Imperial College London

Lecture 5: Binary classification

Introduction to machine learning

Kevin Webster

Department of Mathematics
Imperial College London

Outline

Outline

Classification task

We have already looked at the regression task, and how it can be tackled using linear regression models.

Classification is another type of supervised learning task, where we assume we are given a dataset consisting of N inputs

$$\mathbf{x} = (x_1, x_2, \ldots, x_N), \qquad x_i \in \mathbb{R}^d \quad \forall i$$

and N corresponding output values

$$\mathbf{y} = (y_1, y_2, \ldots, y_N), \qquad y_i \in \mathcal{C} \quad \forall i$$

where $\mathcal{C}$ is a finite set of *classes* or *categories* that each data point belongs to.

Classification task

We wish to find a function f that models the relationship between $\mathbf{x}$ and $\mathbf{y}$. Specifically, given an input data point, we want to predict the correct class for that data point. We assume the classes are mutually exclusive; i.e. every data point belongs to exactly one class.

As in the linear regression case, f is often a parametric function, depending on parameters $\boldsymbol{\theta}$.

Note the difference from this setting and the regression task is the type of output data—for regression we have numerical data, whereas classification uses categorical data.

Binary classification is a particular case of the classification problem where there are just two classes, $\mathcal{C}_1$ and $\mathcal{C}_2$. In this case, the output values y_i are typically assumed to take values in $\{0, 1\}$ or $\{-1, +1\}$.

Classification algorithms

There are many algorithms that have been developed for classification problems:

- Perceptron
- Logistic regression
- Naive Bayes classifier
- Decision trees, random forests
- k-NN classifier
- Support vector machines
- Artificial neural networks
- ...

We will look at the perceptron algorithm, logistic regression and a first look at neural networks.

Outline

Perceptron

The perceptron algorithm is a simple linear classification algorithm invented by Frank Rosenblatt[1]. It is an important algorithm in the development of neural networks.

The research was funded by the United States Office of Naval Research, and the algorithm was originally intended for image classification.

Although initially promising, it was quickly realised that there are many patterns that the perceptron could not recognise. These limitations were removed with the development of more sophisticated algorithms such as neural networks.

[1] Rosenblatt, Frank (1957), The Perceptron–a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory

The perceptron classification rule

Let $x \in \mathbb{R}^d$ be an input data point, with corresponding binary output $y \in \{-1, +1\}$. We denote the k-th dimension of x as $x^{(k)}$.

The perceptron has $k + 1$ weights $w_0, w_1, \ldots, w_d$. We define the function $f : \mathbb{R}^d \mapsto \mathbb{R}$ as

$$f(x) = w_0 + \sum_{k=1}^{d} w_k x^{(k)}.$$

The perceptron class prediction for the input x is then given by sign $f(x)$. That is, the model prediction $\hat{y}$ is given by

$$\hat{y} = \begin{cases} +1 & \text{if } f(x) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

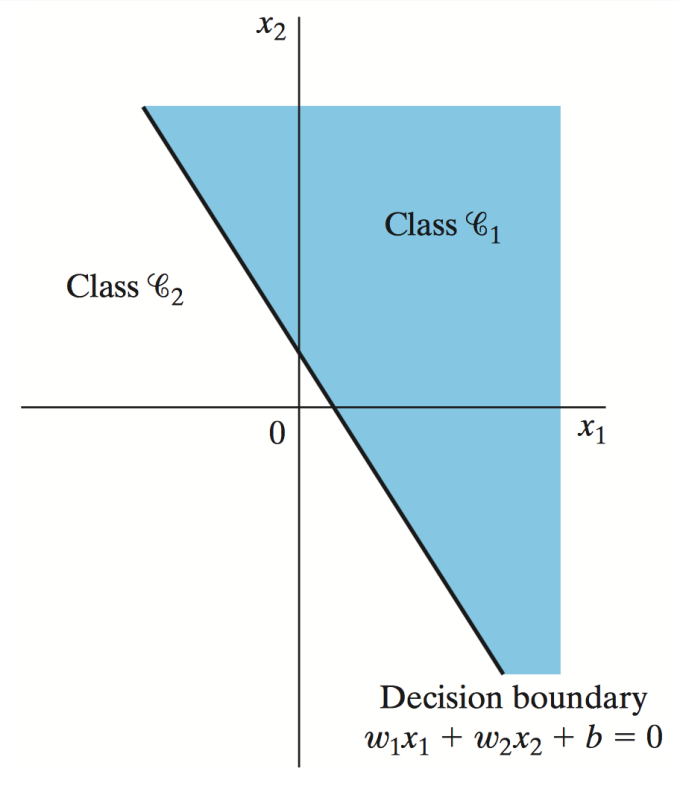
The perceptron classification rule

The perceptron is built around the concept of a *mathematical neuron*. It uses the synaptic weights w_i to compute a linear combination of the inputs, followed by a hard limiter.

Note that similar to linear regression, we can add the constant dummy feature $x^{(0)} = 1$ and denote $w = [w_0, w_1, \ldots, w_d]$. Then the function f can be written more concisely as $f(x) = \langle w, x \rangle$.

The perceptron classification rule

The perceptron classification rule can be thought of as a decision boundary that is a separating hyperplane in the data space:



Training the perceptron

Recall that we have the training data $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_N)$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$.

The perceptron algorithm has the parameters $w_0, w_1, \ldots, w_d$. We need to find the best setting for these parameters according to the training data.

We find these parameters by training the perceptron algorithm according to a learning rule. This is an iterative rule for updating the weight parameters, resulting in a sequence of updates $w(t)$, $t \in \mathbb{N}$.

