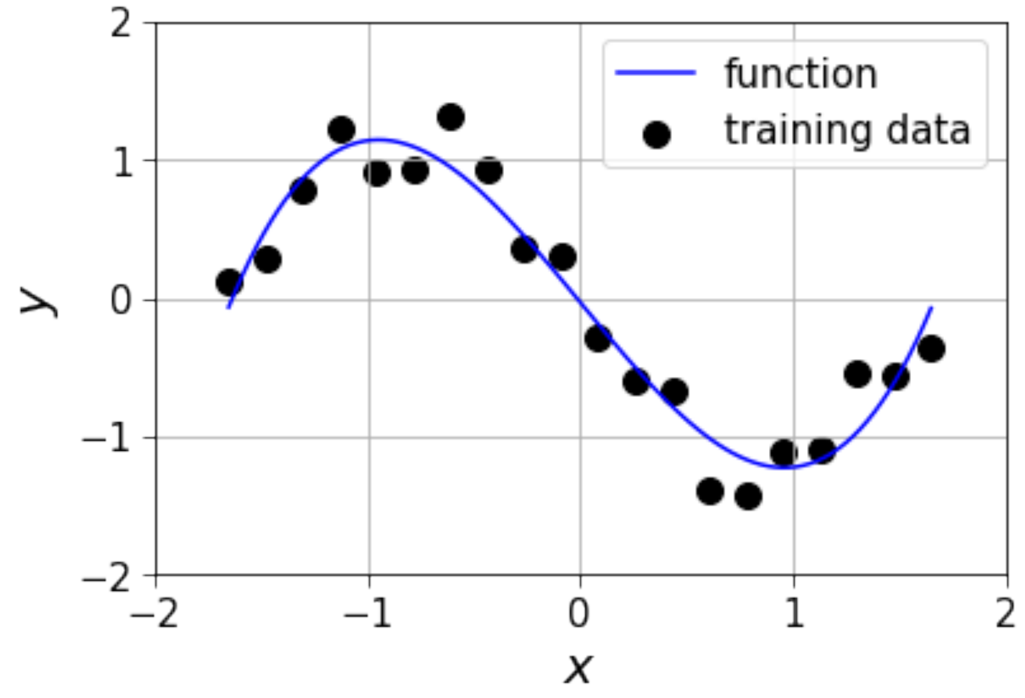
Data Analysis and Machine Learning 4

Week 6: Linear Classification

Elliot J. Crowley, 27th February 2023

Recap

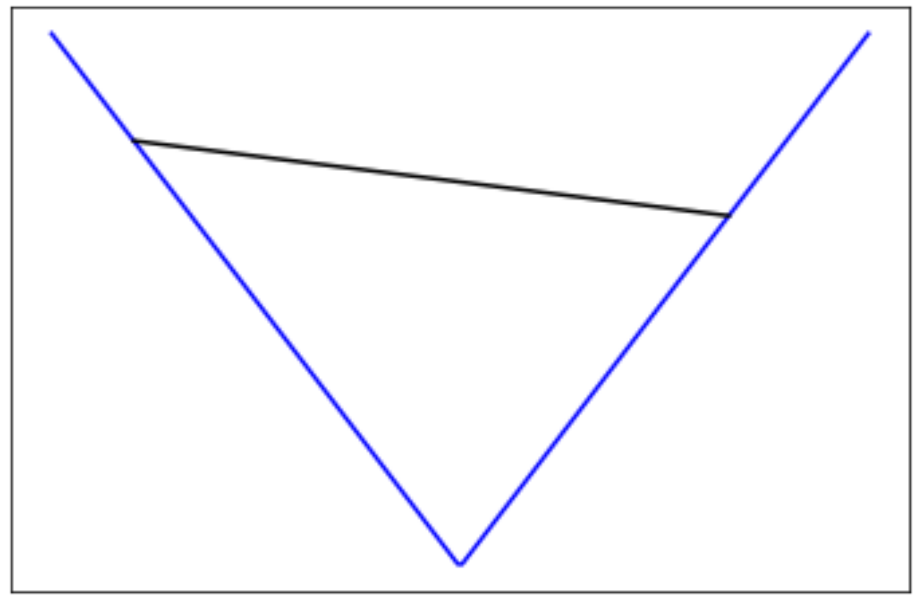
- We learned about different types of linear regression and regularisers

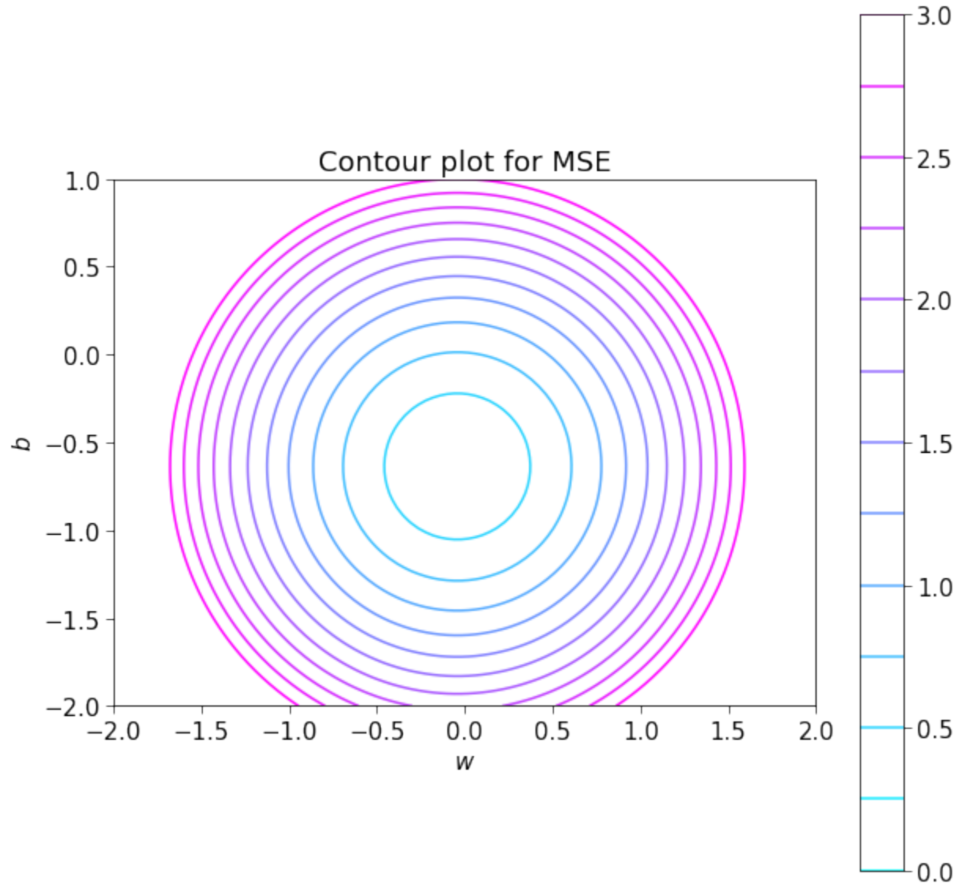


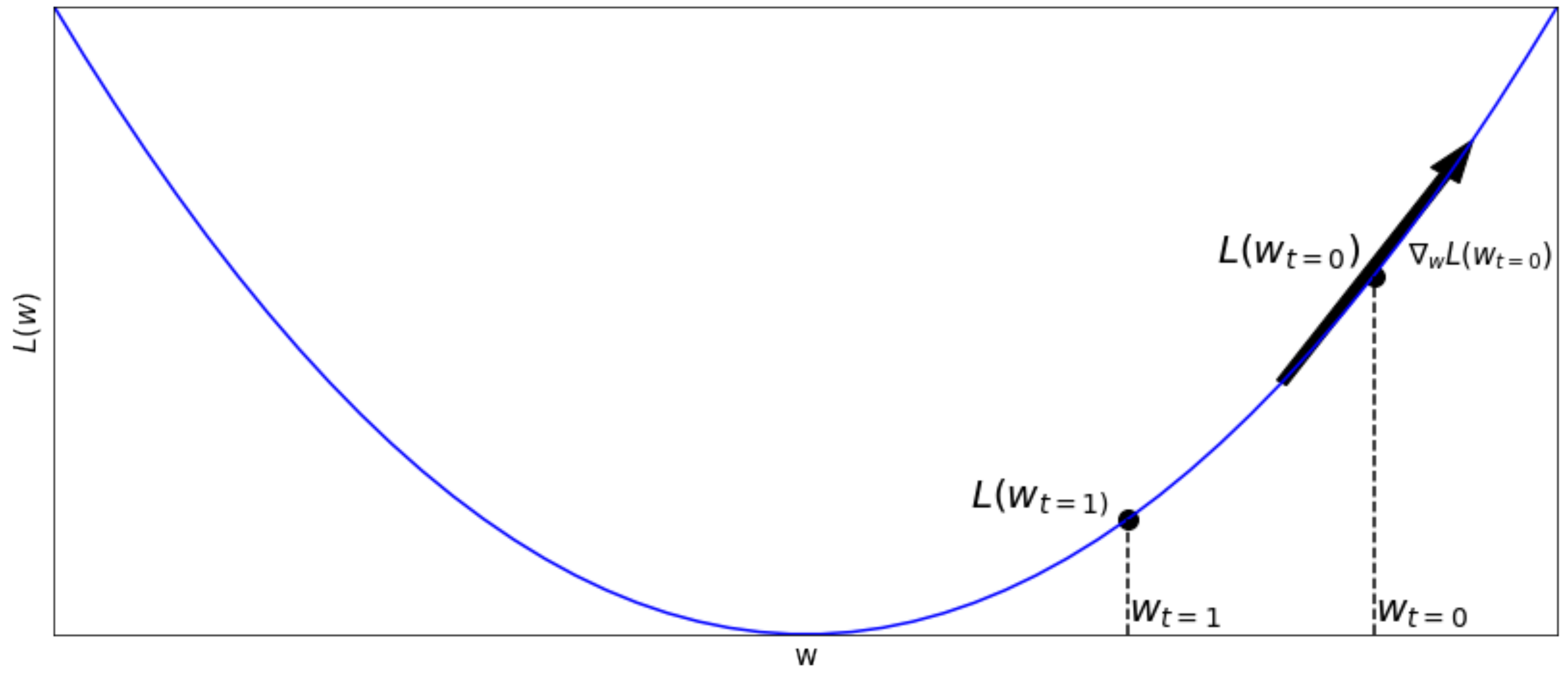
$$f(\mathbf{x}) = \hat{y} = \mathbf{w}^\top \phi(\mathbf{x})$$

$$L_{ridge}(\mathbf{w}) = \underbrace{\|\mathbf{y} - \mathbf{\Phi}\mathbf{w}\|^2}_{SE} + \underbrace{\lambda\|\mathbf{w}\|^2}_{regularisation}$$

- We looked at convex functions and gradient descent





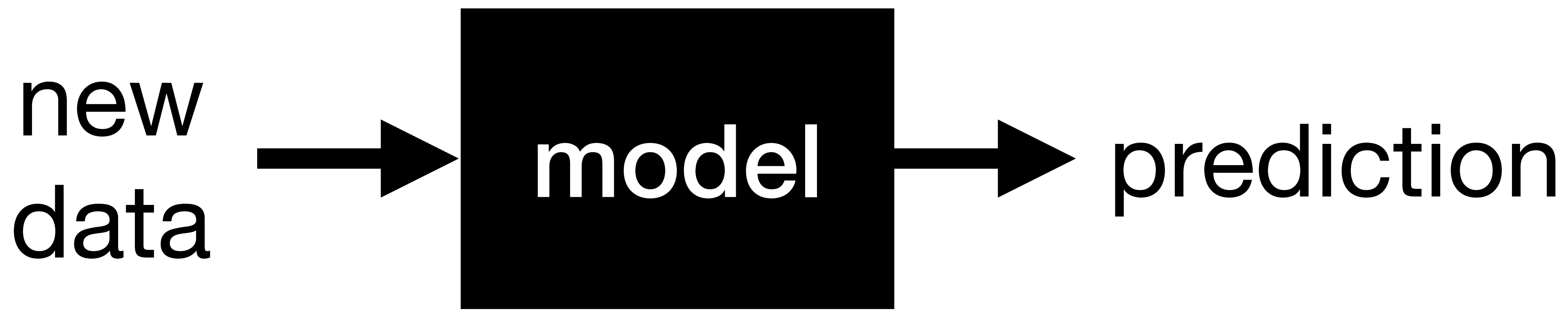


Warning: change in notation!

