Binary Classification

Christopher Yau

October 4, 2021

Overview

Example

The probability of getting a job offer (O) is dependent only on the outcome of an interview (I).

The probability that someone is offered an interview I depends on their having prior experience E and a minimum set of academic grades G.

A person's prior experience and academic record can be considered independent.

1. Draw a DAG representing these dependencies.
2. Factorise the joint probability distribution $P(E, G, I, O)$

TAKE 2 MINUTES TO SEE IF YOU CAN WORK THIS OUT

Assignments

There is no specific template for the first assignment.

A report that contains:

1. Well-commented code,
2. And plots (with textual descriptions) as requested in the assignment

would be sufficient for this exercise.

It is not necessary to include any introductory text or discussion.

Drop-in Class

Submit questions in the Blackboard discussion thread for the Drop-in Class on 6th October 2021 from 14:00-15:00.

QUESTIONS ABOUT WEEK 1 MATERIAL SHOULD BE SUBMITTED BY 09:00, 6 OCTOBER, 2021.

Questions should focus on:

1. Concepts you have not understood in the live or video lectures from the previous week ,
2. Any side questions related to concepts,
3. Any hints or tips required for answering example problems.

Describe your question in sufficient detail.

No detailed support for mathematical derivations and calculations via Zoom.

Attendance is not compulsory – only if you have questions.

Example

The probability of getting a job offer (O) is dependent only on the outcome of an interview (I).

The probability that someone is offered an interview I depends on their having prior experience E and a minimum set of academic grades G.

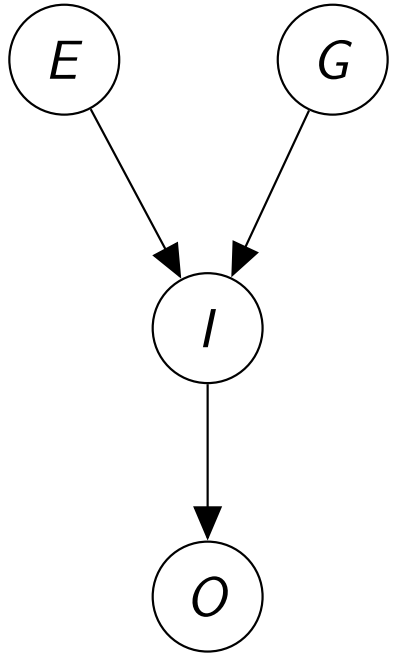
A person's prior experience and academic record can be considered independent.

1. Draw a DAG representing these dependencies.
2. Factorise the joint probability distribution $P(E, G, I, O)$

TAKE 2 MINUTES TO SEE IF YOU CAN WORK THIS OUT

Example

The DAG looks like the following:



This leads to a factorisation of the joint distribution:

$$P(E, G, I, O) = P(O|I)P(I|E, G)P(E)P(G)$$

Example

Q: Compute the probability that you are offered a job $P(O = 1)$.

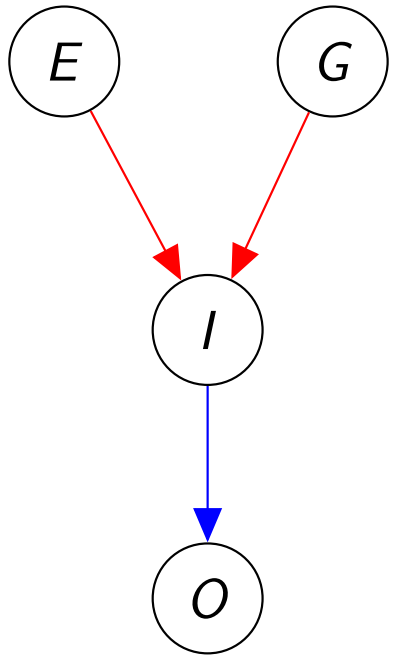
You are given the following (assume all variables are binary):

1. $P(E = 1) = 0.6, P(G = 1) = 0.8,$
2.
   - $P(I = 1|E = 0, G = 0) = 0,$
   - $P(I = 1|E = 1, G = 0) = 0.6,$
   - $P(I = 1|E = 0, G = 1) = 0.4,$
   - $P(I = 1|E = 1, G = 1) = 0.9,$
3. $P(O = 1|I = 1) = 0.2,\ P(O = 1|I = 0) = 0.$

TAKE 2 MINUTES TO SEE IF YOU CAN WORK THIS OUT

Example

The DAG looks like the following:



The DAG shows I can break this computation into two parts:

$$P(O) = P(O|I)P(I),$$
$$P(I) = \sum_{E,G} P(I|E,G)P(E)P(G)$$

Example

Q: Compute the probability that you are offered a job $P(O = 1)$.

$P(E, G, I) = P(I\|E, G)P(E)P(G)$	E	G	I
$(1 - 0.6) \times 0.8 \times 0.4 = 0.128$	0	1	1
$0.6 \times (1 - 0.8) \times 0.6 = 0.072$	1	0	1
$0.6 \times 0.8 \times 0.9 = 0.576$	1	1	1

Using total probability:

$$P(I = 1) = 0.128 + 0.072 + 0.576 = 0.776$$

Recognising that $O = 1$ depends only on $I = 1$:

$$P(O = 1) = P(O = 1|I = 1)P(I = 1) = 0.2 \times 0.776 = \boxed{0.1552}$$

Example

Q: Compute the probability that a person offered a job has experience $P(E = 1|O = 1)$.

You are given the following (assume all variables are binary):

1. $P(E = 1) = 0.6, P(G = 1) = 0.8$
2. $P(I = 1|E = 0, G = 0) = 0$, $P(I = 1|E = 1, G = 0) = 0.6$, $P(I = 1|E = 0, G = 1) = 0.4$, $P(I = 1|E = 1, G = 1) = 0.9$
3. $P(O = 1|I = 1) = 0.2$, $P(O = 1|I = 0) = 0$

TAKE 2 MINUTES TO SEE IF YOU CAN WORK THIS OUT

Example

Q: Compute the probability that a person offered a job has experience $P(E = 1|O = 1)$.

From the definition of conditional probability:

$$P(E = 1|O = 1) = \frac{P(E = 1, O = 1)}{P(O = 1)}, \text{ [CHAIN RULE]}$$

$$= \frac{1}{P(O = 1)} \sum_{G,I} P(O = 1|I)P(I|E = 1, G)P(E = 1)P(G),$$

$$= \frac{P(E = 1)}{P(O = 1)} \underbrace{\sum_{I} P(O = 1|I) \underbrace{\sum_{G} P(I|E = 1, G)P(G)}_{P(I|E=1)}}_{P(O=1|E=1)},$$

$$= (0.6/0.1552) \times 0.2 \times (0.072 + 0.576) = \boxed{0.501}$$

$P(E, G, I) = P(I\|E, G)P(E)P(G)$	E	G	I
$0.6 \times (1 - 0.8) \times 0.6 = 0.072$	1	0	1
$0.6 \times 0.8 \times 0.9 = 0.576$	1	1	1

Key Concepts

1. **Joint Probability:** $P(A, B)$
2. **Conditional Probability:** $P(A, B) = P(B|A)P(A)$
3. **Total Probability:** $P(C) = P(C|B)P(B) + P(C|A)P(A)$
4. **Bayes' Theorem:**
$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$
5. **Bayesian Networks (Directed Acyclic Graphs)**
$$P(A, B, C, D) = P(D|C, B)P(C|B, A)P(B)P(A)$$

(Also, random variables, expectations, discrete probability distributions)

Linear binary classifiers

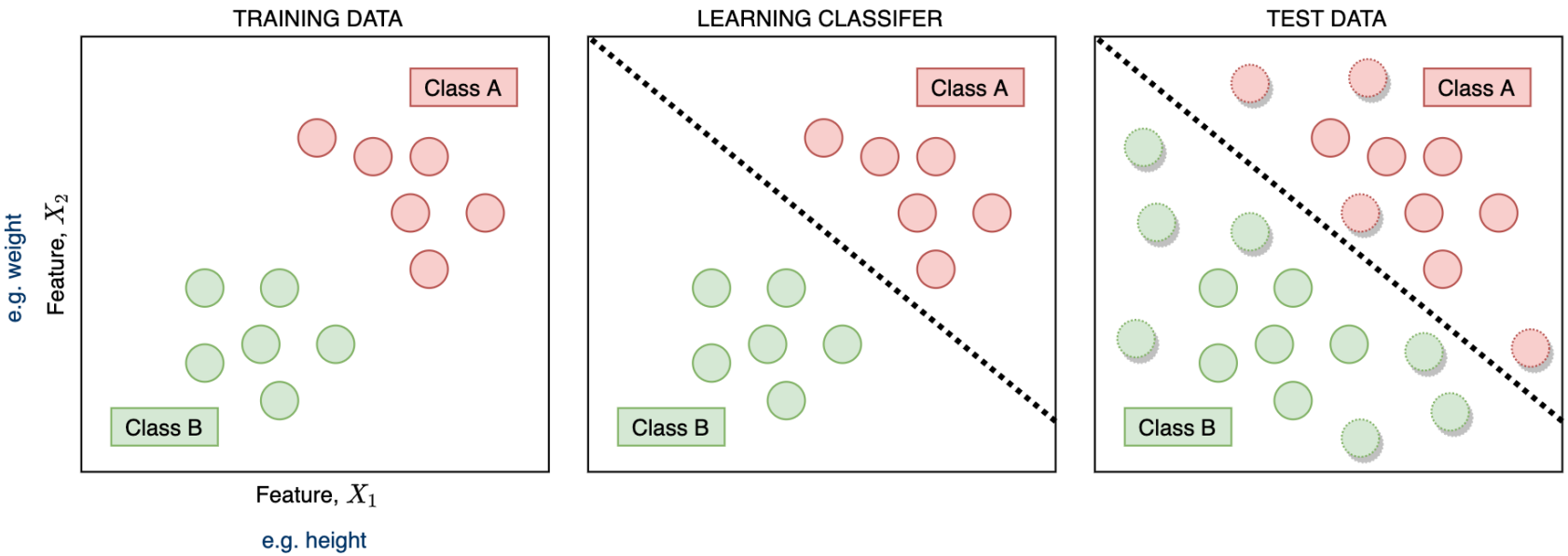
This week we will focus on classifiers that are:

- binary in their output,
- "linear" in input features or predictors,
- "linear" in their decision boundaries.

Examine "deterministic" and stochastic solutions to the same problem (different loss functions) in a **supervised** setting where we have input-output exemplars (**training data**).

Binary Classification

The goal of **binary classification** is to learn how to classify objects described by input **features** (or predictors) into one of two **output classes**.



The classifier is designed using **training data** but needs to work effectively on **test data**.

Week 2 Overview

Key concepts:

1. Binary classification
2. Gradient Descent
3. Stochastic Gradient Descent
4. Perceptron
5. Logistic Regression

Binary classification

Learn mathematical functions f that transform an input vector $\mathbf{x} = \{x_1, x_2, \ldots, x_p\}$ into a binary-valued output $y \in \{0, 1\}$ or $(\{-1, 1\})$:

$$y = f(\mathbf{x})$$

We specifically focus on linear functions such that:

$$y = f_{\mathbf{w}}(w_1 x_1 + w_2 x_2 + \cdots + w_p x_p + w_0)$$

where $\mathbf{w} = \{w_0, w_1, \ldots, w_p\}$ are parameters to be estimated or learnt from training data.

Training a classifier

Given known input-output pairs (**training data**):

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n\}).$$

Find $f_{\mathbf{w}}$ such that the predictions given the inputs match the known outputs as much as possible:

$$f_{\mathbf{w}}(\mathbf{x}_i) \underbrace{\Rightarrow}_{\text{Predict}} \hat{y}_i \underbrace{\Leftrightarrow}_{\text{Compare}} y_i$$

The **loss function** is what we use to make the comparison.

Loss functions

The loss function compares the collection of predictions $\hat{y}$ against the true outputs y:

$$L_{\mathbf{w}}(y, \hat{y}) = \sum_{i=1}^{n} l_{\mathbf{w}}(y_i, \hat{y}_i)$$

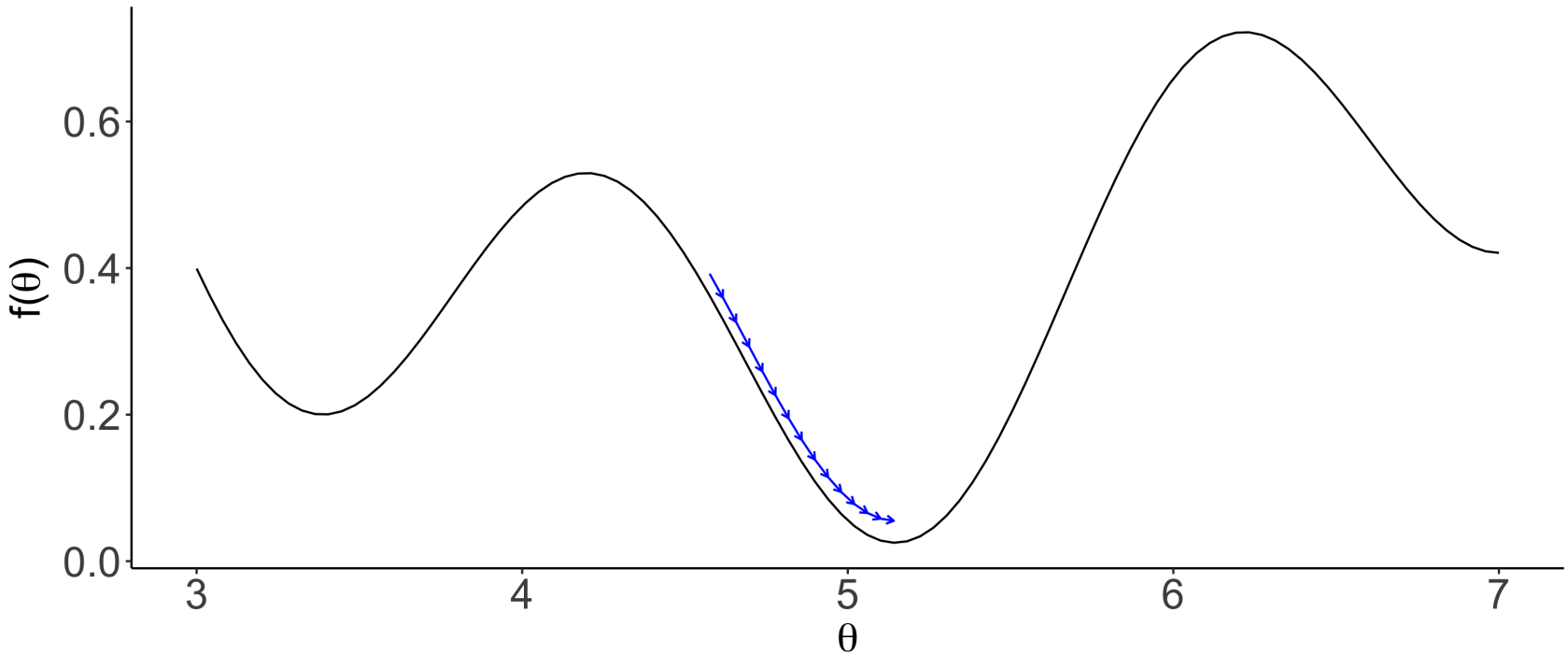
which we can typically write as a sum over the n training samples because we classify each sample independently.

We use the subscript $\mathbf{w}$ to denote that the loss function depends on these parameters.

Goal: We want to minimise the loss function with respect to the parameters $\mathbf{w}$

Gradient Descent

Given a general function $f(\theta)$ which depends on parameters θ.



If we want to find the minimum of f with respect to θ we can adopt a numerical approach known as **gradient descent**.

Gradient Descent

The gradient descent update is given by:

$$\theta^{(t+1)} = \theta^{(t)} - \lambda \frac{\partial f}{\partial \theta}\bigg|_{\theta=\theta^{(t)}}$$

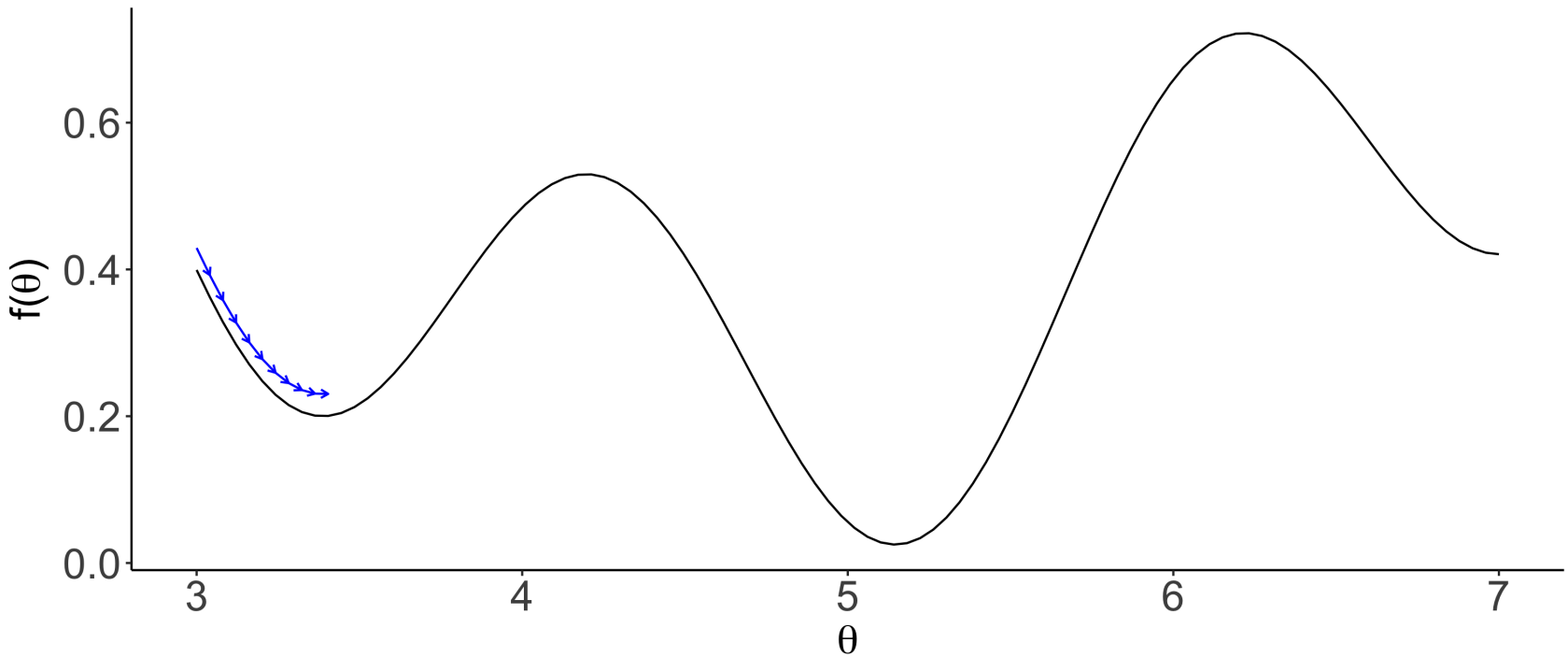
where:

- $\theta^{(t+1)}$ is the new value of θ,
- $\theta^{(t)}$ is the current value of θ,
- λ is a step size (not too big, not too small),
- $\frac{\partial f}{\partial \theta}\Big|_{\theta=\theta^{(t)}}$ at the current value $\theta^{(t)}$

Convergence when $\frac{\partial f}{\partial \theta} = 0$.

Gradient Descent

Multimodal loss functions can lead to problems.



The local minima found depends on starting point initialisation.

Gradient Descent

Given the function:

$$y = x^2 - 6x + 9$$

Questions:

1. Design a gradient descent algorithm to find the value of x that minimises this function.
2. Identify the value of x at the minimum and show that at this value of x the algorithm converges.

TAKE 2 MINUTES TO THINK ABOUT THIS PROBLEM

Gradient Descent

Given the function:

$$y = x^2 - 6x + 9$$

Design a gradient descent algorithm to find the value of x that minimises this function.

SOLUTION:

Compute gradient of y with respect to the x:

$$\frac{dy}{dx} = 2x - 6$$

GD update:

$$\begin{aligned} x' &= x - \lambda \frac{dy}{dx}, \\ &= x - \lambda(2x - 6) \end{aligned}$$

Gradient Descent

Given the function:

$$y = x^2 - 6x + 9$$

Design a gradient descent algorithm to find the value of x that minimises this function.

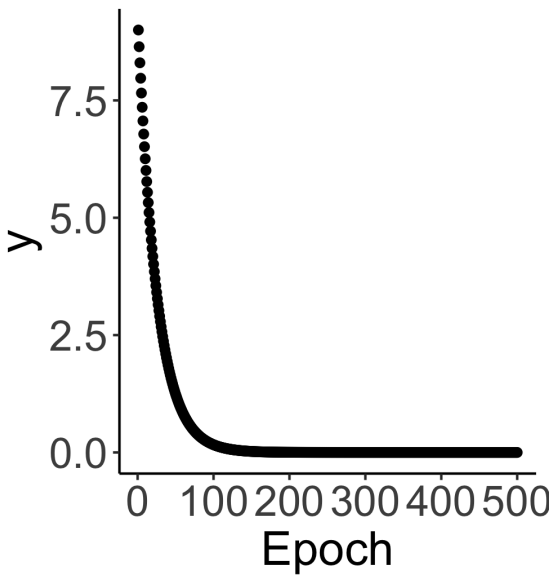
Identify the value of x at the minimum and show that at this value of x the algorithm converges.

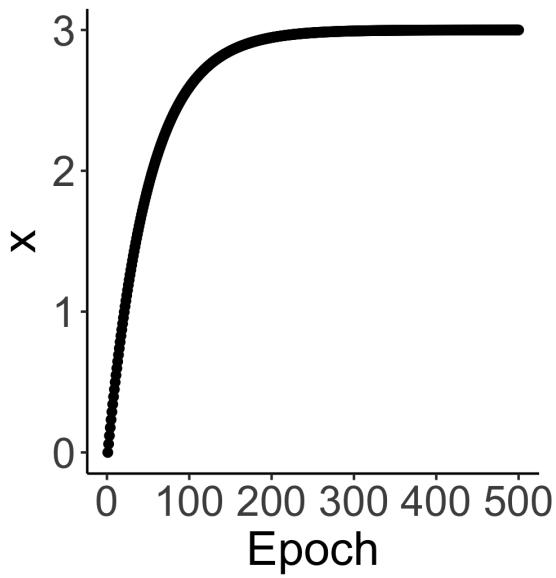
GD update:

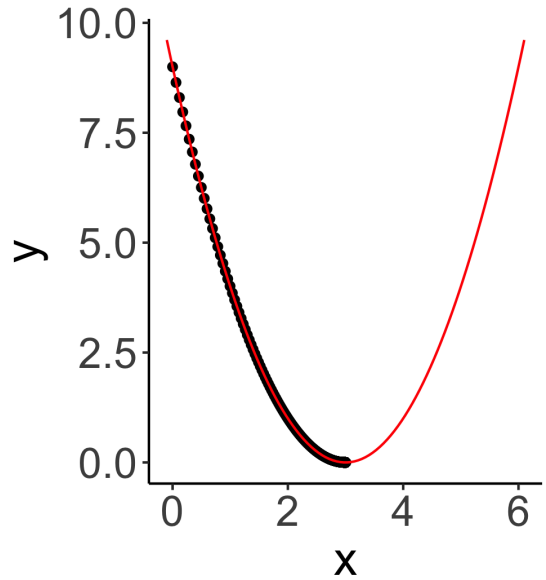
$$\begin{aligned} x' &= x - \lambda \frac{dy}{dx}, \\ &= x - \lambda(\underbrace{2x - 6}_{0}) \end{aligned}$$

Solve: $2x - 6 = 0 \Rightarrow x = 3$ at the minimum.

Gradient Descent: Example







Stochastic Gradient Descent

In modern machine learning, classic gradient descent is not used so often anymore.

Recall that our loss function is given by:

$$L_{\mathbf{w}}(y, \hat{y}) = \sum_{i=1}^{n} l_{\mathbf{w}}(y_i, \hat{y}_i)$$

If we want to minimise the loss function, we need to compute:

$$\begin{aligned} \frac{\partial L}{\partial w_j} &= \frac{\partial}{\partial w_j} \left[\sum_{i=1}^{n} l_{\mathbf{w}}(y_i, \hat{y}_i) \right], \\ &= \sum_{i=1}^{n} \frac{\partial}{\partial w_j} l_{\mathbf{w}}(y_i, \hat{y}_i). \end{aligned}$$

If n is large then the computation of the gradients could be significant.

Stochastic Gradient Descent

Stochastic gradient descent approximates the true gradients required for gradient descent by estimating from a subset of the samples (**mini-batch**).

In the extreme (**online learning**) we randomly select just one of the samples to estimate the gradient:

$$\begin{aligned}\frac{\partial L}{\partial w_i} &= \sum_{i=1}^{n} \frac{\partial}{\partial w_j} l_{\mathbf{w}}(y_i, \hat{y}_i), \\ &\approx n \frac{\partial}{\partial w_j} l_{\mathbf{w}}(y_j, \hat{y}_j)\end{aligned}$$

SGD algorithms still converge although not necessarily as quickly and each step is not guaranteed to reduce the loss function.

Stochasticity can lead to "jumping" of locals modes.

Stochastic Gradient Descent

The following data:

x	-2	-1	0	1	2
y	4.1	2.1	1.2	0	5.0

is believed to come from a function of the form:

$$y = ax + b$$

where a and b are parameters.

Given a loss function:

$$l(a, b) = \sum_{i=1}^{5} (y_i - ax_i - b)^2$$

Apply one iteration of SGD to minimise this loss function starting with $(a, b) = (0, 0)$.

TAKE 2 MINUTES TO THINK ABOUT THIS PROBLEM

Differentiate loss function with respect to a and b:

$$\frac{\partial l}{\partial a} = \sum_i -2x_i(y_i - ax_i - b),$$
$$\frac{\partial l}{\partial b} = \sum_i -2(y_i + ax_i - b),$$

Online SGD update:

$$a' = a + 10\lambda x_i(y_i - ax_i - b)$$
$$b' = b + 10\lambda(y_i - ax_i - b),$$

where i is randomly chosen from 1 to 5.