Perceptron learning rule

The learning rule can be summarised as follows:

1. Initialise the parameters $w(0)$ to zero, or to small random values
2. For each example (x_i, y_i) in the training set:
   2.1 Calculate the model prediction $\hat{y}_i = \text{sign}\, f(x_i)$
   2.2 Update the weights according to

$$w_j(t+1) = w_j(t) + \eta(t)\frac{(y_i - \hat{y}_i)}{2}x_i^{(j)}, \quad j = 0, \ldots, d$$

3. Repeat step 2 until the error $\sum_{i=1}^{N} |y_i - \hat{y}_i|$ is less than a user-defined threshold

In the above, $\eta(t)$ is a learning rate that is used to control the adjustment to the weights at iteration t.

Perceptron learning rule

- The update rule implies that the weights are only updated for data examples that the perceptron classifier incorrectly labels
- The update rule can be equivalently written as

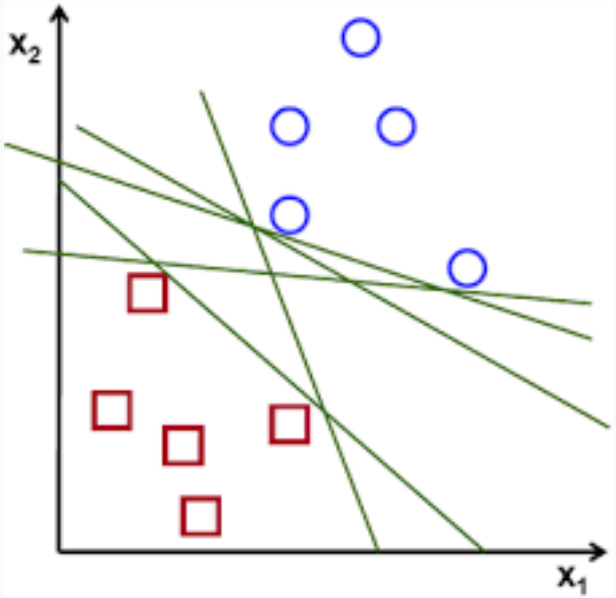
$$w_j(t+1) = \begin{cases} w_j(t) + \eta(t) y_i x_i^{(j)}, & y_i \langle w(t), x_i \rangle \leq 0 \\ w_j(t) & \text{otherwise} \end{cases}$$

  for all $j = 0, \ldots, d$
- If the perceptron labels all data points correctly, the algorithm terminates in finite time
- If $\eta(t) = \eta$ for all iterations, then we have a fixed-rate adaptation rule
- For fixed learning rates, we can arbitrarily choose $\eta = 1$, since any rescaling simply rescales the weights but does not change the sign of the prediction

Linearly separable data

It is clear that the algorithm can only correctly label all examples if the data is *linearly separable*. Precisely, that means there exists a $\gamma > 0$ and $w^* \in \mathbb{R}^{d+1}$ such that $y_i \langle w^*, x_i \rangle > \gamma$ for all $i = 1, \ldots, N$.

In addition, if a separating hyperplane does exist, then it is not unique and there will be many possible solutions for the parameters w.



Perceptron convergence theorem

Theorem
Suppose the data $(x_i, y_i)_{i=1}^N$ *is linearly separable. Then the perceptron learning rule with a constant learning rate* η *converges to a solution that defines a separating hyperplane in a finite number of iterations.*

In the above theorem, note from our earlier observation that we can assume without loss of generality that $\eta = 1$ by just rescaling the weight vector.

Perceptron convergence theorem: proof

Proof. Let w^* be a solution that defines a separating hyperplane, and so have $y_i \langle w^*, x_i \rangle > \gamma$ for all $i = 1, \ldots, N$, for some $\gamma > 0$. We assume the parameters are initialised to zero.

The proof of the convergence theorem will proceed in two stages:

1. we will show that $\langle w^*, w(t) \rangle$ increases at least linearly with each update
2. the squared norm $||w(t)||^2$ increases at most linearly in the number of updates k.

This in turn will show that the cosine of the angle between w^* and $w(t)$ increases by at least a finite amount with each update, and therefore we can only make a finite number of updates.

Perceptron convergence theorem: proof

First part:

When the parameter vector $w(t)$ is updated, we have

$$w(t+1) = w(t) + y_i x_i$$

where we have assumed w.l.o.g. that the learning rate $\eta = 1$. Taking the inner product with w^* on both sides gives

$$\begin{aligned} \langle w^*, w(t+1) \rangle &= \langle w^*, w(t) \rangle + y_i \langle w^*, x_i \rangle \\ &\geq \langle w^*, w(t) \rangle + \gamma \end{aligned}$$

Then after the parameter vector $w(t)$ has been updated k times, we have

$$\langle w^*, w(t) \rangle \geq k\gamma. \tag{1}$$

Perceptron convergence theorem: proof

Second part:

Again, when the parameter vector $w(t)$ is updated:

$$\begin{aligned} ||w(t+1)||^2 &= ||w(t) + y_i x_i||^2 \\ &= ||w(t)||^2 + ||y_i x_i||^2 + 2y_i \langle w(t), x_i \rangle \\ &\leq ||w(t)||^2 + ||x_i||^2 \\ &\leq ||w(t)||^2 + R^2 \end{aligned}$$