- We have previously included the bias term b in the weight vector $\mathbf{w}$
- This makes the maths for linear regression much nicer
- When considering classifiers it's better to separate the weights from the bias
- From now on we will write our linear model as $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$
- **The weight vector does not contain the bias**

Supervised Learning

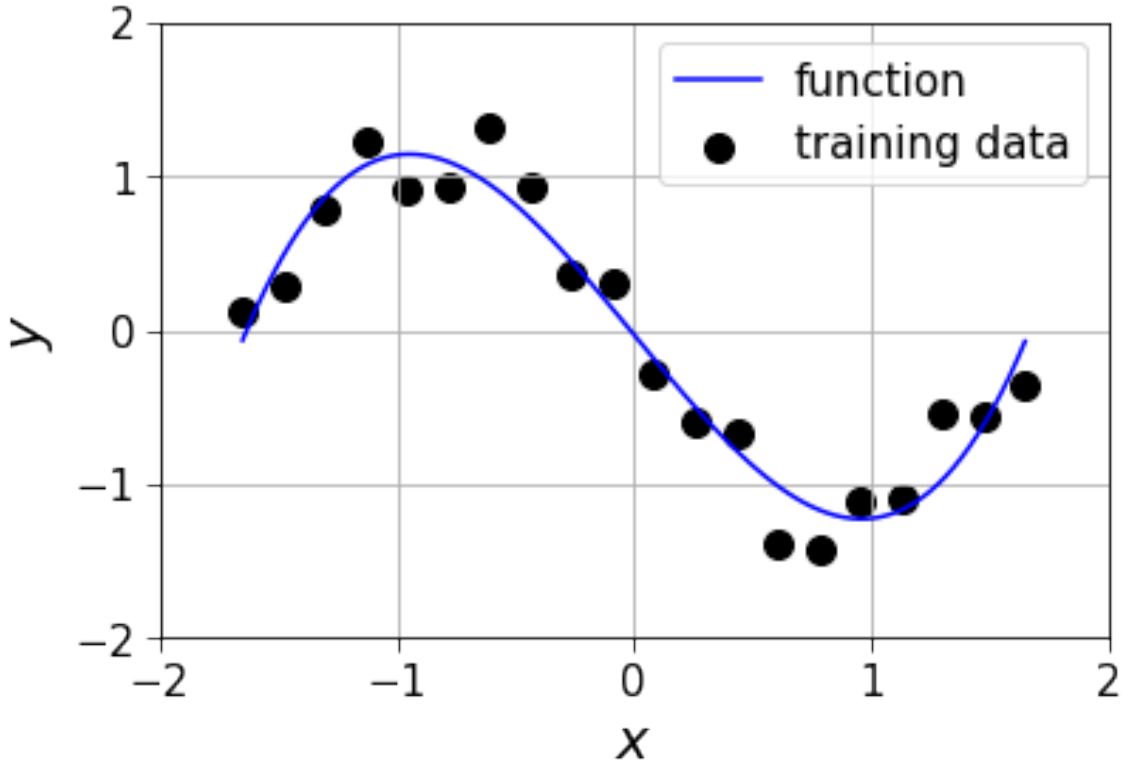
- We want a model that takes in a new data point and outputs a prediction



- For the model to be accurate it must first learn from training data
- Often, models are parameterised functions and learning = finding the best parameters
- Training data is a set of existing data points that have been **labelled**
- The label says what the prediction for that data point **should be**

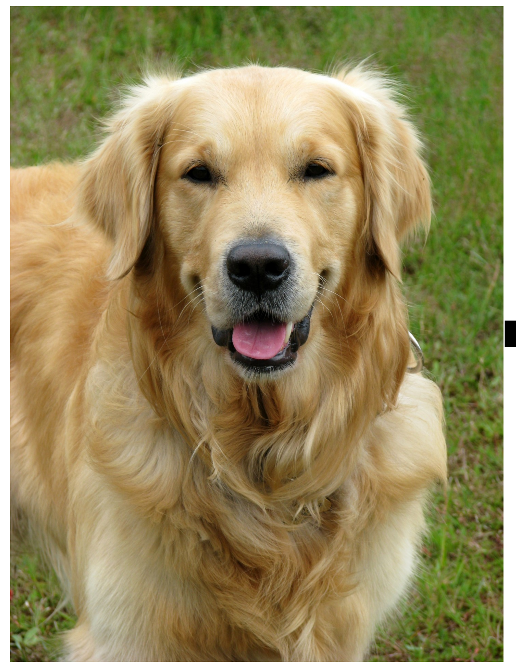
Two canonical problems in supervised learning

- **Regression**: Given input data, predict a continuous output



- **Classification**: Given input data, predict a distinct category

cat



dog

Classification

The classification problem

- Our training set consists of N data point-target pairs $\{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$
- Data points $\mathbf{x} \in \mathbb{R}^D$ are column vectors, targets are class labels $y \in \mathbb{Z}^+_{<K} = \{0,1,\ldots,K-1\}$
- i.e. each data point has been labeled as belonging to 1 of K classes
- ~~Objective: We want a model that classifies our training data **correctly**~~
- **Objective:** We want a model that classifies our held-out test data correctly

The most common way to quantify classification performance is **accuracy**

This is simply the fraction or % of classifications that are correct

Linear Classifiers

Linear classifiers

- These are linear models $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$
- For now we will consider untransformed features so $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$
- For now we will consider **binary classification:** $y \in \{0,1\}$ *or* $y \in \{-1,1\}$
- $f(\mathbf{x}) \in \mathbb{R}^1$ is a classification score: we decide how it is used to make a class prediction $\hat{y}$
- e.g. we could use a thresholding function like $\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) > 0 \\ 0 & \text{if } f(\mathbf{x}) < 0 \end{cases}$

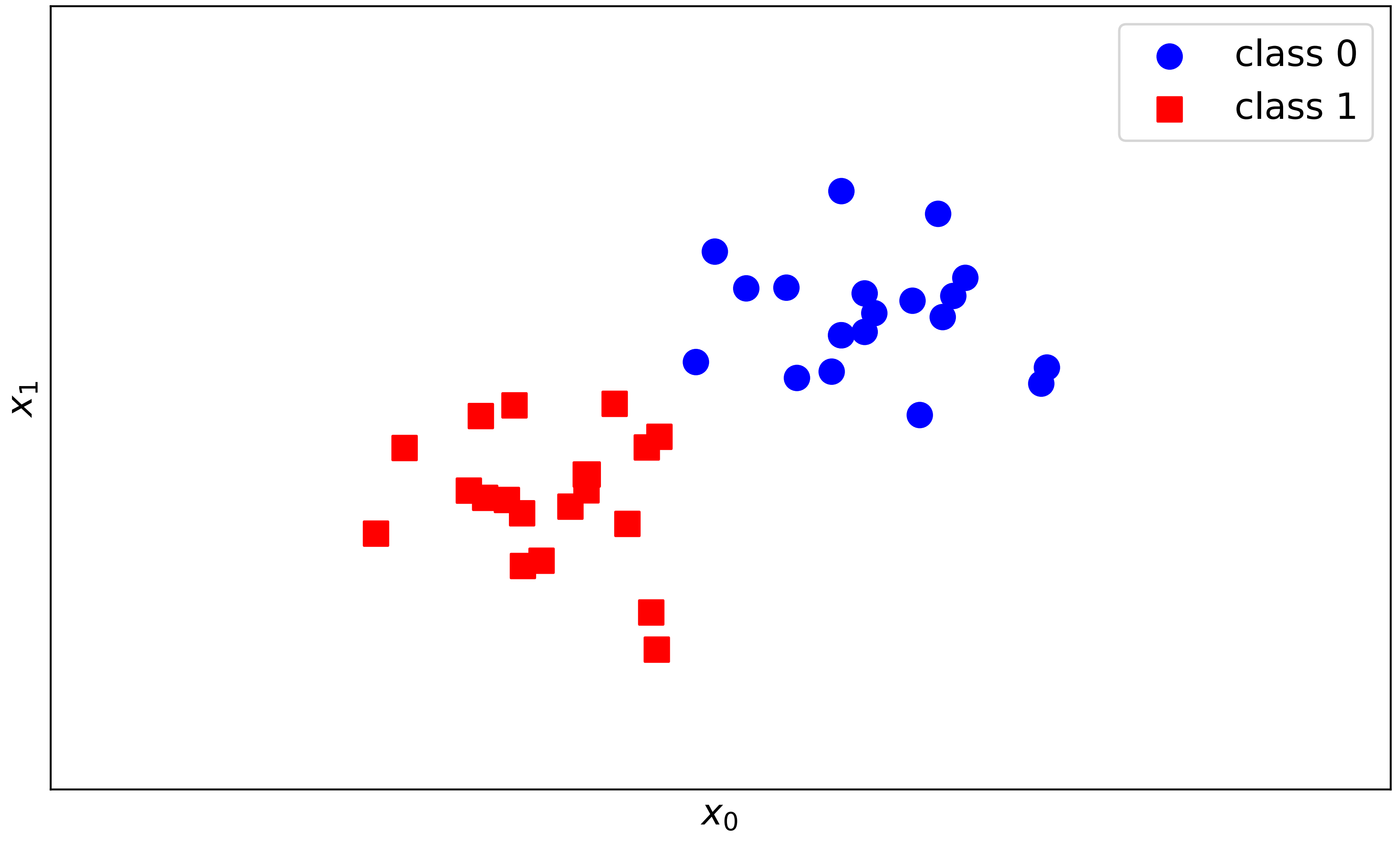
Why linear models?

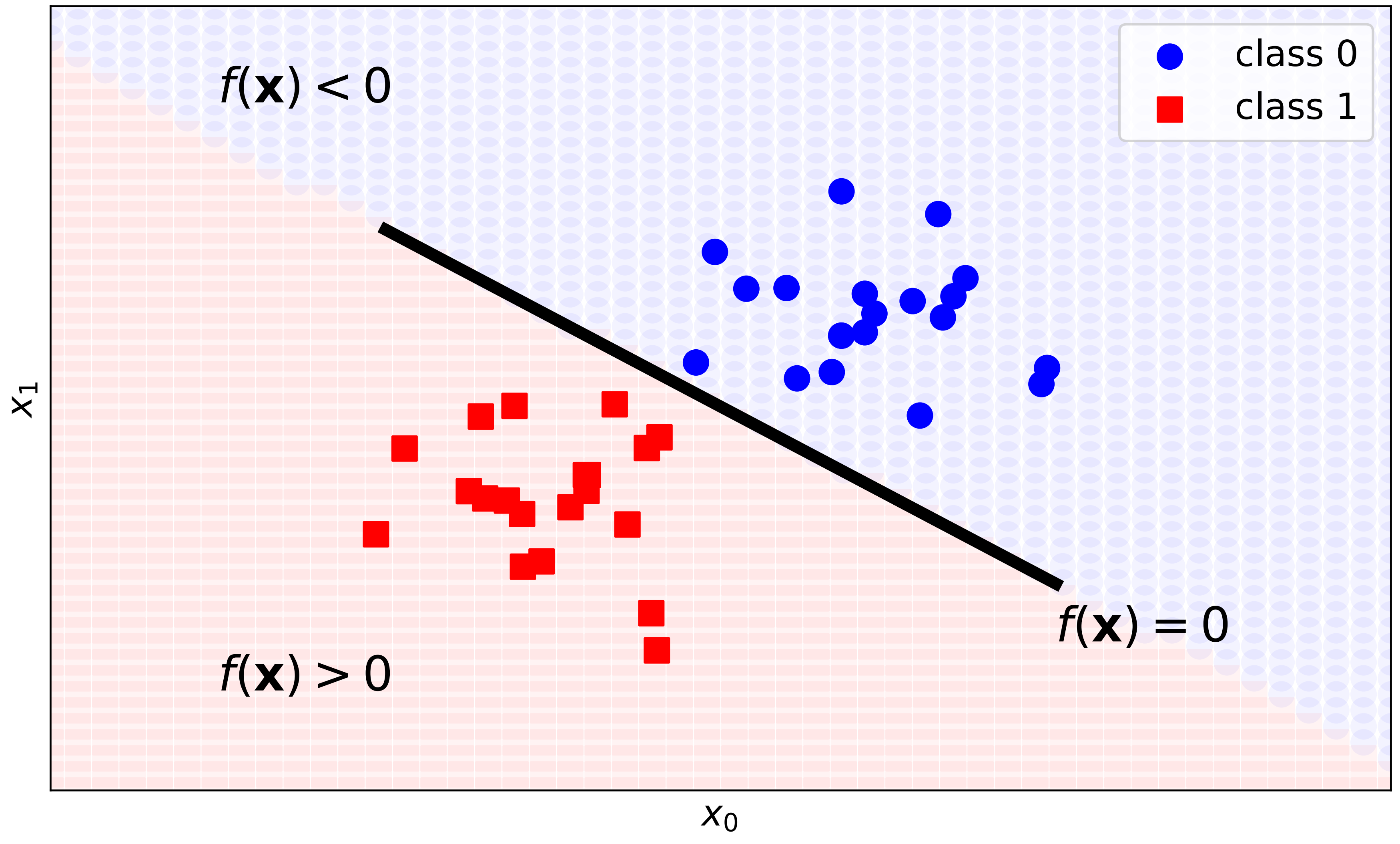
- They are simple and intuitive
- They are interpretable
- They use vectors and matrices (computers love these)
- They work well in many scenarios

Slide inspired by https://sites.google.com/site/christophlampert/teaching/kernel-methods-for-object-recognition

Linear classifier decision boundary in 2D

- Consider a training set $\{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$ with $\mathbf{x} \in \mathbb{R}^2$ and $y \in \{0,1\}$
- We have $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ where $\mathbf{x} = [x_0 \quad x_1]$ and $\mathbf{w} = [w_0 \quad w_1]^\top$
- Predictions are determined using $\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) > 0 \\ 0 & \text{if } f(\mathbf{x}) < 0 \end{cases}$
- $f(\mathbf{x}) = 0$ is a line which forms the decision boundary of the classifier

