sigma_{u,i} &\sim \text{Gaussian}(0, \sigma_{u,i}^2), \quad \text{for } i = 1, \ldots, p \text{ and } j = 1, \ldots, N \\
1/\sigma_{u,i}^2 \mid \sigma_{u,*} &\sim \text{gamma}(\text{mean} = 1/\sigma_{u,*}^2,\ \text{shape} = \ldots), \quad \text{for } i = 1, \ldots, p \\
1/\sigma_{u,*}^2 &\sim \text{gamma}(\ldots)
\end{aligned}
$$

Here, $\sigma_{u,i}$ controls the magnitude of weights from input i to the hidden units. If it is small, this input will largely be ignored. The top-level hyperparameter $\sigma_{u,*}$ controls the distribution of the lower-level $\sigma_{u,i}$.

Gaussian Process Models

The limit of an MLP network with infinitely-many hidden units, using Gaussian priors, is a Gaussian process, in which the joint distribution of the values of the function at any set of input points is multivariate Gaussian. Why not just get rid of the network, and work directly with this (or some other) Gaussian process?

This is indeed an attractive idea. We need only specify a *covariance function* for input points, $C(x, x') = \mathrm{Cov}(o(x), o(x'))$, where $o(x)$ is the function value (analogous to the network output). Neural networks with Gaussian priors produce rather complicated covariance functions. Simpler ones may be just as good, such as the "squared exponential":

$$C(x, x') \;=\; a \exp(-b\,|x - x'|^2)$$

where a and b are constants or (more likely) hyperparameters.

This Gaussian process prior for a function combines nicely with a Gaussian model for the noise in observations of this function — we just add the noise variance to the covariance of an observation with itself.

Prediction for Gaussian Process Regression

In a Gaussian process regression model with Gaussian noise, the posterior distribution for function values is also Gaussian. We can therefore make predictions for test cases using simple matrix operations.

Let C be the matrix of covariances between the function values at all the points in the training set, found using the covariance function we chose, with the variance of the observation noise added to the diagonal. Let k be the vector of covariances between the points in the training set and a test point. Let v be the prior variance for the test observation (covariance with itself plus noise). Let t be the vector of training observations.

Then the predictive mean and variance for the test observation are

$$\begin{aligned} \textbf{mean} &= k^T \mathrm{C}^{-1} t \\ \textbf{variance} &= v - k^T C^{-1} k \end{aligned}$$

This takes $O(n^3)$ time to find C^{-1}, then $O(n)$ time to find the predictive mean for each test case, and $O(n^2)$ time to find the predictive variance.

Gaussian Process Classification Models

Building a classification model using a Gaussian process is less straightforward, requiring the introduction of "latent values" associated with each case.

For a binary classification problem, there is one latent value per case. If this value is o, the probability of class $+1$ is defined to be $1/(1 + \exp(-o))$. This is analogous to how we translated the output of a network to a class probability.

For multi-class problems, we have as many latent values per case as possible classes. The probability of a class is then proportional to the exponential of its latent value, again analogously to the network models.

We give Gaussian process priors to these latent values. We can no longer do inference with just matrix operations, however. We need to sample from the distribution of latent values, or use some approximation to their distribution.

See Neal (1998b) and MacKay (1997) for more on Gaussian process regression and classification.

Gaussian Processes and Support Vector Machines

There are strong connections between Gaussian process (GP) models and Support Vector Machines (SVMs) for regression and classification.

The covariance function corresponds directly with the "kernel function" of a SVM. There is one big advantage to the SVM approach:

- Although both GP and SVM methods take time proportional to n^3 in the worst case, SVMs often take much less time, when the number of "support vectors" is much less than n.

There have been several attempts to speed up the GP computations to overcome their disadvantage in this respect (eg, Seeger, Williams, Lawrence 2003).

The Bayesian GP approach has several advantages over SVMs:

- Classification with more than two classes is easily handled, as are numerous other models (eg, Poisson regression).
- Predictions are probabilistic, incorporating uncertainty due to noise and due to sparseness of training data.
- Hyperparameters in the covariance function can easily be inferred. One can use Automatic Relevance Determination, for example.

