

# COMS30035, Machine learning:

From regression to classification  
and neural networks:

Classification and neural networks

Rui Ponte Costa

Department of Computer Science, SCEEM  
University of Bristol

December 15, 2022

# Textbooks

We will follow parts of the Chapter 4 and 5 of the Bishop book:

- ▶ Bishop, C. M., Pattern recognition and machine learning (2006). Available for free [here](#).

# Agenda

- ▶ Revising linear and nonlinear regression [[see SPS slides](#); Chapter 3, Bishop]
  - ▶ Linear regression
  - ▶ Nonlinear regression
  - ▶ Probabilistic models
  - ▶ Maximum likelihood estimation
- ▶ Sequential Bayesian regression [Chapter 3, Bishop]
  - ▶ Bayesian formulation
  - ▶ Conjugate priors
  - ▶ Example
- ▶ **Classification and neural networks** [Chapter 5, Bishop]
  - ▶ Discriminant functions
  - ▶ Logistic regression
  - ▶ Perceptron
  - ▶ Neural networks (multi-layer perceptron)
    - ▶ Architecture
    - ▶ The backpropagation algorithm
    - ▶ Gradient descent

# Classification

- ▶ It is the classical example of **supervised learning**
- ▶ Goal: Classify input data into one of  $N$  classes
- ▶ Model: *Discriminant function*:
  - ▶ It is a function that takes an input vector  $x$  and assigns it to one of  $K$  classes, denoted  $C_k$ . For simplicity we will focus on  $K = 2$  and will first study linear functions (see the general cases in Bishop).

# Linear discriminant function

- ▶ The simplest linear discriminant (LD) is  $y(x) = w_0 + \mathbf{w}^T \mathbf{x}$ 
  - ▶ where  $y$  is used to predicted class  $C_k$ ,  $x$  the input vector (features)
  - ▶  $w_0$  is a scalar, which we call *bias*
  - ▶  $w_T$  is our vector of parameters, which we call *weights*
- ▶ For  $K = 2$ : An input vector  $x$  is assigned to class  $C_1$  if  $y(x) \geq 0$  and to class  $C_2$  otherwise.
- ▶ Optimisation: We can use a least-square approach similar to what we used for regression <sup>1</sup>, where we want to minimise the cost or error function:

- ▶  $E = \frac{1}{2} \sum_{n=1}^N (\mathbf{w}^T \mathbf{x}_n + w_0 - t_n)^2$  where  $t_n$  are the targets/labels (e.g.  $t_1 = C_1$ ).

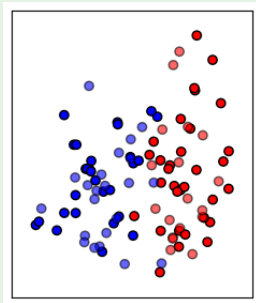
---

<sup>1</sup>See Bishop p184 and p190.

# LD and linear separability

## Example

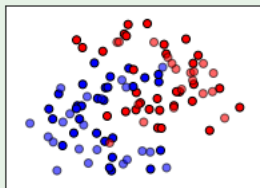
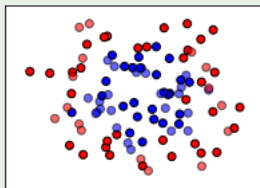
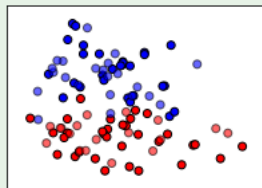
*Linear separability* is when two sets of points are separable by a line. We generated two sets of points using two Gaussians to illustrate this point, which can easily be fit by a LD. A *decision boundary* is the boundary that separates the two given classes, which our models will try to find.



# Linear separability vs nonlinear separability

## Example

Which datasets **are** and **are not** linearly separable<sup>2</sup>?



Only the first dataset is linearly separable!

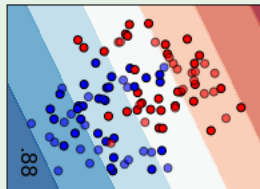
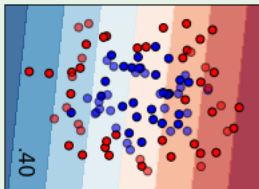
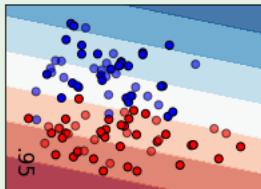
---

<sup>2</sup>Example from Sklearn here.

