Supervised learning II: the Bayesian approach

We now place supervised learning into a probabilistic setting by examining:

- The application of Bayes' theorem to the *supervised learning problem*.
- Priors, the likelihood, and the posterior probability *of a hypothesis*.
- The *maximum likelihood* and *maximum a posteriori* hypotheses, and some examples.
- *Bayesian decision theory*: minimising the error rate.
- Application of the approach to *neural networks*, using approximation techniques.

Reading

There is some relevant material to be found in *Russell and Norvig*, chapters 18 to 20, particularly in chapter 20, although the intersection between that material and what I will cover is small.

Almost all of what I cover can be found in:

- *Machine Learning*, by Tom Mitchell, McGraw Hill 1997, chapter 6.
- *Neural Networks for Pattern Recognition*, by Christopher M. Bishop, Oxford University Press 1995, chapter 1, sections 1.8, 1.9 and 1.10 and Chapter 10, introduction and sections 10.1, 10.2, 10.3 and 10.9.

Supervised learning: a quick reminder

We want to design a *classifier*, denoted $h(\mathbf{x})$

It should take an attribute vector

$$\mathbf{x} = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix}$$

and label it.

What we mean by *label* depends on whether we're doing *classification* or *regression*.

Supervised learning: a quick reminder

In *classification* we're assigning $\mathbf{x}$ to one of a set $\{\omega_1, \ldots, \omega_c\}$ of c *classes*.

For example, if $\mathbf{x}$ contains measurements taken from a patient then there might be three classes:

ω_1 = patient has disease
ω_2 = patient doesn't have disease
ω_3 = don't ask me buddy, I'm just a computer!

We'll often specialise to the case of two classes, denoted C_1 and C_2.

Supervised learning: a quick reminder

In *regression* we're assigning $\mathbf{x}$ to a *real number* $h(\mathbf{x}) \in \mathbb{R}$.

For example, if $\mathbf{x}$ contains measurements taken regarding today's weather then we might have

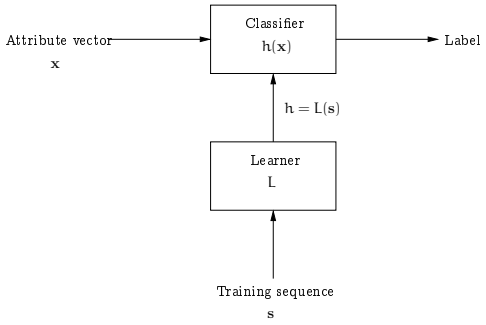
$$h(\mathbf{x}) = \text{estimate of amount of rainfall expected tomorrow}$$

For the two-class classification problem we will also refer to a situation somewhat between the two, where

$$h(\mathbf{x}) = \Pr(\mathbf{x} \text{ is in } C_1)$$

Supervised learning: a quick reminder

We don't want to design h explicitly.



So we use a *learner* L to infer it on the basis of a sequence $\mathbf{s}$ of *training examples*.

Supervised learning: a quick reminder

The *training sequence* $\mathbf{s}$ is a sequence of m *labelled examples*.

$$\mathbf{s} = \begin{pmatrix} (\mathbf{x}_1, y_1) \\ (\mathbf{x}_2, y_2) \\ \vdots \\ (\mathbf{x}_m, y_m) \end{pmatrix}$$

That is, examples of attribute vectors $\mathbf{x}$ with their correct label attached.

So a learner only gets to see the labels for a—most probably small—subset of the possible inputs $\mathbf{x}$.

Regardless, we aim that the hypothesis $h = L(\mathbf{s})$ will usually be successful at predicting the label of an input it hasn't seen before.

This ability is called *generalization*.

Supervised learning: a quick reminder

There is generally a set $\mathcal{H}$ of hypotheses from which L is allowed to select h

$$L(\mathbf{s}) = h \in \mathcal{H}$$

$\mathcal{H}$ is called the *hypothesis space*.