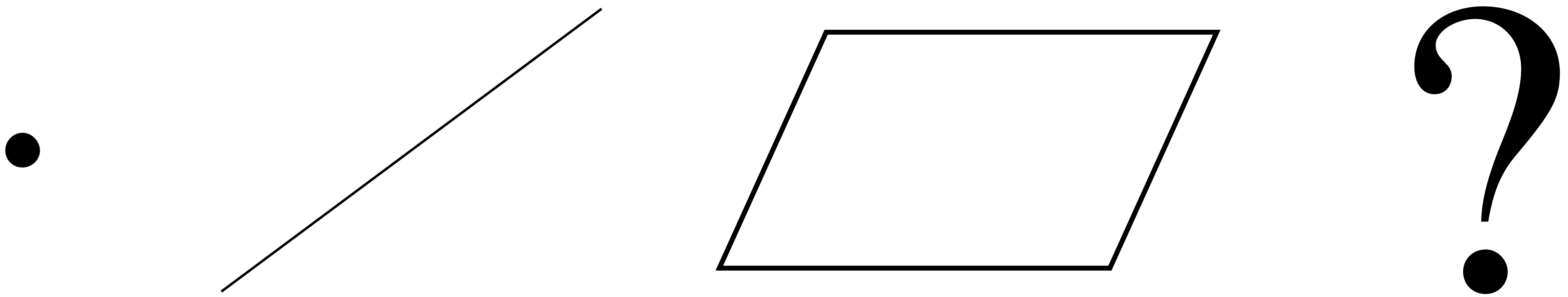




Decision boundary are hyperplanes

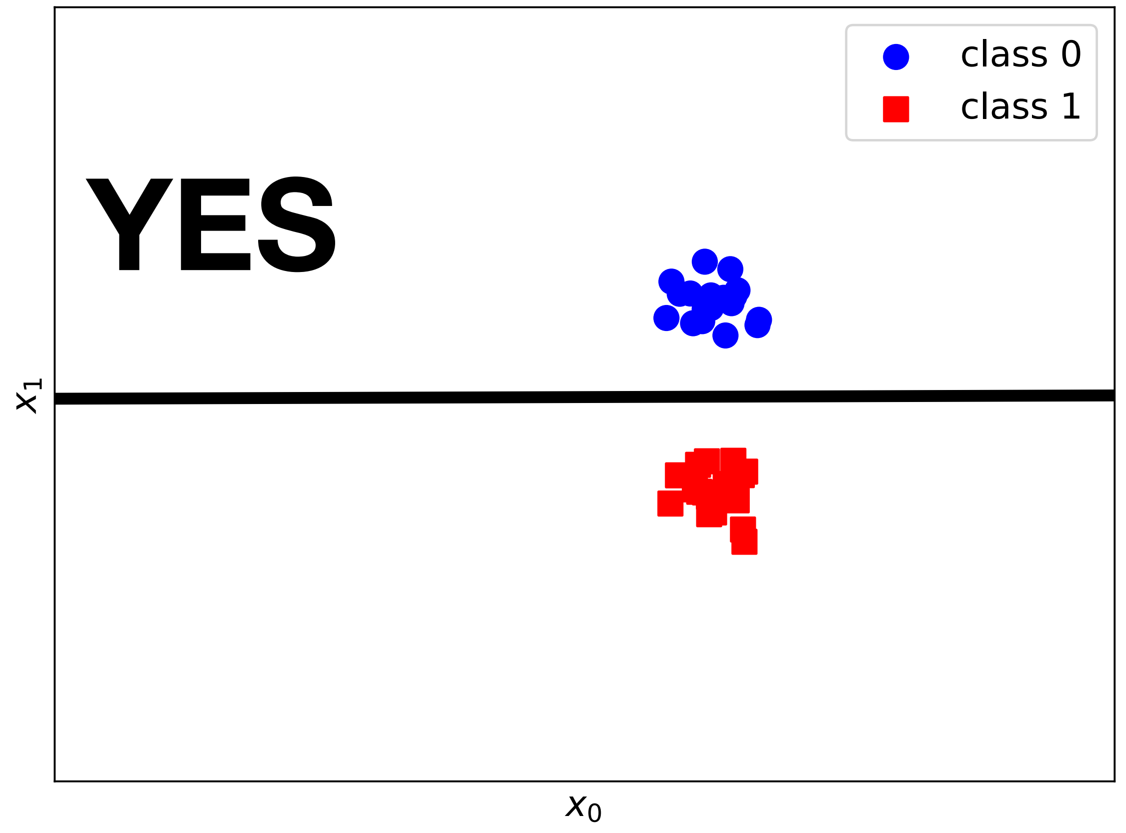
For $\mathbf{x} \in \mathbb{R}^D$ the decision boundary of a linear classifier is in $D - 1$

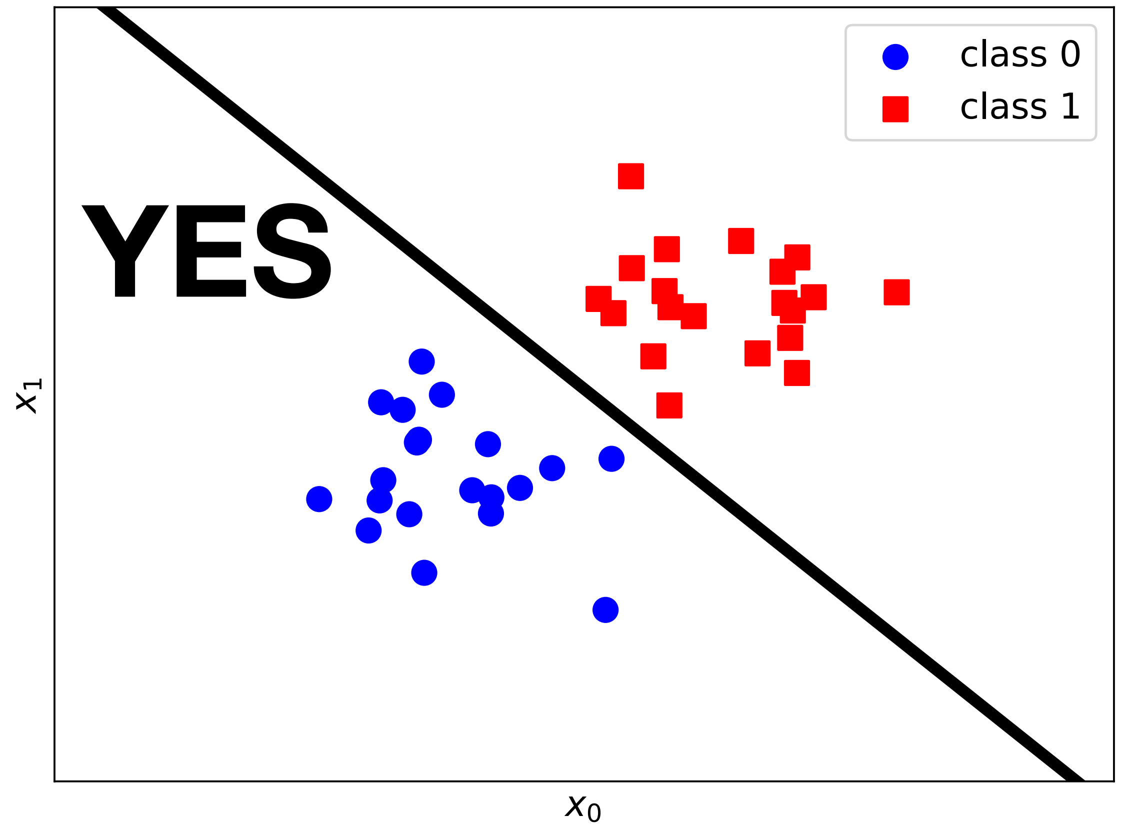
- In 1D the decision boundary is a point
- In 2D the decision boundary is a line
- In 3D the decision boundary is a plane
- In 4D and above the decision boundary is a **hyperplane** we can't visualise but all the maths still works (:

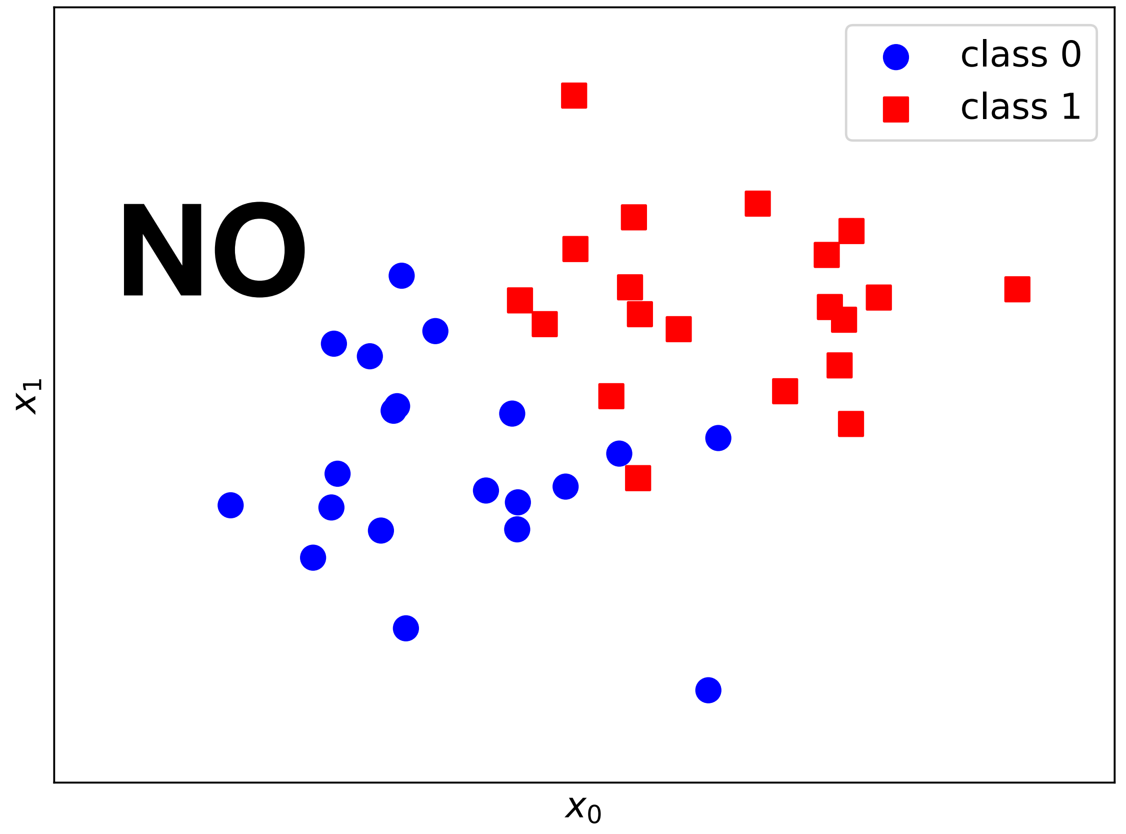


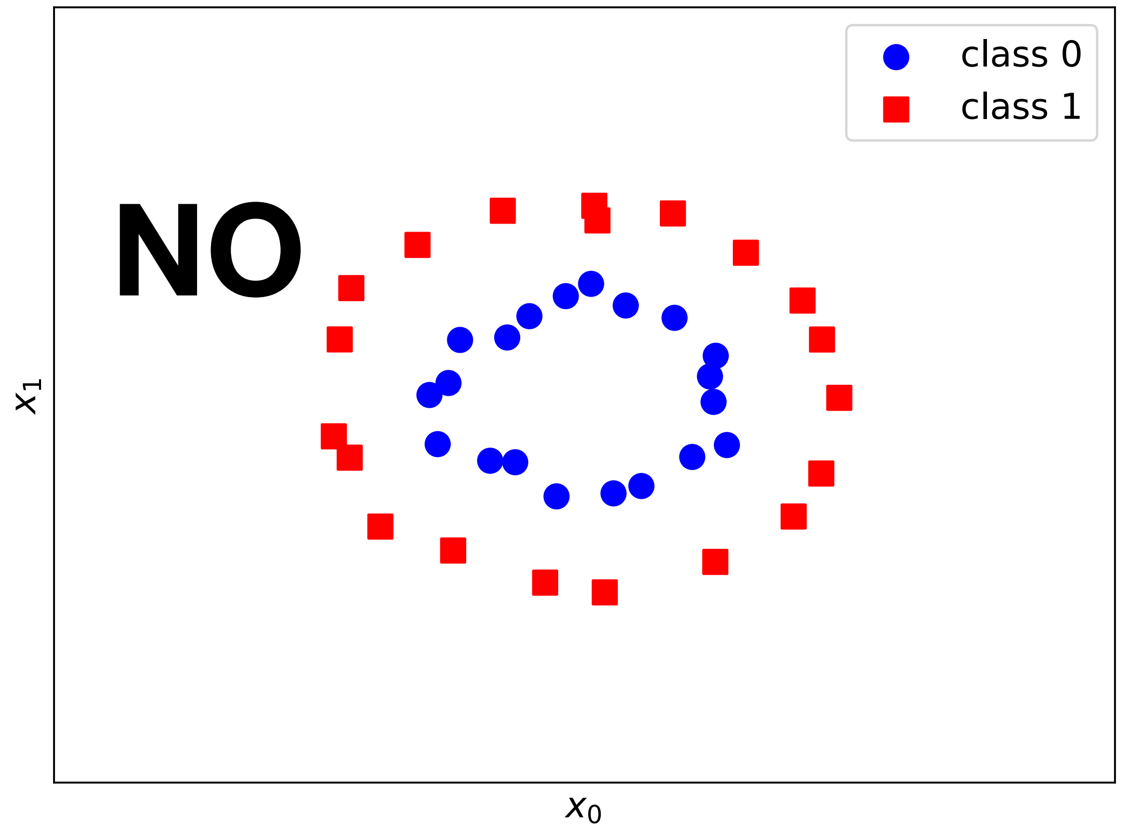
Linear separability

Our training data is linearly separable if we are able to draw a hyperplane that completely separates points from both classes









Learning the weights of linear classifiers

- For the classifier to be any good we have to learn $\mathbf{w}, b$ and on training data
- Once that is done we can **throw away the training set**
- We will now cover two different learning methods:
    1. The perceptron algorithm (**obsolete but foundational**)
    2. Logistic regression (**popular and used a lot**)