Mixture Models for Density Estimation and Clustering

Suppose we have data, $y^{(1)}, \ldots, y^{(n)}$, drawn independently from some unknown distribution. The $y^{(i)}$ may be multidimensional.

We may wish to estimate the probability density of this data.

We may also wish to analyse the data in terms of "clusters", which perhaps correspond to sub-populations — eg, teachers with different instructional styles, or different species of iris. We don't know which sub-population each data point came from. We may not know how many sub-populations (clusters) exist.

Mixtures of simple distributions are suitable models for both density estimation and clustering. We can model the density of y as

$$\sum_{c=1}^{K} p_c \, f(y \,|\, \phi_c)$$

The p_c are the mixing proportions. The ϕ_c parameterize simple densities for each cluster. For example, f might be a Gaussian distribution having a diagonal covariance matrix, with ϕ giving means and variances for each dimension of y.

Bayesian Mixture Models

A Bayesian mixture model requires a prior for the mixing proportions, p_c, and component parameters, ϕ_c.

We can use a symmetric Dirichlet prior for the p_c, with density

$$\frac{\Gamma(\alpha)}{\Gamma(\alpha/K)^K} \prod_{c=1}^{K} p_c^{(\alpha/K)-1} \qquad (p_c \geq 0,\ \sum p_c = 1)$$

When α is large, the p_c tend to be nearly equal; when α is close to zero, a few of the p_c are much bigger than the others.

We will make the ϕ_c be independent under the prior, all with the same distribution, G_0.

There may be higher levels to the model (eg, a prior for α), but we'll ignore that possibility here.

The Model Using Cluster Indicators

We can express the mixture model using latent variables, $c^{(i)}$, that identify the cluster that each $y^{(i)}$ belongs to:

$$\begin{aligned}
y^{(i)} \mid c^{(i)}, \phi &\sim F(\phi_{c^{(i)}}) \\
c^{(i)} \mid p_1, \ldots, p_K &\sim \text{Discrete}\,(p_1, \ldots, p_K) \\
\phi_c &\sim G_0 \\
p_1, \ldots, p_K &\sim \text{Dirichlet}\,(\alpha/K, \ldots, \alpha/K)
\end{aligned}$$

The clusters indicators will have values in $\{1, \ldots, K\}$.

The model is not identifiable, since relabelling the classes changes nothing, but this causes no problems — all that really matters is whether $c^{(i)} = c^{(j)}$ or not.

The Prior After Integrating Out Mixing Proportions

The mixing proportions (p_c) can be eliminated by integrating with respect to their Dirichlet prior. The resulting conditional probabilities follow the well-known "law of succession":

$$P(c^{(i)} = c \mid c^{(1)}, \ldots, c^{(i-1)}) \;\; = \;\; \frac{n_{i,c} + \alpha/K}{i - 1 + \alpha}$$

where $n_{i,c}$ is the number of $c^{(j)}$ for $j < i$ that are equal to c.

We could generate from the prior distribution for the $c^{(i)}$ and $y^{(i)}$ as follows:

- Generate $c^{(1)}$, $c^{(2)}$, ..., $c^{(n)}$ using the above probabilities (note that $P(c^{(1)} = c) \; = \; 1/K$).
- Generate ϕ_c for $c = 1, \ldots, K$ from G_0, independently.
- Generate each $y^{(i)}$ from $F(\phi_{c^{(i)}})$, independently.

How Many Clusters?

How many clusters (K) should we include in our model?

If we set K too small, we won't be able to model the density well. And if we're looking for clusters, we'll miss some.

If we use a large K, the model will overfit if we set parameters by maximum likelihood. With some priors, a Bayesian model with large K may underfit.

A large amount of research has been done on choosing K, by Bayesian and non-Bayesian methods.

But does choosing K actually make sense?

Is there a better way?

Letting the Number of Clusters Go to Infinity

There is often reason to think approximating the real distribution arbitrarily well is only possible as K goes to infinity.

The real number of clusters may also be effectively infinite — eg, can one really say that there are only, say, 14 different teaching styles?

So let's try letting K be infinite...