# Linear discriminant

## Example

Using sklearn we fitted a LD to the data:



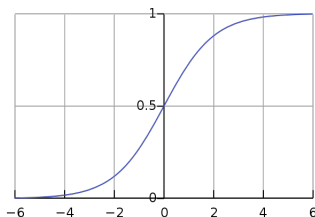
As expected, the LD model only does a good job in finding a good separation in the first dataset.



# Logistic regression

- ▶ We fit a logistic function to the data to perform classification:

$y(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$  where  $\sigma$  denotes the logistic sigmoid function (s-shaped), for example:



- ▶ such that when  $y \rightarrow 0$  we get class 1 and  $y \rightarrow 1$  class 2.
- ▶ We will take a probabilistic formalist and use  $p(C_1|\mathbf{x}) = y(\mathbf{x})$ , with  $p(C_2|\mathbf{x}) = 1 - p(C_1|\mathbf{x})$ .

# Logistic regression – maximum likelihood estimation

Follow MLE recipe:

1. Define likelihood: For a data set  $\{x_n, t_n\}$ , where the targets  $t_n \in \{0, 1\}$

we have  $p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^N y_n^{t_n} (1 - y_n)^{1-t_n}$  where  $y_n = p(C_1|x_n)$ <sup>3</sup>

2. Take negative logarithm of the likelihood:

$$-\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$
<sup>4</sup>

3. Calculate the derivative w.r.t. the parameters  $\mathbf{w}$ :

$$\frac{d \ln p(\mathbf{t}|\mathbf{w})}{d\mathbf{w}} = \sum_{n=1}^N (y_n - t_n) x_n$$
<sup>5</sup>

4. Now we can use Eq. above to directly update  $\mathbf{w}$  using the data  $x$ .

---

<sup>3</sup>Note that the exponent switches from first to second class as needed (i.e. if  $t_n = 1$  we get  $y_n$ ; if  $t_n = 0$  we get  $1 - y_n$ ).

<sup>4</sup>Note that we used the logarithm product and power rule.

<sup>5</sup>This solution makes sense since we want to optimise the difference between the model output  $y$  and the desired targets  $t$ .

# Logistic regression – maximum likelihood estimation

More details on calculating the derivative:

1. From here  $-\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$

2. We get  $\sum_{n=1}^N \left\{ -\frac{t_n}{y_n} + \frac{(1-t_n)}{1-y_n} \right\} \{y_n(1 - y_n)\} x_n$ <sup>6</sup>

3. The above simplifies to  $\sum_{n=1}^N \{-t_n(1 - y_n) + (1 - t_n)y_n\} x_n$

4. And in turn to  $\sum_{n=1}^N \{y_n - t_n\} x_n$ <sup>7</sup>

---

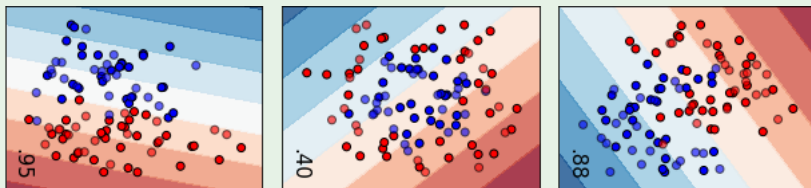
<sup>6</sup>We used the chain rule and  $d \ln(x) = 1/x$ . We also used the derivative of the sigmoid  $dy_n = y(1 - y_n)$ .

<sup>7</sup>You can find the full derivation here.

# Logistic regression

## Example

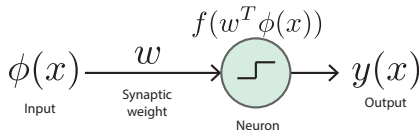
Using sklearn we fitted a logistic regression classifier to the data:



As you can see the results are very similar to LD, but because of probabilistic formulation we have an explicit probability of belonging to one or the other class (not shown); this can be very useful in real-world applications (e.g. self-driving cars or cancer detection).