In the above we have used the fact that $y_i \langle w(t), x_i \rangle \leq 0$ at an update, and chosen a constant R such that $||x_i|| \leq R$ for all i, which is trivially true as the dataset is finite. Then it follows that after k updates, we have

$$||w(t)||^2 \leq kR^2 \tag{2}$$

Perceptron convergence theorem: proof

Let θ denote the angle between the vectors w^* and $w(t)$ (after k updates). We now can combine (1) and (2) to show

$$\begin{aligned} 1 \quad \geq \quad \cos\theta \quad &= \quad \frac{\langle w^*, w(t)\rangle}{||w^*|| \cdot ||w(t)||} \\ &\geq \quad \frac{k\gamma}{||w^*|| \cdot ||w(t)||} \qquad \text{(by (1))} \\ &\geq \quad \frac{k\gamma}{||w^*||\sqrt{k}R} \qquad \text{(by (2))} \end{aligned}$$

We can rewrite the above as

$$k \leq \frac{R^2||w^*||^2}{\gamma^2}$$

which proves the number of updates is finite. □

Outline

Probabilistic models

- The perceptron algorithm outputs class labels for a given input
- This does not account for any uncertainty in the predictions, and so we turn to a probabilistic approach for the classification problem
- We have again the training data $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_N)$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$
- We use the convention $y_i = 1$ if x_i belongs to class $\mathcal{C}_1$, and vice versa
- We also recall the distinction between *discriminative* and *generative* modelling approaches:
  1. The discriminative approach aims to model the conditional probability $p(\mathcal{C}_k|x)$ directly
  2. The generative approach models the class-conditional densities $p(x|\mathcal{C}_k)$ as well as the class priors $p(\mathcal{C}_k)$, and uses these to compute posterior probabilities $p(\mathcal{C}_k|x)$ with Bayes' theorem

Probabilistic generative models

We will first consider the generative modelling approach to binary classification, and see how this leads to a form of the conditional probability $p(\mathcal{C}_k|x)$ that is a generalised linear model under certain assumptions on the class-conditional densities $p(x|\mathcal{C}_k)$.

To begin, recall that from Bayes' theorem we have

$$p(\mathcal{C}_k|x) = \frac{p(x|\mathcal{C}_k)p(\mathcal{C}_k)}{p(x)}, \qquad k = 1, 2$$

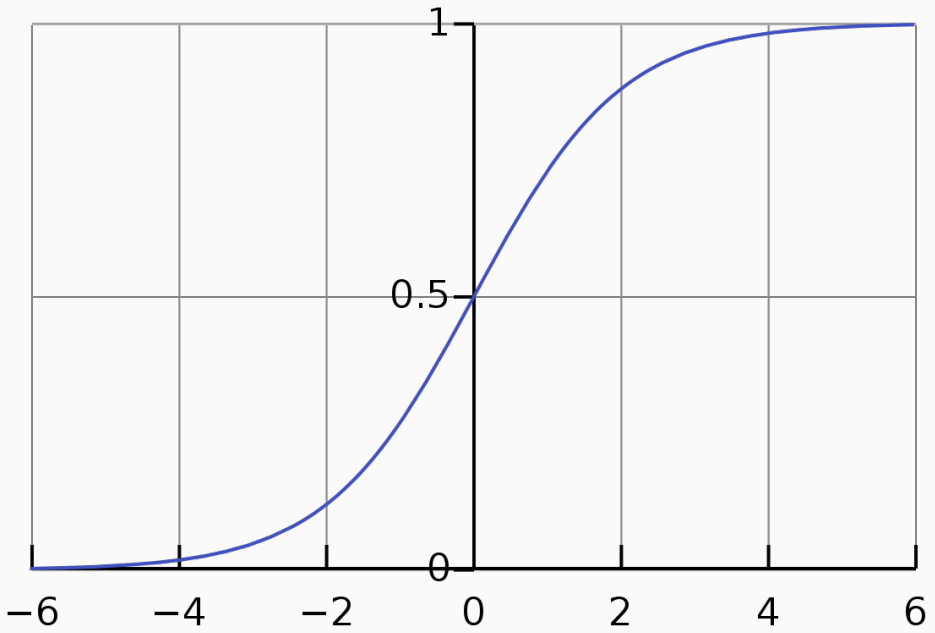
where $p(x) = p(x|\mathcal{C}_1)p(\mathcal{C}_1) + p(x|\mathcal{C}_2)p(\mathcal{C}_2)$.

Logistic sigmoid

We define the **logistic sigmoid** function $\sigma : \mathbb{R} \mapsto (0, 1)$ as

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

This type of function is commonly referred to as a 'squashing function' since it maps the real line into the interval $(0, 1)$. The sigmoid plays an important role in many machine learning algorithms.



Logistic sigmoid: properties

- The logistic sigmoid function has the following symmetry property:

$$\sigma(-x) = 1 - \sigma(x)$$

- The inverse is given by

$$x = \log\left(\frac{\sigma}{1-\sigma}\right)$$

  The inverse is known as the *logit* function

- The derivative is given by

$$\frac{d\sigma}{dx} = \sigma(x)(1 - \sigma(x))$$

Logistic sigmoid

If we define the log-odds

$$a := \log \frac{p(\mathcal{C}_1|x)}{p(\mathcal{C}_2|x)} = \log \frac{p(x|\mathcal{C}_1)p(\mathcal{C}_1)}{p(x|\mathcal{C}_2)p(\mathcal{C}_2)}$$

then it is straightforward to show that

$$\begin{aligned} p(\mathcal{C}_1|x) &= \frac{p(x|\mathcal{C}_1)p(\mathcal{C}_1)}{p(x|\mathcal{C}_1)p(\mathcal{C}_1) + p(x|\mathcal{C}_2)p(\mathcal{C}_2)} \\ &= \sigma(a) \end{aligned}$$

We will now look at two cases where the functional form of the class-conditional density $p(x|\mathcal{C}_k)$ leads to a being a linear function of x.