The limiting form of the "law of succession" as $K \to \infty$ is

$$\begin{aligned} P(c^{(i)} = c \mid c^{(1)}, \ldots, c^{(i-1)}) \quad &= \quad \frac{n_{i,c} + \alpha/K}{i-1+\alpha} \\ &\to \quad \frac{n_{i,c}}{i-1+\alpha} \end{aligned}$$

$$P(c^{(i)} \neq c^{(j)} \text{ for all } j < i \mid c^{(1)}, \ldots, c^{(i-1)}) \quad \to \quad \frac{\alpha}{i-1+\alpha}$$

These give the probabilities that data point $y^{(i)}$ belongs to the same cluster as a previous data point, or that it instead belongs to a new cluster, not previously seen.

The Dirichlet Process View

Let $\theta^{(i)} = \phi_{c^{(i)}}$ be the parameters of the distribution from which $y^{(i)}$ was drawn.

When $K \to \infty$, the "law of succession" for the $c^{(i)}$ together with the prior (G_0) for the ϕ_c lead to conditional distributions for the $\theta^{(i)}$ as follows:

$$\theta^{(i)} \mid \theta^{(1)}, \ldots, \theta^{(i-1)} \quad \sim \quad \frac{1}{i-1+\alpha} \sum_{j<i} \delta(\theta^{(j)}) \; + \; \frac{\alpha}{i-1+\alpha}\, G_0$$

This is the "Polya urn" (aka, "Chinese restaurant process") representation of a Dirichlet process, which implicitly defines a distribution over distributions.

We can write this "Dirichlet process mixture model" as

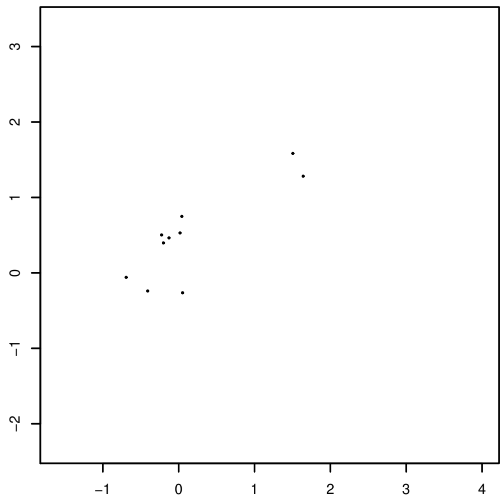
$$\begin{aligned} y^{(i)} \mid \theta^{(i)} &\sim F(\theta^{(i)}) \\ \theta^{(i)} \mid G &\sim G \\ G &\sim D(G_0, \alpha) \end{aligned}$$

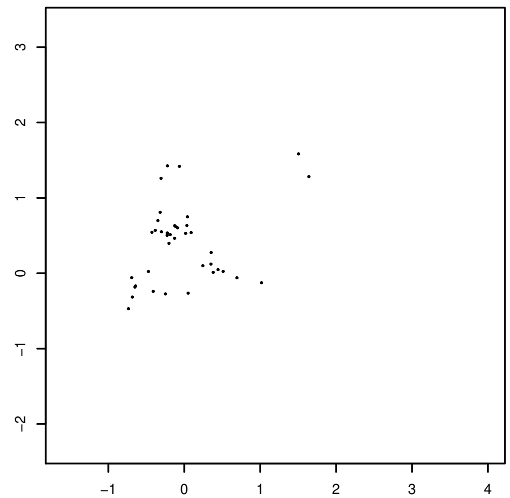
The last line says that G is drawn from a Dirichlet process distribution over distributions. It turns out that G will have its probability concentrated on a countably infinite number of points, corresponding to a countably infinite number of clusters.

