Programme

- What is pattern recognition?
- Setting up the problem
- Theory
- Fitting (feed-forward) neural networks
- Outliers
- Missing data
- Assessing the performance of a network
- Selecting and combining network architectures

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. This has two components:

- Visualizing the data to view groupings
- Formal methods to partition the data into subgroups (*cluster analysis*).

The 'neural networks' contribution to these has been small. See Chapter 9 of PRNN for a survey.

Supervised pattern recognition

‘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.

Examples

Credit scoring

System for a leasing company to accept/decline credit proposals from small businesses. Data available on past cases:

- Legal entity
- Type of business
- Time in business
- Existing hirer? If so for how long.
- Product
- Shareholder’s funds
- Exposure ratio
- Gearing ratio
- Items from last accounts
- Accountant’s reference
- Results of searches
- Any guarantees?

and outcome (settled good or written off).

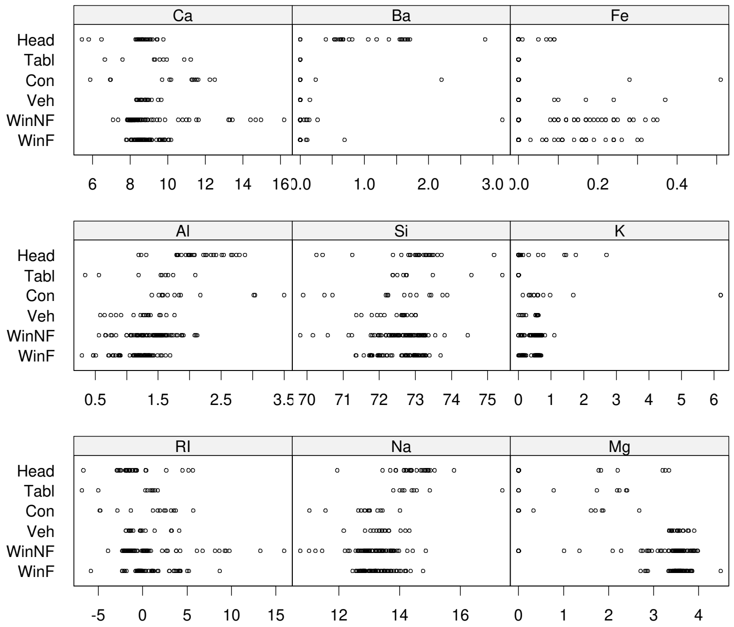
[Consulting problem.]

Forensic glass

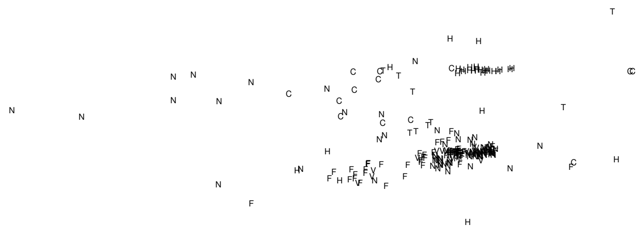
Comes from forensic testing of glass collected by B. German. This has 10 measurements on 214 fragments of glass from forensic testing, the measurements being of the refractive index and composition (percent weight of oxides of Na, Mg, Al, Si, K, Ca, Ba and Fe). The fragments have been classified by six sources, window float glass (70), window non-float glass (76), vehicle window glass (17), containers (13), tableware (9) and vehicle headlamps (29).

The classes seem not be be well-separated, and some classes seem to split into subclasses.

[Treated in several of my papers, and in PRNN.]



Univariate summary plots.



Isotonic multidimensional scaling representation. The groups are plotted by the initial letter, except `F` for window float glass, and `N` for window non-float glass.

ZIP code reading

Given segmented images of hand-written characters, specify the character (0–9 in the US, plus A–Z in the UK and Canada). Many similar problems (reading vehicle license plates, part nos on components, . . .).

Data are often 16×16 or 32×32 greyscale images.

