Pattern Recognition *via* Neural Networks

Brian D. Ripley

Professor of Applied Statistics
University of Oxford

Programme

- Pattern recognition
- Kohonen's SOM and visualization
- Fitting feed-forward neural networks
- Estimating the performance of a network
- Selecting and combining network architectures
- Traps for the unwary

For simplicity, the examples use feed-forward neural networks (MLPs), but the principles apply much more widely, including to radial basis function networks.

'On-line' *vs* 'Batch' algorithms

In the neural networks literature, emphasis is placed on *'on-line'* methods, in which the data are not stored except through the changes the learning algorithm has made.

The theory of such algorithms is studied for a very long stream of examples, but the practical distinction is less clear, as this stream is made up by

- repeatedly cycling through the training set or
- sampling the training examples (with replacement).

In contrast, methods which use all the examples together are called *'batch'* methods.

It is often forgotten that there are intermediate positions, such as using small batches chosen from the training set.

Pattern Recognition

Supervised *vs* unsupervised

Most pattern recognition problems have pre-assigned classes of patterns, and the task is to classify new observations.

Sometimes the task includes finding the patterns (cluster analysis?).

'Statistical' pattern recognition

so called by electrical engineers.

Very little is assumed about the classes of patterns: everything is 'learned from examples'.

Structural pattern recognition

Quite a lot is assumed about the classes of patterns.

Main example is *syntactic* pattern recognition using formal languages.

Very few working examples. [Typeset maths.]

'Topological' Mappings

Kohonen describes his own motivation as:

> 'I just wanted an algorithm that would effectively map similar patterns (pattern vectors close to each other in the input signal space) onto contiguous locations in the output space.'
>
> Kohonen (1995, p. VI)

He interpreted 'contiguous' via a rectangular or hexagonal 2-D lattice.

Multidimensional scaling (MDS) is a family of techniques to map points in feature space into $\mathbb{R}^k$ (usually for $k = 2$) to approximate desirable aspects of the distances between points.

Which aspects are thought desirable varies between methods. The best known outside psychometrics and statistics is *Sammon mapping*, which is one of the cruder methods.

Kohonen's SOM

In K*-means clustering* an example is assigned to the cluster whose representative $\boldsymbol{m}_j$ is nearest to the example.

In SOM the representatives $(\boldsymbol{m}_j)$ are arranged on a regular grid, with representatives at nearby points on the grid are more similar that those which are widely separated.

Examples are presented in turn until convergence. The $\boldsymbol{m}_j$ are initially assigned at random. Whenever an example $\boldsymbol{x}$ is presented, the closest representative $\boldsymbol{m}_j$ is found. Then

$$\boldsymbol{m}_i \leftarrow \boldsymbol{m}_i + \alpha[\boldsymbol{x} - \boldsymbol{m}_i] \qquad \text{for all neighbours } i.$$

Both the constant α and the definition of 'neighbour' change with time.

[Typically α might decline linearly from 1.0 to 0.04 over 1000 examples, then linearly to zero over the second thousand, while the definition of a 'neighbour' is a grid point i within distance r of j, where r declines linearly from 6 to 1 over the first 1000 examples.]

Dissimilarities

The Kohonen algorithms lie on finding the closest 'reference vector' $\boldsymbol{m}_i$ in feature space. How 'closest' is measured is not discussed, but the code uses Euclidean distance. May need more general measures if the features are not purely continuous (and if they are highly correlated).

Dissimilarities generalize distances; they are non-negative and symmetric, but need not satisfy the triangle inequality. Lots are known, e.g. Jaccard matching coefficient, Mahalanobis distance, Manhattan distance and combinations too.

We will assume that we are given a matrix (d_{rs}) of dissimilarities between pairs of objects, and $d_{rr} = 0$ for all r.