Data From a Dirichlet Process Mixture

Data sets of increasing size from a Dirichlet process mixture model with 2D Gaussian distributions for each cluster, with $\alpha = 1$:

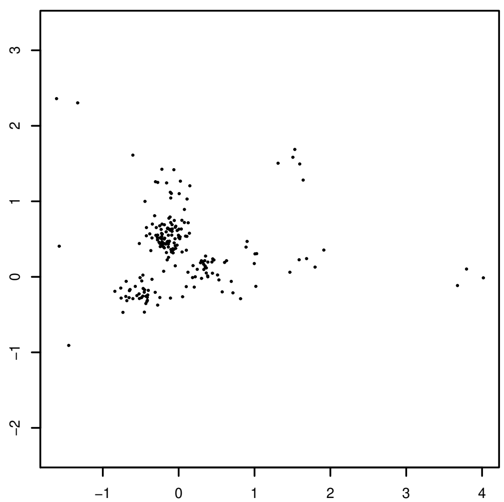
$n = 10$

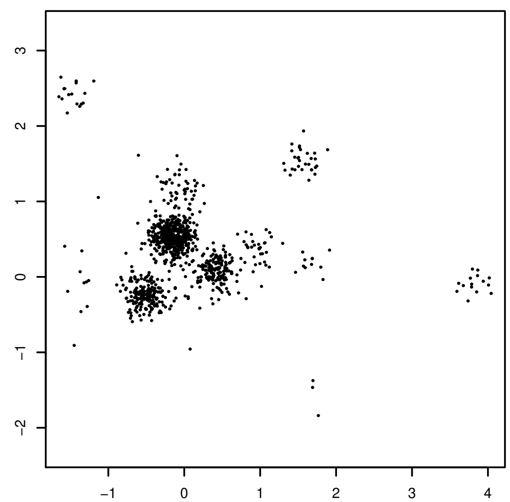




$n = 40$

$n = 200$





$n = 1000$

Further Developments Related to Dirichlet Processes

- Hierarchical generalizations of Dirichlet process mixtures can be used for problems such as document modeling — a document is a mixture of topics (distributions over words), with certain proportions, but topics can be shared between documents. A related approach allows for Hidden Markov Models with an infinite number of states. (Teh, Jordan, Beal, and Blei 2004)
- In another direction, hierarchical clustering can be done by replacing the latent indicators of cluster membership with a latent tree structure over data points. Such "Dirichlet diffusion tree" models are analogous to phylogenetic trees relating species. (Neal 2003a)
- Models based on Dirichlet processes that look at relations between items rather than attributes of single items are also possible. (Kemp, Griffiths, and Tenenbaum 2004)
- Computation for Dirichlet process mixture models typically uses Markov chain Monte Carlo. Some tricks are needed to handle non-conjugate models (Neal 1998a), and to efficiently split and merge clusters (Jain and Neal 2004).

Monte Carlo Methods

A very general approach to Bayesian computation — applicable even to very high-dimensional problems — is to obtain a *sample* of points from the posterior distribution, and use it to make *Monte Carlo* estimates.

A single sample point will contain values for all the unknown parameters, hyperparameters, latent variables, missing data values, etc. — everything not known, except what we don't care about or have integrated away analytically.

We use this sample to approximate expected values by averages over the sample points. For example, from K values, $\theta^{(1)}, \ldots \theta^{(K)}$, for a parameter, sampled from $P(\theta \,|\, \text{data})$, we can approximate the predictive probability that $Y = 1$ by

$$P(Y = 1 \mid \text{data}) \;\approx\; \frac{1}{K} \sum_{j=1}^{K} P(Y = 1 \mid \theta^{(j)})$$