The Perceptron algorithm

The Perceptron algorithm for training a linear classifier

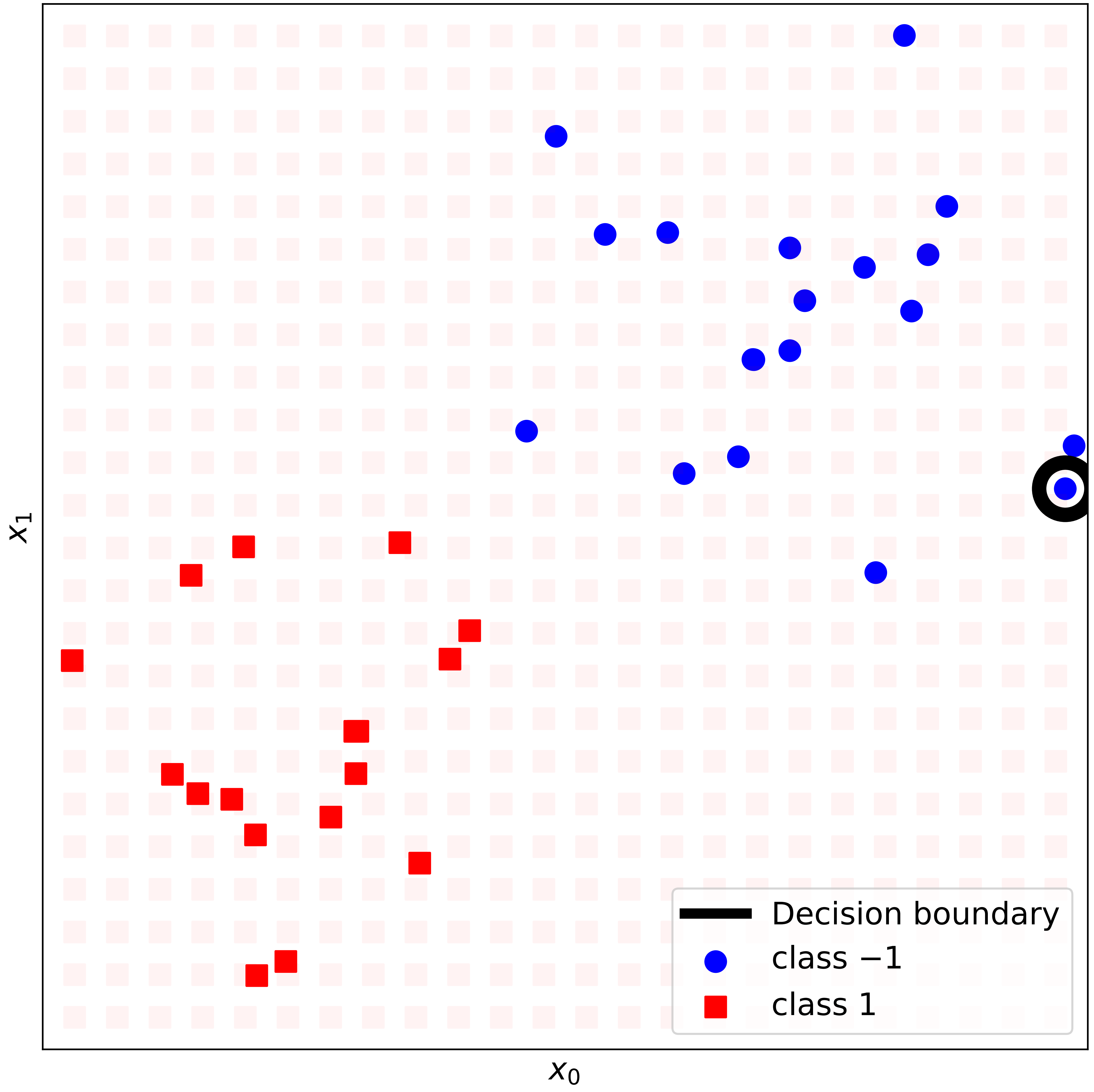
We have a linearly separable training set $\{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$ with $\mathbf{x} \in \mathbb{R}^D$ and $y \in \{-1,1\}$

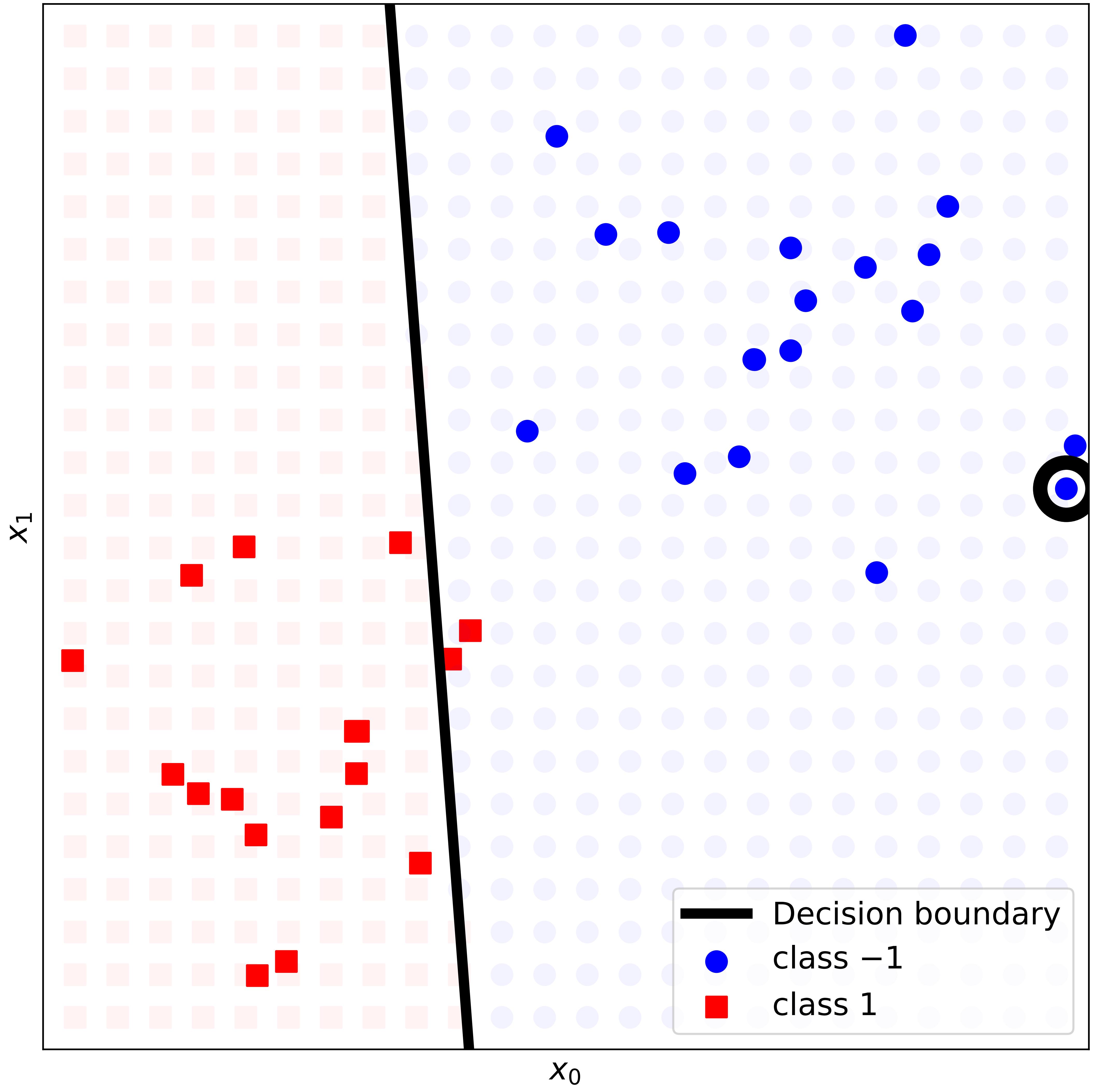
We want $f(\mathbf{x}^{(n)}) = \mathbf{w}^\top \mathbf{x}^{(n)} + b$ s.t. $\hat{y}^{(n)} = \begin{cases} 1 & \text{if } f(\mathbf{x}^{(n)}) \geq 0 \\ -1 & \text{if } f(\mathbf{x}^{(n)}) < 0 \end{cases} \quad \forall n$

Note classes are -1 and 1 here for mathematical ease

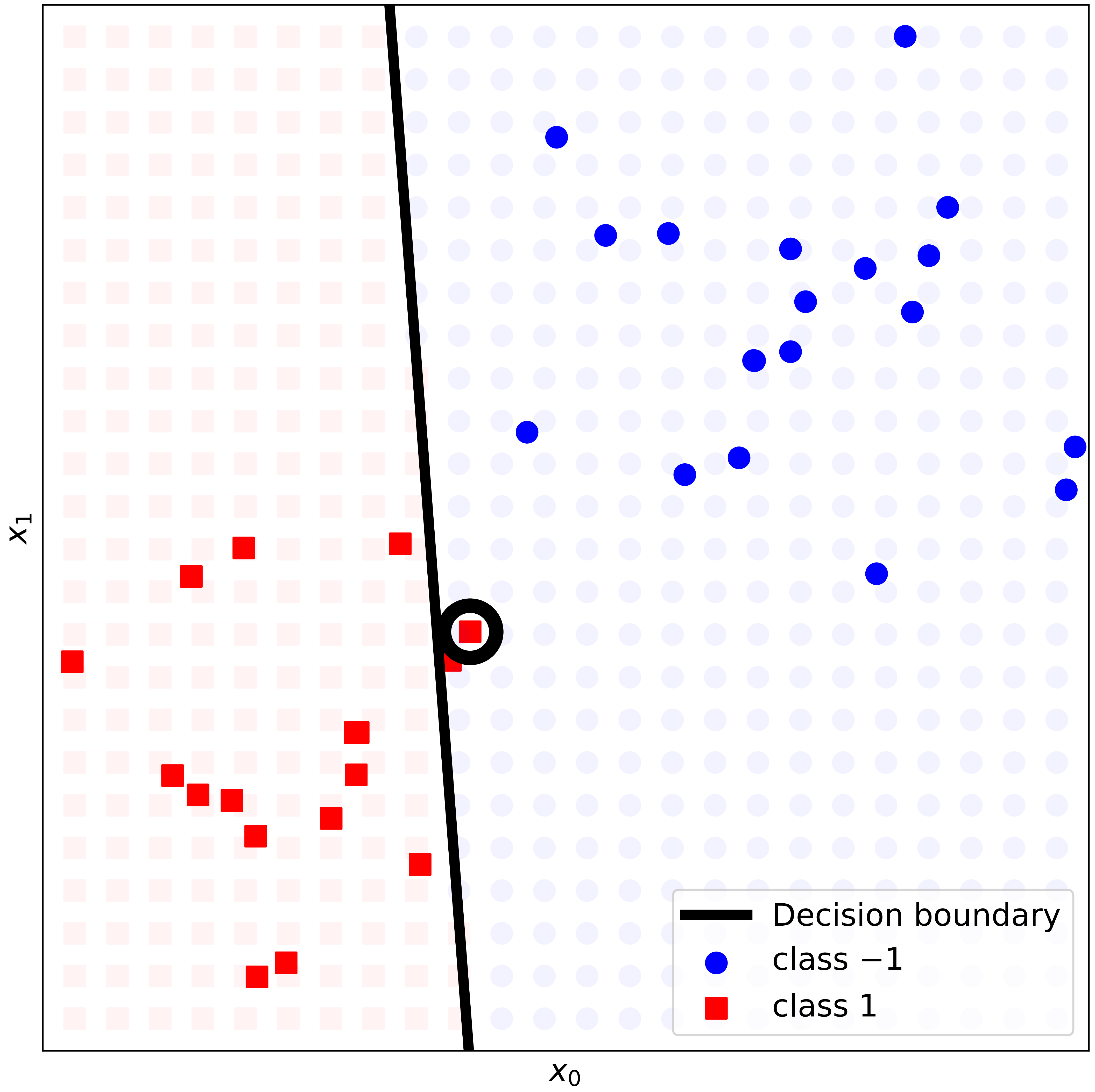
- Initialise $\mathbf{w}$ as $\mathbf{w} = \mathbf{0}$ and b as $b = 0$
- Shuffle, then cycle through $\{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$
    - If $\mathbf{x}^{(n)}$ is misclassified then $\mathbf{w} \leftarrow \mathbf{w} + \alpha\, y^{(n)}\mathbf{x}^{(n)}$, $b \leftarrow b + \alpha\, y^{(n)}$
- Stop when all the data is classified correctly