# Perceptron – a simplified neural network

- ▶ It is the very beginning of neural network models in ML!
- ▶ It is directly inspired on how neurons process information:



- ▶ It is an example of a linear discriminant model given by  $y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x}))$

with a nonlinear *activation function*  $f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$

- ▶ Here the target  $t = \{+1, -1\}$ .

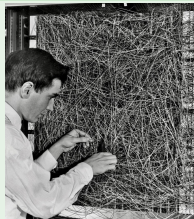
- ▶ And we aim to minimise the following error  $-\sum_{n=1}^N \mathbf{w}^T \phi_n t_n$ <sup>8</sup>

---

<sup>8</sup>Intuitively we want to improve our chances of having  $t_n = y_n = -1$  or  $t_n = y_n = 1$ , which will both decrease our error function.

# Perceptron – a simplified neural network

## Example



### The Perceptron of Rosenblatt (1962)

The perceptron played an important role in the history of machine learning (Rosenblatt 1962). Indeed it represents the very start of the current *deep learning* revolution. Frank Rosenblatt used IBM and special-purpose hardware for a parallel implementation of perceptron learning. Rosenblatt's work was criticized by Marvin Minsky, who showed that such models could only learn *linearly separable problems*. However, this limitation is only true in the case of single layers!

source: Bishop p193.

## Quiz and video time!



Watch this very cool video about the perceptron <sup>9</sup>.

Go to Blackboard unit page » Quizzes » Lecture 3.3

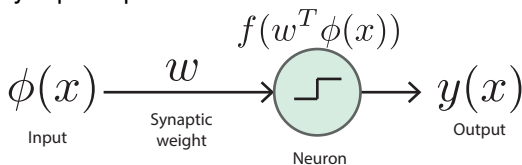
[Should take you less than 5 minutes]

---

<sup>9</sup>Note the comment at the end – it underlies all the recent successes using deep learning!

# Neural networks

From a single layer perceptron:

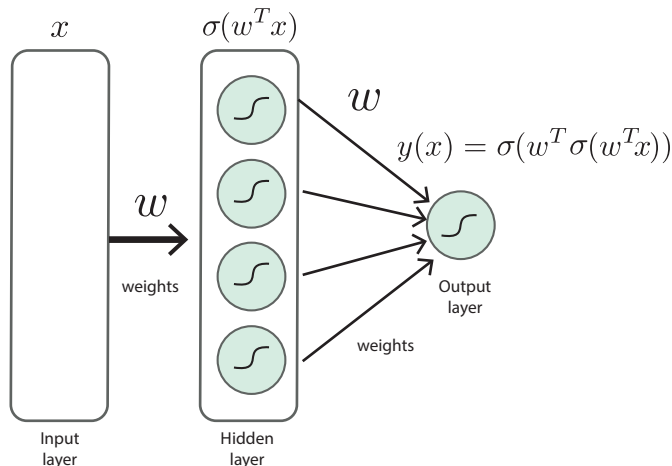


However these and other linear (or near-linear) models have limited expressibility due to the *curse of dimensionality*.



# Neural networks

To a Multiple Layer Perceptron (MLP) <sup>10</sup>:



<sup>10</sup>Although, we call it perceptron, it typically uses logistic sigmoid activation functions (continuous nonlinearities), instead of step-wise discontinuous nonlinearities.

# Neural networks

- ▶ Neural networks are at heart composite functions of linear-nonlinear functions.
- ▶ **Deep learning**<sup>11</sup> refers to neural networks (or MLPs) with more than 1 hidden layer
- ▶ They can be applied in any form of learning, but we will focus on supervised learning and classification in particular
- ▶ MLP recipe <sup>12</sup>:
  - ▶ Define architecture (e.g. how many hidden layers and neurons) <sup>13</sup>
  - ▶ Define cost function (e.g. mean squared error)
  - ▶ Optimise network using backprop:
    1. Forward pass – calculate activations; generate  $y_k$
    2. Calculate error/cost function
    3. Backward pass – use backprop to update parameters

---

<sup>11</sup>If you would like to learn more take our Applied Deep Learning unit in your 4th year.

<sup>12</sup>Here we focus on simple feedforward nnets but the recipe is the same for any neural network.

<sup>13</sup>Note that this makes them parametric models.

