Models for Dice Factories and Amino Acid Probability Vectors. *Draft 1.1*

David J.C. MacKay
Cavendish Laboratory
mackay@mrao.cam.ac.uk

October 5, 1994

Abstract

Protein alignments are commonly characterized by the probability vectors over amino acids in each column of the alignment. This paper develops various models for the probability distribution of these probability vectors. First a simple Dirichlet distribution is used, then a mixture of Dirichlets. Finally a componential model employing a 'density network' is described. These models are optimized and compared using Bayesian methods.

1 Introduction

A protein is a sequence of amino acids. There are 20 different amino acids. A protein family is a set of proteins believed to have similar structure, although their amino acid sequences may be somewhat different. The sequences of the proteins in one protein family can be *aligned*, one protein per row, such that each column corresponds to a particular amino acid location in the common structure. When we look down a column, we find the 20 amino acids occurring with frequencies that differ from column to column. A protein family is commonly characterized by specifying the frequency distribution for each column. Each column is like a die from a dice factory, having a bias to some faces rather than others. This paper describes the modelling of the distribution of probabilities over amino acids.

The development of a model for the probability distribution over amino acid probability vectors is important for two reasons. The first is that such a model defines a prior probability distribution over probability vectors to be encountered in other protein families; with a good prior $P(\{\mathbf{q}_{|j}\})$, we can obtain an accurate model for a new protein family with less data, *i.e.*, fewer examples of proteins from the family. The second reason is that this modelling activity may in itself be interesting, giving new insight into amino acid properties. In this paper three different models for amino acid probabilities are described and adapted to data using Bayesian methods.

2 Dirichlet distributions

The simplest model $P(\{\mathbf{q}_{|j}\})$ that is often used (implicitly or explicitly) in protein family modelling with 'hidden Markov models' is a Dirichlet distribution. Dirichlet distributions are particularly convenient prior distributions to work with where count data is involved. The optimization of Dirichlet distributions is discussed by **?**), and is reviewed here.

The Dirichlet distribution for a probability vector $\mathbf{p}$ with I components is parameterized by a measure (a positive vector) which I will write here as $\mathbf{u} = \alpha\mathbf{m}$, where $\mathbf{m}$ is a normalized measure over the I components ($\sum m_i = 1$), and α is positive:

$$P(\mathbf{p}|\alpha\mathbf{m}) = \frac{1}{Z(\alpha\mathbf{m})} \prod_{i=1}^{I} p^{\alpha m_i - 1} \delta\left(\textstyle\sum_i p_i - 1\right) \equiv \mathrm{Dirichlet}(\mathbf{p}|\alpha\mathbf{m}) \tag{1}$$

The function $\delta(x)$ is the Dirac delta function which here restricts the distribution to the simplex such that $\sum_i p_i = 1$. The normalizing constant is $Z(\alpha\mathbf{m}) = \prod_i \Gamma(\alpha m_i) / \Gamma(\alpha)$. The vector $\mathbf{m}$ is the mean of the probability distribution:

$$\int d^I\mathbf{p}\ \text{Dirichlet}(\mathbf{p}|\alpha\mathbf{m})\mathbf{p} = \mathbf{m} \tag{2}$$

The parameter α measures the sharpness of the distribution, *i.e.*, it measures how different we expect typical samples $\mathbf{p}$ from the distribution to be from the mean $\mathbf{m}$. A large value of α produces a distribution over $\mathbf{p}$ which is sharply peaked around $\mathbf{m}$. As α is decreased, the distribution becomes broader; as α approaches zero, the distribution spreads into the corners of the simplex. For α close to zero, a typical sample $\mathbf{p}$ from Dirichlet$(\mathbf{p}|\alpha\mathbf{m})$ has nearly all its mass on just one component p_i.

If we observe samples from $\mathbf{p}$ and obtain counts $\mathbf{F} = (F_1, F_2, \ldots F_I)$ of the possible outcomes i, then the posterior for the probability vector is another Dirichlet distribution:

$$\begin{aligned} P(\mathbf{p}|\mathbf{F}, \mathbf{m}, \alpha, \mathcal{H}) &\propto \prod_i p_i^{F_i + \alpha m_i - 1} \delta(\textstyle\sum_i p_i - 1) && (3) \\ &= \text{Dirichlet}(\mathbf{p}|\mathbf{F} + \alpha\mathbf{m}). && (4) \end{aligned}$$

The predictive distribution given the data $\mathbf{F}$ is then:

$$P(i|\mathbf{F}, \alpha\mathbf{m}) = \int d^I\mathbf{p}\ \text{Dirichlet}(\mathbf{p}|\mathbf{F} + \alpha\mathbf{m})\ \mathbf{p} = \frac{F_i + \alpha m_i}{\sum_{i'} F_{i'} + \alpha m_{i'}} \tag{5}$$

The value of α defines the number of samples from $\mathbf{p}$ that are required in order that the data dominate over the prior in subsequent predictions. Additional useful formulae and approximations are found in appendix A.

Many methods in genome modelling and other fields make use of predictions having the form of equation (5), in which the empirical frequency $F_i / \sum_{i'} F_{i'}$ is 'smoothed' by an offset αm_i. Rarely, however, are objective methods used to set these smoothing parameters $\{\alpha m_i\}$. **?**) showed how to optimize these parameters, given many sets of counts $\{\mathbf{F}_{|j}\}_1^J$, by viewing $\alpha\mathbf{m}$ as the hyperparameters of a Dirichlet distribution.

The dice factory

It may be helpful to consider this problem in the toy context of a 'dice factory'. Imagine that a factory produces biased I–sided dice. We receive J dice labelled by $j = 1 \ldots J$. Each die j is assumed to have a probability vector $\mathbf{q}_{|j} = \{q_{i|j}\}_{i=1}^I$ that defines the probability of each of the I possible outcomes. Each die j is rolled a number of times F_j, and we are told the counts of the outcomes, $F_{i|j}$, which give us imperfect information about the parameters $\mathbf{q}_{|j}$. Our task is to develop a model for the probability distribution $P(\mathbf{q})$ of the vectors $\mathbf{q}$, given the data $\{\mathbf{F}|j\}$.