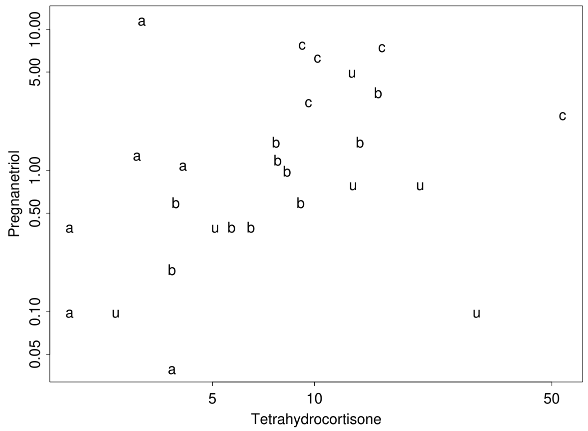
Not all the characters are equally likely, and we need to recognize a group of characters (the whole ZIP code), not just one.

In pictographic languages (such as Chinese) there are very many answers.

Cushing's syndrome diagnosis

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 tetrahydrocortisone and pregnanetriol, and are considered on log scale. Those marked u are future examples to be classified.

[Aitchison & Dunsmore (1975), PRNN]

Breast cancer prognosis

Predict time to relapse at diagnosis of breast cancer. May be just less than / greater than 3 years, or one of a series of intervals, or a survival time distribution. In one study prognostic factors used were:

- Age
- Histological type of tumour
- Degree of differentiation of the tumour (for ductal tumours only)
- Size of tumour
- Number of positive axillary nodes
- Oestrogen receptors (ER)
- Epidermal growth factor receptors (EGFR)
- Type of surgery
- radiotherapy?
- chemotherapy?
- Tamoxifen treatment?

[R.M. Ripley and A.L. Harris, 1996]

Setting up the problem

- Specify the outcomes
- What should we measure?
- What dataset should we collect?
- How should we code the inputs and outputs?
- How will we use the data?

Outcomes

Suppose we are given K predetermined classes.

- Are they mutually exclusive?
- Do we have to make a decision, or can we pass on to an expert / for more measurements, . . . ?
- Are all errors equally costly?
- Are the outcomes structured or ordered in any way?
- Is it possible that none of the classes are correct?
- Are the classes in the dataset known to be correct?

We need to specify a set of possible decisions, and the costs of making each decision for each possible correct class. (We will make implicit choices of costs if we do not specify explicit ones.)

What should we measure?

The items measured on each example are known as *features*.

The naïve answer

> 'as many features as possible'

may not be right!

- Asking for maximal information may restrict the dataset that can be collected, and may also bias it.
- In theory we can make a better classifier using more features, but this may be hard to achieve, and may deflect attention (and CPU time) from more important parts of the problem.

For example, in ZIP code recognition we may reduce the 16^2 or 32^2 features to, say, 30 principal components.

The dataset

The dataset we select for learning needs to be representative of the future examples we will have to classify.

A problem with credit scoring, as we only know about cases that were accepted.

A problem with medical diagnosis/prognosis as referral patterns may change.

We may want to bias the training set and correct later. We might want to do this:

- to collect sufficient information on rare classes
- when incorrect decisions about some classes are much more expensive than about others.

Often the expensive classes are also rare ones.

We can deliberately over-sample certain classes. (‘Get 100 examples of each digit.’) But if the data are already available there are better ways to use it all. (See *weighted training* below.)

Training, validation and test sets

We have one set of data. We can divide it into:

- A *training* set $\mathcal{T}$ used to fit weights.
- A *validation* set used to select input features and 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.)

These terms are much misused! This is *not* cross-validation (see later).

Coding inputs and outputs

Outputs

Several cases:

(a) Two classes. Use a single output, the indicator of one class.

(b) Several non-exclusive classes. One output for each class.

(c) Several mutually exclusive classes. Use a multiple logistic model, aka *softmax*. This has one non-negative output for each class, with outputs summing to one.

(d) Ordered classes have special methods.
(See Mark Mathieson's poster.)

Inputs

In theory with universal approximators the input coding doesn't matter, but a helping hand is a very good idea.

- Categorical inputs — use an indicator variable for all but one of the levels.
- Ordinal inputs — use as numerical inputs or 'thermometer' or orthogonal polynomial encoding.
- Zero or positive — may want to encode as $0/ > 0$ and the actual value.
- Related continuous measurements (e.g. glass components) — may wish to transform to principal components.

For some methods linear transformations of the inputs are in theory irrelevant, but often scaling the inputs so that Euclidean distance is a good measure of similarity will help in fitting the models.