Continuous input x

In the case where the inputs are continuous, suppose we assume the class-conditional densities are Gaussian, with a shared covariance matrix:

$$p(x|\mathcal{C}_k) \sim \mathcal{N}(x|\mu_k, \Sigma)$$

Substituting the multivariate Gaussian pdf into the expression for the posterior distribution $p(\mathcal{C}_1|x)$ above, we get

$$p(\mathcal{C}_1|x) = \sigma(w^T x + w_0)$$

where

$$\begin{aligned} w &= \Sigma^{-1}(\mu_1 - \mu_2) \\ w_0 &= -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \log\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)} \end{aligned}$$

Note the argument to the sigmoid function is linear in x.

Parameter estimation

WIth the class-conditional densities $p(x|\mathcal{C}_k)$ specified, we can estimate their parameter values using maximum likelihood.

Our model has the parameters μ_1, μ_2, Σ and the prior class probability $p(\mathcal{C}_1) = \pi$.

Recall that the likelihood function can be thought of as the probability of the data given the model parameters:

$$\mathcal{L}(\mu_1, \mu_2, \Sigma, \pi|\mathbf{x}, \mathbf{y}) = p(\mathbf{x}, \mathbf{y}|\mu_1, \mu_2, \Sigma, \pi)$$

The maximum likelihood solution is given by

$$(\mu_1, \mu_2, \Sigma, \pi)_{ML} = \underset{\mu_1, \mu_2, \Sigma, \pi}{\arg\max}\, \mathcal{L}(\mu_1, \mu_2, \Sigma, \pi|\mathbf{x}, \mathbf{y})$$

Maximum likelihood solution

Under the usual assumption of i.i.d. data, we have

$$p(\mathbf{x}, \mathbf{y}|\mu_1, \mu_2, \Sigma, \pi) = \prod_{i=1}^{N} p(x_i|\mathcal{C}_i)p(\mathcal{C}_i)$$

where $\mathcal{C}_i \in \{\mathcal{C}_1, \mathcal{C}_2\}$ is the class that x_i belongs to. Recall that we have defined $y_i = 1$ if $\mathcal{C}_i = \mathcal{C}_1$, and $y_i = 0$ if $\mathcal{C}_i = \mathcal{C}_2$. Then we can write

$$p(\mathbf{x}, \mathbf{y}|\mu_1, \mu_2, \Sigma, \pi) = \prod_{i=1}^{N} [\pi N(x_i|\mu_1, \Sigma)]^{y_i} [(1-\pi)N(x_i|\mu_2, \Sigma)]^{1-y_i}$$

where $N(x|\mu, \Sigma)$ denotes the (multivariate) Gaussian pdf.

Maximum likelihood solution

As before, it is convenient to work with the log-likelihood function—recall the the arg max of the likelihood function is the same as for the log-likelihood.

The log-likelihood function is given by

$$\begin{aligned}\log p(\mathbf{x}, \mathbf{y}|\mu_1, \mu_2, \Sigma, \pi) &= \sum_{i=1}^{N} \{y_i \log[\pi N(x_i|\mu_1, \Sigma)] \\ &\quad + (1 - y_i) \log[(1 - \pi) N(x_i|\mu_2, \Sigma)]\}\end{aligned}$$

We want to find the parameters that maximise this function.

Maximum likelihood solution: π

Consider first the class prior probability π (defined by $p(\mathcal{C}_1) = \pi$). The terms in the log-likelihood that depend on π are

$$\mathcal{L}(\pi|\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} y_i \log \pi + (1 - y_i) \log(1 - \pi) + \text{const.}$$

Setting the derivative with respect to π equal to zero, we obtain:

$$\begin{aligned} \pi_{ML} &= \frac{1}{N} \sum_{i=1}^{N} y_i \\ &= \frac{N_1}{N} = \frac{N_1}{N_1 + N_2} \end{aligned}$$

where N_1, N_2 denote the number of examples in class $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. Thus the maximum likelihood solution π_{ML} is just the proportion of data points in class $\mathcal{C}_1$.

Maximum likelihood solution: μ_1 and μ_2

Similarly, we can see the terms in the log-likelihood function that depend on μ_1 are

$$\mathcal{L}(\mu_1|\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N} y_i (x_i - \mu_1)^T \Sigma^{-1} (x_i - \mu_1) + \text{const.}$$

This time, setting the derivative with respect to μ_1 to zero gives

$$\mu_{1,ML} = \frac{1}{N_1} \sum_{i=1}^{N} y_i x_i$$

A similar argument also shows $\mu_{2,ML} = \frac{1}{N_2} \sum_{i=1}^{N} (1 - y_i) x_i$.

Thus the maximum likelihood solutions $\mu_{1,ML}$, $\mu_{2,ML}$ are just the means of the input vectors belonging to the corresponding classes.