3 Modelling the dice factory with a Dirichlet distribution

We might model the probability vector $\mathbf{q}$ of a single die as coming from a Dirichlet prior, with unknown hyperparameters $\mathbf{m}$ (a normalized vector, representing the average probability vector of a die from the factory) and α (which determines the peakedness of the probability distribution of probability vectors about this average):

$$P(\mathbf{q}|\alpha\mathbf{m}) = 1/Z \prod_{i=1}^{I} q_i^{\alpha m_i - 1} \delta\left(\textstyle\sum_i q_i - 1\right) \tag{6}$$

Under this model, the task of modelling the factory's properties boils down to the inference of the parameters $\alpha\mathbf{m}$ from the count data. We may distinguish two levels of inference. At the first level we assume we know the hyperparameters, and we infer the likely probability vector $\mathbf{q}_{|j}$ of die j given the data. At the second level, we infer the hyperparameters themselves, given the data.

4 The dice factory — count data

Level 1

Consider a particular die j. For a given value of $\alpha\mathbf{m}$, we can infer the posterior (**??**) and obtain the predictive distribution:

$$\langle q_{i|j} \rangle = \frac{F_{i|j} + \alpha m_i}{F_j + \alpha} \tag{7}$$

Level 2

The posterior for $\mathbf{m}, \alpha$ is proportional to the following 'evidence' term:

$$P(D|\alpha\mathbf{m}) = \prod_j \left(\frac{\prod_i \Gamma(F_{i|j} + \alpha m_i)}{\Gamma(F_j + \alpha)} \frac{\Gamma(\alpha)}{\prod_i \Gamma(\alpha m_i)} \right) \tag{8}$$

To find the most probable $\alpha\mathbf{m}$, we differentiate the gamma functions.

$$\frac{\partial}{\partial u_i} \log P(D|\alpha\mathbf{m}) = \sum_j \left[\Psi(F_{i|j} + u_i) - \Psi(F_j + \textstyle\sum_{i'} u_{i'}) + \Psi(\textstyle\sum_{i'} u_{i'}) - \Psi(u_i) \right] \tag{9}$$

Formulae for more general algorithms

Algorithms given in (**?**) were based on series expansions of $\Psi(u)$ about $u = \infty$ and $u = 0$ respectively, and were specialised for particular regimes of α and u_i. The following formula, although it is not part of a series expansion, gives an approximation to the difference $\Psi(F + u) - \Psi(u)$ that is accurate to within 2% for all u and all positive integers F:

$$\Psi(F + u) - \Psi(u) \simeq \frac{1}{u} + \log \frac{F + u - 1/2}{u + 1/2} \tag{10}$$

This approximation has been used in an algorithm that can efficiently cope with all values of α and u_i. It is used to evaluate the derivative (9), which is then passed to a conjugate gradients optimizer in order to find the hyperparameters $\alpha\mathbf{m}$ that maximize the evidence.

Mixture models

If we believe that in proteins there are different types of column, such that for each type of column the probabilities $\mathbf{q}$ are similar, then a mixture model may be appropriate. A mixture model $\mathcal{H}_{M(C)}$ defines a density over $\mathbf{q}$ as a weighted combination of C independently parameterized simple distributions, where each mixture component $c = 1 \ldots C$ could be a Dirichlet or entropic distribution. Various algorithms can be used to implement mixture models: both Monte Carlo methods (**?**) and Gaussian approximations (Hanson, Stutz and Cheeseman 1991). The basic ideas of mixture modelling are reviewed in appendix **??** which I will probably cut from future drafts.

In this section I report the results of applying a mixture-of-Dirichlets model to data from protein families. These models are relevant to language modelling, speech modelling, text compression, and the modelling of DNA and protein sequences.

NB, a mixture model in this context is not asserting that there are types of amino acid; it asserts that there are types of distribution over amino acid, and each probability distribution belongs to one type.

5 Mixture models

A mixture model $\mathcal{H}_{M(C)}$ defines a density over $\mathbf{q}$ as a weighted combination of C independently parameterised simple distributions, where each mixture component $c = 1 \ldots C$ could be a Dirichlet or entropic distribution. Various algorithms can be used to implement mixture models: both Monte Carlo methods (**?**) and Gaussian approximations (Hanson et al. 1991). Many algorithms depend on the evaluation of the probability of the data given various assumptions, and the gradient of the log probability with respect to the model parameters.

Mixture modelling - details

A mixture model $\mathcal{H}_{M(C)}$ defines a density over a variable $\mathbf{q}$ as a weighted combination of C independently parameterised simple distributions. Labelling the mixture components with $c = 1 \ldots C$, we give each component a weight p_c^M such that $\sum_{c=1}^{C} p_c^M = 1$, and a parameter vector $\mathbf{u}^c$ that parameterizes a density $P(\mathbf{q}|\mathbf{u}^c)$. We denote the entire collection of parameters $\{\mathbf{u}^c, p_c^M\}_{c=1}^{C}$ by $\mathbf{U}$. We consider a collection of vectors $\mathbf{Q} = \{\mathbf{q}^{(j)}\}_{j=1}^{J}$, and use the notation $j \in c$ to denote the proposition that the vector labelled n comes from class c. Noting that $P(j \in c|\mathbf{U}, \mathcal{H}_{M(C)}) = p_c^M$, we write down:

$$P(\mathbf{Q}|\mathbf{U}, \mathcal{H}_{M(C)}) = \prod_{j=1}^{J} \left[\sum_{c=1}^{C} p_c^M P(\mathbf{q}^{(j)}|n \in c, \mathbf{u}^c) \right]. \tag{11}$$

There are various computations we will want to perform.

Classification

If the vectors $\mathbf{q}$ are observable, and a particular value of the parameters $\mathbf{U}$ of the mixture model is given, then we can infer the probability that an individual vector $\mathbf{q}^{(j)}$ belongs to cluster c, which we will abbreviate to $p_{c|j}$.

$$p_{c|j} \equiv P(j \in c|\mathbf{q}^{(j)}, \mathbf{U}, \mathcal{H}_{M(C)}) = \frac{p_c^M P(\mathbf{q}^{(j)}|j \in c, \mathbf{u}^c)}{\sum_{c'=1}^{C} p_{c'}^M P(\mathbf{q}^{(j)}|j \in c', \mathbf{u}^{c'})} \tag{12}$$

In the case of language modelling, the vectors $\mathbf{q}^{(j)}$ are not directly observable, but are measured via data $\mathbf{F}^{(j)}$, and we integrate over $\mathbf{q}^{(j)}$ to obtain for each class c the evidence:

$$P(\mathbf{F}^{(j)}|n \in c, \mathbf{u}^c) = \int d\mathbf{q}^{(j)} \; P(\mathbf{F}^{(j)}|\mathbf{q}^{(j)}) P(\mathbf{q}^{(j)}|j \in c, \mathbf{u}^c). \tag{13}$$