# Neural networks – forward pass step-by-step

1. Calculate activations of the hidden layer  $h$ :  $a_j = \sum_{i=1}^D w_{ji} x_i^{(h)} + w_{j0}^{(h)}$  [linear]
2. Pass it through a nonlinear function:  $z_j = \sigma(a_j)$  [nonlinear<sup>14</sup>]
3. Calculate activations of the output layer  $o$ :  $a_k = \sum_{j=1}^D w_{kj} z_j^{(o)} + w_{k0}^{(o)}$  [linear]
4. Compute predictions using a sigmoid:  $y_k = \sigma(a_k)$  [nonlinear<sup>15</sup>]
5. All together:  $y_k = \sigma \left( \sum_{j=1}^D w_{kj} \sigma \left( \sum_{i=1}^D w_{ji} x_i^{(h)} + w_{j0}^{(h)} \right) + w_{k0}^{(o)} \right)$

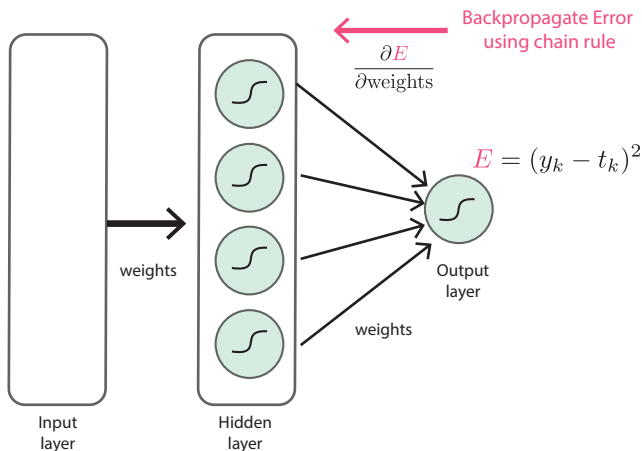
---

<sup>14</sup>In MLP we typically use sigmoid functions.

<sup>15</sup>For classification problems we use a sigmoid at the output, where each output neuron codes for one class.

# Neural networks – backward pass

We now need to optimise our weights, and as before we use derivatives to find a solution. Effectively backpropagating the output error signal across the network – backpropagation algorithm.



# Neural networks – backward pass

We now need to optimise our weights, and as before we use derivatives to find a solution. Effectively backpropagating the output error signal across the network – backpropagation algorithm.

1. Compute the error (or cost) function: e.g.:  $E = \frac{1}{2} \sum_{n=1}^N (\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n)^2$

2. Use the *chain rule* to compute the gradients w.r.t.  $\mathbf{w}$ ,  $\frac{dE}{d\mathbf{w}}$

3. For the output weights  $w_{kj}$  we get:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial \sigma} \frac{\partial \sigma}{\partial w_{kj}} = \sigma'(y_n - t_n) z_j^{16}$$

4. Whereas for the input weights  $w_{ji}$  we get:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial \sigma} \frac{\partial \sigma}{\partial z_j} \frac{\partial z_j}{\partial \sigma} \frac{\partial \sigma}{\partial w_{ji}} = \sigma'(y_n - t_n) w_{kj}^T \sigma' x_i^{17}$$

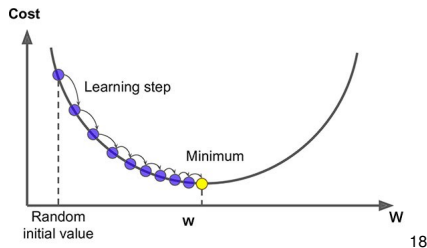
---

<sup>16</sup>  $\sigma'$  denotes the derivative of the sigmoid activation function.

<sup>17</sup> Note that the updates for the bias terms  $w_0$  do not depend on the activity of the previous layer  $z_j$  and  $x_i$ .

# Neural networks – gradient descent <sup>19</sup>

In many ML methods is common to iteratively update the parameters by descending the gradient.



In our neural network this means to update the weights using:

- ▶  $w_{ji} = w_{ji} - \Delta w_{ji}$ , where  $\Delta w_{ji} = \sigma'(y_n - t_n) w_{kj}^T \sigma' x_i$
- ▶  $w_{kj} = w_{kj} - \Delta w_{kj}$ , where  $\Delta w_{kj} = \sigma'(y_n - t_n) z_j$
- ▶ This is often done in mini-batches – using a small number of samples to compute  $\Delta w$ .