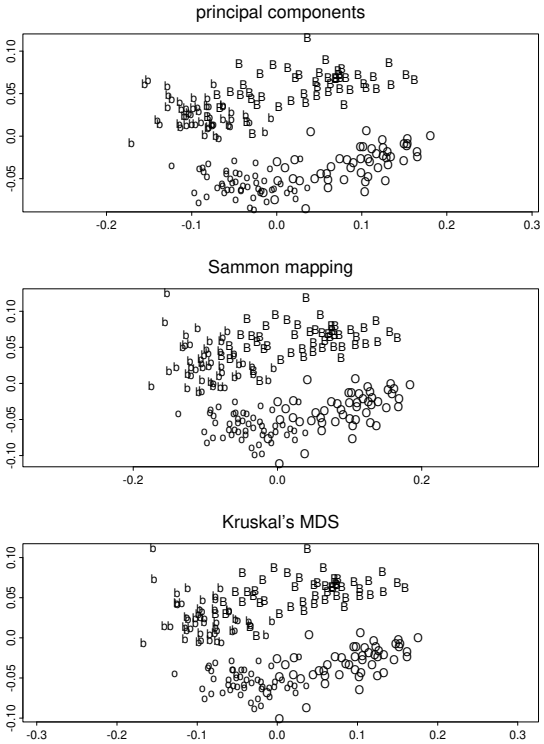
Leptograpsus Crabs

Campbell & Mahon (1974) studied rock crabs of the genus *Leptograpsus*. One species, *L. variegatus*, had been split into two new species, previously grouped by colour form, orange and blue. Preserved specimens lose their colour, so it was hoped that morphological differences would enable museum material to be classified. Carapaces (shells) may need to be sexed.

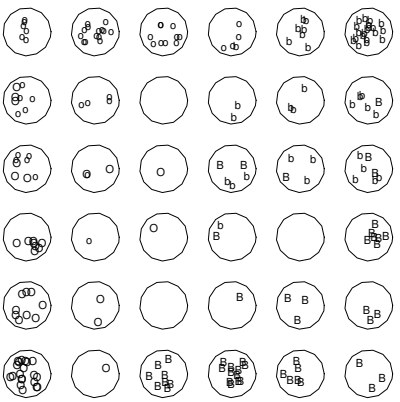
Data are available on 50 specimens of each sex of each species, collected on sight at Fremantle, Western Australia. Each specimen has measurements on

- the width of the frontal lip of the carapace
- the rear width,
- length along the midline,
- the maximum width of the carapace,
- body depth

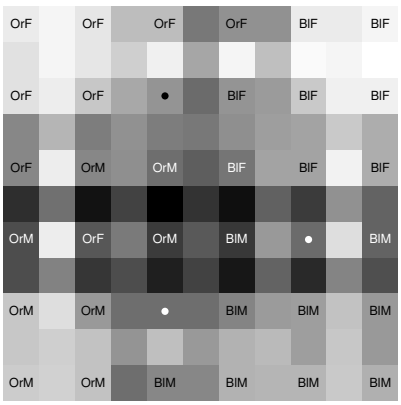
A $\log$ transformation was used for all the variables.



Multidimensional scaling for the Crabs data. The coding is upper case for males, lower case for females, 'B' for blue and 'O' for orange.



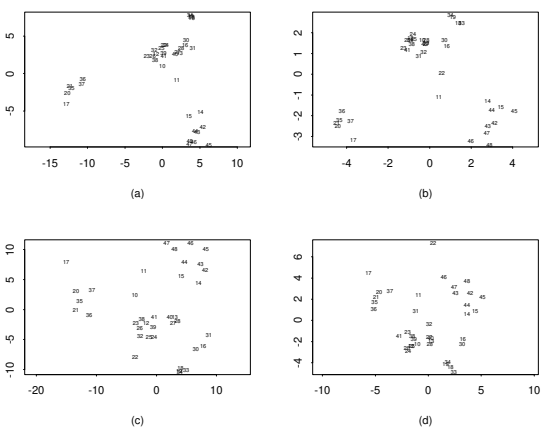
SOM mapping of the crabs data to a 6×6 grid. The labels of those examples mapped to each cluster are distributed randomly within the circle representing the cluster.



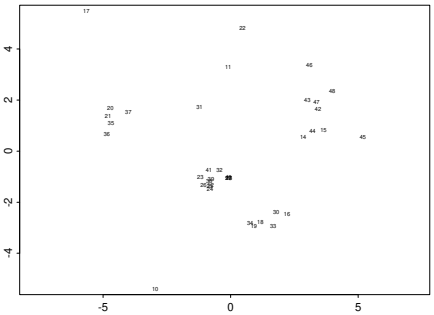
Ultsch representation of the SOM representatives for previous figure. The rows and columns between the units represent the magnitude of the gradient (black being high); the greylevel at each unit represents the median of the surrounding gradients. The label is the most common class mapped to that representative.

Viruses

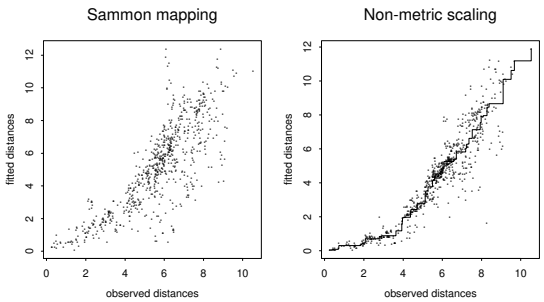
This is a dataset on 39 viruses with rod-shaped particles affecting various crops (tobacco, tomato, cucumber ...). There are 18 measurements on each virus, the number of amino acid residues per molecule of coat proteins.