Maximum likelihood solution: Σ

Finally, consider the shared covariance matrix Σ:

$$\begin{aligned}
\mathcal{L}(\Sigma|\mathbf{x},\mathbf{y}) &= -\frac{1}{2}\sum_{i=1}^{N} y_i \log|\Sigma| - \frac{1}{2}\sum_{i=1}^{N} y_i (x_i-\mu_{1,ML})^T\Sigma^{-1}(x_i-\mu_{1,ML}) \\
&\quad -\frac{1}{2}\sum_{i=1}^{N}(1-y_i)\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{N}(1-y_i)(x_i-\mu_{2,ML})^T\Sigma^{-1}(x_i-\mu_{2,ML}) \\
&\quad + \text{ const.} \\
&= -\frac{N}{2}\log|\Sigma| - \frac{N}{2}\,\mathrm{Tr}\{\Sigma^{-1}S\}
\end{aligned}$$

where

$$\begin{aligned}
S &= \frac{N_1}{N}S_1 + \frac{N_2}{N}S_2 \\
S_k &= \frac{1}{N_k}\sum_{i\in\mathcal{C}_k}(x_i-\mu_{k,ML})(x_i-\mu_{k,ML})^T, \qquad k=1,2.
\end{aligned}$$

Continuous input x: summary

We have seen that for continuous inputs x, a particular choice of model for the class-conditional density $p(x|\mathcal{C}_k)$ (multivariate Gaussian density) leads to the posterior class probabilities $p(\mathcal{C}_k|x)$ being given by generalised linear models with logistic sigmoid activation

$$p(\mathcal{C}_k|x) = \sigma(w^T x + w_0)$$

where the parameters $w \in \mathbb{R}^d$ and $w_0 \in \mathbb{R}$ are functions of the model parameters $\mu_1, \mu_2, \Sigma, \pi$.

We have also shown how the model parameters $\mu_1, \mu_2, \Sigma, \pi$ can be estimated using maximum likelihood estimation.

We now show how a similar result holds under certain modelling assumptions for discrete inputs.

Discrete input x

Now assume the inputs x are discrete; in particular we assume binary feature values $x^{(j)} \in \{0, 1\}$ for $j = 1, \ldots, d$.

The number of possible inputs is 2^d, and a general model of the discrete distribution $p(x|\mathcal{C}_k)$ would then have $2^d - 1$ parameters (due to the constraint that the probabilities must sum to one).

Since this is exponential in d, we make the simplifying modelling assumption that the feature values are independent given the class:

$$p(x|\mathcal{C}_k) = \prod_{j=1}^{d} \mu_{kj}^{x^{(j)}} (1 - \mu_{kj})^{1-x^{(j)}}$$

This is known as the **naive Bayes** assumption.

Discrete input x

Now we have

$$\log[p(x|\mathcal{C}_k)p(\mathcal{C}_k)] = \sum_{j=1}^{d}\{x^{(j)}\log\mu_{kj} + (1-x^{(j)})\log(1-\mu_{kj})\} + \log p(\mathcal{C}_k)$$

and using our earlier result that

$$p(\mathcal{C}_1|x) = \sigma(a)$$

where

$$a = \log\frac{p(x|\mathcal{C}_1)p(\mathcal{C}_1)}{p(x|\mathcal{C}_2)p(\mathcal{C}_2)}$$

we see that again our posterior class probabilities are generalised linear models of the form $p(\mathcal{C}_k|x) = \sigma(w^T x + w_0)$, where the argument of the logistic sigmoid is linear in the input features.

Discrete input x

Similar to the continuous input model, we can estimate the generative model parameters μ_{kj}, $k = 1, 2$, $j = 1, \ldots, d$ using maximum likelihood estimation.

The likelihood function in this case is given by

$$\begin{aligned}\mathcal{L}(\boldsymbol{\mu}, \pi | \mathbf{x}, \mathbf{y}) \quad = \quad & \sum_{i=1}^{N} \Bigg\{ y_i \sum_{j=1}^{d} \{ x^{(j)} \log \mu_{1j} + (1 - x^{(j)}) \log(1 - \mu_{1j}) \} + \log \pi \\ & + (1 - y_i) \sum_{j=1}^{d} \{ x^{(j)} \log \mu_{2j} + (1 - x^{(j)}) \log(1 - \mu_{2j}) \} + \log(1 - \pi) \Bigg\}\end{aligned}$$

Once the model parameters have been estimated, we can compute the posterior class probabilities $p(\mathcal{C}_1 | x)$ to make predictions for new inputs.

Outline

Probabilistic discriminative model

- In our generative approach, we modelled the class-conditional densities $p(x|\mathcal{C}_k)$ and class priors $p(\mathcal{C}_k)$
- The model parameters were estimated using maximum likelihood estimation
- We then used Bayes' theorem to compute the posterior class distributions, which we can then use to make predictions on test inputs x^*
- We have seen that for certain modelling assumptions on the class-conditional densities $p(x|\mathcal{C}_k)$ that the posterior class distributions take the form of the generalised linear model

$$p(\mathcal{C}_k|x) = \sigma(w^T x + w_0) \tag{3}$$

Probabilistic discriminative model

- An alternative approach is to not model the class-conditional density $p(x|\mathcal{C}_k)$ at all, and assume the functional form of a generalised linear model (3) directly
- In this case, we are likely to have fewer parameters to estimate—we now only have w and w_0
- We also do not make any assumption about the nature of the class-conditional input density
- In practice, this often leads to improved performance when the model density assumption leads to a poor approximation to the true density
- This is the *discriminative* approach—we aim to model $p(\mathcal{C}_k|x)$ directly

Basis functions

With our generalised linear modelling assumption that we have taken with the discriminative modelling approach, we can make use of increased representational power through the use of nonlinear basis functions $\{\phi_m\}_{m=1}^M$.

In this case, we have the feature vector

$$\boldsymbol{\phi}(x) := [\phi_1(x), \phi_2(x), \ldots, \phi_M(x)]$$

Notice the similarity to linear regression. We also often choose one of the basis functions to be constant, say $\phi_1(x) = 1$.

In the case of logistic regression, nonlinear basis functions mean that the decision boundary can be nonlinear.