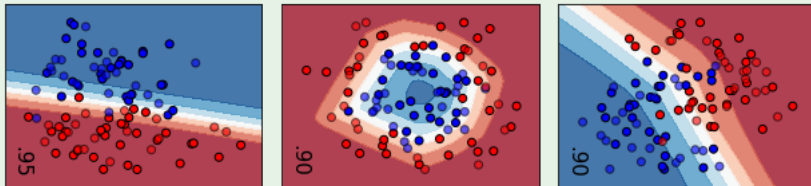
<sup>18</sup> Figure from <https://mc.ai/an-introduction-to-gradient-descent-2/>

<sup>19</sup> Its called descent because we are minimising the cost function, so descending on the function landscape, which can be quite hilly!

# Neural networks

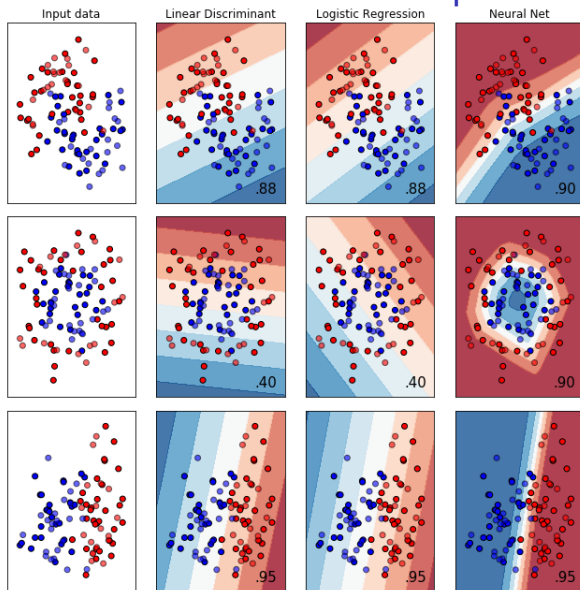
## Example

Using sklearn we fitted a MLP classifier to the data:



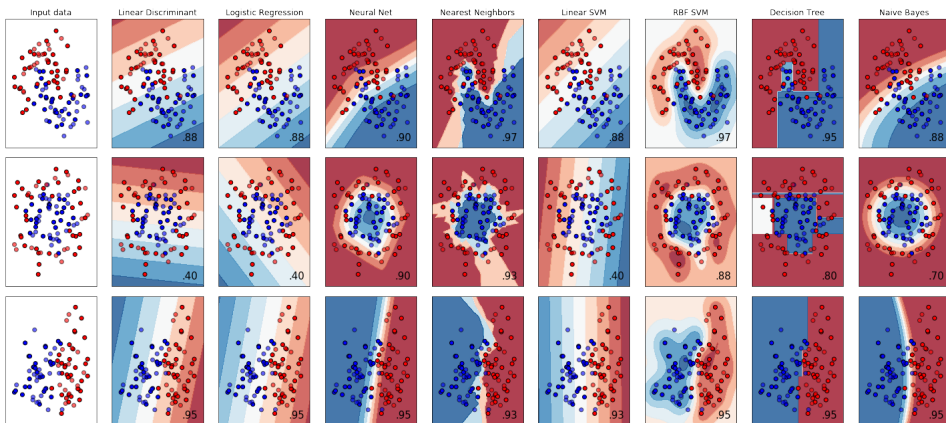
As you can see a MLP (with one hidden layer) can indeed perform very well in nonlinear classification problems. Note, however, because MLPs are highly flexible models they can easily *overfit* the data. To prevent this methods such as *early stopping* (stop when test performance starts decreasing) and *dropout* (randomly drop units in the network) are used.

# Classification methods – overall comparison





# Classification methods – overall comparison [including methods from the upcoming lectures]



# Tasks

- ▶ Live lecture week 2 (Tue 9-10): Questions about ML concepts, regression and nnets
  - ▶ You should bring questions or post them on Teams > QA channel.
- ▶ Next lab (Week 2): Linear and nonlinear regression, nnets and SVMs
  1. Join meeting on your Bubble [from 9am to 12pm on Thu]
  2. See link to lab 2 on BB