Theory

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

Cases $(\boldsymbol{X}, C)$ (features $+$ class) 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. The *Bayes' rule* minimizes the expected cost of future decisions by choosing the decision which minimizes

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

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

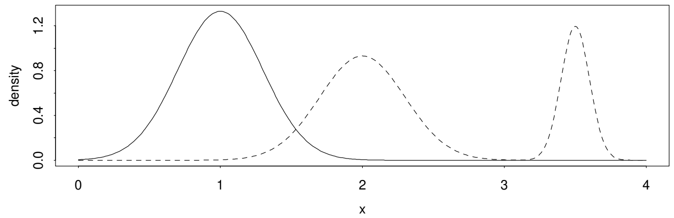
Prior probabilities π_k describe abundance of each class in the population.

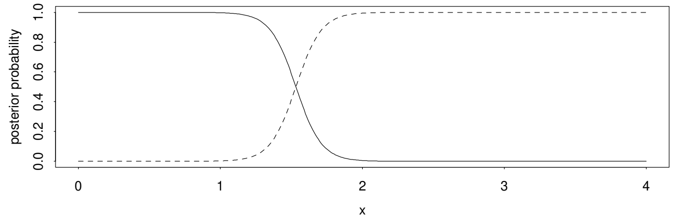
Often $C_{jk} = 1$ except cost $d < 1$ for doubt. Then Bayes rule is

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

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, but here only around 0.5.

Biased training sets

Suppose we knew (or estimated) the posterior probabilities $q(k \mid \boldsymbol{x})$ from a population with prior probabilities ψ_k not π_k. Then

$$p(k \mid \boldsymbol{x}) = \frac{\pi_k p_k(\boldsymbol{x})}{\sum_j \pi_j p_j(\boldsymbol{x})}$$

$$q(k \mid \boldsymbol{x}) = \frac{\psi_k p_k(\boldsymbol{x})}{\sum_j \psi_j p_j(\boldsymbol{x})}$$

using Bayes' lemma, so

$$p(k \mid \boldsymbol{x}) \propto q(k \mid \boldsymbol{x}) \pi_k / \psi_k$$

and $\sum_k p(k \mid \boldsymbol{x}) = 1$ fixes the constant. Thus

$$p(k \mid \boldsymbol{x}) = \frac{q(k \mid \boldsymbol{x}) \pi_k / \psi_k}{\sum_j q(j \mid \boldsymbol{x}) \pi_j / \psi_j}$$

Estimating the $p(k \mid \boldsymbol{x})$

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

$$\operatorname{logit} p(2 \mid \boldsymbol{x}) = \log \tfrac{p(2 \mid \boldsymbol{x})}{p(1 \mid \boldsymbol{x})}$$

In *logistic discrimination* we assume this is linear ($= \alpha + \boldsymbol{\beta}^T \boldsymbol{x}$) and fit α and $\boldsymbol{\beta}$ by maximum likelihood, that is 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 sometimes known as 'entropy minimization'.

Think of this in two stages: 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 linear output units.

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 $\boldsymbol{\beta}_1 = 0$ since only ratios of probabilities matter (as they sum to one).

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

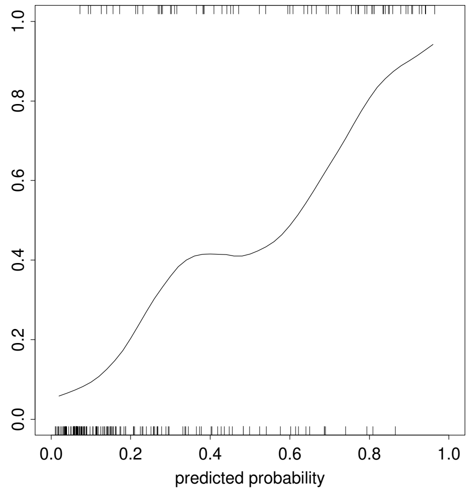
Now approximate the function $f\colon \mathcal{X} \to \mathbb{R}^K$ by a K-output neural net with linear outputs. This is sometimes known as *softmax*. It is not the procedure with K logistic output units, when the outputs are not guaranteed to sum to one.

Calibration

How will we know if we have estimated the $p(k \mid \boldsymbol{x})$ well?

One answer is by the performance of our rule.