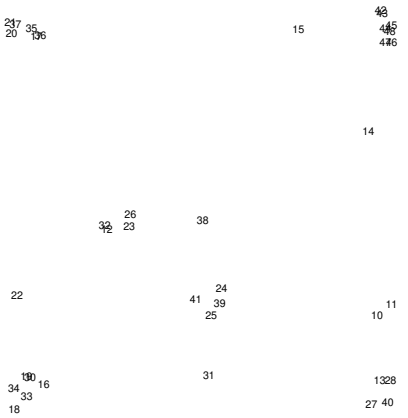
Principal component (top row) and Sammon mapping (bottom row) plots of the viruses. The plots in the left column are of the raw data, those in the right column with variables rescaled to have unit variance.



Non-metric multidimensional scaling plot of the viruses. The variables were scaled before Euclidean distance was used.



Distortion plots of Sammon mapping and non-metric multidimensional scaling for the viruses data. For the right-hand plot the fitted isotonic regression is shown as a step function.



SOM mapping of the viruses data to a 5×5 grid.

Automated Sleep EEG Analysis

From

Roberts, S. & Tarassenko, L. (1995) Automated sleep EEG analysis using an RBF network. In *Neural Network Applications*, ed. A. F. Murray, pp. 305–322. Kluwer.

Coefficients of AR(10) fit to 128-sample frames of EEG produced every second.

Populated cells of 10×10 SOM mapping of AR coefficients.

Statistical Pattern Recognition

K classes, plus 'doubt' $\mathcal{D}$ and 'outlier' $\mathcal{O}$.

Cases $(\boldsymbol{X}, Y)$ are drawn independently.

Training set $\mathcal{T}$ is a set of n classified cases.

Future cases present a pattern $\boldsymbol{X}$ for a decision.

Assign costs C_{jk} to wrong decisions. Bayes' rule chooses decision which minimizes

$$\sum_{k \neq j} C_{jk}\, p(j \mid \boldsymbol{x})$$

Posterior probabilities $p(k \mid \boldsymbol{x})$ are central.

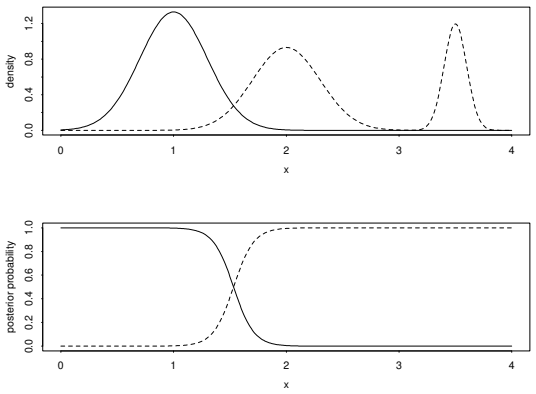
Prior probabilities π_k describe abundance of three class in the population.

Often $C_{jk} = 1$ except cost $d < 1$ for doubt. Then Bayes rule is

$$k(\boldsymbol{x}) = \begin{cases} k & \text{if } p(k \mid \boldsymbol{x}) = \max_j p(j \mid \boldsymbol{x}) > 1 - d \\ \mathcal{D} & \text{if } \max_j p(j \mid \boldsymbol{x}) \leq 1 - d \end{cases}$$

Learning Probabilities

It is not always necessary to model the class-conditional densities (upper figure) accurately, as the posterior probabilities in the lower figure are effectively unchanged by most aspects of modelling the right peak of the class-conditional density shown dashed. Only the densities in the interval $[1, 2]$ matter.



So posterior probabilities $p(k \mid \boldsymbol{x})$ are most important.

Logistic Discrimination

First consider just 2 classes. Look at the log-odds

$$\operatorname{logit} p(2 \mid \boldsymbol{x}) = \log \frac{p(2 \mid \boldsymbol{x})}{p(1 \mid \boldsymbol{x})} = \alpha + \boldsymbol{\beta}^T \boldsymbol{x}$$

and assume these are linear. Fit α and $\boldsymbol{\beta}$ by maximum likelihood, that is to maximize

$$\sum_{i \in \mathcal{T}} \log p(y_i \mid \boldsymbol{x}_i)$$

Now consider this as a feed-forward neural network with no hidden layer and one logistic unit. This is a form of training known there as 'entropy minimization'.

Think of this in two parts. A linear function + a logistic output stage. Now replace the linear function by a non-linear function $f(\boldsymbol{x}) = \operatorname{logit} p(2 \mid \boldsymbol{x})$ and approximate that by a neural network with hidden layer(s).