Logistic regression

- With the increased flexibility obtained by using basis functions, the posterior class probability can be written as a logistic sigmoid activation function with an argument that is linear in the model parameters:

$$p(\mathcal{C}_1|\phi(x)) = \sigma(w^T\phi(x))$$

  and we have $p(\mathcal{C}_2|\phi(x)) = 1 - p(\mathcal{C}_1|\phi(x))$

- For a logistic regression model with M basis functions, there are M parameters to estimate
- Contrast this with a Gaussian class conditional density generative model:
  - $2M$ parameters for the means μ_1, μ_2
  - $M(M+1)/2$ parameters for the shared covariance matrix
  - 1 parameter for the class prior $p(\mathcal{C}_1) = \pi$

  $\Rightarrow$ total of $M(M+5)/2 + 1$ parameters

Likelihood function

We will again use maximum likelihood to determine the parameters of the logistic regression model.

The likelihood function is given by

$$\begin{aligned}
\mathcal{L}(w|\mathbf{x},\mathbf{y}) &= p(\mathbf{y}|\mathbf{x},w) \\
&= \prod_{i=1}^{N} p(\mathcal{C}_1|w^T\phi(x_i))^{y_i} p(\mathcal{C}_2|w^T\phi(x_i))^{1-y_i} \\
&= \prod_{i=1}^{N} \sigma(w^T\phi(x_i))^{y_i} (1-\sigma(w^T\phi(x_i)))^{1-y_i}
\end{aligned}$$

Cross entropy loss

As with linear regression, we define an error function by taking the negative logarithm of the likelihood, which gives

$$-\log \mathcal{L}(w|\mathbf{x}, \mathbf{y}) = -\sum_{i=1}^{N} \left\{ y_i \log \sigma(w^T \phi(x_i)) + (1 - y_i) \log(1 - \sigma(w^T \phi(x_i))) \right\}$$

If we denote $\hat{y}_i := \sigma(w^T \phi(x_i))$ as the model probability for the data point x_i to belong to class $\mathcal{C}_1$, we can write the negative log likelihood more concisely as

$$L(w) := -\log \mathcal{L}(w|\mathbf{x}, \mathbf{y}) = -\sum_{i=1}^{N} \{ y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \}$$

The above is the **binary cross entropy** loss function.

Cross entropy loss

We would like to minimise the cross entropy loss. Taking the derivative, we get

$$\begin{aligned}
\frac{dL}{dw} &= -\sum_{i=1}^{N}\left\{\frac{y_i}{\sigma}\sigma'\phi(x_i) - \frac{(1-y_i)}{(1-\sigma)}\sigma'\phi(x_i)\right\} \\
&= -\sum_{i=1}^{N}\left\{\frac{y_i}{\sigma}\sigma(1-\sigma)\phi(x_i) - \frac{(1-y_i)}{(1-\sigma)}\sigma(1-\sigma)\phi(x_i)\right\} \\
&= -\sum_{i=1}^{N}\{y_i(1-\sigma)\phi(x_i) - (1-y_i)\sigma\phi(x_i)\} \\
&= \sum_{i=1}^{N}(\sigma - y_i)\phi(x_i)
\end{aligned}$$

In the above we have denoted $\sigma := \sigma(w^T\phi(x_i))$ for brevity, and used the derivative property of the logistic sigmoid function.

Parameter estimation

- Unfortunately, there is no closed-form available for the solution of the logistic regression problem as was the case for linear regression
- However, it can be shown that the error function is in fact convex, and so has a unique minimum
- To approximate the minimum, we need to resort to gradient-based methods, such as gradient descent
- Note that with maximum likelihood estimation there is no regularisation, and if the classes are linearly separable in feature space, then severe over-fitting can occur as $||w||$ will tend to infinity
- This can be avoided by inclusion of a prior and seeking the MAP solution for w, or equivalently adding regularisation to the cross entropy loss

Gradient descent

We compute the gradient averaged over the dataset:

$$G(w) = \frac{1}{N} \sum_{i=1}^{N} (\sigma(w^T \phi(x_i)) - y_i)\phi(x_i)$$

To approximate the weight parameters using gradient descent:

1. Initialise the weights w_0 to zero, or small random values
2. Until convergence, repeat for each step $t = 0, 1, \ldots$:
   2.1 Compute $G(w_t)$
   2.2 Update the weights:

$$w_{t+1} = w_t - \eta G(w_t)$$

In the above, $\eta > 0$ is a learning rate that is selected by the user.

Stochastic gradient descent

Gradient descent requires processing the entire dataset to make one weight update, and this can be inefficient.

Stochastic gradient descent (SGD) updates the weights with noisy but unbiased estimates of the true gradient.

To perform SGD updates, the full dataset is broken up into *minibatches* $\{\mathcal{B}_j\}_{j=1}^{M}$ and the gradient is estimated on a minibatch:

1. Initialise the weights w_0 to zero, or small random values
2. Until convergence, repeat for each step $t = 0, 1, \ldots$:
   2.1 Compute the estimated gradient
   $$\hat{G}(w_t) = \frac{1}{|\mathcal{B}_j|} \sum_{x_i, y_i \in \mathcal{B}_j} (\sigma(w_t^T \phi(x_i)) - y_i)\phi(x_i)$$
   2.2 Update the weights:
   $$w_{t+1} = w_t - \eta \hat{G}(w_t)$$