In this case, we replace the hidden $\mathbf{q}^{(j)}$ in (12) by the observable $\mathbf{F}$ and define $p_{c|j}$ by:

$$p_{c|j} \equiv P(j \in c|\mathbf{F}^{(j)}, \mathbf{U}, \mathcal{H}_{M(C)}) = \frac{p_c^M P(\mathbf{F}^{(j)}|j \in c, \mathbf{u}^c)}{\sum_{c'=1}^{C} p_{c'}^M P(\mathbf{F}^{(j)}|j \in c', \mathbf{u}^{c'})} \tag{14}$$

Inference of U

Given data $\mathbf{F} = \{\mathbf{F}^{(j)}\}$, we wish to infer the hyperparameters of the mixture model. I proceed by evaluating the gradient of the evidence with respect to the hyperparameters so as to optimize them, to give a mode and error bars as a summary of the posterior distribution of the hyperparameters. The first and second derivatives of the log of the probability (11) can be found as follows.

$$\begin{aligned} \frac{\partial}{\partial \mathbf{u}^c} \log P(\mathbf{F}|\mathbf{U}, \mathcal{H}_{M(C)}) &= \sum_{j=1}^{J} \left[\frac{p_c^M \frac{\partial}{\partial \mathbf{u}^c} P(\mathbf{q}^{(j)}|j \in c, \mathbf{u}^c)}{\sum_{c'=1}^{C} p_{c'}^M P(\mathbf{F}^{(j)}|j \in c', \mathbf{u}^{c'})} \right] \\ &= \sum_{j=1}^{J} \left[p_{c|j} \frac{\partial}{\partial \mathbf{u}^c} \log P(\mathbf{q}^{(j)}|j \in c, \mathbf{u}^c) \right]. \end{aligned} \tag{15}$$

Thus the gradient of the total log probability with respect to the parameters $\mathbf{u}^c$ is a sum over all examples $\mathbf{q}^{(j)}$ of log probability gradients, each of them weighted by the probability $p_{c|j}$ that example j does indeed belong to class c.

The derivative with respect to $\mathbf{p}^M$ is also simple. It is computationally most convenient to represent these probabilities by logarithms, defining $\mathbf{p}^M$ in terms of $\mathbf{l}$ thus:

$$p_c^M = \frac{\exp l_c}{\sum_c' \exp l_{c'}}. \tag{16}$$

This representation has one excess degree of freedom (addition of a constant to all l_c), but this need not cause any difficulty. The gradient of the total log probability is:

$$\frac{\partial}{\partial l_c} \log P(\mathbf{F}|\mathbf{U}, \mathcal{H}_{M(C)}) = \sum_{j=1}^{J} p_{c|j} - p_c^M J. \tag{17}$$

The two terms on the right hand side are: $\left[\sum_{j=1}^J p_{c|j}\right]$, the inferred effective number of examples j that belong to class c; and $p_c^M J$, the expected number of examples in class c if the probability were indeed p_c^M.

Using the gradient vector

The gradients above could be used by generic gradient-based optimisation algorithms. Alternatively, computational methods known as re-estimation methods[1] use this gradient information in the following way: the classifications $p_{c|j}$ are computed using the current parameters $\mathbf{U}$; then keeping those assignments fixed, the parameters $\mathbf{u}^c$ are adjusted (in a single step, or several steps) so as to improve the fit of component c to its assigned points. For one sub-class of re-estimation algorithms known as E-M algorithms (Expectation-Maximization) there is a convergence proof for this procedure. See also the paper by Neal and Hinton for generalizations of these methods.

For computational purposes one can evaluate derivatives like (16) in time less than $O(JC)$ by omitting the terms where $p_{c|n}$ is below some numerical threshold. The probable classes c for an example $\mathbf{q}^{(n)}$ can be efficiently identified by the use of data structures such as k-d trees (Lewicki 1994). A stochastic re-estimation method used by Lewicki (1994) randomly assigns each example n to just one of the classes c, with probability given by $p_{c|n}$; the parameters of class c, $\mathbf{u}^c$ are then optimized as if its randomly assigned members belong to class c with certainty.

Curvature

Differentiating a second time, we obtain the curvature:

$$\begin{aligned}\frac{\partial^2}{\partial \mathbf{u}^c \partial \mathbf{u}^{c'}} \log P(\mathbf{F}|\mathbf{U}, \mathcal{H}_{M(C)}) \quad = \quad & \sum_{j=1}^{J} \Bigg[\delta_{cc'} p_{c|j} \Bigg(\frac{\partial^2}{\partial \mathbf{u}^c \partial \mathbf{u}^{c'}} \log P(\mathbf{q}^{(j)}|j \in c, \mathbf{u}^c) \\ & + \frac{\partial}{\partial \mathbf{u}^c} \log P(\mathbf{q}^{(j)}|j \in c, \mathbf{u}^c) \frac{\partial}{\partial \mathbf{u}^{c'}} \log P(\mathbf{q}^{(j)}|j \in c, \mathbf{u}^c) \Bigg) \\ & - p_{c|j} p_{c'|j} \frac{\partial}{\partial \mathbf{u}^c} \log P(\mathbf{q}^{(j)}|j \in c, \mathbf{u}^c) \frac{\partial}{\partial \mathbf{u}^{c'}} \log P(\mathbf{q}^{(j)}|j \in c', \mathbf{u}^{c'}) \Bigg].\end{aligned} \tag{18}$$

In problems where most of the examples j are firmly classified (*i.e.*, for most j there is one c such $p_{c|j} \simeq 1$), the first of these three terms will dominate, and the other two may be negligible for computational purposes.

The curvature is a measure of how well determined the hyperparameters $\mathbf{U}$ are by the examples.

If there are symmetries in the model, for example, the components of the mixture have the same computational form, then the probability (11) will have multiple maxima. If one is fortunate, then these maxima may all be symmetrically related to each other, and the predictions obtained by integrating over a single maximum are equivalent to those obtained by integrating over the entire space. But in general, a mixture model may have more than one set of maxima. If this is the case, it is desirable to locate a representative of each distinct set (by incorporating a random element into the search

[1] Ask Radford if this is acceptable terminology. Or would 'carpet jumping algorithms' be better?

algorithm, say). Having found distinct optima, we evaluate the posterior probability associated with each one by integrating over the local parameter space $\mathbf{U}$, and multiplying by the appropriate symmetry factor. When performing this integral approximately, the curvature evaluated above is likely to be useful. Subsequent inferences are obtained by integrating together the predictions of the optima, each weighted by the probability associated with it. The same integrated probability is also used to compare the optima of the fitted mixture model with alternative models, either mixture models with a different number of components C, or models of other forms.