More classes

Suppose the classes are not ordered in any way. Natural generalization is

$$f_k(\boldsymbol{x}) = \log p(k \mid \boldsymbol{x}) = \alpha_k + \boldsymbol{\beta}_k^T \boldsymbol{x}$$

a *log-linear model* or *multiple logistic regression*. Note that we can take $\alpha_1 = \boldsymbol{\beta}_1 = 0$ since only ratios of probabilities matter (since they sum to one).

The parameters are still chosen by maximum likelihood with the same formula.

Now approximate the function $f : \mathcal{X} \to [0, 1]^K$ by a K-output feedforward neural net.

This is sometimes known as *softmax*. It is not the usual procedure with K logistic output units, when the outputs are not guaranteed to sum to one.

Using a Flexible Family for Classification

Suppose we are given a family $f(x; \theta)$, how do we use it to do discrimination? How do we use linear functions?

(a) We model the log posterior probabilities

$$\log P(C = c \mid X = x) = g_c(x; \theta)$$

and fit by maximum (conditional) likelihood. Allocate future examples to class with highest $g_c(x; \widehat{\theta})$.
[*Logistic discrimination*]

(b) Fit the regression

$$E(I(C = c) \mid X = x) = g_c(x; \theta),$$

allocate to class with largest $g_c(x; \widehat{\theta})$, equivalent to the class with nearest 'target'
$\boldsymbol{t}_c = (0, \ldots, 0, 1, 0, \ldots)$ to $\boldsymbol{g}(x; \widehat{\theta})$.
[*'Minimum (mean) square classifier'*]

It has been known since Fisher that (b) is equivalent to LDA for two classes, and so mode efficient under a normal model with equal within-class covariances.

This is not so for more than two classes, and it has been suggested (Breiman & Ihaka) to generalize the following characterization of LDA:

(i) Fit the regression

$$E(I(C = c) \mid X = x) = g_c(x; \theta)$$

(ii) Apply standard LDA in the lower-dimensional space of fitted values.

Dietterich & Bakiri (1991, 1995) have suggested choosing higher-dimensional targets which are well-spaced in m dimensions for $m > K - 1$.

'Plug-in' *vs* 'Predictive'

Statistical decision theory tells us that as we don't know $P(C = c \,|\, X = x)$, but should use our best estimate $P(C = c \,|\, X = x, \mathcal{T})$, where $\mathcal{T}$ denotes the training data. This is called the *predictive* approach, whereas using $P(C = c \,|\, X = x; \widehat{\theta})$ is called the *plug-in* approach.

We have

$$P(c \,|\, x, \mathcal{T}) = \int P(c \,|\, x; \theta) p(\theta \,|\, \mathcal{T}, x) \, \mathrm{d}\theta$$

$$p(\theta \,|\, \mathcal{T}, x) \propto \prod_{i=1}^{n} P(c_i \,|\, x_i; \theta) p(x_i; \theta) \; p(\theta) p(x; \theta)$$

and normally assume that $p(x; \theta)$ does not in fact depend on θ (and hence $p(\theta \,|\, \mathcal{T}, x)$ does not).

There will be appreciable differences only if $p(c \,|\, x; \theta)$ is not highly peaked *and* $p(c \,|\, x; \theta)$ is not linear around the posterior mode for θ. The differences are small in LDA and logistic discrimination, but can be large in LDA and non-linear logistic methods.

Training a FFNN

Fitting a FFNN is normally done by least-squares or maximum likelihood, although unconventional optimization algorithms are used.

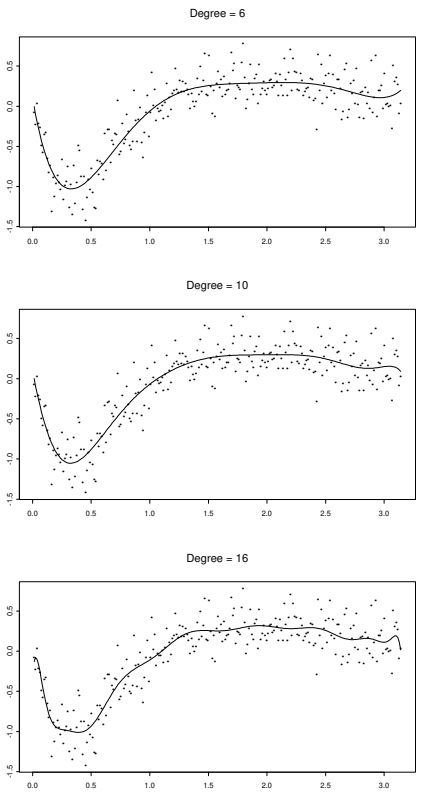
What's different about neural networks, compared to curve-fitting as done in statistics?