Obtaining a Sample by Simulating a Markov Chain

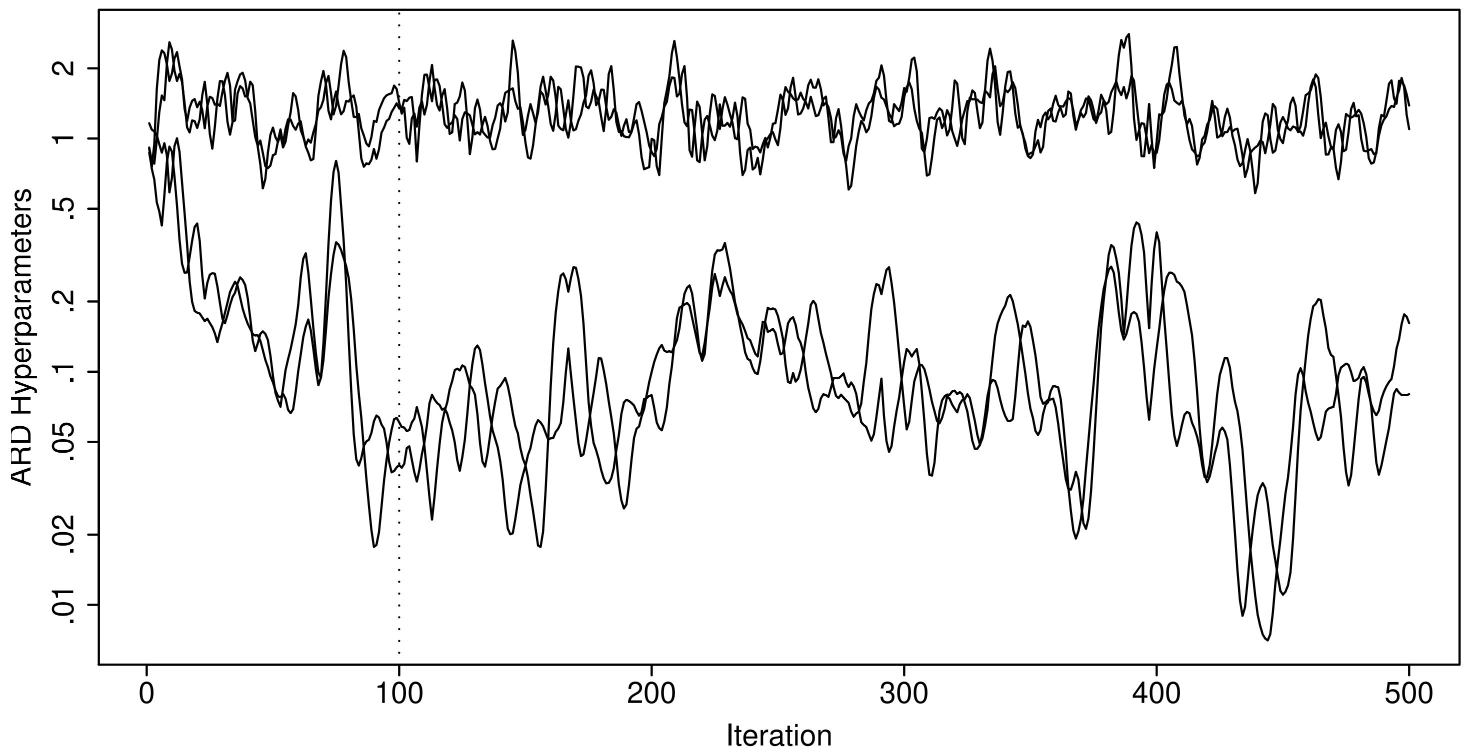
When the posterior distribution is too complex to sample from directly, we can instead simulate a *Markov chain* that will converge (asymptotically) to the posterior distribution.

States from the latter portion of this Markov chain will come (approximately) from the posterior distribution, though they will not be independent. We can use these states to make Monte Carlo estimates.

Finding such a Markov chain sounds hard, but fortunately there are general schemes that make this possible even for difficult problems. This *Markov chain Monte Carlo (MCMC)* approach is therefore very general. MCMC can also be very slow in some circumstances, but despite this, it is often the only viable approach to Bayesian inference using complex models.

An Example of Markov Chain Sampling

Here is a Markov chain run for a Gaussian process classification model, showing the four hyperparameters controlling the relevance of the four input variables:



The chain starts out in a state that isn't typical of the posterior distribution, but by about iteration 100, the distribution seems to have stabilized. We would use iterations from there on to make predictions.

Note the high autocorrelation for the two hyperparameters with low values. Fortunately, the exact degree of irrelevance of largely irrelevant inputs isn't crucial.

Fundamental Requirements for Markov Chain Monte Carlo

Suppose we want to sample from a distribution $\pi(x)$ by simulating a Markov chain — eg, π could be the posterior, x the parameter vector. For notational simplicity, we'll assume here that x is discrete, but everything generalizes.

We need to find transition probabilities, $T(x, x')$, for the chain to move from state x to state x' that will lead to convergence to π.

A fundamental requirement for this is that the transitions *leave* π *invariant*:

$$\pi(x') \;\; = \;\; \sum_x \pi(x)\, T(x, x'), \quad \text{for all } x'$$

This says that if we ever reached the distribution π at some time in the simulation, the distribution for the next state would also be π.