The learner can output a hypothesis explicitly or—as in the case of a multilayer perceptron—it can output a vector

$$\mathbf{w} = \begin{pmatrix} w_1 & w_2 & \cdots & w_W \end{pmatrix}$$

of *weights* which in turn specify h

$$h(\mathbf{x}) = f(\mathbf{w}; \mathbf{x})$$

where $\mathbf{w} = L(\mathbf{s})$.

Supervised learning: a quick reminder

In AI I you saw the *backpropagation algorithm* for training multi-layer perceptrons, in the case of *regression*.

This worked by minimizing a function of the weights representing the *error* currently being made:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m} (f(\mathbf{w}; \mathbf{x}_i) - y_i)^2$$

The summation here is over the training examples. The expression in the summation grows as fs prediction for $\mathbf{x}_i$ diverges from the known label y_i.

Backpropagation tries to find a $\mathbf{w}$ that minimises $E(\mathbf{w})$ by performing *gradient descent*

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \left.\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}_t}$$

Difficulties with classical neural networks

There are some well-known difficulties associated with neural network training of this kind.

Think of the process as follows:

- Nature picks an $h' \in \mathcal{H}$ but doesn't reveal it to us.
- Nature then shows us a training sequence $\mathbf{s}$ where each $\mathbf{x}_i$ is labelled as $h'(\mathbf{x}_i) + \epsilon_i$ where ϵ_i is noise of some kind.

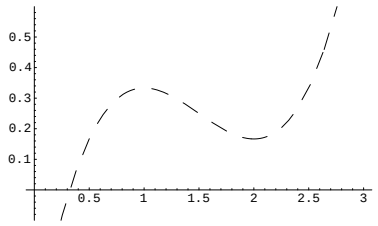
Our job is to try to infer what h' is on the basis of $\mathbf{s}$ only.

This is easy to visualise in one dimension: it's just fitting a curve to some points.

Difficulties with classical neural networks

For example, if $\mathcal{H}$ is the set of all polynomials of degree 3, then nature might pick

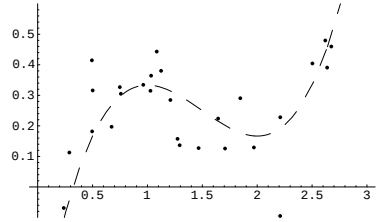
$$h'(x) = \frac{1}{3}x^3 - \frac{3}{2}x^2 + 2x - \frac{1}{2}$$



The line is dashed to emphasise the fact that *we don't get to see it*.

Difficulties with classical neural networks

We can now use h' to obtain a training sequence $\mathbf{s}$ in the manner suggested..



Here we have,

$$\mathbf{s} = ((x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m))$$

where each x_i and y_i is a number.

Difficulties with classical neural networks

Lets use the learning algorithm L that operates in exactly the same way as backpropagation: it picks an $h \in \mathcal{H}$ minimising the following quantity,

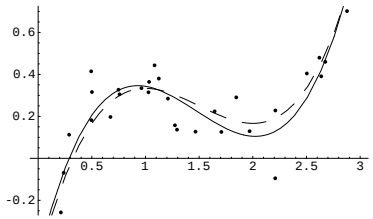
$$E = \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

In other words

$$h = L(s) = \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

Difficulties with classical neural networks

If we pick h using this method then we get:



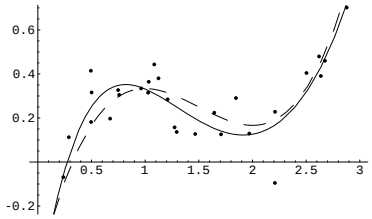
The chosen h is close to the target h', even though it was chosen using only a small number of noisy examples. It is not quite identical to the target concept; however, if we were given a new point x' and asked to guess the value $h'(x')$, then guessing $h(x')$ might be expected to do quite well.

Difficulties with classical neural networks

We don't know what $\mathcal{H}$ nature is using. What if the one we choose doesn't match? We can make *our* $\mathcal{H}$ 'bigger' by defining it as,

$$\mathcal{H} = \{h : h \text{ is a polynomial of degree at most } 5\}$$

If we use the same learning algorithm then we get:



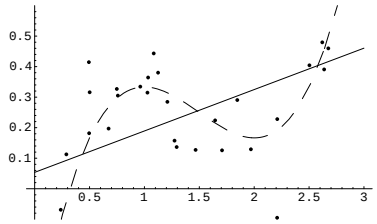
The result in this case is similar to the previous one: h is again quite close to h', but not quite identical.