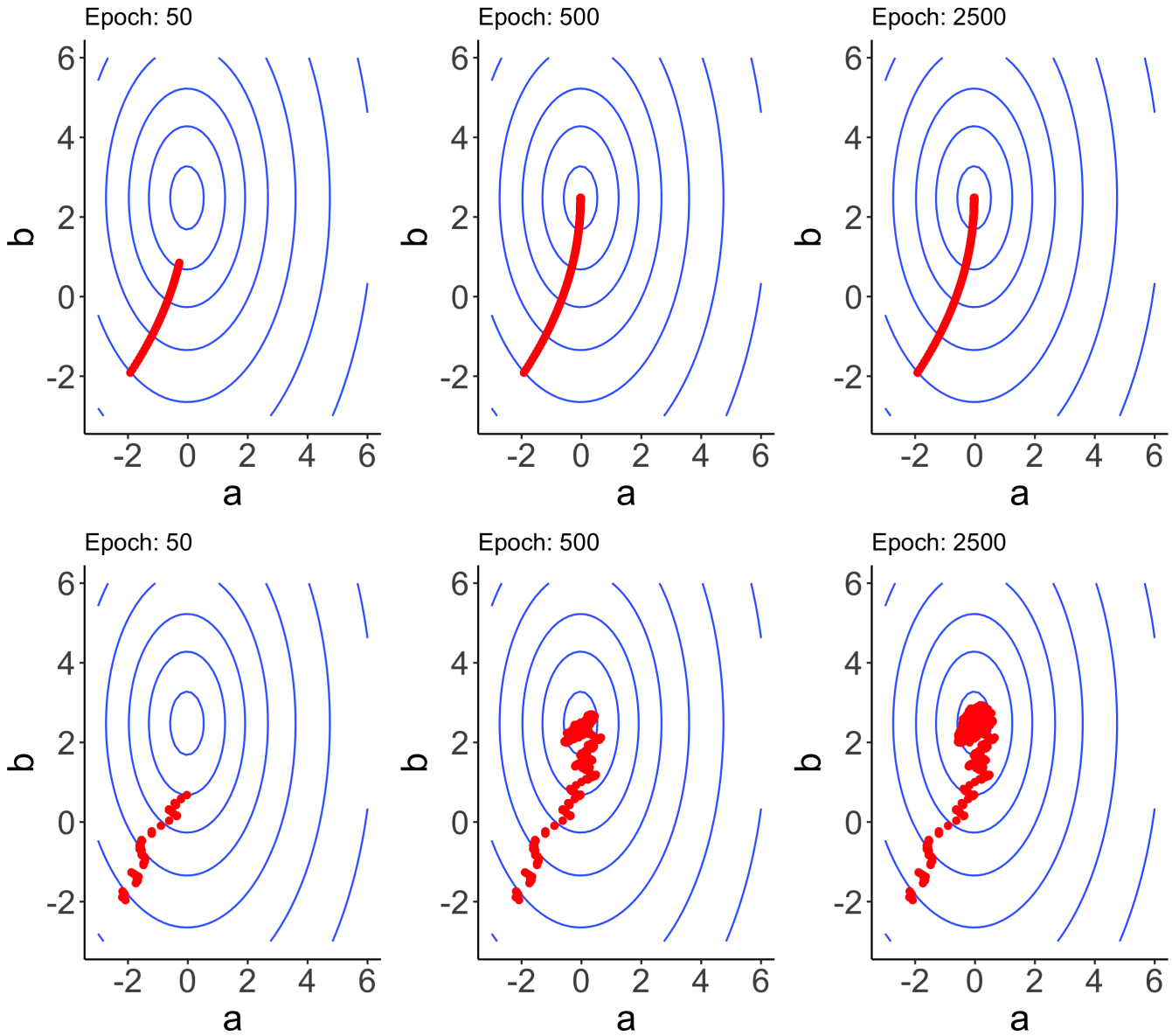


Figure: **Top**: Gradient descent. **Bottom**: SGD.

What is the objective of machine learning?

- Input X
- Output Y
- Model:
  - Deterministic: $Y = f(X)$ ("one-to-one")
  - Stochastic: $Y \sim F(X)$ where $\sim$ means "distributed" ("one-to-many")
- Learning problem: what is f or F?
- Supervised machine learning solution:

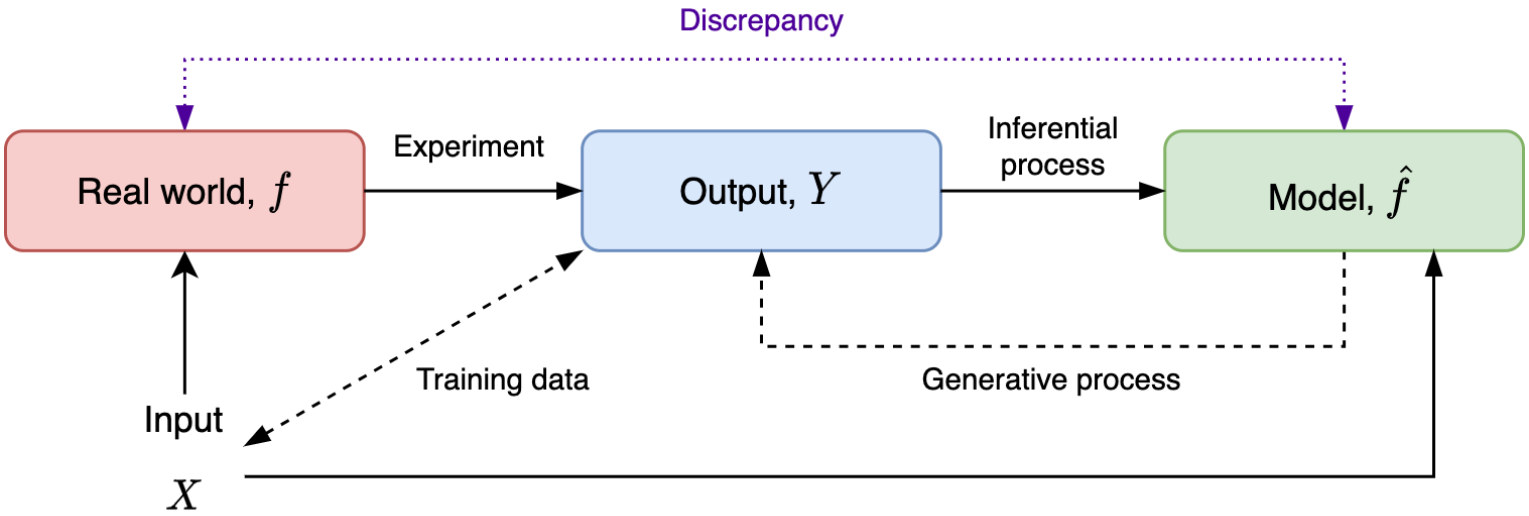
$$(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$$

Learn from paired examples (**training data**).

Why do we care what f or F is?

Generative processes

One school of thought in machine learning centres on the production of realistic models of real world processes.



Inference describes the task of learning those models *from* data.

A *generative process* describes a model for simulating real-world data.

When testing an inferential algorithm - we typically only know we have learnt *the truth* when we have a suitable generative process.

Generative processes

To simulate from a model f with parameters $\mathbf{w}$:

Generative model simulation

Repeat for $i = 1, \ldots, N$:

1. Generate input features $\mathbf{x}_i$,
2. Apply model to inputs to obtain an output, $y_i = f_{\mathbf{w}}(\mathbf{x}_i)$

The training samples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ are therefore *known* to be obtained from model f with parameters $\mathbf{w}$.

This would not typically be the case with real-world data.

Perceptron

A classic binary classification algorithm from the 1958 (Rosenblatt) whose features are still relevant today.

Generative process - how is data generated?

Linear model:

$$z = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_p x_p$$

Classify:

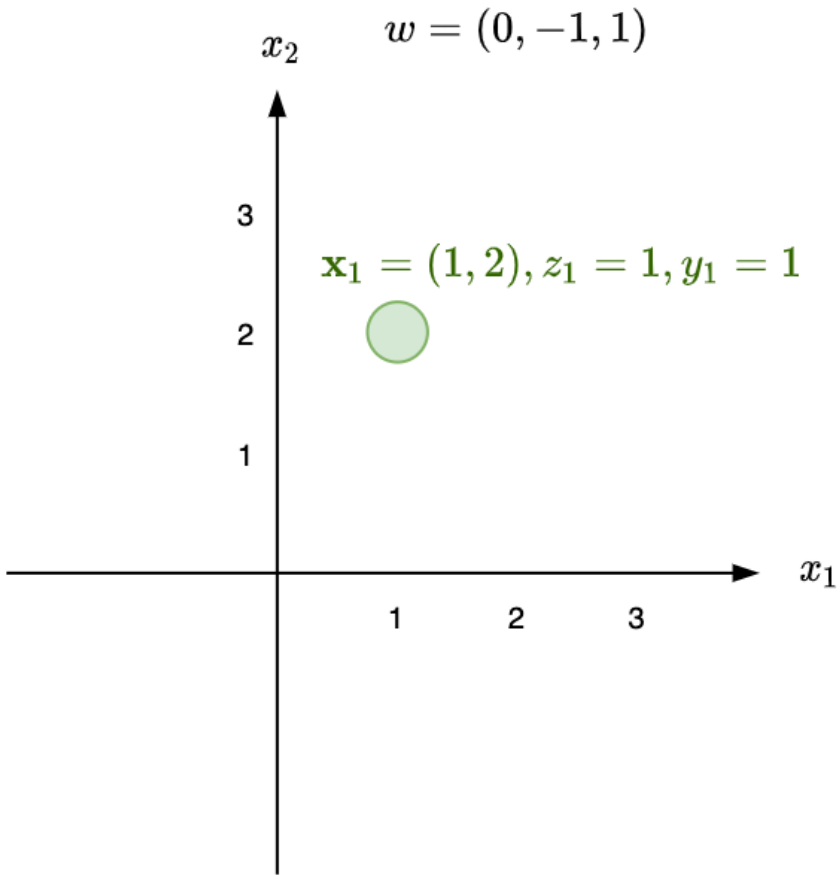
$$y = \begin{cases} -1, & z < 0, \\ +1, & z \geq 0. \end{cases}$$

or $y = \operatorname{sign}(z)$.