We also need the chain to be *ergodic* — roughly speaking, it shouldn't get trapped in one part of the state space.

These two conditions are enough to (more-or-less) guarantee that Monte Carlo estimates based on states from the chain converge to the correct expectations with respect to π.

Proving Invariance From Detailed Balance

Our first challenge is to find transition probabilities that leave π invariant.

One way is to show that the transitions satisfy a stronger condition known as *detailed balance*:

$$\pi(x)\,T(x,x') \;=\; \pi(x')\,T(x',x), \quad \text{for all } x \text{ and } x'$$

If this is true, the chain is also said to be *reversible* with respect to π.

It's easy to prove that detailed balance implies invariance:

$$\begin{aligned}\sum_x \pi(x)\,T(x,x') \;&=\; \sum_x \pi(x')\,T(x',x) \\ &=\; \pi(x')\sum_x T(x',x) \;=\; \pi(x')\end{aligned}$$

The converse is not true: *nonreversible* chains that leave π invariant are possible.

Combining Transitions

If the transitions $T_0(x, x')$ and $T_1(x, x')$ both leave π invariant, then so does any mixture of the two, define for some $\alpha \in [0, 1]$ by

$$T_\alpha(x, x') \quad = \quad (1 - \alpha)T_0(x, x') + \alpha T_1(x, x')$$

The transition defined by first applying T_0 and then applying T_1 will also leave π invariant. If we view transition probabilities as a matrix, this combined transition matrix is just $T_0 T_1$.

Two applications:

– We can combine several transitions each of which changes only part of the state (leaving the rest unchanged). As long as each part is changed by at least one of these transitions, the combined transition may be ergodic.

– We can combine several types of transitions in the hopes that at least one of them will work well. It may be that we need one type in one part of the state space, another type in another part.

The Metropolis Algorithm

The original MCMC method, applied to a statistical physics problem by Metropolis, *et al.* in 1953, is still widely used, because it is very widely applicable.

The *Metropolis algorithm* does a transition from state x to state x' as follows:

1) A "candidate", x^*, is proposed according to some probabilities $S(x, x^*)$, satisfying the symmetry condition, $S(x, x^*) = S(x^*, x)$.

2) This candidate, x^*, is accepted as the next state with probability

$$\min\left[1,\ \pi(x^*)/\pi(x)\right]$$

If x^* is accepted, then $x' = x^*$. If x^* is instead rejected, then $x' = x$.

One can easily show that transitions defined in this way satisfy detailed balance, and hence leave π invariant.

Crucially, it's enough to be able to evaluate some function proportional to $\pi(x)$. We don't need the normalizing constant; it cancels when computing $\pi(x^*)/\pi(x)$.

The choice of proposal distribution, S, is important, since it determines whether the chain is ergodic, and if so, whether it converges rapidly.

Gibbs Sampling

The *Gibbs sampling* algorithm, popular in statistics, updates each component of the state by sampling from its conditional distribution given other components.

Let the current state be $x = (x_1, \ldots, x_n)$. The next state, $x' = (x'_1, \ldots, x'_n)$, is produced as follows:

- Sample x'_1 from $x_1 \mid x_2, \ldots, x_n$.
- Sample x'_2 from $x_2 \mid x'_1,\ x_3, \ldots, x_n$.

  $\cdots$
- Sample x'_i from $x_i \mid x'_1, \ldots, x'_{i-1},\ x_{i+1}, \ldots, x_n$.

  $\cdots$
- Sample x'_n from $x_n \mid x'_1, \ldots, x'_{n-1}$.

In many interesting problems we can easily sample from these conditional distributions, even though the joint distribution cannot easily be sampled from.

In many other problems, we can update some of the components this way, and use other updates (eg, Metropolis) for the others.

Other Markov Chain Monte Carlo Methods

Slice sampling (Neal 2003b) is a general MCMC approach that is more "robust" than the Metropolis algorithm (behaves reasonably well even when badly tuned).

Methods based on *Hamiltonian dynamics* (see Neal 1993; Liu 2001) can avoid the "random walk" behaviour that simple Metropolis and Gibbs sampling methods exhibit. This can produce large speedups.