Main differences are:

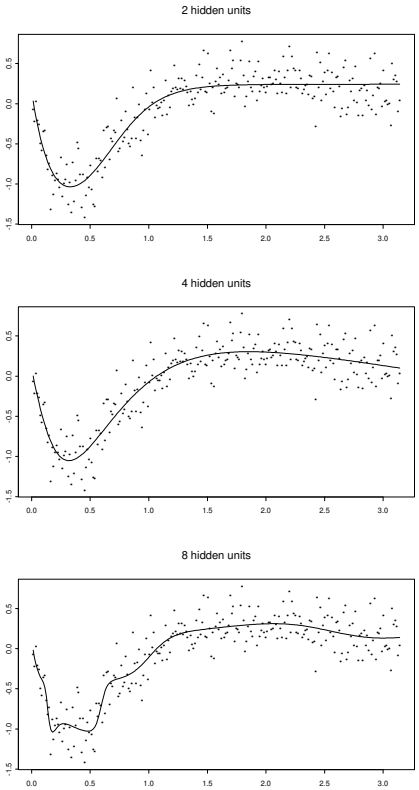
- Multiple local minima are common (especially for large models).
- Over-fitting is controlled by non-likelihood fitting.
- Sequence of possible architectures is not necessarily nested.

Consequence:

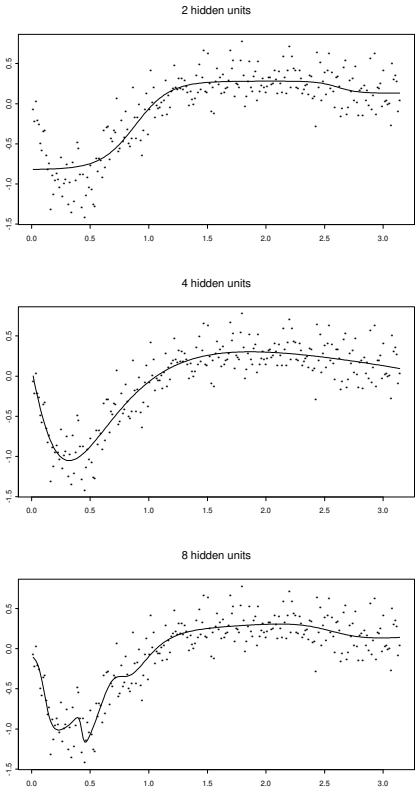
Everyone fitting polynomials will get the same answer, but almost no two people fitting neural networks will. (*Exception:* networks without hidden layers.)



Fits with polynomials.



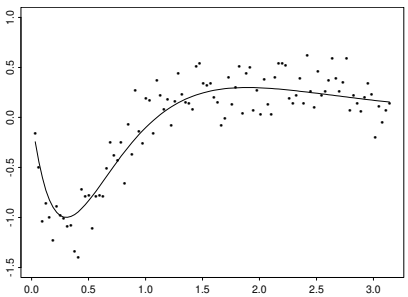
Fits with feed-forward neural networks.



More fits with feed-forward neural networks.

Demonstration example

Taken from Wahba & Wold (1975) where it illustrates smoothing splines.

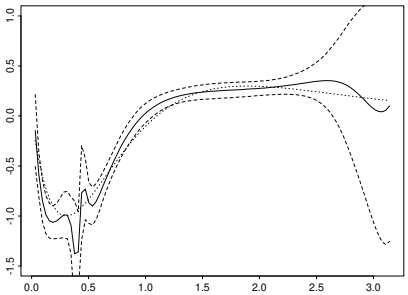


100 data points from a curve-fitting problem, with the true curve.

Throughout we assume that the true noise variance, $(0.2)^2$, is known.

Regression Neural Networks

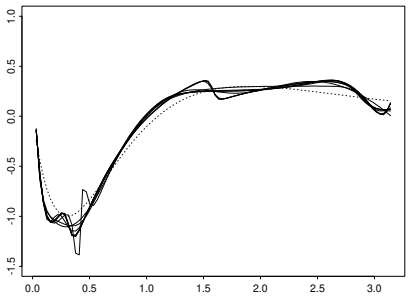
Model selection amounts to choosing the number of hidden units.



Curve fitted by 1-8-1 neural network, with ± 2 standard error bands and true curve (dotted).

The error limits come from standard regression theory, and are based on local linearization.