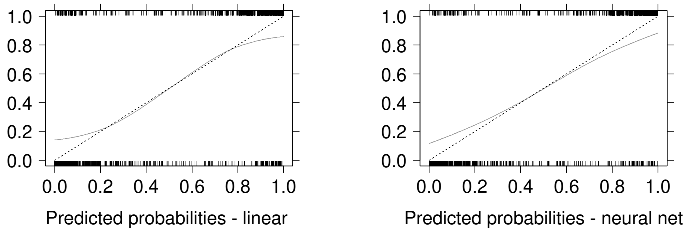
Another is that the probability forecasts are *well-calibrated*; about 30% of the events we say have probability about 0.3 should happen.



Calibration plot for a two-class medical diagnosis problem.

Do this on the validation set. Smooth curve is a non-linear curve-fitting problem!

More often we see something like:



Breast cancer – predict years to relapse as 0, 1, 2, 3, 4+.

in which the extreme probabilities (near zero or one) are estimated as too extreme.

The reason is that we are not using $p(k \mid \boldsymbol{x})$ but $\widehat{p}(k, \mid \boldsymbol{x}; \widehat{\boldsymbol{w}})$, a *plug-in* estimate in which we assume that the fitted weights $\widehat{\boldsymbol{w}}$ are the true weights (and that the neural network represents the true posterior probabilities). We are ignoring the uncertainty in the $\widehat{\boldsymbol{w}}$.

If necessary, we can use such graphs to recalibrate, but usually they indicate a deeper problem.

The predictive approach

Theory says we should use $P(C = c \mid \boldsymbol{X} = \boldsymbol{x}, \mathcal{T})$, where $\mathcal{T}$ denotes the training data. This is called the *predictive* approach. Now

$$P(c \mid \boldsymbol{x}, \mathcal{T}) \;=\; \int p(c \mid \boldsymbol{x}; \boldsymbol{w}) p(\boldsymbol{w} \mid \mathcal{T}, \boldsymbol{x}) \, \mathrm{d}\boldsymbol{w}$$

$$p(\boldsymbol{w} \mid \mathcal{T}, \boldsymbol{x}) \;\propto\; \prod_{i=1}^{n} p(c_i \mid \boldsymbol{x}_i; \boldsymbol{w}) p(\boldsymbol{x}_i; \boldsymbol{w}) \; p(\boldsymbol{w}) p(\boldsymbol{x}; \boldsymbol{w})$$

and we normally assume that $p(\boldsymbol{x}; \boldsymbol{w})$ does not in fact depend on $\boldsymbol{w}$ so

$$p(\boldsymbol{w} \mid \mathcal{T}) \propto p(\boldsymbol{w}) \prod_{i=1}^{n} p(c_i \mid \boldsymbol{x}_i; \boldsymbol{w})$$

There will be appreciable differences from the plug-in approach only if $p(c \mid \boldsymbol{x}; \boldsymbol{w})$ is not highly peaked *and* $p(c \mid \boldsymbol{x}; \boldsymbol{w})$ is not linear around the posterior mode for $\boldsymbol{w}$. Unfortunately, both are normally the case.

The predictive approach is normally computationally impossible, but various approximations have been proposed. For many purposes it is as good as any to average the $\widehat{p}(k \mid \boldsymbol{x}; \widehat{\boldsymbol{w}})$ over many fits of the network.

Early stopping and regularization

We are trying to replace an average over weights $\boldsymbol{w}$ by the predictions at a single set of weights $\widehat{\boldsymbol{w}}$. The best choice looks like the maximizer of

$$p(\boldsymbol{w} \mid \mathcal{T}) \propto p(\boldsymbol{w}) \prod_{i=1}^{n} p(c_i \mid \boldsymbol{x}_i; \boldsymbol{w})$$

This is not the maximum likelihood fit, as it has the term $p(\boldsymbol{w})$. *A priori* we expect a smooth function $p(k \mid \boldsymbol{x})$, so we are biased against large weights in the input-to-hidden layer (at least). A simple way to achieve this is by *weight decay*:

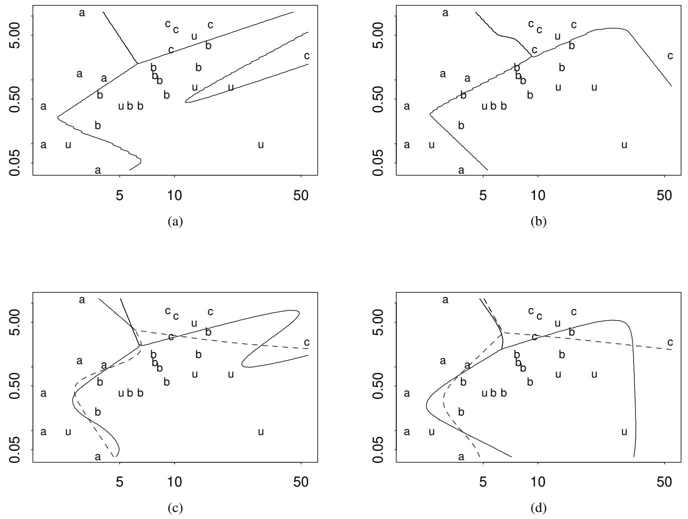
$$p(\boldsymbol{w}) \propto \exp -\lambda \|\boldsymbol{w}\|^2 = \exp -\lambda \sum_{ij} w_{ij}^2$$

Early stopping tests single $\boldsymbol{w}$ on the validation set, but there seem no rationale for taking the path from a starting point to the maximum likelihood estimator as the set of values $\boldsymbol{w}$ to test. In fact we will still need a validation set to choose the model, so should test on a fourth *stopping* set.

Computational effort seems best spent approximating the average over weights.

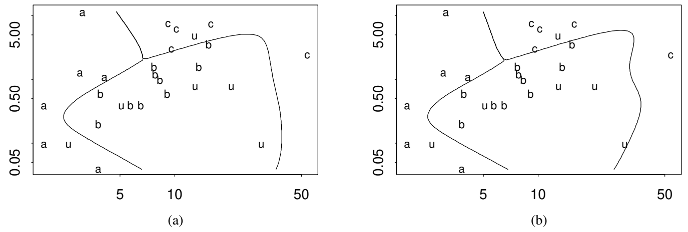
An example of estimation

Use the data on Cushing's syndrome.

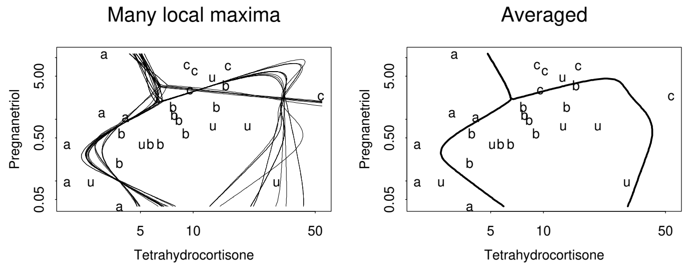


Neural network fits with two hidden units and connections from the input to output layer. Panels (a) and (b) are two solutions without weight decay which fit perfectly. Panels (c) and (d) show two local minima each, for $\lambda = 0.001$ and $\lambda = 0.01$ respectively. The dashed lines correspond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

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.

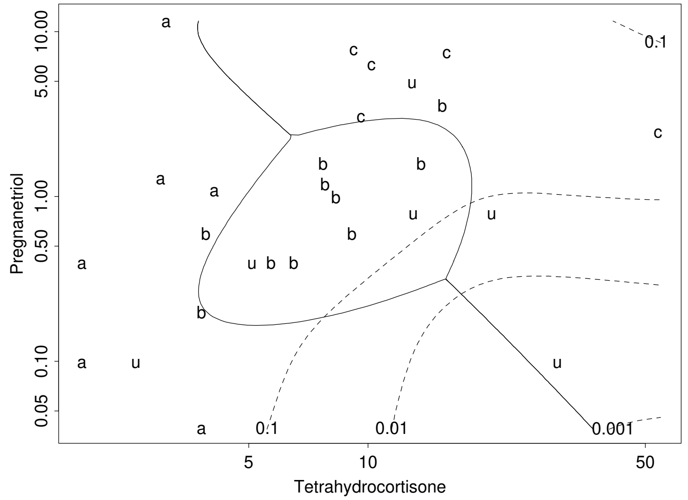


Neural networks with three hidden units and $\lambda = 0.01$ applied to the Cushing's syndrome data. The right panel shows their averaged prediction.