6 A componential density model

Do we believe in mixture models? We might believe 'small/large' and 'hydrophobic/hydrophilic' to be two relevant attributes of an amino acid. A mixture model would have to use four categories to capture all four combinations of these binary attributes, whereas only two independent degrees of freedom are really present. This prior expectation might motivate a *combinatorial* representation of underlying variables.

Geoff Hinton (personal communication) pointed out that a mixture model does not capture the **componential** nature of a context. A particular column of our alignment might consist of 'small hydrophobic' residues, another might consist of 'small hydrophilic', and so forth. The categories 'small' and 'hydrophobic' should not therefore be represented as exclusive clusters; but this is what a mixture model does.

We are therefore interested in componential models, that is, models that assign a density over $\mathbf{q}$ that is indexed by hyperparameters $\mathbf{x}$ that are components in a space of context types.

We are therefore interested in componential models, that is, models that assign over $\mathbf{q}$ a density indexed by a vector of latent variables $\mathbf{x}$ that are components in a space of context types. If the reader is familiar with factor analysis, then a rough analogy may be helpful: componential modelling is like a non-linear factor analysis appropriate to the probability simplex. Factor analysis should be contrasted with principal components analysis, which blah blah (GH pc). The construction in terms of eigenvectors and eigenvalues is not needed, nor is the artificial notion of orthogonality. PCA is obsessed with high variance but what is of interest is opportunities to capture regularities, i.e. correlations between observables. I will develop a full probabilistic model which can be objectively compared with other models (such as the Dirichlet model or mixture model) by evaluating the evidence. The evidence will also be used to infer automatically the most probable dimensionality of the componential model. The density in the particular models studied here will correspond to a Gaussian in $\{\log q_i\}$ space, but it will be straightforward to generalize the model to more complicated low-dimensional densities over $\mathbf{q}$.

The density network (**?**) models probabilities $\mathbf{q}$ as coming from a non-linear manifold parameterized by latent variables. This model can also be used to capture correlations between probability vectors, but here it is used in its simplest form with only one probability vector. Performs a well-defined factor analysis of the probability vectors.

Denoting the hidden vector by $\mathbf{x}$, and the observable output vector by $\mathbf{q}$, the task is as follows:

> Assume that $\mathbf{q}$ depends on $\mathbf{x}$, and that all we can observe are examples of $\mathbf{q}$. Deduce the nature of the $\mathbf{x}$ and the dependence of $\mathbf{q}$ on $\mathbf{x}$, so as to create a model for the density $P(\mathbf{q})$.

To be precise, in this particular paper I will furthermore assume that $\mathbf{q}$ itself is not observable, but is measured indirectly through measurements $\{\mathbf{F}\}$. If $\mathbf{q}$ lies in a high dimensional space, it is possible automatically to infer whether the observed distribution of $\mathbf{q}$ might be explained by a lower dimensional distribution. We will be able to do this without explicitly limiting the dimensionality of $\mathbf{x}$. As usual, this 'complexity control' will be an automatic feature of Bayesian inference.

The model

We define the 'hidden components' of $\mathbf{q}$ to be a vector $\mathbf{x}$ indexed by $h = 1 \ldots H$. The dimensionality of this hidden space is H but the effective dimensionality may be smaller, as some of the components x_h may be effectively unused by the model. I introduce a matrix $\mathbf{w}$ that defines the relationship between the hidden components and the observable probability. For each context j, there is a single unknown vector $\mathbf{x}^j$ which defines the probability $\mathbf{q}_{|j}$ thus:

$$q_{i|j}(\mathbf{x}^j; \mathbf{w}) = \frac{1}{Z(\mathbf{x}^j; \mathbf{w})} \exp\left(\sum_h w_{ih} x_h^j + w_{i0}\right), \tag{19}$$

where

$$Z(\mathbf{x}^j; \mathbf{w}) = \sum_i \exp\left(\sum_h w_{ih} x_h^j + w_{i0}\right). \tag{20}$$

In order to define a probability distribution on $\mathbf{q}$, we must assign a value to $\mathbf{w}$, and a probability distribution to $\mathbf{x}$. There is some redundancy in the parameterization $\mathbf{w}$ and the prior on $\mathbf{x}$. For example, if the distribution over $\mathbf{x}$ is modelled as a normal distribution with diagonal covariance matrix ξ, then we can obtain an identical distribution over $\mathbf{q}$ by setting ξ to the identity matrix and rescaling each parameter w_{ih} by an appropriate factor. For this reason, I fix this distribution to be a unit Gaussian and just adapt the parameters $\mathbf{w}$ to the data. There is nothing special about the choice of a Gaussian. A non-Gaussian distribution would be equally easy to handle with the Monte Carlo algorithm that follows; indeed the evidence could be used to infer the most appropriate distribution on $\mathbf{x}$.

We could create fancier distributions over $\mathbf{q}$ by defining $\mathbf{q}$ to be a more complex ('multilayer') function of the hidden variable $\mathbf{x}$.

Our tasks are:

1. Given $\mathbf{w}$, and data $\mathbf{F}_{|j}$ for a particular context j, to infer $\mathbf{x}^j$ (the hidden components of the context) and predict the probability of 'the next i'.

2. Given data $\{\mathbf{F}_{|j}\}$ for many contexts j, to infer $\mathbf{w}$.

These tasks define two levels of inference. To solve the second task we will need to find the normalizing constant from the first.

For a few equations that follow I will make explicit the conditioning assumption $\mathcal{H}_\mathrm{C}$, which denotes the form of componential density model, and the assumption that the hidden vectors $\mathbf{x}^j$ are independent from one j to another.