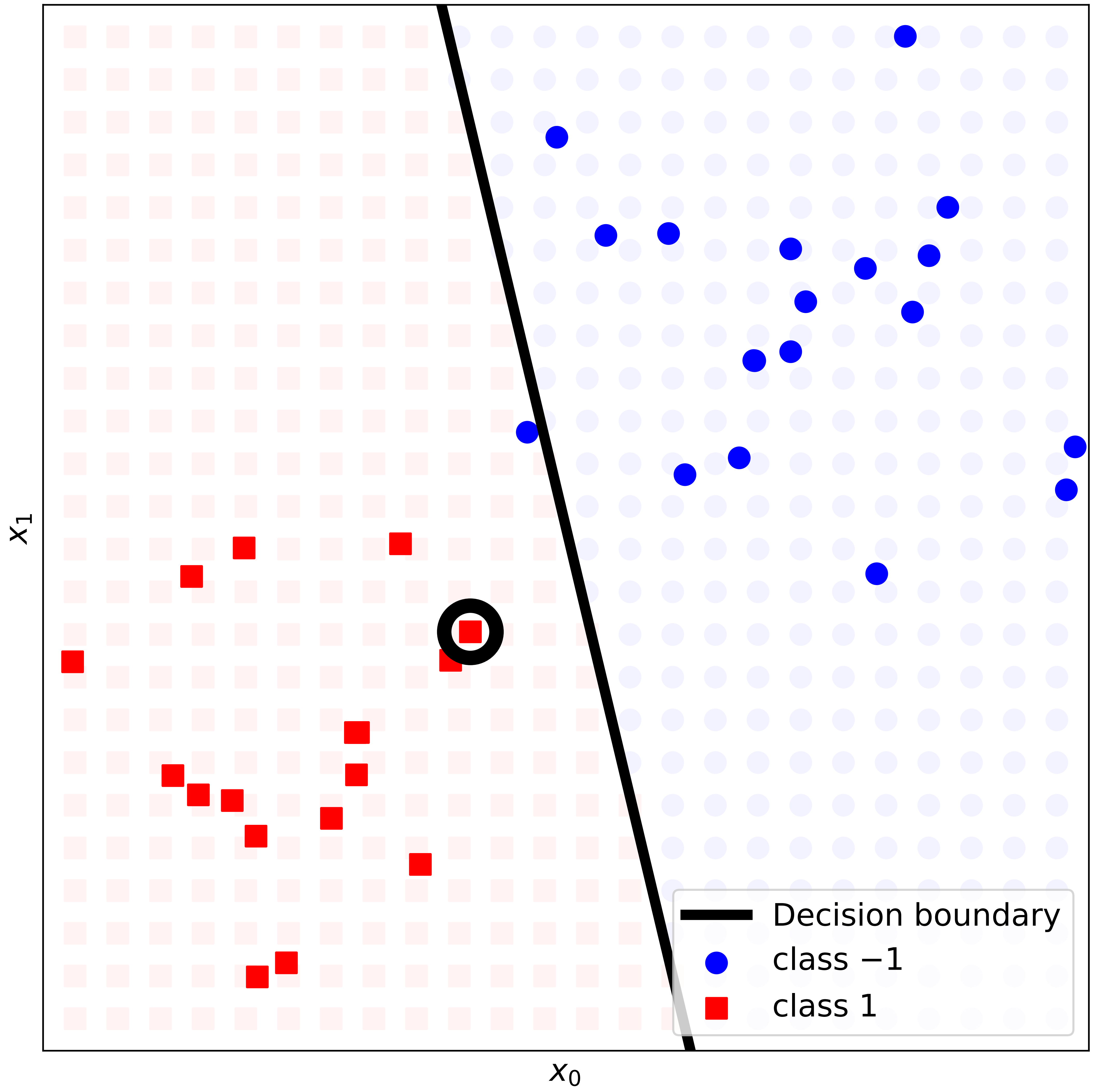
Perceptron Learning: Update 0



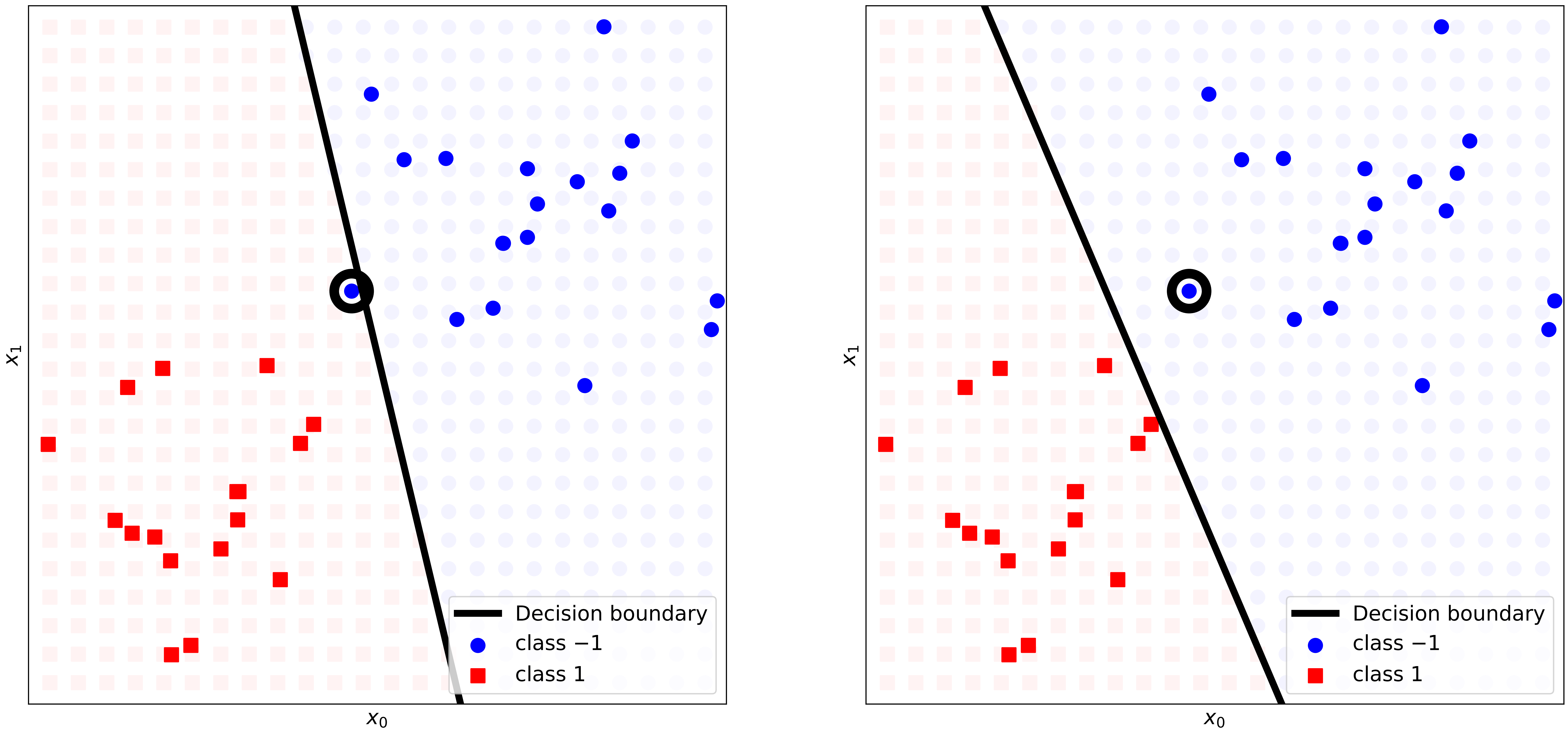


Perceptron Learning: Update 1





Perceptron Learning: Update 2



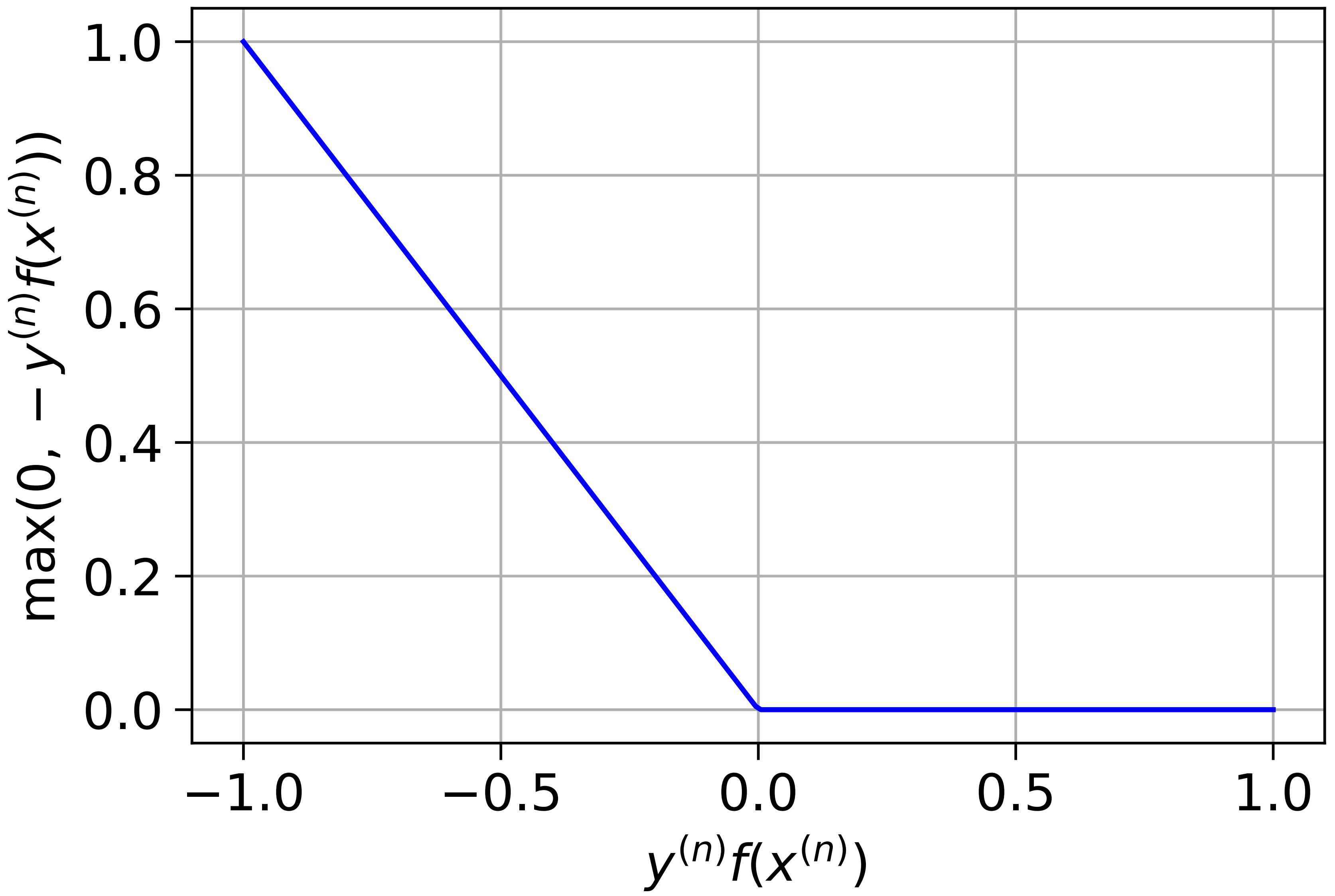
Perceptron loss

- Can we phrase the perceptron algorithm as minimising some loss function?
- Yes. It is minimising a **hinge loss** using **stochastic gradient descent (SGD)**

$$L_{hinge} = \frac{1}{N} \sum_{n} \max\left(0, -y^{(n)} f(\mathbf{x}^{(n)})\right)$$

The optimisation problem is to solve

$$\underset{\mathbf{w},b}{\text{minimise}} \; L_{hinge}$$

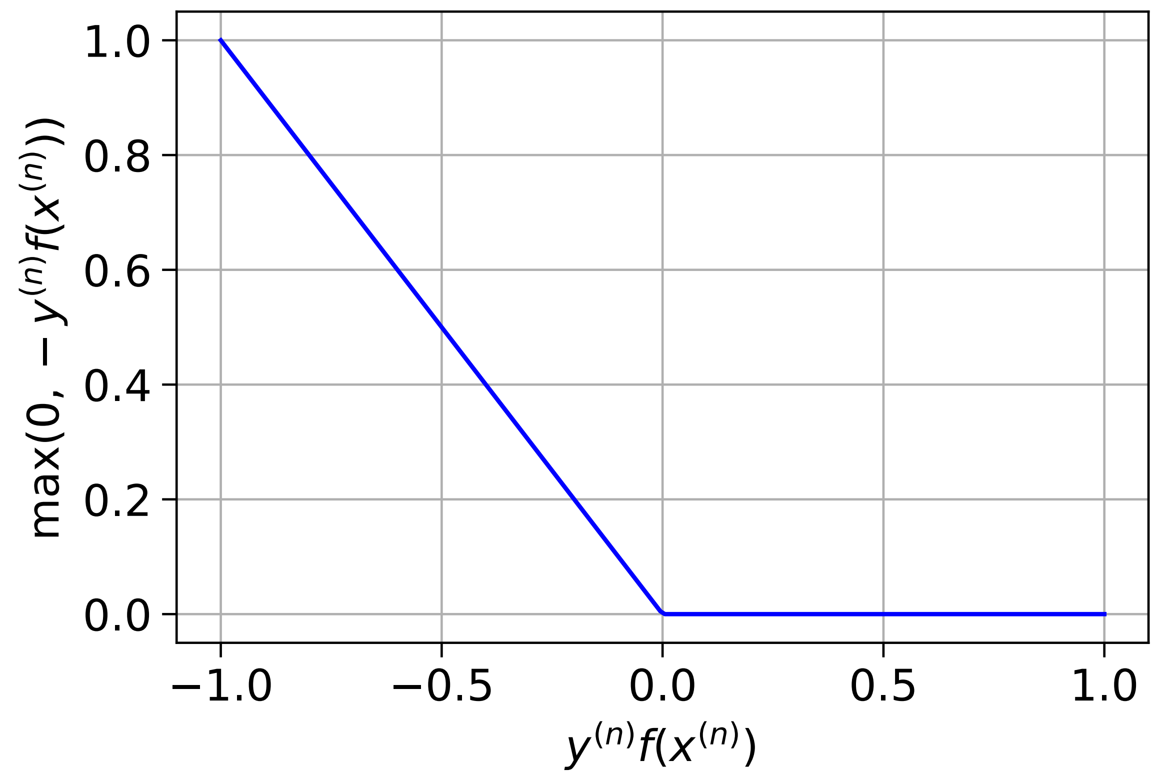


Stochastic gradient descent (SGD)

- SGD is identical to GD except at each step the gradient is computed on a random subset of the data; for the perceptron this is a single data point

- Recall in GD the update is $\mathbf{w}_{t=i+1} = \mathbf{w}_{t=i} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_{t=i})$

$$\tilde{L}_{hinge} = \max\left(0, -y^{(n)} f(\mathbf{x}^{(n)})\right) = \max\left(0, -y^{(n)}(\mathbf{w}^\top \mathbf{x}^{(n)} + b)\right)$$



$$\nabla_{\mathbf{w}} \tilde{L}_{hinge} = \begin{cases} -y^{(n)}\mathbf{x}^{(n)} & \text{if } y^{(n)} f(\mathbf{x}^{(n)}) < 0 \\ 0 & \text{if } y^{(n)} f(\mathbf{x}^{(n)}) > 0 \end{cases}$$

- Plugging into update gives

$$\mathbf{w}_{t=i+1} = \mathbf{w}_{t=i} + \begin{cases} \alpha y^{(n)}\mathbf{x}^{(n)} & \text{if } y^{(n)} f(\mathbf{x}^{(n)}) < 0 \\ 0 & \text{if } y^{(n)} f(\mathbf{x}^{(n)}) > 0 \end{cases}$$

This is the same as the perceptron weight update!