Learning process - what parameters best fit the data?

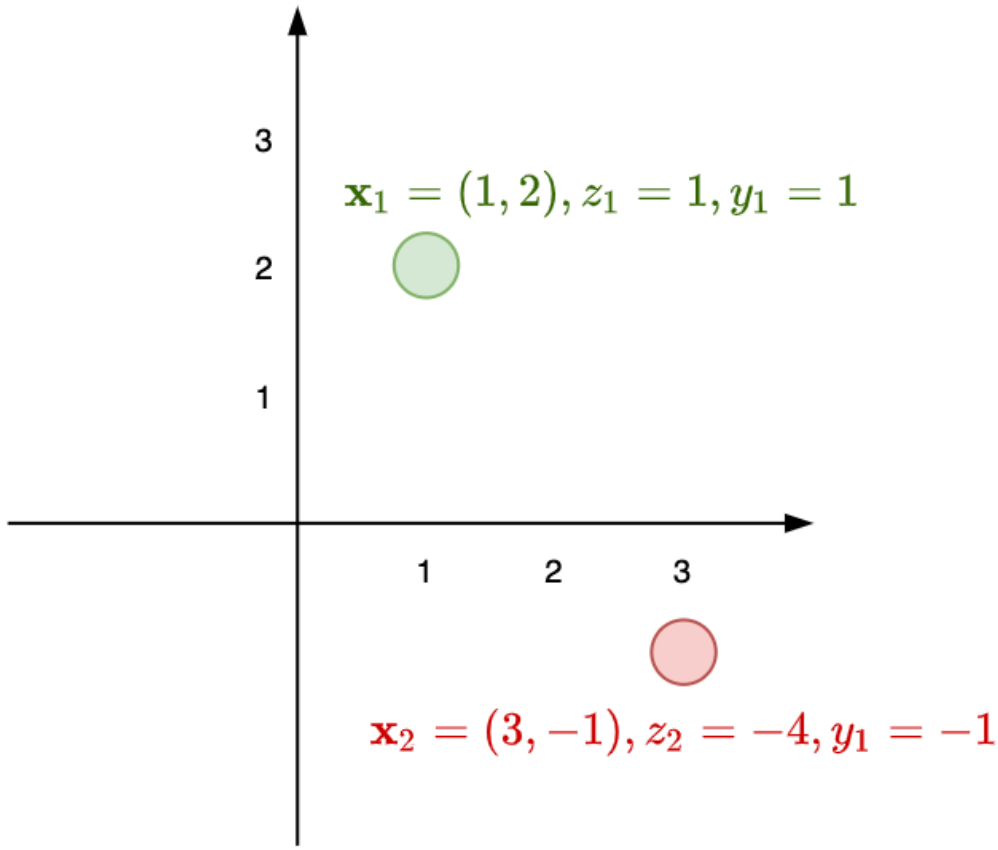
$$\underbrace{\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n\})}_{\text{Training data}} \Rightarrow \underbrace{\mathbf{w}}_{\text{Parameter Estimate}}$$

Perceptrons

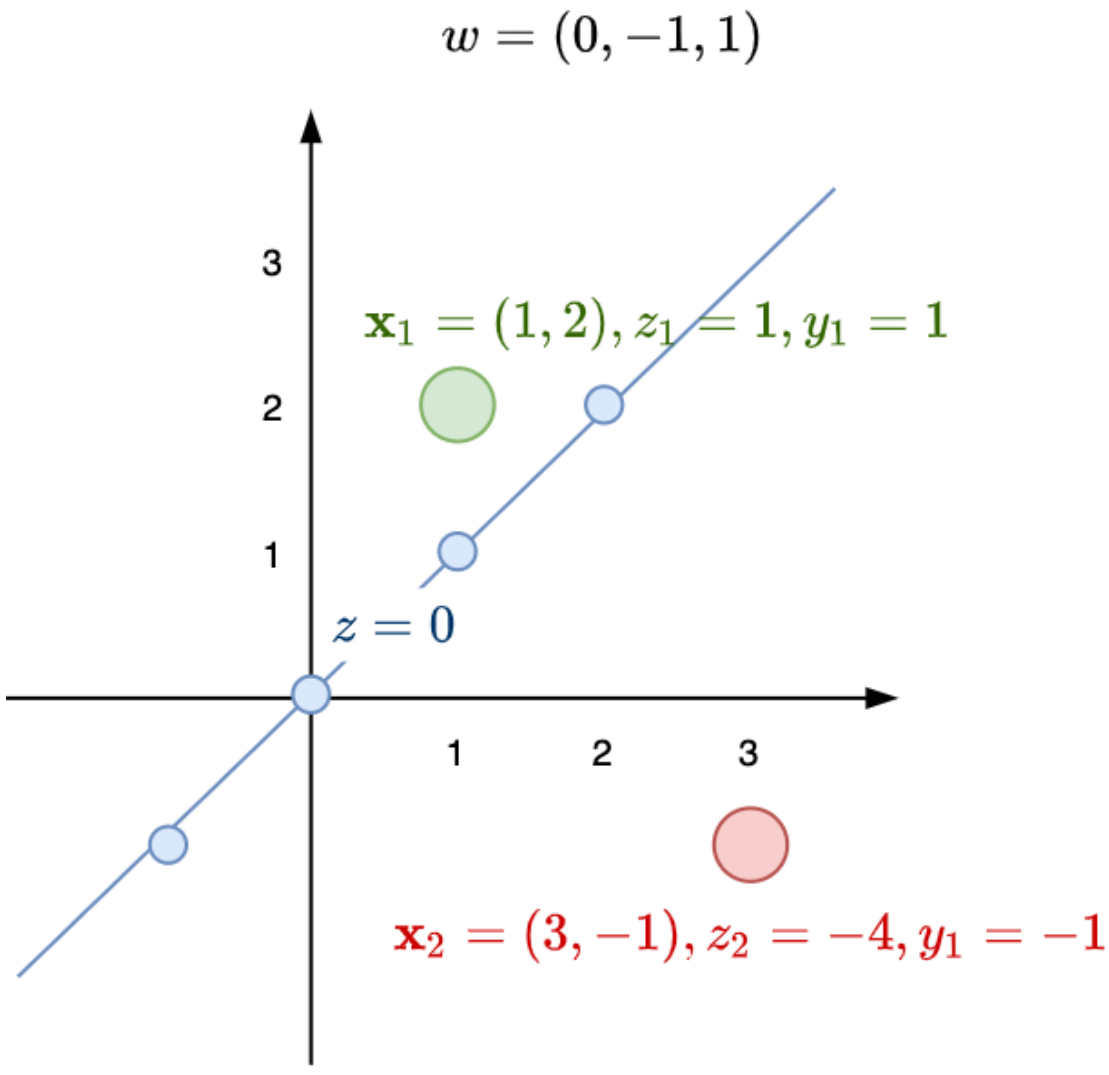


Perceptrons

$$w = (0, -1, 1)$$



Perceptrons



Perceptron

Learning process

Given a current set of parameters $\mathbf{w}$, if I misclassify the i-th sample, i.e. $\hat{y}_i \neq y_i$:

The classical perceptron update:

$$w_j' = w_j + y_i x_{ij}, \ j = 1, \ldots, p,$$

or $\mathbf{w}' = \mathbf{w} + y_i \mathbf{x}_i$ in vector form (for w_0 use $x_{ij} = 1$).