Level 1

We start by assuming particular values for $\mathbf{w}$ and ξ. The posterior distribution of the hidden components is:

$$P(\mathbf{x}^j | \mathbf{F}_{|j}, \mathbf{w}, \mathcal{H}_\mathrm{C}) = \frac{P(\mathbf{F}_{|j} | \mathbf{x}^j, \mathbf{w}, \mathcal{H}_\mathrm{C}) P(\mathbf{x}^j | \mathcal{H}_\mathrm{C})}{P(\mathbf{F}_{|j} | \mathbf{w}, \mathcal{H}_\mathrm{C})} \tag{21}$$

The likelihood $P(\mathbf{F}_{|j} | \mathbf{x}^j, \mathbf{w}, \mathcal{H}_\mathrm{C})$ measures how well the model with its parameters set to $\mathbf{w}$ predicts the observed counts:

$$P(\mathbf{F}_{|j} | \mathbf{x}^j, \mathbf{w}, \mathcal{H}_\mathrm{C}) = \prod_{i=1}^{I} q_i(\mathbf{x}^j; \mathbf{w})^{F_{i|j}}. \tag{22}$$

The predictive distribution (the probability of 'the next i') is obtained by integrating over $\mathbf{x}$:

$$P(i | \mathbf{F}_{|j}, \mathbf{w}, \mathcal{H}_\mathrm{C}) = \int d^H\mathbf{x}\, P(\mathbf{x}^j | \mathbf{F}_{|j}, \mathbf{w}, \mathcal{H}_\mathrm{C}) q_i(\mathbf{x}^j; \mathbf{w}) \tag{23}$$

It may be useful to notice that this integral can be written as a ratio of 'evidences' (on which more later). If we define $\mathbf{F}_{|j}^{+i}$ to be the vector of counts with count i incremented by one, then we can write (using $P(A|B,C) = P(AB|C)/P(B|C)$):

$$P(i|\mathbf{F}_{|j}, \mathbf{w}, \mathcal{H}_{\mathrm{C}}) = \frac{P(\mathbf{F}_{|j}^{+i}|\mathbf{w}, \mathcal{H}_{\mathrm{C}})}{P(\mathbf{F}_{|j}|\mathbf{w}, \mathcal{H}_{\mathrm{C}})}$$

Level 2

We now wish to infer $\mathbf{w}$ given data $\{\mathbf{F}_{|j}\}$ for many different contexts j. The inference is:

$$P(\mathbf{w}, \xi|\{\mathbf{F}_{|j}\}, \mathcal{H}_{\mathrm{C}}) = \frac{P(\{\mathbf{F}_{|j}\}|\mathbf{w}, \mathcal{H}_{\mathrm{C}})P(\mathbf{w}, \mathcal{H}_{\mathrm{C}})}{P(\{\mathbf{F}_{|j}\}|\mathcal{H}_{\mathrm{C}})} \tag{24}$$

I will leave out the conditioning assumption $\mathcal{H}_{\mathrm{C}}$ for the rest of the section, but note here that in order to compare this model with other density models, we would want to evaluate the normalizing constant $P(\{\mathbf{F}_{|j}\}|\mathcal{H}_{\mathrm{C}})$.

The data dependent term in (24) separates into a product of factors, each of which is a normalizing constant from equation (21):

$$\begin{aligned} P(\{\mathbf{F}_{|j}\}|\mathbf{w}, \mathcal{H}_{\mathrm{C}}) &= \prod_{j=1}^{J} P(\mathbf{F}_{|j}|\mathbf{w}, \mathcal{H}_{\mathrm{C}}) \\ &= \prod_{j=1}^{J} \int d^H\mathbf{x} \prod_{i=1}^{I} q_i(\mathbf{x}^j; \mathbf{w})^{F_{i|j}} P(\mathbf{x}|\mathcal{H}_{\mathrm{C}}) \end{aligned}$$

Now the task is to evaluate the derivative of these evidence terms with respect to $\mathbf{w}$. Let us ignore for the moment the question of the prior on $\mathbf{w}$.

When we maximise the evidence over $\mathbf{w}$, our model will, as if by magic, create a componential model for the density over $\mathbf{q}$. If the vectors $\mathbf{q}$ do lie in a simple low-dimensional subspace, then the evidence maximization will automatically discover this; the parameters $\mathbf{w}$ will adjust themselves so that the appropriate number of degrees of freedom in $\mathbf{x}$ are used.

For brevity, the fixed assumptions $\mathcal{H}_{\mathrm{C}}$ will be omitted here, and I define

$$L_j(\mathbf{w}) = \log P(\mathbf{F}_{|j}|\mathbf{w}) \quad \text{and} \quad L(\mathbf{w}) = \sum_j L_j(\mathbf{w}) \tag{25}$$

and write:

$$P(\mathbf{F}_{|j}|\mathbf{w}) = \int d^H\mathbf{x} \, \exp(G_j(\mathbf{x}; \mathbf{w}))P(\mathbf{x}) \tag{26}$$

where

$$G_j(\mathbf{x}; \mathbf{w}) = \sum_i F_{i|j} \log q_i(\mathbf{x}; \mathbf{w}). \tag{27}$$

We now evaluate the derivative of the log evidence term.

$$\frac{\partial}{\partial w_{ih}} L_j(\mathbf{w}) = \frac{1}{P(\mathbf{F}_{|j}|\mathbf{w})} \int d^H\mathbf{x} \, \exp(G_j(\mathbf{x}; \mathbf{w}))P(\mathbf{x}) \frac{\partial}{\partial w_{ih}} G_j(\mathbf{x}; \mathbf{w})$$

This final derivative is easy to evaluate:

$$\frac{\partial}{\partial w_{ih}} G_j(\mathbf{x}; \mathbf{w}) = \Big(F_{i|j} - F_j q_i(\mathbf{x}; \mathbf{w})\Big) x_h. \tag{28}$$

Evaluation of the evidence and its derivatives using a simple Monte Carlo algorithm

The evidence and its derivatives with respect to $\mathbf{w}$ both involve integrals over the hidden components $\mathbf{x}$. For a hidden vector of sufficiently small dimensionality, a simple Monte Carlo approach to the evaluation of these integrals might prove effective.

Let $\{\mathbf{x}^{(r)}\}_{r=1}^{R}$ be random samples from $P(\mathbf{x})$. Then we can approximate the log evidence by:

$$\begin{aligned} L(\mathbf{w}) &= \sum_j \log \int d^H\mathbf{x}\ \exp(G_j(\mathbf{x};\mathbf{w}))P(\mathbf{x}) \\ &\simeq \sum_j \log\left[\frac{1}{R}\sum_r \exp(G_j(\mathbf{x}^{(r)};\mathbf{w}))\right]. \end{aligned}$$

Similarly the derivative can be approximated by:

$$\frac{\partial}{\partial w_{ih}} L(\mathbf{w}) \simeq \sum_j \frac{\sum_r \exp(G_j(\mathbf{x}^{(r)};\mathbf{w}))\left(F_{i|j} - F_j q_i(\mathbf{x}^{(r)};\mathbf{w})\right) x_h^{(r)}.}{\sum_r \exp(G_j(\mathbf{x}^{(r)};\mathbf{w})).}$$