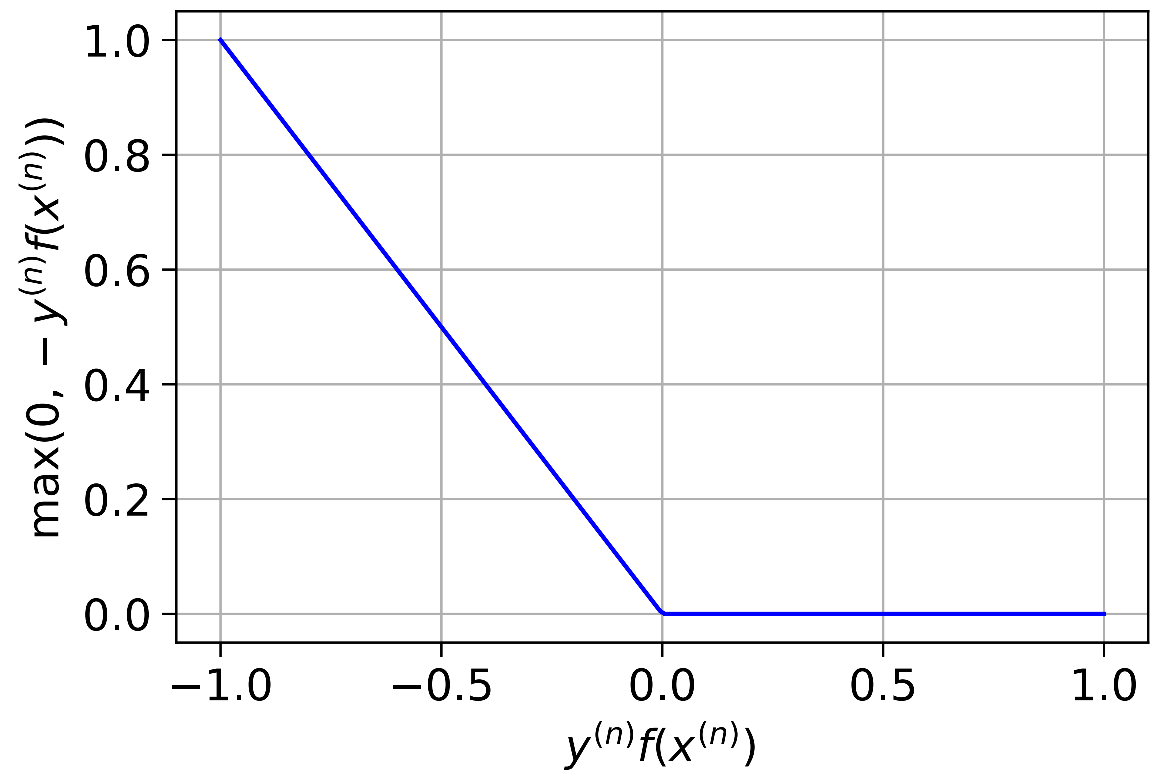
We also have a bias!

- The bias needs to be updated too. This happens alongside each weight update

- The update for the bias is $b_{t=i+1} = b_{t=i} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_{t=i})$

$$\tilde{L}_{hinge} = \max\left(0, -y^{(n)}(\mathbf{w}^\top \mathbf{x}^{(n)} + b)\right)$$

$$\nabla_b \tilde{L}_{hinge} = \begin{cases} -y^{(n)} & \text{if } y^{(n)} f(\mathbf{x}^{(n)}) < 0 \\ 0 & \text{if } y^{(n)} f(\mathbf{x}^{(n)}) > 0 \end{cases}$$



- Plugging into update gives

$$b_{t=i+1} = b_{t=i} + \begin{cases} \alpha y^{(n)} & \text{if } y^{(n)} f(\mathbf{x}^{(n)}) < 0 \\ 0 & \text{if } y^{(n)} f(\mathbf{x}^{(n)}) > 0 \end{cases}$$

This is the the same as the perceptron bias update!

Logistic Regression

Classification as regression

- Consider a training set $\{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$ where $\mathbf{x} \in \mathbb{R}^D$ and $y \in \{0,1\}$
- Let's treat y as continuous $y \in \mathbb{R}^1$: *it just happens to be 0/1 for training data*
- We can perform linear regression $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ to predict this "continuous" label
- This can be achieved by e.g. minimising $L_{MSE} = \dfrac{1}{N} \sum_n (y^{(n)} - \mathbf{w}^\top \mathbf{x}^{(n)} - b)^2$
- Can we predict something more meaningful?

Logistic Regression

- Probabilities are meaningful as they quantify uncertainty
- We want to predict $p(y = 1 \mid \mathbf{x})$: the probability that $\mathbf{x}$ belongs to class 1
- We can't predict this with our linear model $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ however
- This is because probabilities must lie between 0 and 1 and $f(\mathbf{x})$ is unbounded
- Let's instead predict an unbounded quantity that is related to $p(y = 1 \mid \mathbf{x})$

$$f(\mathbf{x}) = \log \frac{p(y = 1 \mid \mathbf{x})}{1 - p(y = 1 \mid \mathbf{x})}$$

The log-odds, or logit

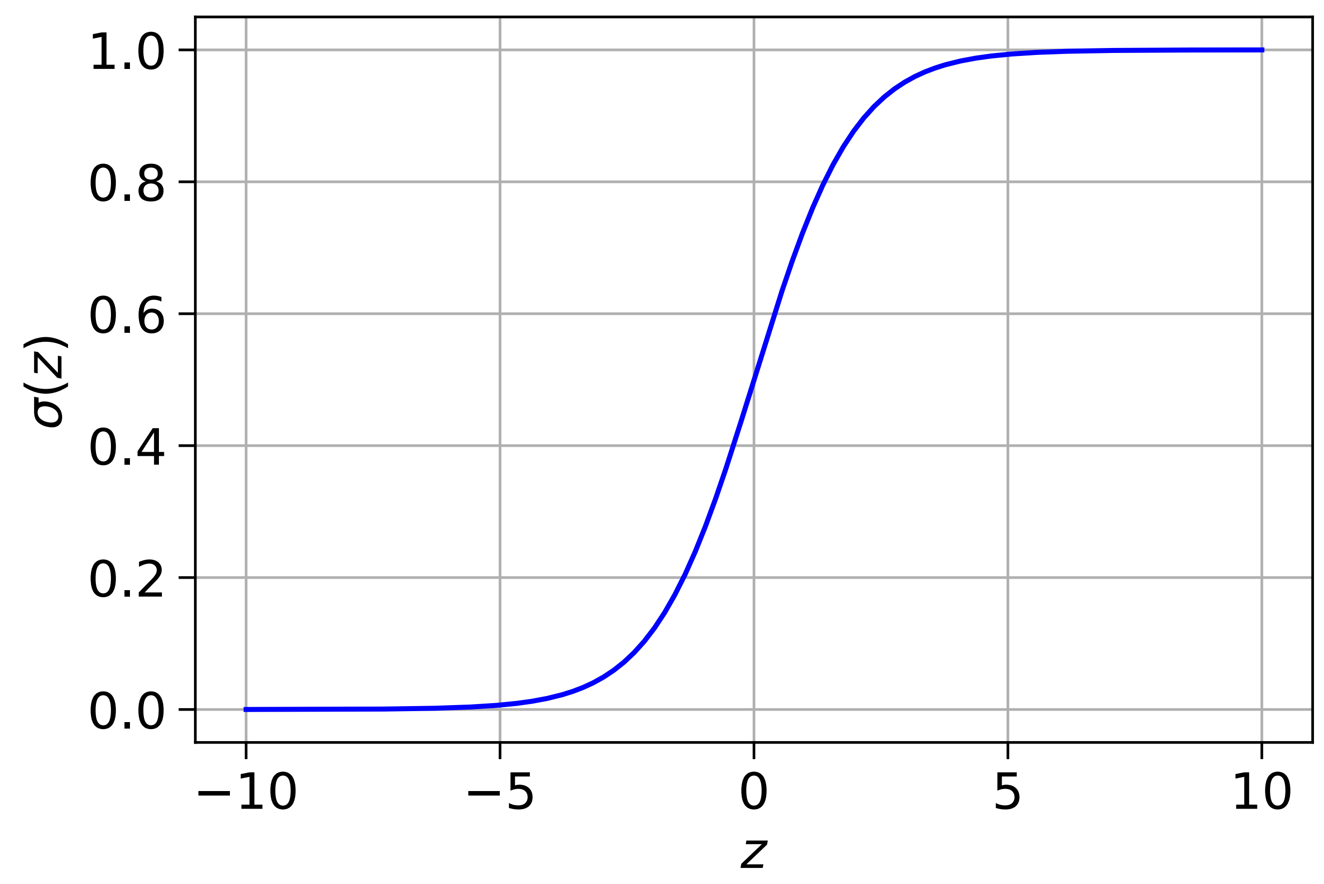
The sigmoid function

- In logistic regression our model predicts log-odds from data

$$f(\mathbf{x}) = \log \frac{p(y=1 \mid \mathbf{x})}{1 - p(y=1 \mid \mathbf{x})}$$

- We can rearrange to express $p(y=1 \mid \mathbf{x})$ in terms of log-odds

$$p(y=1 \mid \mathbf{x}) = \frac{1}{1+e^{-f(\mathbf{x})}} = \sigma(f(\mathbf{x})) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$$



σ **is the sigmoid function.**
It squashes numbers to be between 0 and 1

Making decisions

- We can convert log-odds to probabilities through $p(y = 1 \mid \mathbf{x}) = \sigma(f(\mathbf{x}))$
- It follows that $p(y = 0 \mid \mathbf{x}) = 1 - \sigma(f(\mathbf{x}))$ as there are only two classes
- How do we make a class prediction $\hat{y}$?
- The obvious approach is $\hat{y} = \begin{cases} 1 & \text{if } p(y = 1 \mid \mathbf{x}) \geq 0.5 \\ 0 & \text{if } p(y = 1 \mid \mathbf{x}) < 0.5 \end{cases}$
- But what if $\mathbf{x}$ represents a patient, class 1/0 are cancer/not-cancer diagnoses and you get $p(y = 1 \mid \mathbf{x}) = 0.49$?

Learning weights using Maximum likelihood estimation (MLE)

- We can write $p(y \mid \mathbf{x}) = \sigma(f(\mathbf{x}))^{y}(1 - \sigma(f(\mathbf{x}))^{1-y}$
- The likelihood of your training data is a sensible quantity to maximise

$$\ell = \prod_n p(y^{(n)} \mid \mathbf{x}^{(n)}) = \prod_n \sigma(f(\mathbf{x}^{(n)}))^{y^{(n)}}(1 - \sigma(f(\mathbf{x}^{(n)}))^{1-y^{(n)}}$$

Independence assumption

- Maximising ℓ is the same as minimising $-\frac{1}{N}\log \ell$

$$-\frac{1}{N}\log \ell = -\frac{1}{N}\sum_n \left[y^{(n)} \log \sigma(f(\mathbf{x}^{(n)})) + (1 - y^{(n)})\log(1 - \sigma(f(\mathbf{x}^{(n)}))\right]$$

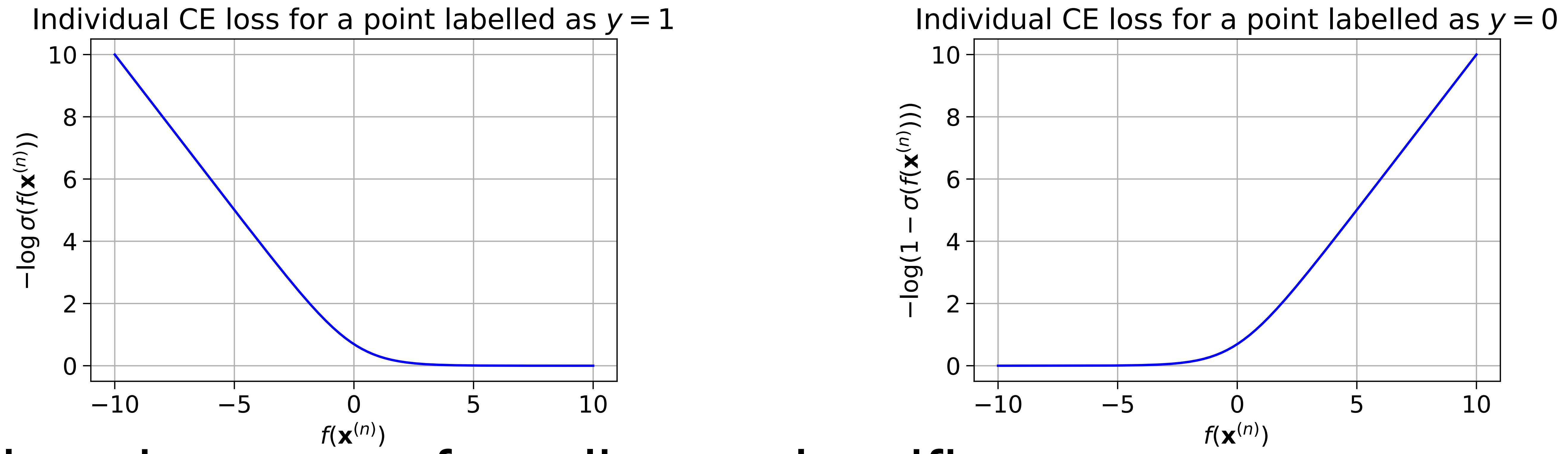
Cross-entropy loss

$$L_{CE} = -\frac{1}{N}\sum_{n}\left[y^{(n)}\log\sigma(f(\mathbf{x}^{(n)})) + (1 - y^{(n)})\log(1 - \sigma(f(\mathbf{x}^{(n)}))\right]$$