Difficulties with classical neural networks

So what's the problem? Repeating the process with,

$$\mathcal{H} = \{h : h \text{ is a polynomial of degree at most } 1\}$$

gives the following:



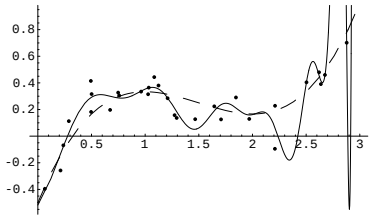
In effect, we have made *our* $\mathcal{H}$ too 'small'. It does not in fact contain any hypothesis similar to h'.

Difficulties with classical neural networks

So we have to make $\mathcal{H}$ huge, right? WRONG!!! With

$$\mathcal{H} = \{h : h \text{ is a polynomial of degree at most } 25\}$$

we get:



BEWARE!!! This is known as *overfitting*.

Difficulties with classical neural networks

An experiment to gain some further insight: using

$$h'(x) = \frac{1}{10}x^{10} - \frac{1}{12}x^8 + \frac{1}{15}x^6 + \frac{1}{3}x^3 - \frac{3}{2}x^2 + 2x - \frac{1}{2}.$$

as the unknown underlying function we can look at how the degree of the polynomial the training algorithm can output affects the generalization ability of the resulting h.

We use the same training algorithm, and we train using

$$\mathcal{H} = \{h : h \text{ is a polynomial of degree at most } d\}$$

for values of d ranging from 1 to 30

Difficulties with classical neural networks

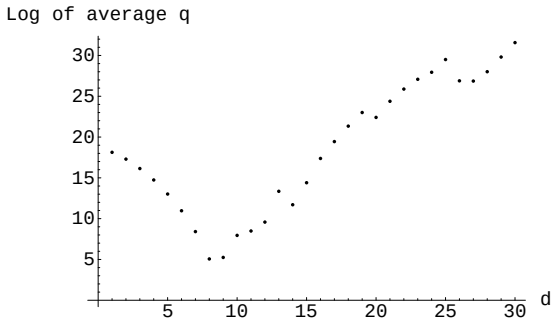
- Each time we obtain an h of a given degree—call it h_d—we assess its quality using a further 100 inputs x_i' generated at random and calculating

$$q(d) = \frac{1}{100}\sum_{i=1}^{100}(h'(x_i') - h_d(x_i'))^2$$

- As the values $q(d)$ are found using inputs that are not necessarily included in the training sequence *they measure generalization*.
- To smooth out the effects of the random selection of examples we repeat this process 100 times and average the values $q(d)$.

Difficulties with classical neural networks

Here is the result:



Clearly: we need to choose $\mathcal{H}$ sensibly if we want to obtain *good generalisation performance*.

Sources of uncertainty

So we have to be careful. But let's press on with this approach for a little while longer...

The model used above suggests two sources of uncertainty that we might treat with probabilities.

- Let's *assume* we've selected an $\mathcal{H}$ to use, *and it's the same one nature is using*.
- We don't know how nature chooses h' from $\mathcal{H}$. We therefore model our uncertainty by introducing the *prior* distribution $\Pr(h)$ on $\mathcal{H}$.
- There is noise on the training examples.

It's worth emphasizing at this point that in modelling noise on the training examples *we'll only consider noise on the labels*. The input vectors $\mathbf{x}$ are not modelled using a probability distribution.

The likelihood

We model our uncertainty in the training examples by specifying a *likelihood*:

$$\Pr(Y|h, \mathbf{x})$$

Translation: the probability of seeing a given label Y, when the input vector is $\mathbf{x}$ and the underlying hypothesis is h.