This expression can be simplified by introducing the quantities:

$$b_{rj} = \frac{\exp(G_j(\mathbf{x}^{(r)};\mathbf{w}))}{\sum_{r'} \exp(G_j(\mathbf{x}^{(r')};\mathbf{w}))} \tag{29}$$

The quantity b_{rj} is like a posterior probability over the vectors r. We obtain:

$$\frac{\partial}{\partial w_{ih}} \sum_j L_j(\mathbf{w}) \simeq \sum_j \sum_r b_{rj} \left(F_{i'|j} - F_j q_i(\mathbf{x}^{(r)};\mathbf{w})\right) x_h^{(r)}.$$

A rough scaling law for the number of samples needed

The Monte Carlo method will fall down badly if, given any one data example $\mathbf{F}_j$, the posterior distribution over $\mathbf{x}$ does not receive any representation among the random samples $\{\mathbf{x}^{(r)}\}$. If the output probability vector is well determined, then the volume of the posterior bubble in $\mathbf{x}$ space will be of order $\sqrt{1/F_j}^D$ where the exponent D is the min of $(I-1)$, the dimension of the output space, and the dimension of the hidden space, H. The Jacobean between $\mathbf{x}$ and $\mathbf{q}$ is treated as a constant here. Thus the number of samples drawn randomly from the prior must scale at least as $F_j^{D/2}$. This scaling law does not take into account the wonderful adaptiveness of neural nets, however, which may make it possible to get by with fewer samples. If a density net is trained with a very small number of inputs R, it can adjust its weights $\mathbf{w}$ so as to make best use of them.

Notice that the algorithm has no difficulty scaling with increasing numbers of examples J.

This simple Monte Carlo algorithm is therefore best when there are lots of examples, for each of which $\mathbf{q}$ is not very well determined.

Some practical details

Computation time can be saved by omitting from the gradient calculations points with small b_{rj} that are expected to make negligible contribution. In these demonstrations I omitted the terms rj with $b_{rj} < 1/(100R)$.

The algorithm was implemented by evaluating the evidence and its gradient and feeding them into a conjugate gradient routine based on the `frprmn` code in Numerical Recipes. The random points $\{\mathbf{x}^{(r)}\}$ were kept fixed, so that the objective function and its gradient were not noisy quantities during the optimization.

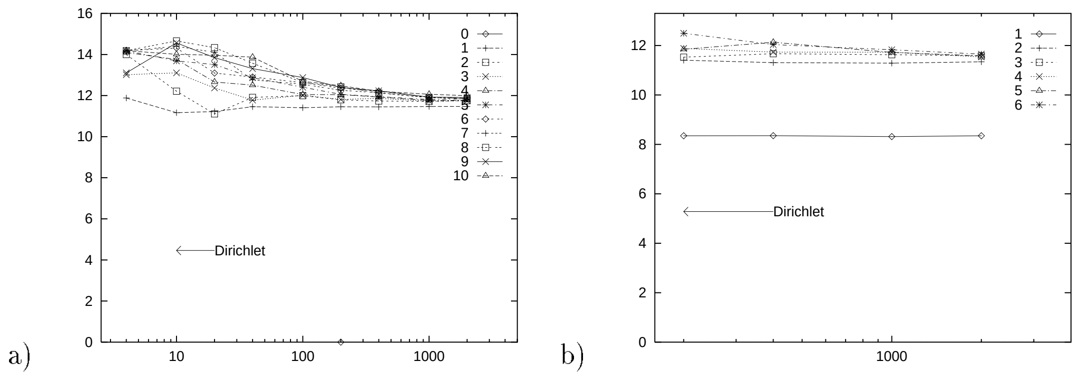

Figure 1: **Toy examples. Estimated evidence.**
Log evidence (y axis) is shown as a function of R (number of Monte Carlo samples, x axis), for models with different numbers of hidden components (H between 0 to 7).
The evidence for the optimized Dirichlet model is also marked. All values are log evidences relative to the null model $\mathcal{H}_0$.
a) Toy example number 1. b) Toy example number 2.

More efficient evaluation of the evidence using importance sampling

If we create a sampling distribution $Q_j(\mathbf{x})$ that is similar to the posterior distribution $P(\mathbf{x}|\mathbf{F}_j)$ then the evidence integral can be approximated in terms of $\{\mathbf{x}^{(r)}\}_{r=1}^{R}$, which are random samples from $Q(\mathbf{x})$.:

$$\begin{aligned} L_j(\mathbf{w}) &= \log \int d^H\mathbf{x}\ \exp(G_j(\mathbf{x};\mathbf{w}))P(\mathbf{x}) \\ &\simeq \log \left[\frac{1}{R} \sum_r \exp(G_j(\mathbf{x};\mathbf{w})) \frac{P(\mathbf{x})}{Q(\mathbf{x})} \right] \end{aligned}$$

Later, I use this expression to evaluate accurately the evidence for a model that has been adapted by the simple Monte Carlo method above. The sampling distribution $Q_j(\mathbf{x})$ is set to a Gaussian with mean $\bar{\mathbf{x}}_j$ and diagonal covariance matrix Σ_j obtained from statistics returned by the simple algorithm.

7 Examples

In the following examples I compute the log evidence ratio for the model relative to a null model, $\mathcal{H}_0$, that states that $\mathbf{q}$ is the same for all examples. This model is optimised by setting $\mathbf{q} = \mathbf{q}^0$, defined by $q_i^0 \equiv F_i/F$. A density network with $H = 0$, *i.e.*, no hidden inputs, is identical to $\mathcal{H}_0$.

The new model is also compared with a Dirichlet model which models the probability distribution over $\mathbf{q}$ as coming from a single distribution. This is not a componential model. The Dirichlet model has I free parameters which are optimized by evidence maximization as discussed in (**?**).

Toy problems

I created two toy data sets corresponding to sets of dice from two unrelated factories. In each data set, six five-sided dice were rolled a few times giving the data shown in table 1.

The data in TOY 1 has been constructed to show a roughly one-dimensional underlying component in the data, with some dice appearing to favour small values of i and some large values. In TOY 2 wrap-around has been added, so that a human might infer that the vectors $\mathbf{q}$ seem to lie in or on a circle, requiring two real dimensions.