One problem is multiple local minima:



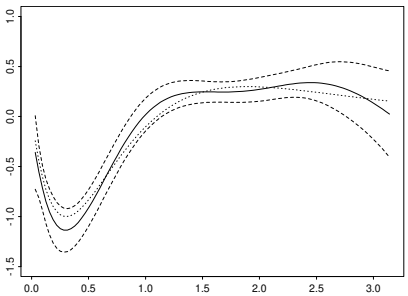
1-8-1 neural networks with different starting weights.

We clearly have over-fitting. Many ways to avoid this. Perhaps the best is to use *regularization* as used for smoothing splines. Somewhat easier for neural networks to use *weight decay*:

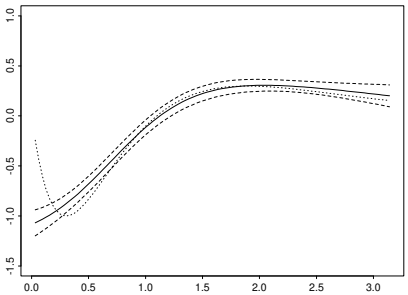
$$\sum_{i=1}^{n}[y_i - f(x_i; \mathbf{w})]^2 + \lambda \sum_{ij} w_{ij}^2$$

with the inputs and outputs suitably scaled.

Effect of Weight Decay

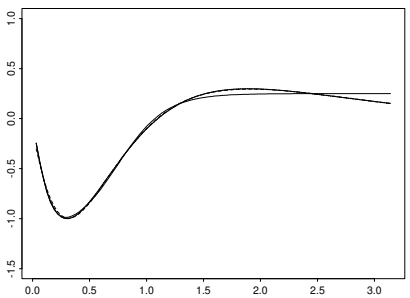


Curve fitted by 1-8-1 neural network, with ± 2 standard error bands and true curve $\lambda = 10^{-3}$ (above) and $\lambda = 0.1$ (below).



Bias/variance Compromises

We now have two ways to control the fit, the number of hidden units and λ, the degree of regularization. The true curve is not exactly representable by our model, so we have some bias.



Fitting to the true curve with 2 and 3 hidden units, and 8 units with $\lambda = 10^{-3}$.

Choosing fewer hidden units leads to more bias, as does adding a regularizer. But it also reduces their variability, so reduces the mean-square error. Can manipulate λ and the number of hidden units as a function of n to get the right compromise for very large n. Doesn't help in real problems!

Selection Techniques

Many classical techniques work with *nested* families of models, e.g. degree of a polynomial regression, number of hidden units in a 'raw' FFNN.

- *Forwards selection* Add terms until the fit is good enough (cascade correlation).
- *Backwards selection* Build a large model then drop terms (projection pursuit regression, MARS, . . .).
- *Stepwise selection* Try both adding and dropping at each step.

Larger models will fit better, so need either significance tests or penalty for complexity. Many such

(a) *AIC* Akaike's 'An Information Criterion'

(b) *BIC* 'Bayesian Information Criterion'

(c) *MDL/MML* 'Minimum Description (or Message) Length— Rissanen and Wallace schools.

These work for non-nested models, but a search in model space is needed.

Cross-Validation

Suppose we have:

- A training set $\mathcal{T}$ used to fit weights.
- A validation set used to select architectures (models).
- A test set used to estimate performance of selected model.

We can then use the performance on the *validation* set to choose the best model. ('Hold-out' method of Highleyman, 1962.)

Wasteful of data, so split the training set into V parts. Use $(V-1)$ of these to train, and test the performance on the Vth part. Do this for all V parts. This creates a pseudo test set of size n. The performance measure is (almost) unbiased, since the parts are (almost) independent, but it is for performance at size $(1-1/V)n$.

Idea from the mid-1970s (M. Stone, Geisser, ...).

How do we choose V? Traditionally *leave-one-out*, so $V = n$. This gives very variable estimates of performance, and can be seriously wrong. Often better to choose $V \approx \sqrt{n}$ or 10, and much cheaper computationally.

This estimates the effect of over-fitting by $O(1/\sqrt(n))$ perturbations of the problem, the order of the effects of variation between training sets. The *Bootstrap* is based on same notion.

Caveat

(Cross-)validation applies to a well-defined procedure, so troubles with multiple local minima, early stopping and so on.

AIC, NIC, p_{eff}

Akaike's (1973, 4) 'An Information Criterion'.
ML measure of fit is the deviance

$$2\sum \log p(X_i, Y_i)/p(X_i, Y_i; \widehat{\theta})$$

on the training set (the supplied data). Consider a test set, that is an independent sample from the same process of the same size. We would expect the deviance for the test set to be larger.