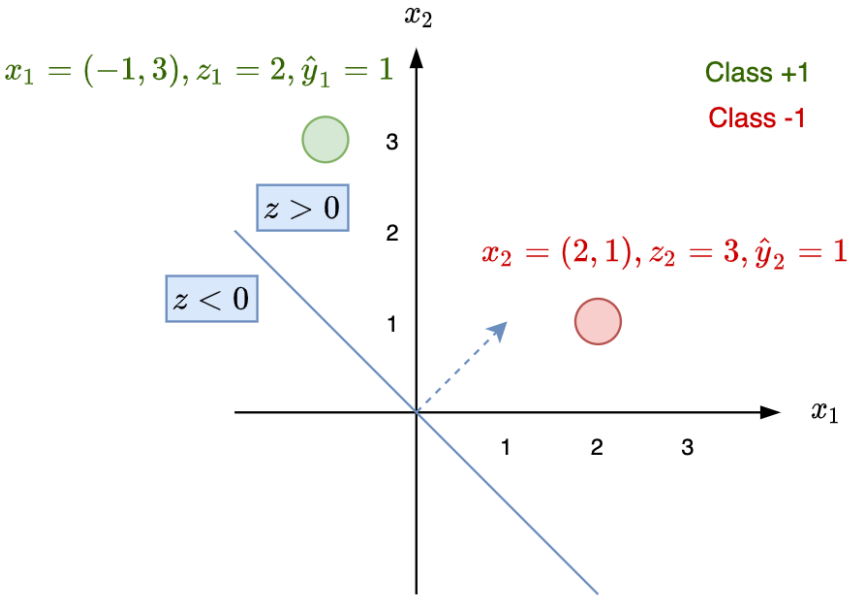
Update each weight w_j by adding on the input x_{ij} in the direction of the correct classification.

The perceptron can be viewed as stochastic gradient descent applied to a **hinge loss** (lose 0 for no misclassification, lose 1 for a misclassification).

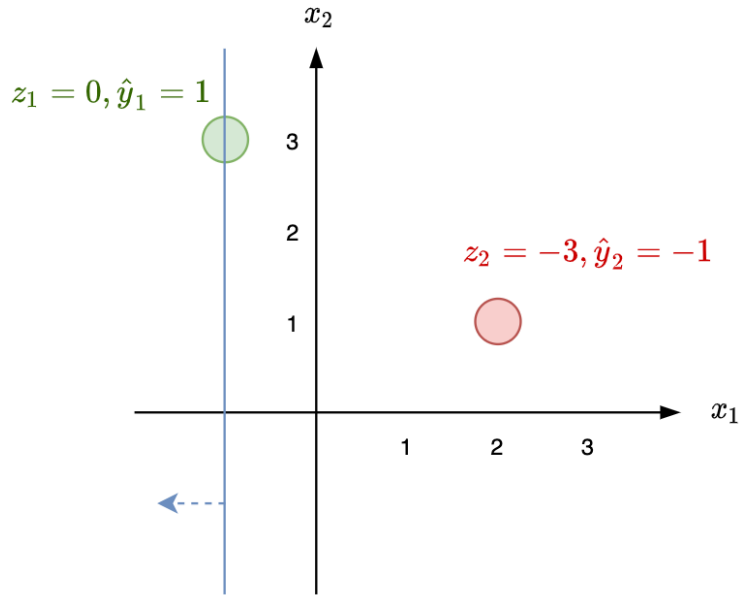
(See video lectures and Python notebook for derivation and implementation)

Perceptrons

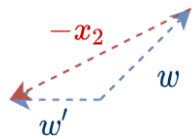
$w = (w_0, w_1, w_2) = (0, 1, 1)$



$w = (-1, -1, 0)$



$w' = (0, 1, 1) + (-1)(1, 2, 1) = (-1, -1, 0)$



Perceptron

Apply one step of the perceptron learning algorithm, starting with weights $(w_0, w_1, w_2) = (1, 0, 1)$, to the following four training data points:

Input, (x_1, x_2)	Output, y
(-1, 2)	-1
(-2, 2)	-1
(1, 0)	1
(2, 1)	1

Classical perceptron update: $w_j' = w_j + y_i x_{ij},\ j = 1, \ldots, p$, or $\mathbf{w}' = \mathbf{w} + y_i \mathbf{x}_i$ in vector form (for w_0 use $x_{ij} = 1$).

TAKE 2 MINUTES TO THINK ABOUT THIS PROBLEM

Perceptron

Starting with weights $(w_0, w_1, w_2) = (1, 0, 1)$.

Latent variable: $z = w_1 x_1 + w_2 x_2 + w_0$

Prediction: $\hat{y} = \text{sign}(z)$

First iteration $\mathbf{x} = (-1, 2), y = -1$:

$$z = \underbrace{0(-1)}_{w_1 x_1} + \underbrace{1(2)}_{w_2 x_2} + \underbrace{1}_{w_0} = 3$$

which leads to $\hat{y} = 1$ which is *not* negative and is a misclassification.

Perceptron update:

$$\begin{aligned} w_0' &= w_0 - 1 = 1 - 1 = 0, \\ w_1' &= w_1 - x_1 = 0 - (-1) = 1, \\ w_2' &= w_2 - x_2 = 1 - 2 = -1. \end{aligned}$$

Use the updated weights $(w_0, w_1, w_2) = (0, 1, -1)$ and apply the perception update to the next data point $(x_1, x_2) = (-2, 2)$ and $y = -1$.

TAKE 1 MINUTE TO THINK ABOUT THIS PROBLEM

Perceptron

Check classification with weights $(w_0, w_1, w_2) = (0, 1, -1)$:

$$w_1 x_1 + w_2 x_2 + w_0 = 1(-2) + -1(2) + 0 = -4$$

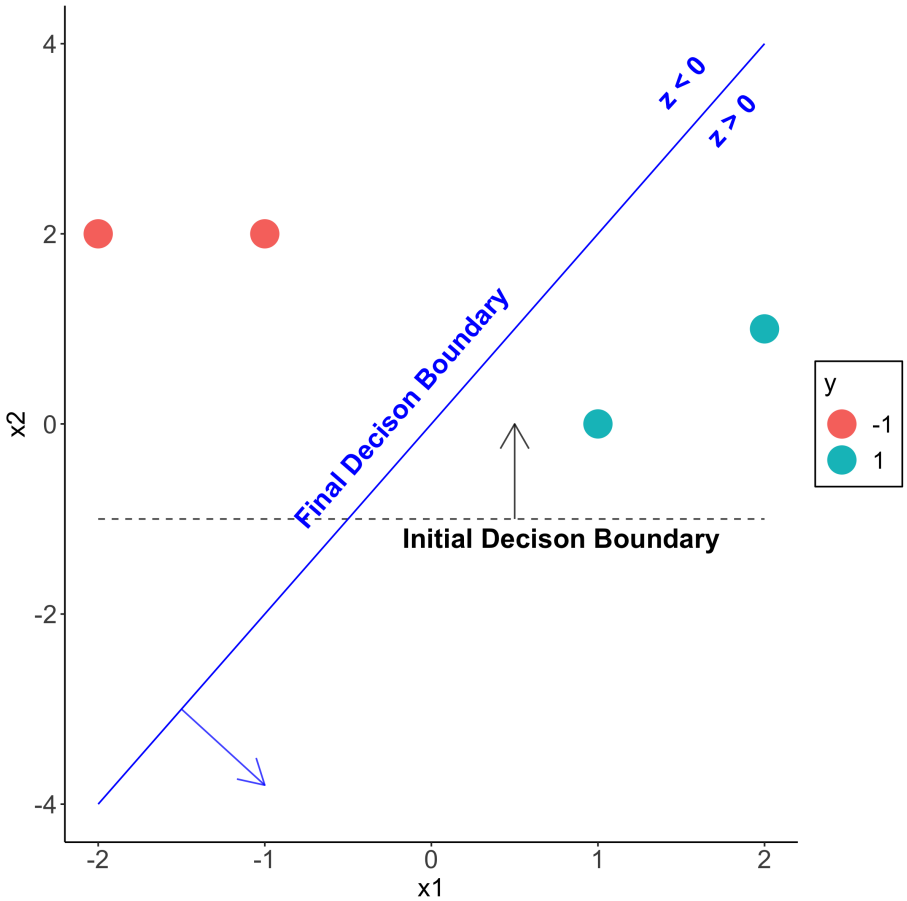
so $\hat{y} = -1$ and $y = -1$. Correct classification. No update required.

Iterate through each data point:

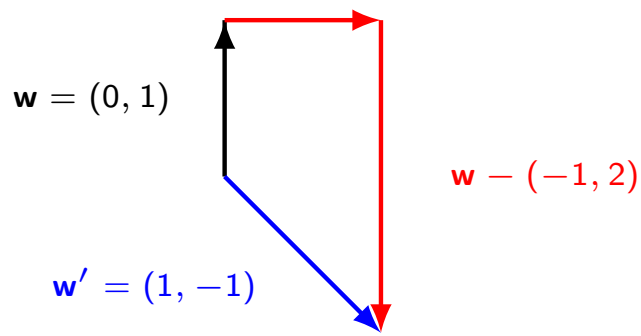
$\mathbf{x}$	y	$\hat{y}$ (z)	Update	$\mathbf{w}' = (w_0, w_1, w_2)$
(-1, 2)	-1	1 (3)	(-1, 1, -2)	(0, 1, -1)
(-2, 2)	-1	-1 (-4)	(0, 0, 0)	(0, 1, -1)
(1, 0)	1	1 (1)	(0, 0, 0)	(0, 1, -1)
(2, 1)	1	1 (1)	(0, 0, 0)	(0, 1, -1)
(-1, 2)	-1	-1 (-3)	(0, 0, 0)	(0, 1, -1)

All data correctly classified. Algorithm has converged.