	TOY 1		TOY 2
Data	(data table A below)	(data table B below)	

Data (TOY 1, first table):

j \ i	1	2	3	4	5
1	5	2	0	0	0
2	2	3	1	0	0
3	0	5	3	0	0
4	0	1	2	4	1
5	0	0	1	3	4
6	1	1	1	1	1

Data (second table):

j \ i	1	2	3	4	5
1	5	2	0	0	1
2	2	3	1	0	0
3	0	5	3	0	0
4	0	1	2	4	1
5	0	0	1	3	4
6	2	0	0	2	3

Inferrrred parameters

i	Bias	Input 1
1	-0.45	2.62
2	0.86	1.59
3	0.72	0.09
4	-0.26	-2.10
5	-1.11	-2.78

ONE HIDDEN COMPT.

i	Bias	Input 1	Input 2
1	-0.53	-1.79	1.92
2	0.87	-1.42	0.62
3	0.72	-0.48	-0.29
4	-0.16	1.44	-1.07
5	-0.94	2.05	-1.14

TWO HIDDEN COMPONENTS

i	Bias	Input 1	Input 2
1	-0.09	-1.09	-1.22
2	0.18	-1.32	0.87
3	-0.09	-0.48	1.20
4	-0.12	1.51	-0.10
5	0.07	1.18	-0.70

TWO HIDDEN COMPONENTS (TOY 2)

Table 1: **Parameters of models for the TOY problems**

The simple Monte Carlo algorithm gave the results illustrated in figure 1, as H and R were varied. The graphs show the evidence as a function of R. Notice that for R greater than 10 or so, the evidence value settles down, and increasing R makes negligible difference.

In the case of data TOY 1, as H is increased beyond 1, the evidence does not become either substantially larger or substantially smaller, even when the hidden vector has a dimensionality bigger than the dimensionality of the output space. This means that the model is finding a density of effective dimensionality about 1. There is apparently no overfitting problem.

In the case of data TOY 2, the results are similar, except that the model with a two-dimensional componential representation is significantly more probable than the one-dimensional density network.

One way to understand what a model is doing is to look at its parameters (at least for small H). Table 1 shows the parameters for the nets with $H = 1$ and $H = 2$, ordered from $i = 1$ to 5 vertically (c.f. horizontal in the data table earlier). Notice that the weights from the inputs in the TOY 1 cases capture the one dimension apparent to the human eye. When there are two inputs, the weight vectors for those inputs are not orthogonal; they are virtually identical (except for a change of sign). This similarity of the vectors of weights from the two inputs produces a low effective dimensionality in the output space.

When it is adapted to the TOY 2 data set, the parameters of the density network with two hidden components are very different. The two vectors over i are here virtually orthogonal, so that a fully two-dimensional distribution is produced in the output space.

Amino acid probabilities in aligned protein families

Figure 2 shows the estimated evidence, for $J = 60$ examples, each with a count of $F_j \simeq 177$. Clearly many Monte Carlo samples are needed for a convergent estimate of the evidence.

The evidence for the Dirichlet model is also displayed. According to these results, a componential model with 13 components is more probable than the Dirichlet model.

8 Application of mixture model to amino acid data

Here are some raw results on the probability of Dirichlet mixture models with increasing numbers of components. For each run a random start was used, with each $\mathbf{l} = \log \mathbf{u}$ vector being given a random value drawn from a Gaussian. One mixture component was initialized to a smaller α value than the

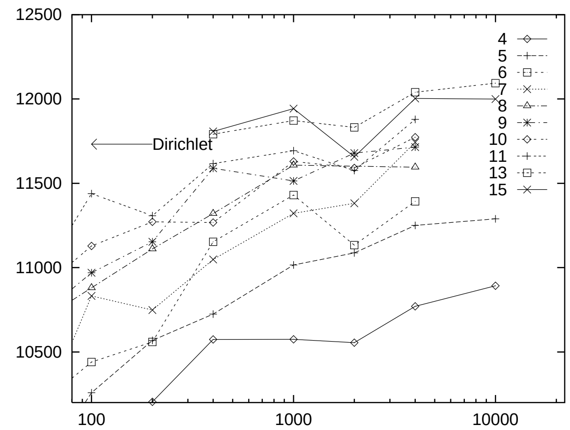

Figure 2: **Amino acid modelling.**
Estimated evidence, as a function of R (number of Monte Carlo samples, x axis), for models with different numbers of hidden components ($H = 3$ to 15).
The evidence for the optimized Dirichlet model is also marked. The evidence for other traditional Dirichlet models can also be reported: $\log P(D|\mathbf{u} = (1, 1, \ldots, 1)) = 10894.5$; $\log P(D|\mathbf{u} = (.05, .05, \ldots, .05)) = 11356.7$.
All values are log evidences relative to the null model $\mathcal{H}_0$.

others to make the anticipated 'twiglets' component. The amino acids are given in 'alphabet' order. For each mixture component, the probability p_c^M is given, and then the values of α and $\mathbf{u}$.

#c	p[c]	alpha	alani	cyste	aspar	glutt	pheny	glyci	histi	isole	lysin	leuci	methi	aspar	proli	glutn	argin	serin	threo	valin	trypt	tyros
1	0.271	0.412	0.034	.....	.....	.....	0.031	0.036	.....	.....	.....	.....	.....	.....	.....	.....	.....	0.032	.....	.....	.....	.....
2	0.275	2.18	0.207	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	0.156	.....	.....	.....	0.256	0.171	.....	.....	.....
3	0.030	18.6	0.749	.....	1.288	9.667	.....	.....	.....	.....	1.202	.....	.....	.....	.....	1.021	.....	.....	.....	.....	.....	.....
4	0.017	23.4	1.677	.....	.....	.....	.....	.....	.....	.....	.....	0.565	.....	.....	14.18	.....	.....	1.325	1.234	.....	.....	.....
5	0.037	8.79	.....	.....	.....	.....	3.001	.....	.....	0.343	.....	0.535	.....	.....	.....	.....	.....	.....	.....	0.362	.....	1.831
6	0.118	2.35	.....	.....	.....	.....	.....	.....	.....	0.531	.....	0.494	0.179	.....	.....	.....	.....	.....	.....	0.540	.....	.....
7	0.010	19.3	.....	.....	.....	.....	.....	.....	.....	3.291	.....	1.128	.....	.....	.....	.....	.....	1.117	2.777	2.360	.....	.....
8	0.004	19.6	.....	.....	.....	1.321	.....	.....	1.211	.....	.....	1.819	.....	.....	.....	.....	.....	.....	.....	.....	6.510	0.930
9	0.009	20.2	.....	.....	.....	2.323	.....	.....	.....	.....	2.548	1.708	.....	.....	.....	1.860	2.490	.....	.....	.....	.....	.....
10	0.079	9.06	.....	.....	.....	0.720	.....	.....	.....	.....	0.937	.....	.....	0.670	.....	.....	0.926	0.791	.....	.....	.....	.....
11	0.021	29.8	1.336	.....	.....	.....	.....	.....	.....	.....	14.39	.....	.....	.....	.....	.....	1.318	1.568	.....	1.258	.....	.....
12	0.049	16	.....	.....	.....	.....	0.758	.....	.....	3.397	.....	3.832	0.781	.....	.....	.....	.....	.....	.....	2.928	.....	.....
13	0.009	32.7	1.197	.....	.....	.....	.....	1.404	.....	.....	.....	.....	.....	18.15	.....	.....	.....	2.467	1.313	.....	.....	.....
14	0.013	34	.....	.....	.....	0.817	.....	.....	.....	.....	.....	0.906	.....	.....	.....	.....	.....	2.591	21.80	0.799	.....	.....
15	0.010	22	10.90	.....	.....	.....	.....	1.703	.....	.....	.....	0.917	.....	.....	.....	.....	.....	1.630	.....	1.271	.....	.....
16	0.033	35.1	.....	.....	.....	.....	1.394	.....	.....	1.647	.....	24.44	1.235	.....	.....	.....	.....	.....	.....	2.050	.....	.....
17	0.002	15	.....	.....	.....	.....	.....	.....	.....	.....	0.916	1.443	.....	.....	.....	.....	.....	1.118	0.919	.....	.....	0.972
18	0.006	25.8	2.357	.....	.....	.....	.....	.....	.....	.....	.....	2.193	.....	.....	.....	.....	2.901	2.934	3.688	.....	.....	.....