AIC corrects this bias:

$$E\big[\text{deviance}_{\text{test}} - \text{deviance}_{\text{train}}\big] \sim 2p$$

for p parameters and for large n *under the model*. Choose model(s) which have small values of

$$\text{deviance}_{\text{train}} + 2p$$

Since AIC is based on maximum likelihood estimations, it has nothing to say in our example if regularization is used.

If we drop the requirements that the model is true, and that fitting is by ML, we find

$$E\big[\text{deviance}_{\text{test}} - \text{deviance}_{\text{train}}\big] \sim 2p_{\text{eff}}$$

where $p_{\text{eff}} = \text{tr}[KJ^{-1}]$ and

$$J = -E\frac{\partial^2 \log p(X_i,Y_i;\theta_0)}{\partial\theta\,\partial\theta^T}, K = \text{Var}\frac{\partial \log p(X_i,Y_i;\theta_0)}{\partial\theta}$$

and θ_0 is the 'least false' value of θ, that is minimizes

$$-E\log p(X, Y; \theta)$$

This is due to Moody (1991, 1992) and Murata, Yoshizawa & Amari (1991,3,4) (hence NIC). Derived (but not stated) by Stone (1977). Proof is easy, based on second-order Taylor expansions of the log-likelihood, *about a single local maximum*.

Note that AIC, NIC only correct the bias as an estimate of the expected test-set deviance. We have to consider the fluctuations. For large n, AIC-like procedures never choose too small a model, but may choose too large a one.

Asymptotically LOO CV is the same as NIC (Stone, 1977), and often accurately so in small samples too. NIC is much less work!

BIC and Bayes Factors

'BIC' often refers to the penalty $p \log n$. An approximation to the use of *Bayes factors*.

The predictive Bayesian approach assesses models via $P(\text{model} \mid \mathcal{T})$, integrating out 'weights' θ. Now

$$P(\text{model} \mid \mathcal{T}) \propto P(\text{model}) P(\mathcal{T} \mid \text{model})$$

Strictly we average predictive distributions with weights $P(\text{model} \mid \mathcal{T})$. But if we choose, a crucial factor is

$$B_{st} = \frac{P(\mathcal{T} \mid \text{model}_s)}{P(\mathcal{T} \mid \text{model}_t)}$$

called the Bayes factor, since

$$\frac{P(\text{model}_s \mid \mathcal{T})}{P(\text{model}_t \mid \mathcal{T})} = B_{st} \frac{P(\text{model}_s)}{P(\text{model}_t)}$$

Often

$$\begin{aligned} \log P(\mathcal{T} \mid \text{model}) \;\approx\; & \log \ell(\widehat{\theta}; \mathcal{T}) + \log p(\widehat{\theta}) \\ & + \tfrac{p}{2} \log 2\pi - \tfrac{1}{2} \log |H| \end{aligned}$$

for H the Hessian of the sum of the first two terms, so can be approximated.

The last approximation depends on a single local maximum, but we can sum up over multiple local maxima.

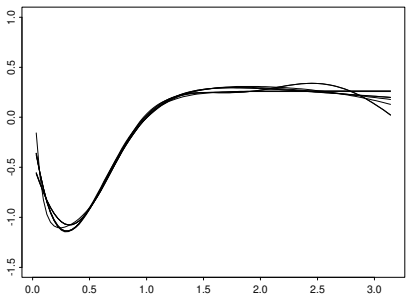
In example we look at $\sigma^2 \times \log P(\mathcal{T} \mid \text{model})$, since $\sigma^2 \times \log \ell(\widehat{\theta}; \mathcal{T})$ is the residual sum of squares.

Results for Example

Size	λ	SSq	SSq_{CV}	p_{eff}	NIC	Bayes
2	0	3.47	4.29	7	4.03	—
2	10^{-3}	3.57	4.78	5	3.98	3.30
3	0	4.12	4.81	10	4.92	—
3	10^{-3}	3.40	4.54	6	3.93	3.30
3	10^{-2}	3.61	4.91	6	4.11	3.72
3	10^{-1}	5.40	6.20	6	5.94	6.08
5	0	3.14	4.57	16	4.42	—
5	10^{-3}	3.21	4.51	17	4.59	3.38
5	10^{-2}	3.61	4.91	6	4.11	4.11
5	10^{-1}	5.38	6.18	6	5.92	5.98
8	0	2.89	5.35	25	4.89	—
8	10^{-3}	3.21	4.54	18	4.67	4.74
8	10^{-2}	3.58	4.94	7	4.16	4.63
8	10^{-1}	5.36	6.14	7	5.90	5.83

Column SSq_{CV} is based on 10-fold cross-validation, the same sets for all the models.

Model Uncertainty



The fitted curves for all the well-supported models.