Outline

Logistic regression neuron

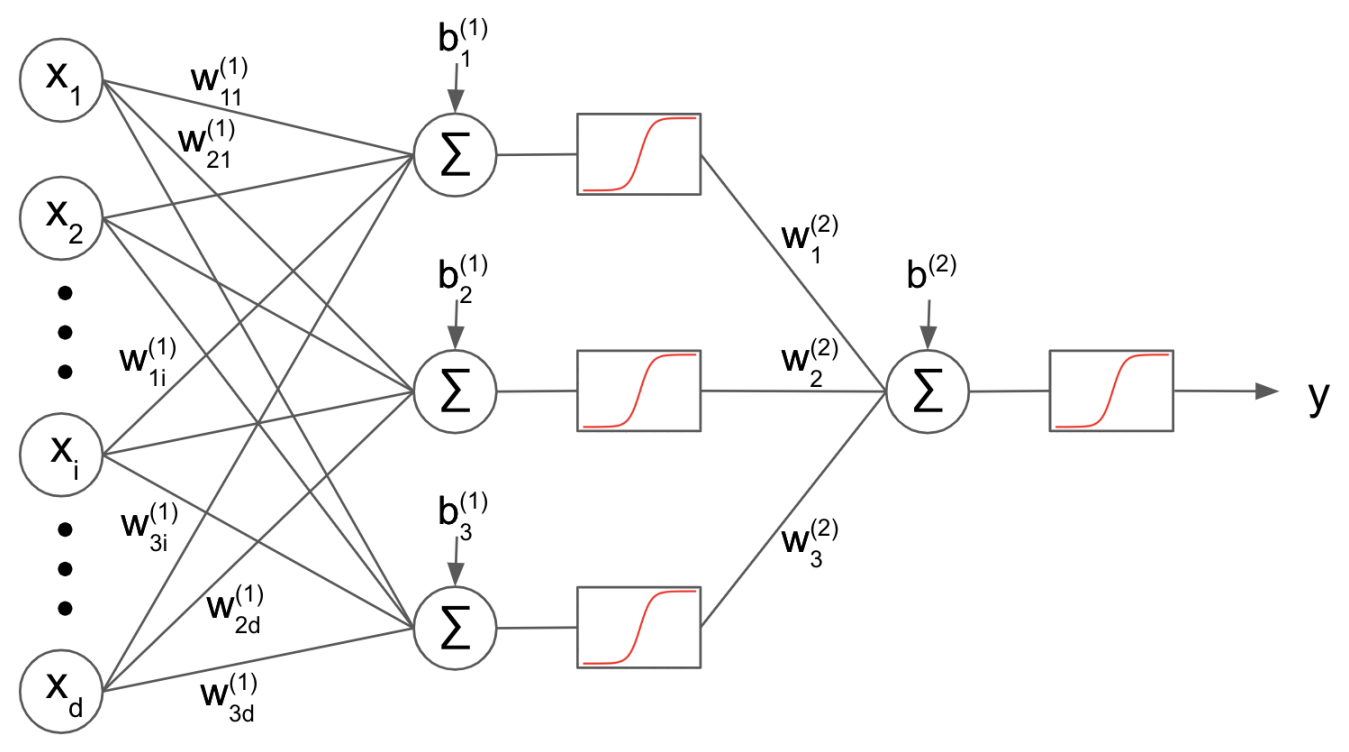
We have already seen the concept of a mathematical neuron in the perceptron classification rule.

In the perceptron, the activation function is a step function, meaning the perceptron is a 'hard classifier'.

The logistic regression classifier can also be viewed as a neuron. This time, the activation function is the logistic sigmoid function. The output y is interpreted as the probability that the input x belongs to class $\mathcal{C}_1$.