Annealing methods (eg, Neal 2001) can sometimes handle distributions with multiple modes, in which the Markov chain can be trapped in one mode for a long period of time. They can also estimate the normalizing constant for the distribution — which for a posterior distribution will be the marginal likelihood needed for model comparison.

But... MCMC is still largely an art. A substantial effort is often needed to get good results. For difficult problems, MCMC will in practice merely produce a "best guess" at the real posterior. This is all we can expect, since guaranteeing correct answers would often require solving NP-complete problems.

Bayesian Inference Using Variational Approximation

Since MCMC can be tedious, we sometimes seek faster approximate solutions. Recently, there has been much work on *variational approximations*, which are related to "mean field" and other approximation methods from physics.

In one family of variational methods, we seek a distribution, $Q(\theta)$, that best approximates the posterior, $\pi(\theta)$, in terms of KL divergence:

$$D(Q \,||\, \pi) \;=\; \int Q(\theta) \log \frac{Q(\theta)}{\pi(\theta)} \, d\theta$$

For models with latent variables, Q may be a joint distribution over both the parameters and the latent variables in all cases.

The best choice is obviously $Q = \pi$, but we assume that's not feasible. Instead, we define a family Q_ϕ of distributions that we can handle (often because of independence properties), and optimize $D(Q_\phi \,||\, \pi)$ with respect to ϕ. See Ghahramani and Beal (2000) for details of a "variational EM" algorithm that operates in this way.

Limitations of Bayesian Methods

The challenges of specifying an appropriate model and prior and of performing the required Bayesian computations are not always easy to meet. Here are some currently difficult situations for Bayesian methods:

Problems requiring specific priors in vague situations.

> An example: We have a sample of points that we know come from a convex polyhedron, whose volume we wish to estimate. A Bayesian method will need a prior over possible polyhedra — which could be done, but probably requires a lot of thought. But a simple non-Bayesian estimate based on cross validation is (usually) available.

Problems where the likelihood has an intractable normalizing constant.

> Boltzmann machines are an example — even maximum likelihood is hard, and Bayesian inference seems out of the question at the moment.

Problems with complex, unknown error distributions.

> We can try to model the error, but it may be difficult. A bad model may lead to "overfitting" data where the model thinks the error is less than it is. A cross-validation approach to regularization can sometimes work better in such situations.

Misguided “Bayesian” Methods

Some attempts at Bayesian inference are just mis-guided — either falling prey to various problems, or failing from the start because they don’t take the Bayesian framework seriously. Here are some commonly observed errors:

Improper posterior distributions: Priors that are “improper” – eg, uniform over $(-\infty, +\infty)$ – can sometimes be convenient, but not if the posterior ends up improper.

Ridiculous priors: Surprisingly many people use a gamma(0.001,0.001) prior for inverse variances, even though it’s absurd.

Relying on MAP estimation: Using the mode of the posterior (MAP estimate) can be a crude but useful approximation. But if a method “works” only because of this approximation, it’s not Bayesian.

Meaningless marginal likelihoods: Often based on priors that aren’t well considered, or computed using hopelessly inaccurate methods.

Fear of unidentifiability: Irrationally worried that some transformations of the model (eg, permutations of hidden units) leave the probability of the data unchanged, some people impose constraints that destroy interpretability, introduce arbitrary asymmetries in the prior, and hinder MCMC convergence.

A final category: **counterproductive creativity**. Physicists would frown on modifying Schrödinger’s equation just because it seems to help on one particular problem. Modifying Bayes’ Rule is a similar mistake.

Successes of Bayesian Methodology

Bayesian **neural network** models and **Gaussian process** models have been applied to many practical problems, with excellent results. See Lampinen and Vehtari (2001) for examples.

Bayesian neural networks produced winning results in the NIPS*2003 feature selection challenge (`http://www.nipsfsc.ecs.soton.ac.uk`), with some help from Dirichlet diffusion tree models — see the entries by myself and J. Zhang.

Dirichlet process mixture models are widely used in the statistical literature, and they and their hierarchical extensions are becoming popular as models of documents for information retrieval.