Outliers

In the strict theory 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).

Note that there is *no* way to say what an outlier is without giving an idea of their distribution. (Looks quite different on linear rather than log scales.)

Note too that the most common neural network approach does not model the class-conditional densities $p_k(x)$, so outliers are easily overlooked. They are much less obvious in 5 input dimensions, let alone 30!

Weighted training

We have seen that plug-in methods with regularization tend to estimate middling probabilities better that extreme ones. Consider data on an SGA[1] study, in which SGA babies are rare (10% by definition) and the cost of missing SGA is taken as $15\times$ that of a false positive. We will decide SGA when

$$p(\text{normal} \mid \boldsymbol{x}) < 15\, p(\text{SGA} \mid \boldsymbol{x})$$

as these are the two $\sum_{j \neq k} C_{jk} p(j \mid \boldsymbol{x})$. Now $p(\text{normal} \mid \boldsymbol{x}) + p(\text{SGA} \mid \boldsymbol{x}) = 1$, so we declare SGA if $p(\text{SGA} \mid \boldsymbol{x}) > 1/16$.

We will have trouble estimating $p(\text{SGA} \mid \boldsymbol{x})$ around $1/16$ by a plug-in rule. Suppose we randomly selected 1/10 th of the normal cases. We would then estimate $q(\text{SGA} \mid \boldsymbol{x}) = 11\, p(\text{SGA} \mid \boldsymbol{x})$ which we might predict quite accurately at around $11/16$.

Why waste 82% of the data? Instead give each normal case weight 1/10 th in the log-likelihood.

[1] Small for Gestational Age, of newborn babies.

Cross-validation

It seems wasteful to have to reserve both validation and test sets, and both have to be quite large to be much use. (For example, error counts between methods on the same validation set need to be at least 5 apart, and often much more, to show superiority.)

The idea of cross-validation dates from the early 1970's. Divide the available dataset into V pieces. In turn, keep one piece back for validation and use the rest of the data as the training set, then predict on the validation set. Compute the measure of performance on the complete set of predictions.

How do we choose V? Originally $V = n$ but

- Far too much computation.
- Too little difference in the training sets, so the cross-validated measure had a high variance.

In this context $V \approx 10$ seems better than the *bootstrap*. May even want to average over several 10–fold splits.

Caveat

(Cross-)validation applies to a well-defined procedure, so troubles with multiple local minima, early stopping and so on. Can just use a randomly-chosen solution throughout.

Cross-testing

It is also wasteful to keep back a large test set (although to ensure 'honesty' this is sometimes done). The idea of nested cross-validation again dates from the 1970's. Divide the whole dataset into T pieces, and keep each back in turn for testing, then use the rest for cross-validation. This fits $V \times T$ models (and each may need to average across local minima).

We can look at this another way. Suppose we use one of the T pieces for testing, one for validation and the remaining $T - 2$ pieces form the training set. (That is $V = T - 1$ and same divisions are used throughout). We then do $T(T - 1)$ fits, but it seem better to randomly subdivide to form the validation sub-pieces.

Missing values

Missing values are very common in medical applications (in the breast cancer example, there were 1227 patients, 709 of which had complete records) and often in financial ones. Further, which cases are complete is often correlated with other measurements (e.g. tumour type in the breast cancer data).

There are two halves to dealing with missing data:

1. *predict* class for cases with missing data
2. using incomplete cases in *training*

We consider each in turn, but if there are very few (or no) complete cases we have to solve the second before implementing the first. Remember that the complete cases may be atypical ones.

Prediction with missing data

Current approaches include:

- Fill in missing values by the overall mean (or median).
- Fill in missing values by a single close match (nearest neighbour imputation).
- Build separate models for each pattern of missingness.
- Choose special models to make the theory easy. (E.g. naïve Bayes, Gaussian mixture models.)
- Case splitting
- 'Multiple imputation'

If the posterior probabilities $p(k \mid \boldsymbol{x}_O, \boldsymbol{x}_U)$ depend heavily on the unobserved $\boldsymbol{x}_U$, the first two can be totally inadequate.

Building separate models is attractive if there are few patterns of missingness, but can be inconsistent.