Example: two-class classification. A common likelihood is

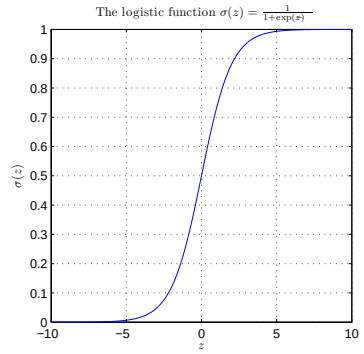
$$\Pr(Y = C_1|h, \mathbf{x}) = \sigma(h(\mathbf{x}))$$

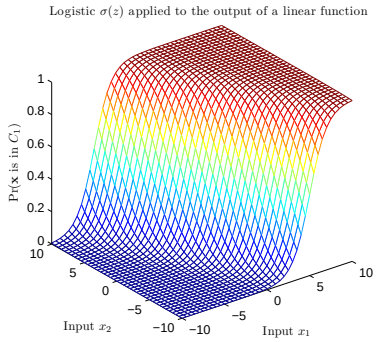
where

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

(Note: strictly speaking, $\mathbf{x}$ should not appear in these probabilities because it's not a random variable. It is included for clarity.)

The likelihood





The likelihood

So: if we're given a training sequence, *what is the probability that it was generated using some* h?

For an example $(\mathbf{x}, y)$, y can be C_1 or C_2. It's helpful here to rename the classes as just 1 and 0 respectively because this leads to a nice simple expression. Now

$$\Pr(Y|h, \mathbf{x}) = \begin{cases} \sigma(h(\mathbf{x})) & \text{if } Y = 1 \\ 1 - \sigma(h(\mathbf{x})) & \text{if } Y = 0 \end{cases}$$

Consequently *when y has a known value* we can write

$$\Pr(y|h, \mathbf{x}) = [\sigma(h(\mathbf{x}))]^y \, [1 - \sigma(h(\mathbf{x}))]^{(1-y)}$$

If we assume that the examples are independent then the probability of seeing the labels in a training sequence $\mathbf{s}$ is straightforward.

The likelihood

Collecting the inputs and outputs in $\mathbf{s}$ together into seperate matrices, so

$$\mathbf{y} = \begin{pmatrix} y_1 & y_2 & \cdots & y_m \end{pmatrix}$$

and

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \end{pmatrix}$$

we have the *likelihood of the training sequence*

$$\begin{aligned} \Pr(\mathbf{y}|h, \mathbf{X}) &= \prod_{i=1}^{m} \Pr(y_i|h, \mathbf{x}_i) \\ &= \prod_{i=1}^{m} [\sigma(h(\mathbf{x}_i))]^{y_i} [1 - \sigma(h(\mathbf{x}_i))]^{(1-y_i)} \end{aligned}$$

The likelihood

Another example: regression. A common likelihood works in the regression case by assuming that examples are corrupted by Gaussian noise with mean 0 and some specified variance σ^2

$$y = h(\mathbf{x}) + \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, \sigma^2)$$

As usual, the density for $\mathcal{N}(\mu, \sigma^2)$ is

$$p(Z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right)$$

by adding $h(\mathbf{x})$ to ϵ we just shift its mean, so

$$p(y|h, \mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - h(\mathbf{x}))^2}{2\sigma^2}\right)$$

The likelihood

Consequently if the examples are independent then the likelihood of a training sequence $\mathbf{s}$ is

$$\begin{aligned} p(\mathbf{y}|h, \mathbf{X}) &= \prod_{i=1}^{m} p(y_i|h, \mathbf{x}_i) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - h(\mathbf{x}_i))^2}{2\sigma^2}\right) \\ &= \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{m} (y_i - h(\mathbf{x}_i))^2\right) \end{aligned}$$

where we've used the fact that

$$\exp(a)\exp(b) = \exp(a + b)$$

Bayes' theorem appears once more...

Right: we've take care of the uncertainty by introducing the *prior* $p(h)$ and the *likelihood of the training sequence* $p(\mathbf{y}|h, \mathbf{X})$.