Bayesian methods have increased in popularity in statistics since MCMC methods were popularized around 1990. Complex **hierarchical models**, with many layers of parameters are often used, and are the only viable approach to some problems.

As the fields of machine learning and statistics merge together, I believe that Bayesian methods derived from both fields will be applied successfully to ever more challenging problems.

General References

J. M. Bernardo, A. F. M. Smith (1994) *Bayesian Theory*, Wiley.

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Rubin (2003) *Bayesian Data Analysis*, 2nd edition, Chapman&Hall/CRC.

J. S. Liu (2001) *Monte Carlo Strategies in Scientific Computing*, Springer-Verlag.

R. M. Neal (1993) *Probabilistic Inference Using Markov Chain Monte Carlo Methods.*
`http://www.cs.utoronto.ca/~radford/review.abstract.html`

R. M. Neal (1996) *Bayesian Learning for Neural Networks*, Springer-Verlag.

D. J. C. MacKay (2003) *Information Theory, Inference, and Learning Algorithms.*
`http://wol.ra.phy.cam.ac.uk/mackay/itila/book.html`

Links to Online Resources

Tom Griffiths' reading list: `http://www-psych.stanford.edu/~gruffydd/bayes.html`

MCMC Preprint Service: `http://www.statslab.cam.ac.uk/~mcmc`

My software for flexible Bayesian modeling:
`http://www.cs.toronto.edu/~radford/fbm.software.html`

The new on-line journal *Bayesian Analysis*: `http://ba.stat.cmu.edu`

Papers on Particular Topics

P. Rujoriginally (1997) Playing billiards in version space, *Neural Computation*, **9**, 99-122.

A. Gelman, X.-L. Meng (1998) Simulating normalizing constants: From importance sampling to bridge sampling to path sampling, *Statistical Science*, **13**, 163-185.

Z. Ghahramani, M. J. Beal (2000) Graphical models and variational methods. `http://www.cse.buffalo.edu/faculty/mbeal/papers/advmf.pdf`

R. Herbrich, T. Graepel, C. Campbell (2001) Bayes point machines, *JMLR*. `http://jmlr.csail.mit.edu/papers/v1/herbrich01a.html`

S. Jain, R. M. Neal (2004) A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model, *Journal of Computational and Graphical Statistics*, **13**, 158-182.

C. Kemp, T. L. Griffiths, J. B. Tenenbaum (2004) Discovering latent classes in relational data. `http://www-psych.stanford.edu/~gruffydd/papers/blockTR.pdf`

J. Lampinen, A. Vehtari (2001) Bayesian approach for neural networks — review and case studies, *Neural Networks*, **14**, 7-24.

D. J. C. MacKay (1997), Introduction to Gaussian Processes. `http://www.inference.phy.cam.ac.uk/mackay/gpB.pdf`

R. M. Neal (1998a) Markov chain sampling methods for Dirichlet process mixture models. `http://www.cs.toronto.edu/~radford/mixmc.abstract.html`

Papers on Particular Topics (Continued)

R. M. Neal (1998b) Regression and classification using Gaussian process priors, in J. M. Bernardo, et al. (editors) *Bayesian Statistics 6*.

R. M. Neal (2001) Annealed importance sampling, *Statistics and Computing*, **11**, 125-139.

R. M. Neal (2003a) Density modeling and clustering using Dirichlet diffusion trees, in J. M. Bernardo, et al. (editors) *Bayesian Statistics 7*.

R. M. Neal (2003b) Slice sampling, *Annals of Statistics*, **31**, 705-767.

C. E. Rasmussen (1996) *Evaluation of Gaussian Processes and other Methods for Non-Linear Regression*. `http://www.kyb.mpg.de/publications/pss/ps2304.ps`

M. Seeger, C. K. I. Williams, N. Lawrence (2003) Fast forward selection to speed up sparse Gaussian process regression, *AI-STATS 2003*. `http://www.dai.ed.ac.uk/homes/ckiw/postscript/aistats03-final.ps.gz`

Y. W. Teh, M. I. Jordan, M. J. Beal, D. M. Blei (2004) Hierarchical Dirichlet processes. `http://www.cs.toronto.edu/~ywteh/research/npbayes/report.pdf`