Perceptron



Geometric Interpretation



Limitations

The perceptron is conceptually simple, yet surprising effective, however:

- Challenges with (high-dimensional) multiple, correlated input features,
- Classes need to be linearly separable,
- Convergence can be tricky depending on the variant of perceptron used,
- Outputs are deterministic - there is no uncertainty.

Logistic Regression

Generative process

Linear model:

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_p x_p$$

Transform z into a probability h using the logistic transform:

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

Probabilistic classification:

$$p(y = 1) = h, \;\; p(y = 0) = 1 - h$$

The output is not a deterministic function of the input.

The same input could lead to both $y = 0$ or $y = 1$ in different experiments.

Logistic Function

The **logistic function** maps the real line to $(0, 1]$ via the transformation:

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

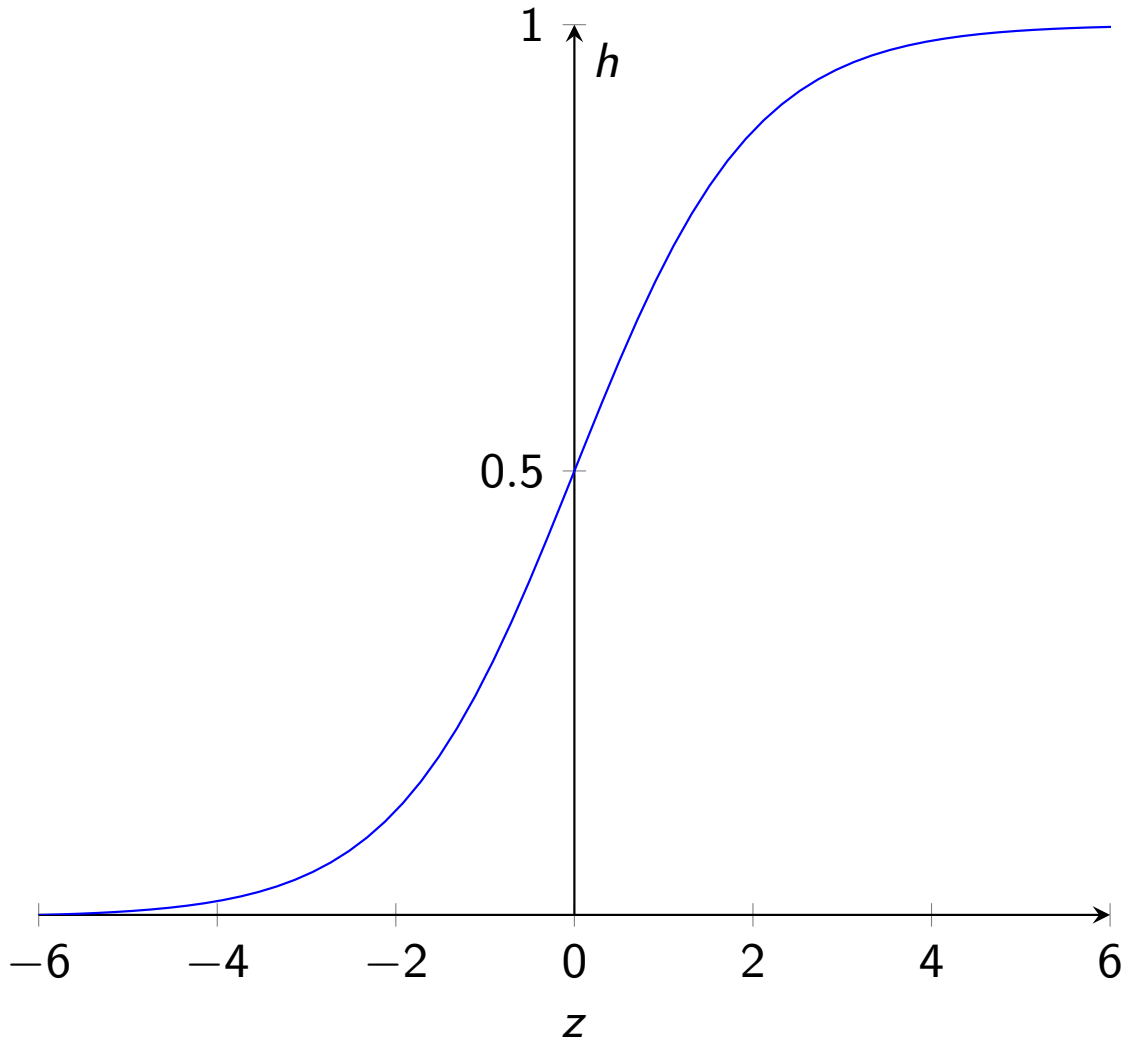
Note that as:

- $z \to \infty$: $h \to 1$,
- $z \to -\infty$: $h \to 0$,
- $z = 0$: $h = 0.5$

So our previous statements are equivalent to saying:

$$p(y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\underbrace{[\mathbf{w}^t\mathbf{x} + w_0]}_{z})}$$

Logistic Function



Logistic Regression

Given training data:

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n\}).$$

Our loss function takes the form of a probability function.

The **likelihood** or joint probability:

$$p_{\mathbf{w}}(y_1, y_2, \ldots, y_n | \mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{i=1}^{n} p_{\mathbf{w}}(y_i | \mathbf{x}_i)$$

The likelihood tells me the probability of observing those outputs y given these inputs $\mathbf{x}$ given the regression weights $\mathbf{w}$.

In logistic regression, we want to learn $\mathbf{w}$ to **maximise** the likelihood of the data (or to minimise the negative likelihood).

Logistic Regression

Fit a logistic regression model to this data set using gradient-based optimisation:

Input, x	Output, y
-1	0
-2	0
1	1
2	1

Logistic Regression: Loss function

For each training sample, compute:

$$z_i = w_1 x_i + w_0$$

and

$$p(y_i = 1) = \frac{1}{1 + \exp(-z_i)}$$

Then compute the likelihood:

$$\begin{aligned} p(y_1 = 0, y_2 = 0, y_3 = 1, y_4 = 1) &= (1 - p(y_1 = 1))(1 - p(y_2 = 1)) \\ &\quad \times p(y_3 = 1) p(y_4 = 1), \\ &= \frac{\exp(-z_1)}{1 + \exp(-z_1)} \times \frac{\exp(-z_2)}{1 + \exp(-z_2)} \\ &\quad \times \frac{1}{1 + \exp(-z_3)} \times \frac{1}{1 + \exp(-z_4)} \end{aligned}$$

Logistic Regression: Deriving updates

Define the loss function as the negative log-likelihood:

$$\begin{aligned} f &= -\ln p(y_1 = -1, y_2 = -1, y_3 = 1, y_4 = 1) \\ &= \sum_{i=1}^{2} [z_i + \ln(1 + \exp(-z_i))] + \sum_{i=3}^{4} \ln(1 + \exp(-z_i)), \\ &= \sum_{i=1}^{2} z_i + \sum_{i=1}^{4} \ln(1 + \exp(-z_i)). \end{aligned}$$

Then differentiate the loss function with respect to each parameter:

$$\begin{aligned} \frac{\partial f}{\partial w_1} &= \sum_{i=1}^{2} x_i + \sum_{i=1}^{4} \frac{-\exp(-z_i)}{1 + \exp(-z_i)} \frac{\partial z_i}{dw_1}, \\ &= \sum_{i=1}^{2} x_i - \sum_{i=1}^{4} \frac{\exp(-z_i)}{1 + \exp(-z_i)} x_i = \sum_{i=1}^{2} x_i - \sum_{i=1}^{4} (1 - p(y_i)) x_i. \end{aligned}$$

Logistic Regression: Gradient descent updates

For this problem, the gradients are:

$$\frac{\partial f}{\partial w_1} = \sum_{i=1}^{2} x_i - \sum_{i=1}^{4} (1 - p(y_i)) x_i,$$