By this point you hopefully want to apply Bayes' theorem and write

$$p(h|\mathbf{y}) = \frac{p(\mathbf{y}|h)p(h)}{p(\mathbf{y})}$$

where

$$p(\mathbf{y}) = \sum_{h \in \mathcal{H}} p(h, \mathbf{y}) = \sum_{h \in \mathcal{H}} p(\mathbf{y}|h)p(h)$$

and to simplify the expression we have now dropped the mention of $\mathbf{X}$ as the inputs are fixed. $p(h|\mathbf{y})$ is called the *posterior distribution*.

The denominator $Z = p(\mathbf{y})$ is called the *evidence*, and leads on to fascinating issues of its own. Unfortunately, we won't have time to explore them.

Bayes' theorem appears once more...

The boxed equation on the last slide has a very simple interpretation: *what's the probability that this specific* h *was used to generate the training sequence I've been given?*

Two natural learning algorithms now present themselves:

1. The *maximum likelihood hypothesis*
$$h_{ML} = \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, p(\mathbf{y}|h)$$
2. The *maximum a posteriori hypothesis*
$$\begin{aligned} h_{MAP} &= \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, p(h|\mathbf{y}) \\ &= \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, p(\mathbf{y}|h)p(h) \end{aligned}$$

Obviously, h_{ML} corresponds to the case where the prior $p(h)$ is uniform.

Example: maximum likelihood learning

We derived an exact expression for the likelihood in the regression case above:

$$p(\mathbf{y}|h) = \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2\right)$$

Proposition: under the assumptions used, *any* learning algorithm that works by minimising the sum of squared errors on $\mathbf{s}$ finds h_{ML}.

This is clearly of interest: the notable example is the *backpropagation algorithm*.

We now prove the proposition...

Example: maximum likelihood learning

The proposition holds because:

$$\begin{aligned} h_{ML} &= \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, p(\mathbf{y}|h) \\ &= \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, \log p(\mathbf{y}|h) \\ &= \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, \log\left[\frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2\right)\right] \\ &= \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, \log\left[\frac{1}{(2\pi\sigma^2)^{m/2}}\right] - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2 \\ &= \underset{h \in \mathcal{H}}{\operatorname{argmax}} -\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2 \\ &= \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2 \end{aligned}$$

Example: maximum likelihood learning

Note:

- If the distribution of the noise is not Gaussian a different result is obtained.
- The use of log above to simplify a maximization problem is a standard trick.
- The Gaussian assumption is sometimes, but not always a good choice. (Beware the Central Limit Theorem!).

The next step...

We have so far concentrated throughout our coverage of machine learning on choosing a *single hypothesis*.

Are we asking the right question though?

Ultimately, we want to generalise.

That means being presented with a new $\mathbf{x}$ and asking the question: *what is the most probable classification of* $\mathbf{x}$*?*

Is it reasonable to expect a single hypothesis to provide the optimal answer?

We need to look at what the optimal solution to this kind of problem might be...

Bayesian decision theory

What is the *optimal* approach to this problem?

Put another way: how should we make decisions in such a way that the outcome obtained is, on average, the best possible? Say we have:

- Attribute vectors $\mathbf{x} \in \mathbb{R}^d$.
- A set of *classes* $\{\omega_1, \ldots, \omega_c\}$.
- Several possible *actions* $\{\alpha_1, \ldots, \alpha_a\}$.

The actions can be thought of as saying *"assign the vector to class 1"* and so on.

There is also a *loss* $\lambda(\alpha_i, \omega_j)$ associated with taking action α_i when the class is ω_j.

The loss will sometimes be abbreviated to $\lambda(\alpha_i, \omega_j) = \lambda_{ij}$.

Bayesian decision theory

Say we can also *model* the world as follows:

- Classes have probabilities $\Pr(\omega)$ of occurring.
- The probability of seeing $\mathbf{x}$ when the class is ω has density $p(\mathbf{x}|\omega)$.

Think of nature choosing classes at random (although not revealing them) and showing us a vector selected at random using $p(\mathbf{x}|\omega)$.

As usual Bayes rule tells us that

$$\Pr(\omega|\mathbf{x}) = \frac{p(\mathbf{x}|\omega)\Pr(\omega)}{p(\mathbf{x})}$$

and now the denominator is

$$p(\mathbf{x}) = \sum_{i=1}^{c} p(\mathbf{x}|\omega_i)\Pr(\omega_i).$$

Bayesian decision theory

Say nature shows us $\mathbf{x}$ and we take action α_i.