Case splitting and multiple imputation are similar; both rely on averaging over the conditional distribution of $\boldsymbol{X}_U \mid \boldsymbol{X}_O$.

Multiple imputation

If missingness is uninformative (given the observed values) then we can use

$$p(k \mid \boldsymbol{x}_O) = E\big[p(k \mid \boldsymbol{x}_O, \boldsymbol{X}_U) \;\big|\; \boldsymbol{X}_O\big].$$

We can estimate this by sampling several (5–100×) from $\boldsymbol{X}_U \mid \boldsymbol{X}_O$, predicting using the imputed values and averaging the predicted posterior probabilities.

How do we find $\boldsymbol{X}_U \mid \boldsymbol{X}_O$? Another modelling problem with missing data!

Training with missing data

In principle this is easy: impute the missing values in the training set, average to estimate $p(k \mid \boldsymbol{x}_O; \boldsymbol{w})$ and use this in your likelihood. But

(a) Probably needs completely rewritten software.

(b) We have to know how to impute for all the patterns of missingness that occurred.

Case splitting is an approximation in which each case with missing features is replaced by fractional cases. This averages the $\log p(k \mid \boldsymbol{x}_O, \boldsymbol{X}_U)$ not the probabilities.

Imputation models

Sometimes the patterns of missingness are simple: in the breast cancer data mostly either EGFR or (ER, EGFR) are missing. Thus we can build two imputation models:

1. for ER given everything else but EGFR
2. for EGFR given the rest

and do a two-pass imputation (of ER then EGFR) if ER is also missing. (General idea is called *monotone* pattern of missingness.)

Another simple pattern is to have A or B but not both (e.g. destructive tests).

Can build general imputation models by cycling round sampling each missing variable given the rest. Now called the *Gibbs sampler* (but some of us did this in 1975). Can include parameters in the sampling.

Why no mention of the EM algorithm? We want to average: EM is an indirect way to maximize.

Assessment of performance

The ultimate measure of performance is the expected loss on future examples the *Bayes' risk* (or this is not pattern recognition!). With a large enough test set (or cross-testing) we should be able to give a good prediction of future performance.

How can we know if we have done well enough — would more data or a better neural-network program help? We can get some idea from lower bounds on the Bayes' risk, which measure the complexity of the problem. For example (PRNN, pp. 196ff):

1. Scale the input data so Euclidean distance is reasonable.
2. For each item in the whole dataset find the three nearest items (including itself), breaking ties at random.
3. Report 1/3 the frequency with precisely two neighbours of one class.

In the forensic glass example the best error rate was around 23.5%, and the Bayes' risk was estimated to be at least 14%.

In the breast cancer study with the binary outcome of three years free of relapse (or not) the best error rate was around 21%, and the lower bound was (again) 14%.

This suggests that in each study there is at best only a little more to be gained by using a better classifier.

Selecting network complexity

The only satisfactory way I know for general non-linear models of the posterior probabilities $p(k \mid \boldsymbol{x})$ is to test the performance (expected cost) on the validation (not test) set.

If the models are well-calibrated, can get a less variable measure of performance by averaging rather than counting, that is by using $\sum_{j \neq k} C_{jk} \widehat{p}(j \mid \boldsymbol{x})$ instead of C_{jk} for the chosen k. But some models will not be well-calibrated, and how can we test this?

If we have to fit a large number of models to select one, why not use them all in prediction? Theory tells us to average $\widehat{p}(k \mid \boldsymbol{x})$ across networks as well as across unknown weights. Stone (1974) suggested this, and using cross-validation to choose the weighting to give to networks. Rediscovered as Wolpert's (1992) *stacked generalization*.

Selecting input features

A special case of selecting network complexity, so the general principles apply.

Also:

- Impossible to do by looking at one or two features at a time. Suppose $p(k \mid \boldsymbol{x})$ really depends on the unmeasured z and

$$x_1 = z + \epsilon + \eta \qquad x_2 = z + \epsilon - \eta$$
$$x_3 = z - \epsilon + \eta \qquad x_4 = z - \epsilon - \eta$$

  for 'noise' vars ϵ and η of large variance (compared to the 'signal' z).

- For prediction it is better to shrink coefficients (weights) than to drop variables. Regularization will do this automatically.

- Feature selection is only useful when measurements are expensive, and then a cost-benefit analysis is needed.