- This quantity is the cross entropy loss (averaged across data item)
- Minimising this is equivalent to maximising likelihood
- Cross-entropy is a quantity that crops up in information theory
- It measures how much the probabilities produced by our model differ from the true probabilities (so low = good)

Cross-entropy loss

$$L_{CE} = -\frac{1}{N}\sum_n \left[y^{(n)} \log \sigma(f(\mathbf{x}^{(n)})) + (1 - y^{(n)})\log(1 - \sigma(f(\mathbf{x}^{(n)}))\right]$$



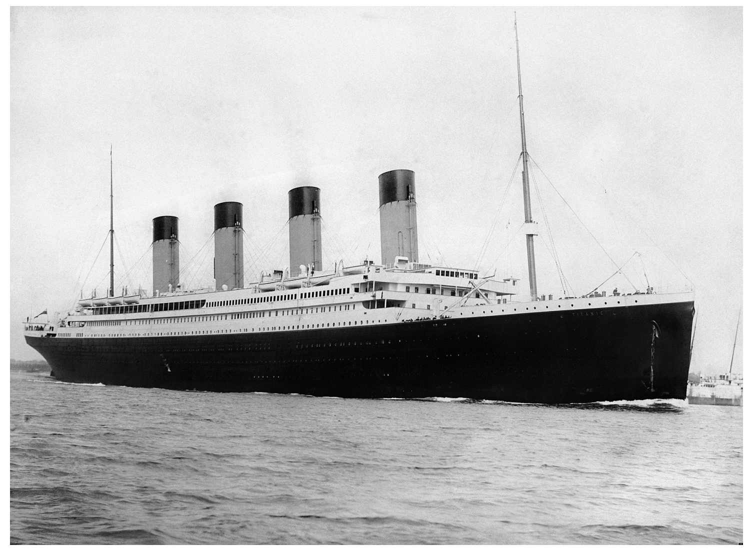
- This loss is convex for a linear classifier
- We can use e.g. GD or SGD to to solve $\underset{\mathbf{w},b}{\text{minimise}}\ L_{CE}$ using:

$$\nabla_{\mathbf{w}} L_{CE} = -\frac{1}{N}\sum_n \left(y^{(n)} - \sigma(f(\mathbf{x}))\right)\mathbf{x}^{(n)} \text{ and } \nabla_b L_{CE} = -\frac{1}{N}\sum_n \left(y^{(n)} - \sigma(f(\mathbf{x}))\right)$$

Decision boundary for logistic regression

- We have placed a sigmoid function over a linear model to turn its log-odds outputs into probabilities: $p(y=1\,|\,\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$
- Classifications are usually made with $\hat{y} = \begin{cases} 1 & \text{if } p(y=1\,|\,\mathbf{x}) \geq 0.5 \\ 0 & \text{if } p(y=1\,|\,\mathbf{x}) < 0.5 \end{cases}$
- The decision boundary is at $p(y=1\,|\,\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b) = 0.5$
- $\sigma(\mathbf{w}^\top \mathbf{x} + b) = 0.5$ when $\mathbf{w}^\top \mathbf{x} + b = 0$ which is a still a **hyperplane**
- We could rewrite the classification rule as $\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) \geq 0 \\ 0 & \text{if } f(\mathbf{x}) < 0 \end{cases}$

Titanic Dataset



- We can use historical data about passengers to learn a linear classifier to predict survival using logistic regression

Pclass	Sex	Age	SibSp	Parch	Fare	Survived
3	0	22.0	1	0	7.2500	0
1	1	38.0	1	0	71.2833	1
3	1	26.0	0	0	7.9250	1
1	1	35.0	1	0	53.1000	1
3	0	35.0	0	0	8.0500	0
...	...	...	...	...	...	...
3	1	39.0	0	5	29.1250	0
2	0	27.0	0	0	13.0000	0
1	1	19.0	0	0	30.0000	1
1	0	26.0	0	0	30.0000	1
3	0	32.0	0	0	7.7500	0

For "Sex", *male* has been mapped to 0 and *female* to 1 arbitrarily

- If we standardise data then the weights we learn are interpretable

$$\mathbf{w} = [\overset{\text{Pclass}}{-0.97} \quad \overset{\text{Sex}}{1.27} \quad \overset{\text{Age}}{-0.52} \quad \overset{\text{SibSp}}{-0.27} \quad \overset{\text{Parch}}{-0.03} \quad \overset{\text{Fare}}{0.16}]^{\top}$$

- Survival more probable for people who are in first class, female, young

Gets 80% on held-out data so is a reasonable model

Perceptron vs. Logistic regression

- Both give linear classifiers $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$
- The main difference is in the classification loss used for optimisation
- In logistic regression the quantities being predicted, and the loss are meaningful
- We can add a regularisation term to either loss as before
- This could be L2 or L1 (or whatever!). L2 is most common

$$L_{total} = \underbrace{L_{clf}}_{classification} + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{regularisation}$$

Remember that the bias goes unregularised

Multi-class classification with linear classifiers

We have $\{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$ with $y \in \mathbb{Z}^{+}_{<K} = \{0,1,\ldots,K-1\}$.

There are three different approaches to solving this:

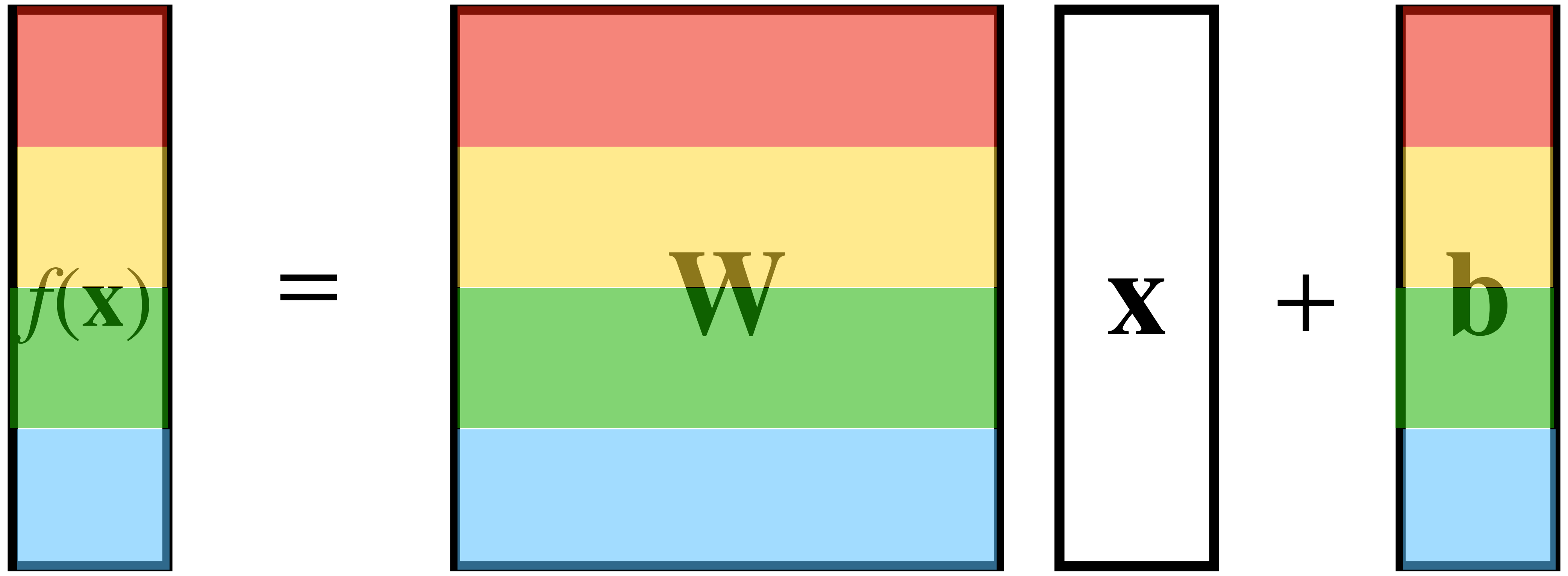
1. We could learn K one-vs-rest classifiers: $f_0(\mathbf{x}), f_1(\mathbf{x}), \ldots, f_{K-1}(\mathbf{x})$ and classify points according to the highest score

2. We could learn $(K(K-1))/2$ one-vs-one classifiers and classify points according to the majority vote

3. We could make our classifier output a **vector** where each element is a score for a different class and select the class with the highest score

Multi-class linear classifiers

- In the binary case $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ where $\mathbf{x} \in \mathbb{R}^D$ and $f(\mathbf{x}) \in \mathbb{R}^1$
- For our classifier to output a score for each of K classes we can:
  1. Replace the vector $\mathbf{w} \in \mathbb{R}^D$ with a matrix $\mathbf{W} \in \mathbb{R}^{K \times D}$
  2. Replace the bias vector $b \in \mathbb{R}^1$ with a vector $\mathbf{b} \in \mathbb{R}^K$
- This gives us $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$ where $f(\mathbf{x}) \in \mathbb{R}^K$



Like having
K classifiers
side-by-side

Multinomial logistic regression

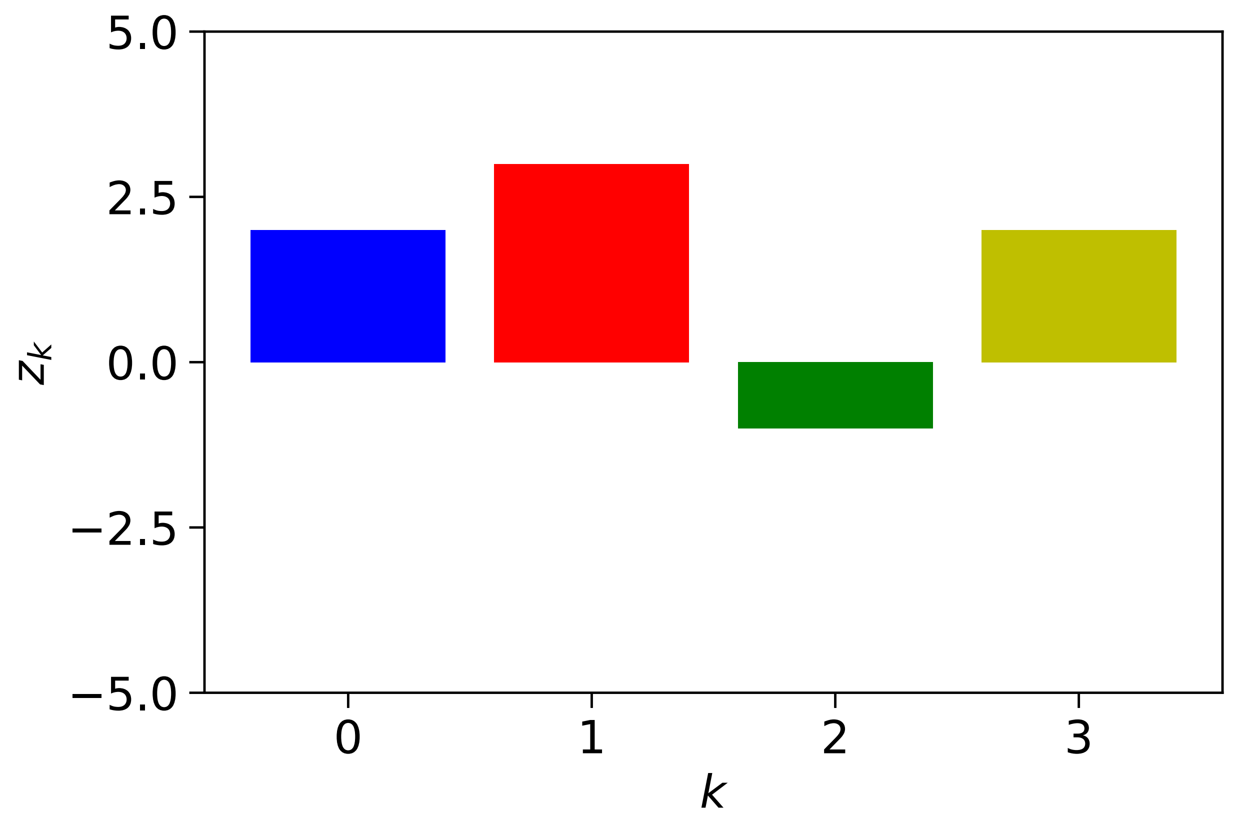
- Logistic regression naturally extends to multi-class problems
- In the binary setting, we only had to consider $p(y = 1 \mid \mathbf{x})$ in terms of $f(\mathbf{x})$
- In the multi-class setting we need to consider all the different probabilities
- Let's store the probabilities in a vector $\mathbf{p}$
- We will write $\mathbf{p}$ as some function S of $f(\mathbf{x})$ — the vector of logits