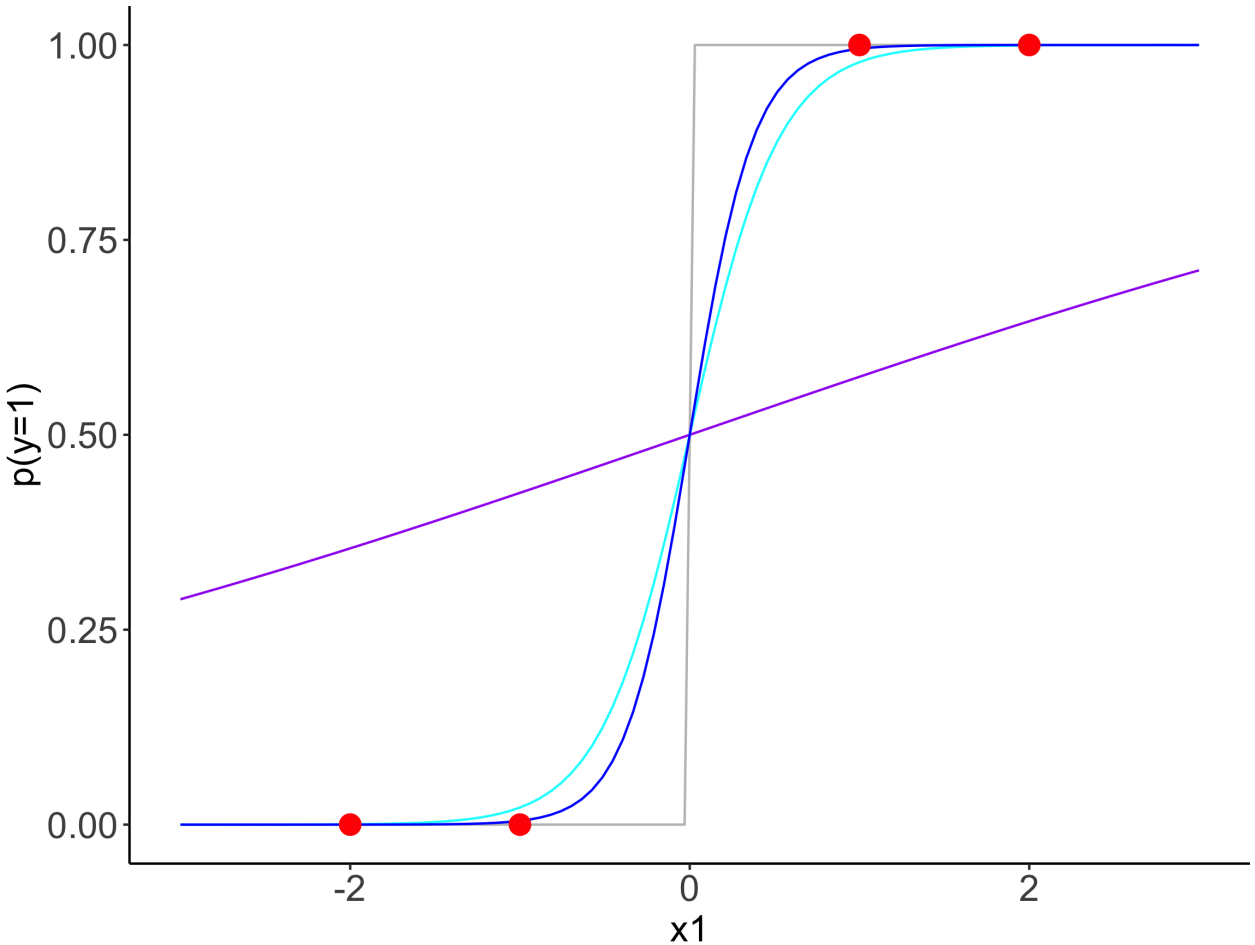
$$\frac{\partial f}{\partial w_0} = 2 - \sum_{i=1}^{4} (1 - p(y_i)),$$

and we can then proceed to update the parameters using gradient descent or stochastic gradient descent as required using:

$$w_1' = w_1 - \lambda \frac{\partial f}{\partial w_1},$$

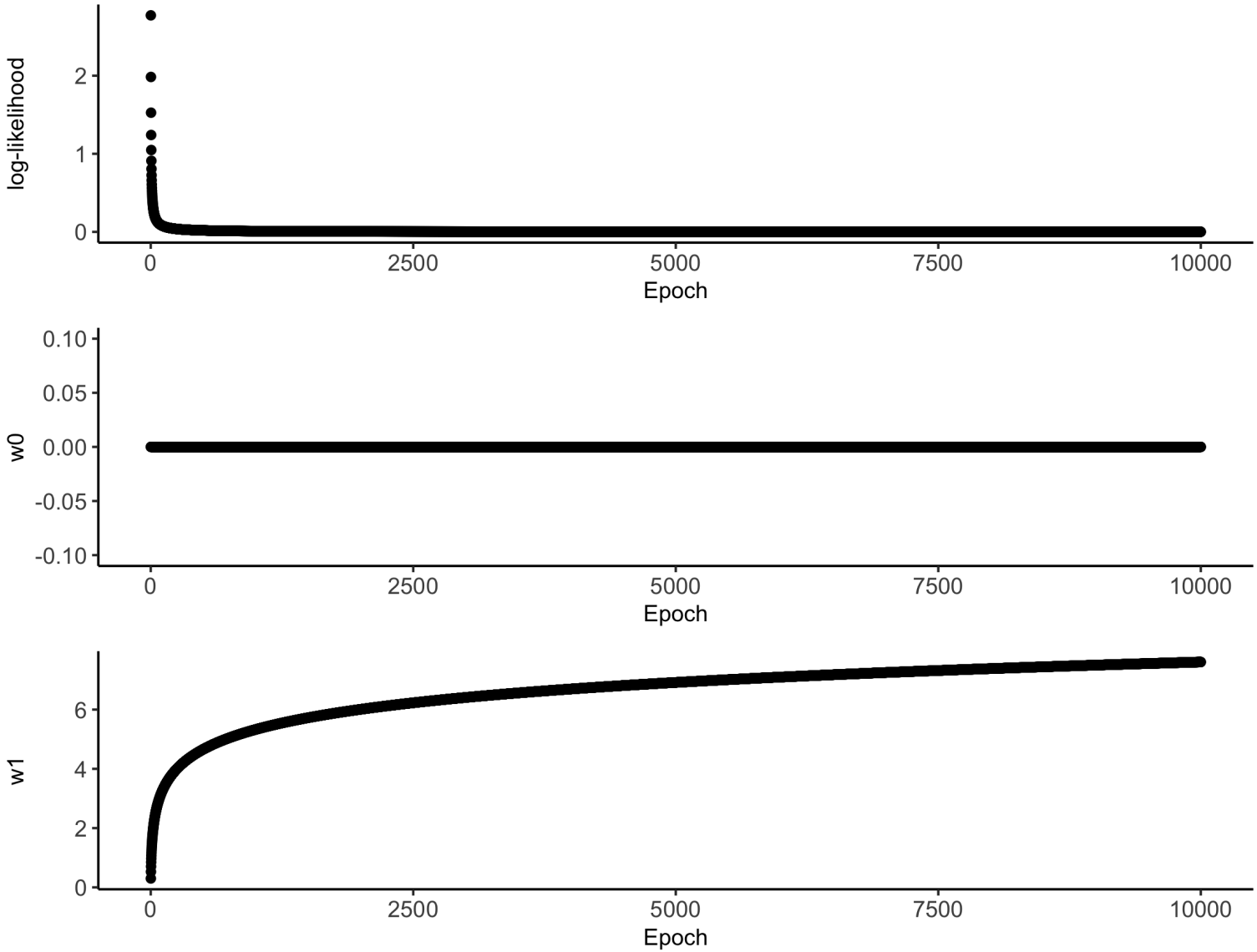
$$w_0' = w_0 - \lambda \frac{\partial f}{\partial w_0}$$

Logistic Regression: Decision function



Legend: 1st epoch, 1,000th epoch, 10,000th epoch

Logistic Regression: Checking convergence



Logistic Regression: Why no convergence?

At convergence, we require:

$$\frac{\partial f}{\partial w_1} = 0 = \sum_{i=1}^{2} x_i - \sum_{i=1}^{4} (1 - p(y_i = 1)) x_i$$

This requires:

$$\begin{aligned} x_3 + x_4 &= \sum_{i=1}^{4} p(y_i = 1) x_i, \\ &= \underbrace{p(y_1)}_{=0} x_1 + \underbrace{p(y_2)}_{=0} x_2 + \underbrace{p(y_3)}_{=1} x_3 + \underbrace{p(y_4)}_{=1} x_4 \end{aligned}$$

In this instance, as the data perfectly separates, the algorithm only *converges* when the classifier also *perfectly* classifies the data, i.e.

$$p(y_1 = 1) = 0, p(y_2 = 1) = 0, p(y_3 = 1) = 1, p(y_4 = 1) = 1$$

Limitations

Logistic regression is conceptually simple, yet also surprising effective, however:

- Challenges with (high-dimensional) multiple, correlated input features,
- Classes need to be linearly separable,
- Convergence can be tricky (e.g. regularisation for collinear features).

The mathematical framework for perceptrons and logistic regression are the foundations of modern machine learning.

Week 2 Summary

Key concepts:

1. Binary classification
2. Gradient Descent (differentiate loss function wrt to parameters)
3. Stochastic Gradient Descent (estimate gradients)
4. Perceptron (deterministic classifier)
5. Logistic Regression (probabilistic classifier, likelihood)

Online Lab (Friday) will explore practical implementations of Perceptrons and Logistic Regression.