If we *always* take action α_i when we see $\mathbf{x}$ then the *average* loss on seeing $\mathbf{x}$ is

$$R(\alpha_i|\mathbf{x}) = \mathbb{E}_{\omega \sim p(\omega|\mathbf{x})}[\lambda_{ij}|\mathbf{x}] = \sum_{j=1}^{c} \lambda(\alpha_i, \omega_j)\Pr(\omega_j|\mathbf{x}).$$

The quantity $R(\alpha_i|\mathbf{x})$ is called the *conditional risk*.

Note that this particular $\mathbf{x}$ is *fixed*.

Bayesian decision theory

Now say we have a *decision rule* $\alpha : \mathbb{R}^d \to \{\alpha_1, \ldots, \alpha_a\}$ telling us what action to take on seeing *any* $\mathbf{x} \in \mathbb{R}^d$.

The average loss, or *risk*, is

$$\begin{aligned} R &= \mathbb{E}_{(\mathbf{x},\omega)\sim p(\mathbf{x},\omega)}\left[\lambda(\alpha(\mathbf{x}),\omega)\right] \\ &= \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\mathbb{E}_{\omega\sim \Pr(\omega|\mathbf{x})}\left[\lambda(\alpha(\mathbf{x}),\omega)|\mathbf{x}\right]\right] \\ &= \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[R(\alpha(\mathbf{x})|\mathbf{x})\right] \qquad (1) \\ &= \int R(\alpha(\mathbf{x})|\mathbf{x})p(\mathbf{x})d\mathbf{x} \end{aligned}$$

where we have used the standard result from probability theory that

$$\mathbb{E}\left[\mathbb{E}\left[X|Y\right]\right] = \mathbb{E}\left[X\right].$$

(See the supplementary notes for a proof.)

Bayesian decision theory

Clearly the risk is minimised for the decision rule defined as follows:

$$\alpha \textit{ outputs the action } \alpha_i \textit{ that minimises } R(\alpha_i|\mathbf{x}), \textit{ for all } \mathbf{x} \in \mathbb{R}^d.$$

The provides us with the minimum possible risk, or *Bayes risk* $R^\star$.

The rule specified is called the *Bayes decision rule*.

Example: minimum error rate classification

In supervised learning our aim is often to work in such a way that we *minimise the probability of error*.

What loss should we consider in these circumstances? From basic probability theory

$$\Pr(A) = \mathbb{E}\left[I(A)\right]$$

where

$$I(A) = \begin{cases} 1 & \text{if } A \text{ happens} \\ 0 & \text{otherwise} \end{cases}$$

(See the supplementary notes for a proof.)

Example: minimum error rate classification

So if we are addressing a supervised learning problem with c classes $\{\omega_1, \ldots, \omega_c\}$ and we interpret action α_i as meaning 'the input is in class ω_i', then a loss

$$\lambda_{ij} = \begin{cases} 1 & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

means that the risk R is

$$R = \mathbb{E}\left[\lambda\right] = \Pr(\alpha(\mathbf{x}) \text{ is in error})$$

and the Bayes decision rule minimises the probability of error.

Example: minimum error rate classification

Now, what is the Bayes decision rule?

$$\begin{aligned} R(\alpha_i|\mathbf{x}) &= \sum_{j=1}^{c} \lambda(\alpha_i, \omega_j) Pr(\omega_j|\mathbf{x}) \\ &= \sum_{i \neq j} Pr(\omega_j|\mathbf{x}) \\ &= 1 - Pr(\omega_i|\mathbf{x}) \end{aligned}$$

so $\alpha(\mathbf{x})$ should be *the class that maximises* $Pr(\omega_i|\mathbf{x})$.

THE IMPORTANT SUMMARY: Given a new $\mathbf{x}$ to classify, choosing the class that maximises $Pr(\omega_i|\mathbf{x})$ is the best strategy if your aim is to obtain the minimum error rate!