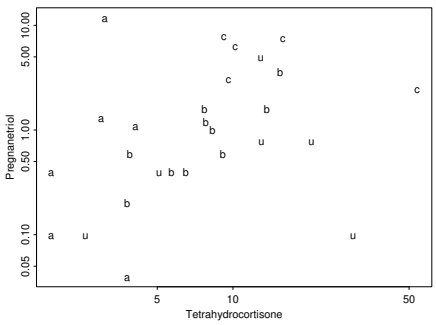
Finding the Bayes factors and weighted average is very time-consuming, *and* assumes the models are true. Stone (1974) suggested using LOO cross-validation to choose the weights in a weighted average, as have many people since.

Clearly we should take a weighted average over local minima, and it *may* suffice to take an unweighted average. In the Bayes approach we extend this to averaging over *models*: Wolpert's *stacked generalization*.

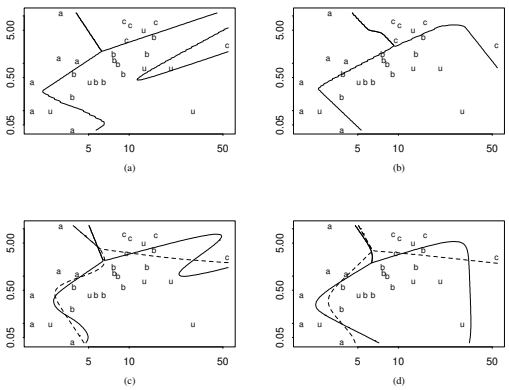
Cushing's Syndrome

Data on diagnostic tests on patients with Cushing's syndrome, a hypertensive disorder associated with over-secretion of cortisol by the adrenal gland.

This dataset has three recognised types of the syndrome represented as a, b, c. (These encode 'adenoma', 'bilateral hyperplasia' and 'carcinoma'.)



The observations are urinary excretion rates (mg/24h) of the tetrahydrocortisone and pregnanetriol, and are considered on log scale. Those marked u are future examples to be classified.

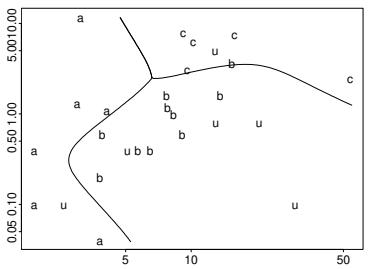


Neural network fits with two hidden units and connections from the input to output layer. Figures (a) and (b) are two solutions without weight decay which fit perfectly. Figures (c) and (d) show two local minima each, for $\lambda = 0.001$ and $\lambda = 0.01$ respectively. The dashed lines correspond to the local minimum that fits less well.

We can compare the seventeen local minima found with $\lambda = 0.01$. Six of them carry 5% or more of the total mass, with fitted values and weights

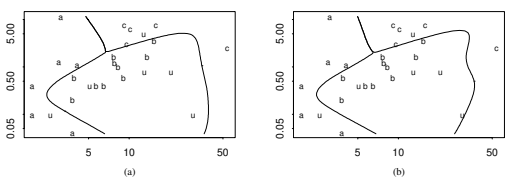
value	4.544	5.928	6.198	4.502	4.269	4.180
%	24.4	21.1	17.5	6.9	6.1	5.7

The second and sixth are shown at (d).



Predictive neural network fit to the data on Cushing's syndrome with two hidden units and $\lambda = 0.01$.

Adding more hidden units while keeping the weight decay constant makes only a little difference to the solution:

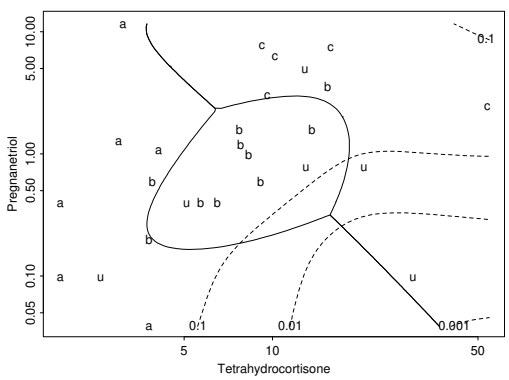


Neural network fits to the data on Cushing's syndrome with $\lambda = 0.01$. Part (a) had five hidden units, part (b) twenty.

Outliers

Strictly outliers are handled as a separate class with a specified distribution (vague?). A vague distribution for class $\mathcal{O}$ declares outlier if $\boldsymbol{x}$ is implausible for each other class, under the usual cost function.

An alternative view is to take an outlier to be any observation with low marginal density $p(\boldsymbol{x})$, roughly the same (if there are few classes).



A Bayes' rule for the Cushing's data, with low marginal density lines (dashed).