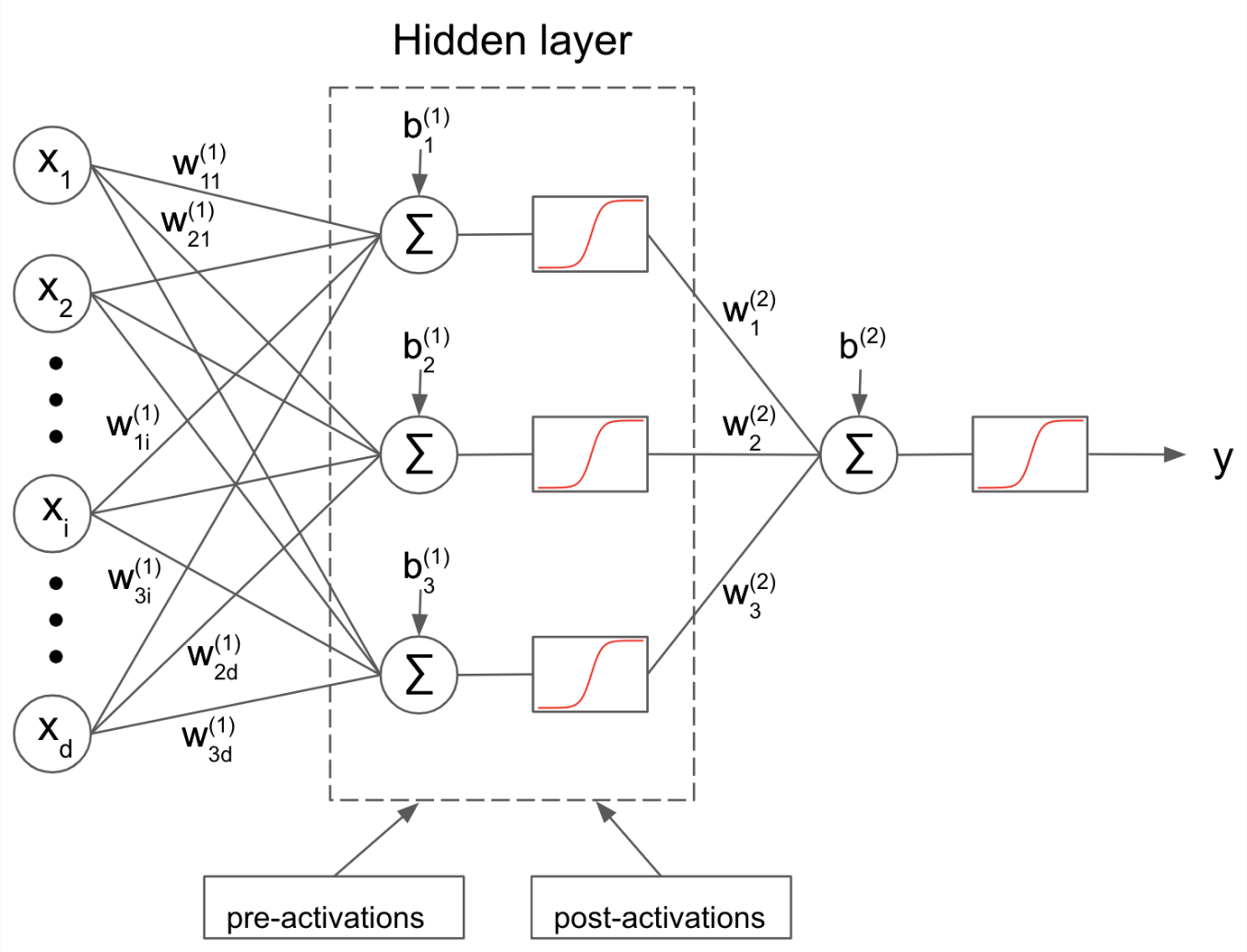
Artificial neural network

We previously increased the representational power of logistic regression using basis functions. An artificial neural network increases the flexibility by creating an intermediate layer of neurons, the outputs of which feed into the final output neuron:



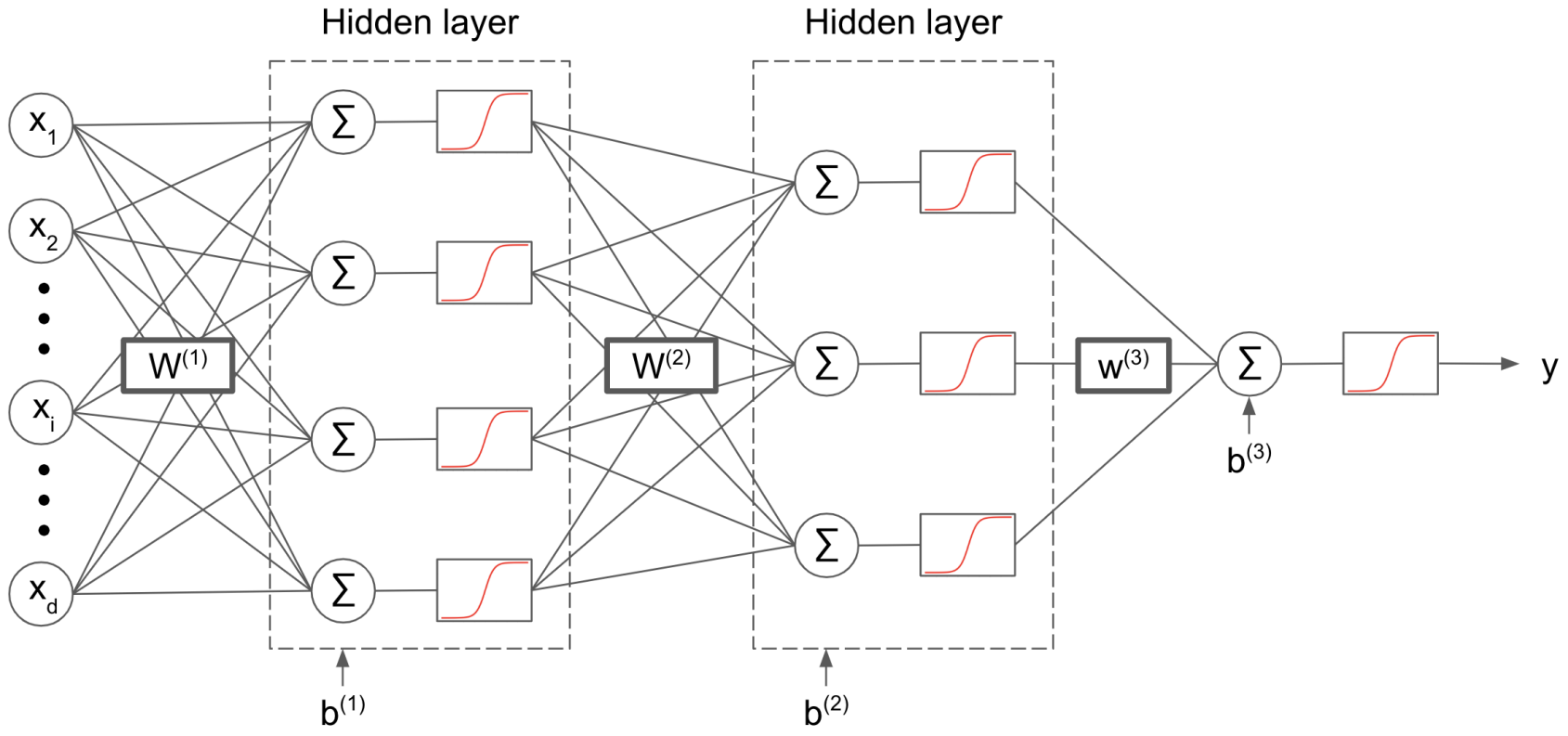
Artificial neural network

This intermediate layer of neurons is known as a **hidden layer**:



Artificial neural network

In addition, hidden layers can be stacked up to create deeper networks.



Additional hidden layers increase the flexibility of the function approximation for the distribution $p(\mathcal{C}_1|x)$.