9 Discussion

Here I have cut a major corner by doing maximum evidence (maximum likelihood) fitting of the hyperparameters. This means that the Occam factors for models with excessive numbers of components are not included, and there may be a potential overfitting problem.

The advantages of a fully Bayesian attitude to data modelling are, firstly, that one is forced to make all one's assumptions explicit; and secondly, that once the model is defined, all inferences and predictions are mechanically defined by the rules of probability theory.

There are two major omissions in the models for amino acid probabilities. The first is that the evolutionary relationship between the inidividuals in the data have been ignored. The different amino acid observations in one column have been viewed as independent rolls of a die belonging to that column, whereas actually they constitute the leaves of a tree and are therefore not independent. The second omission is also to do with evolution. The amino acids are coded for by codons in DNA, and the amino acid in any one column has changed because the underlying codon has mutated. Our knowledge of the genetic code can give us an *a priori* model for the amino acids in the evolutionary tree of one column. Evolutionary effects are certainly present in the data studied here. If, for example, we count the number of times that an amino acid couplet consists of two amino acids that are neighbours in the genetic code (*i.e.*, could intermutate with just one nucleotide change)

A Toolbox

The Gamma function

The Gamma function is defined by $\Gamma(x) \equiv \int_0^\infty du\; u^{x-1} e^{-u}$, for $x > 0$. In general, $\Gamma(x+1) = x\Gamma(x)$, and for integer arguments, $\Gamma(x+1) = x!$. The digamma function is defined by $\Psi(x) \equiv \frac{d}{dx} \log \Gamma(x)$.

For large x (for practical purposes, $0.1 \leq x \leq \infty$), the following approximations are useful:

$$\log \Gamma(x) \simeq \left(x - \tfrac{1}{2}\right) \log(x) - x + \tfrac{1}{2} \log 2\pi + O(1/x) \tag{30}$$

$$\Psi(x) = \frac{d}{dx} \log \Gamma(x) \simeq \log(x) - \frac{1}{2x} + O(1/x^2) \tag{31}$$

And for small x (for practical purposes, $0 \leq x \leq 0.5$):

$$\log \Gamma(x) \simeq \log \frac{1}{x} - \gamma_e x + O(x^2) \tag{32}$$

$$\Psi(x) \simeq -\frac{1}{x} - \gamma_e + O(x) \tag{33}$$

where γ_e is Euler's constant. The digamma function satsfies the following recurrence relation exactly:

$$\Psi(x+1) = \Psi(x) + \frac{1}{x}. \tag{34}$$

Looking at the evidence again

It is interesting to approximate the evidence term appearing in (8) so as to see what properties it has. Using approximations for large u_i from appendix A, we obtain for a particular j:

$$\begin{aligned}\log P(\{F_{i|j}\}|\alpha\mathbf{m}, \mathcal{H}_{\mathcal{D}}) &= \log \left[\frac{\prod_i \Gamma(F_{i|j} + \alpha m_i)}{\Gamma(F_j + \alpha)} \frac{\Gamma(\alpha)}{\prod_i \Gamma(\alpha m_i)} \right] \\ &\simeq \sum_i F_{i|j} \log \langle q_{i|j} \rangle + \sum_i \alpha m_i \log \frac{\langle q_{i|j} \rangle}{m_i} - \frac{1}{2} \sum_i \log \frac{F_{i|j} + \alpha m_i}{\alpha m_i} + \frac{1}{2} \log \frac{F_j + \alpha}{\alpha}\end{aligned}$$

This decomposition of the evidence bears a detailed comparison with the evidence for interpolation and image reconstruction (Gull 1989, MacKay 1992). The first term is the "best fit log likelihood", achieved when the parameters $q_{i|j}$ are set to the posterior mean. The second term is the "best fit prior cost", measuring the distance of the best fit parameters from the null prior parameter value. The final terms are the "log volume factor" measuring how much the parameter space collapses when the data arrive. There are J negative terms and 1 positive term, corresponding to the collapse in the $J - 1$ dimensional parameter space $\mathbf{q}_{|j}$.

References

D. J. C. MacKay (1992). Bayesian interpolation, *Neural Computation* **4**(3): 415–447.

M. Lewicki (1994). Bayesian modeling and classification of neural signals, *Neural Computation* **6**(5): 1005–1030.

R. Hanson, J. Stutz and P. Cheeseman (1991). Bayesian classification with correlation and inheritance, *Proceedings of the 12th International Joint Conference on Artificial Intelligence, Sydney, Australia*.

S. F. Gull (1989). Developments in maximum entropy data analysis, *in* J. Skilling (ed.), *Maximum Entropy and Bayesian Methods, Cambridge 1988*, Kluwer, Dordrecht, pp. 53–71.

I thank Radford Neal, Geoff Hinton, Tim Hubbard, Sean Eddy and Graeme Mitchison for helpful discussions.

I thank Peter Brown, Radford Neal, Geoff Hinton, Phil Woodland, David Robinson, Martin Oldfield, Steve Gull, John Bridle and Graeme Mitchison for helpful discussions, and the Isaac Newton Institute for hospitality.