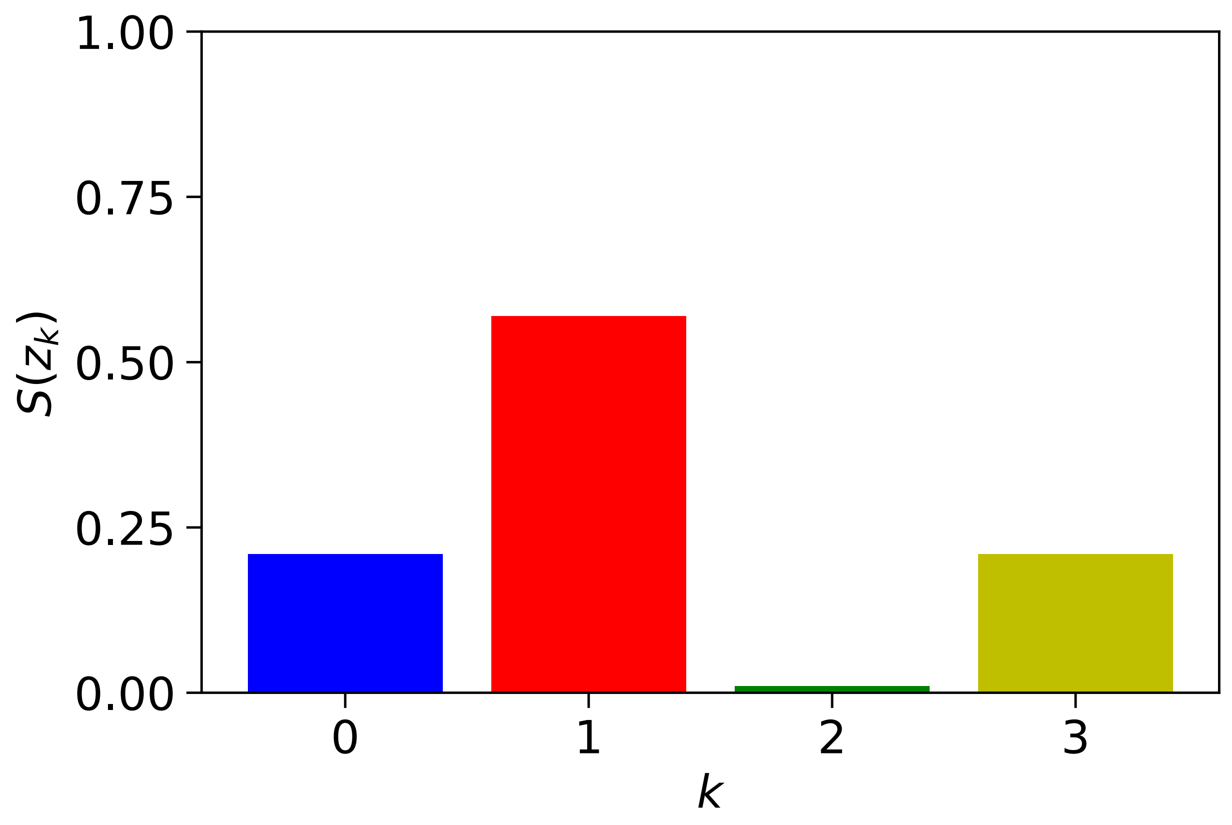
$$\mathbf{p} = \begin{bmatrix} p(y = 0 \mid \mathbf{x}) \\ p(y = 1 \mid \mathbf{x}) \\ p(y = 2 \mid \mathbf{x}) \\ \vdots \\ p(y = K - 1 \mid \mathbf{x}) \end{bmatrix} = S(f(\mathbf{x}))$$

Softmax

- $\mathbf{p}$ must sum to 1 so we need a function that normalises $f(\mathbf{x})$
- We will use the softmax function S which squashes $f(\mathbf{x})$ so it sums to 1

$$S(\mathbf{z}) = S\left(\begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{K-1} \end{bmatrix}\right) = \begin{bmatrix} \frac{\exp z_0}{\sum_{k=0}^{K-1} \exp z_k} \\ \frac{\exp z_1}{\sum_{k=0}^{K-1} \exp z_k} \\ \vdots \\ \frac{\exp z_{K-1}}{\sum_{k=0}^{K-1} \exp z_k} \end{bmatrix}$$





Learning for multinominal logistic regression

- We can minimise cross-entropy $L_{CE} = -\frac{1}{N}\sum_{n} \mathbf{y}^{(n)} \log \mathbf{p}^{(n)}$ wrt. $\mathbf{W}$ and $\mathbf{b}$
- Here, $\mathbf{y} \in \mathbb{R}^K$ is a one-hot encoding of y which is 1 for the element corresponding to class k and zero elsewhere
- e.g. for $K = 6$ and $y^{(t)} = 2$ we have $\mathbf{y}^{(t)} = [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0]^\top$
- We can use GD/SGD with $\nabla_{\mathbf{W}} L_{CE}$ and $\nabla_{\mathbf{b}} L_{CE}$

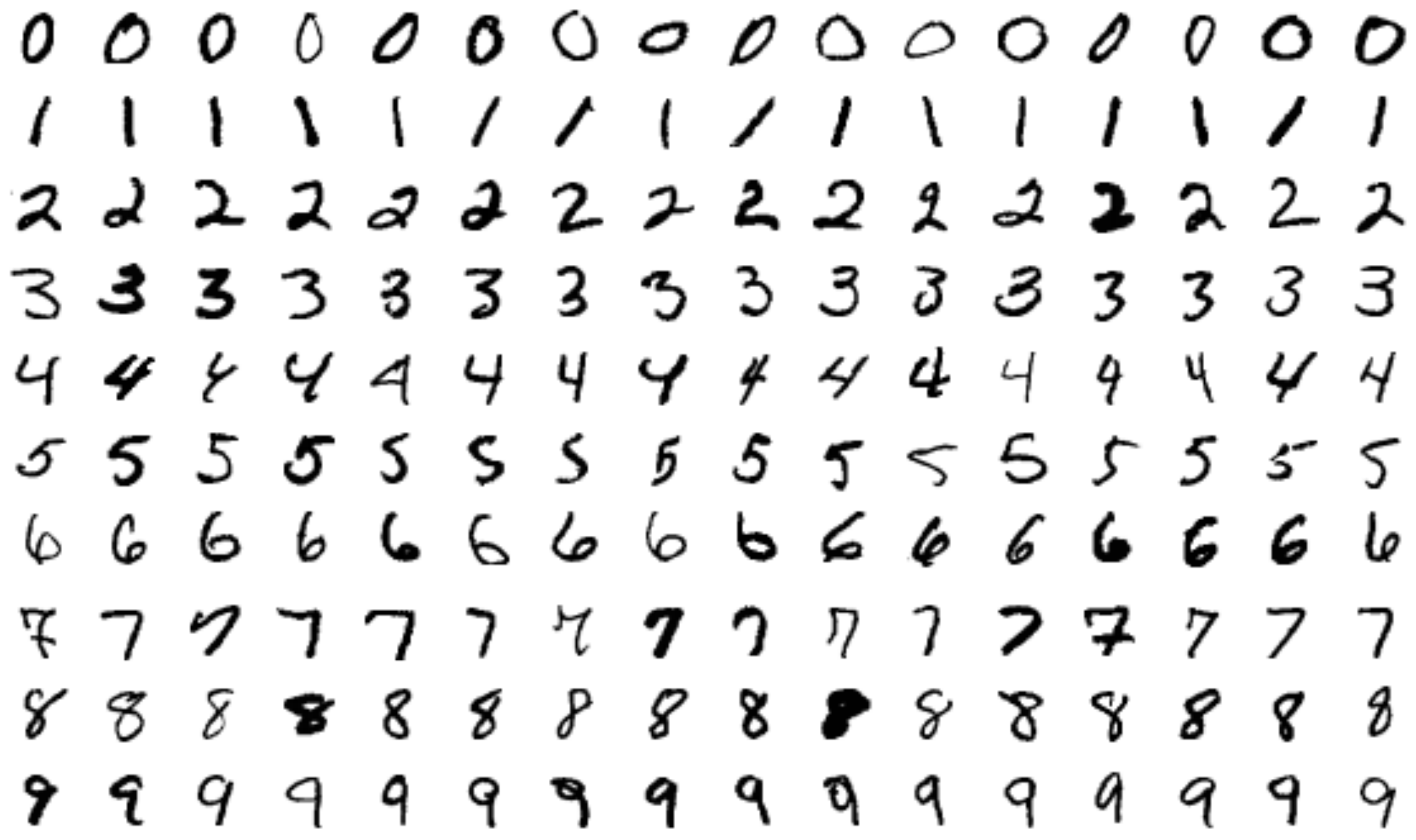
$$\nabla_{\mathbf{W}} L_{CE} = \frac{1}{N}\left[\sum_{n} \mathbf{x}_n(\mathbf{p}^{(n)} - \mathbf{y}^{(n)})^\top\right]^\top$$

See Murphy p346 for the $\nabla_{\mathbf{W}} L_{CE}$ derivation. There are differences in notation, and Murphy's gradient is the transpose of mine.

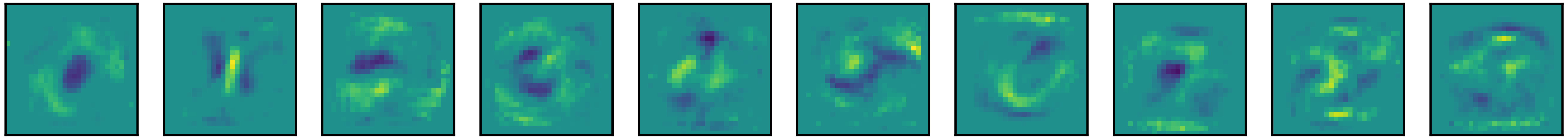
What is the shape of this matrix? Can you tell from sight what $\nabla_{\mathbf{b}} L_{CE}$ is?

Digit classification on MNIST

- MNIST dataset has 60k images (50k train, 10k test)
- Images are 28×28 so can vectorise to get $\mathbf{x} \in \mathbb{R}^{784}$
- Each image is labelled as a digit 0-9 so $y \in \mathbb{Z}^{+}_{<10}$
- Let's perform multinomial logistic regression with L1 regularisation
- Classify according to most probable class
- Test accuracy: 89.4% (or test error: 10.6%)



https://en.wikipedia.org/wiki/MNIST_database#/media/File:MnistExamples.png



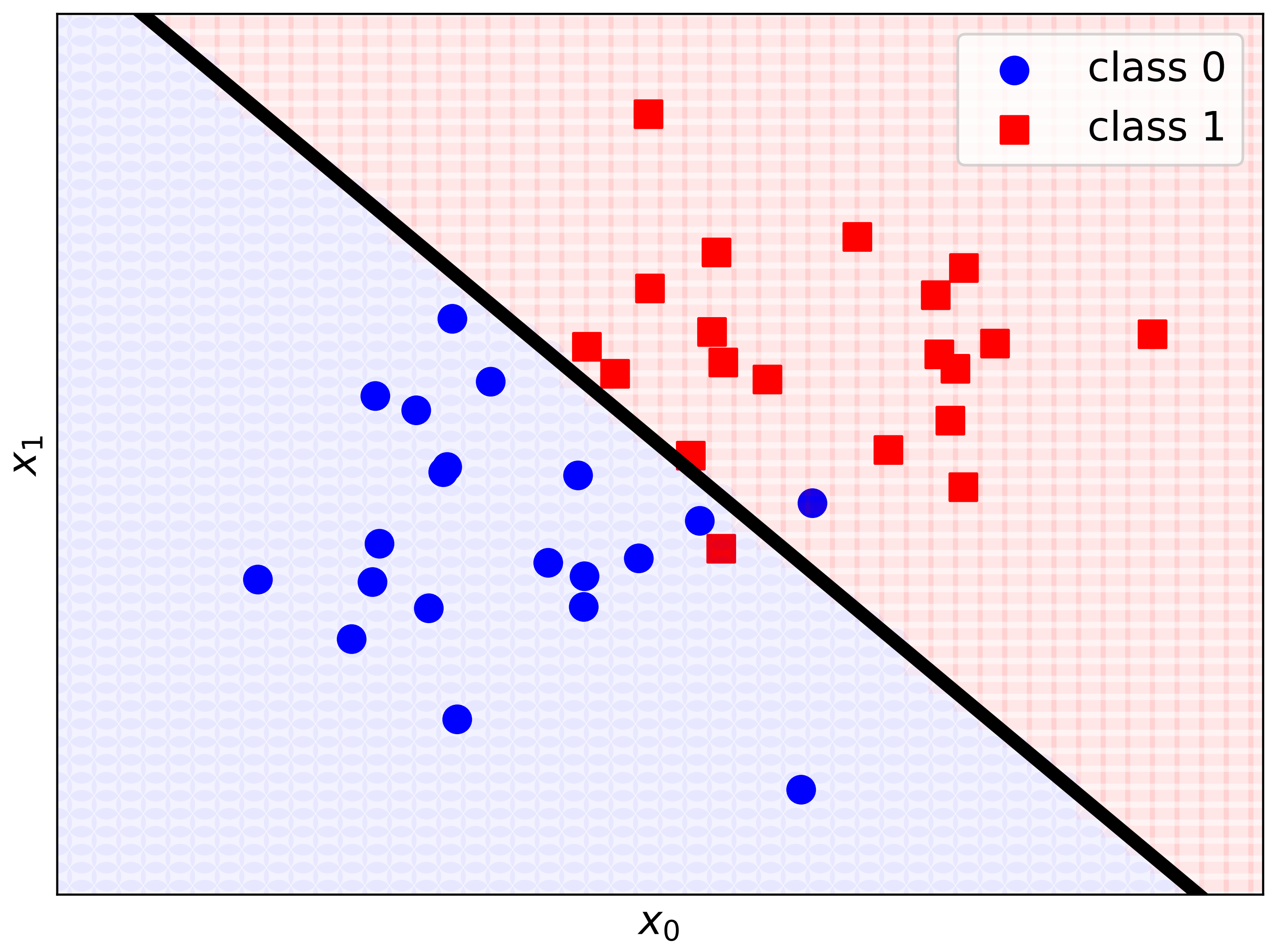
0 1 2 3 4 5 6 7 8 9

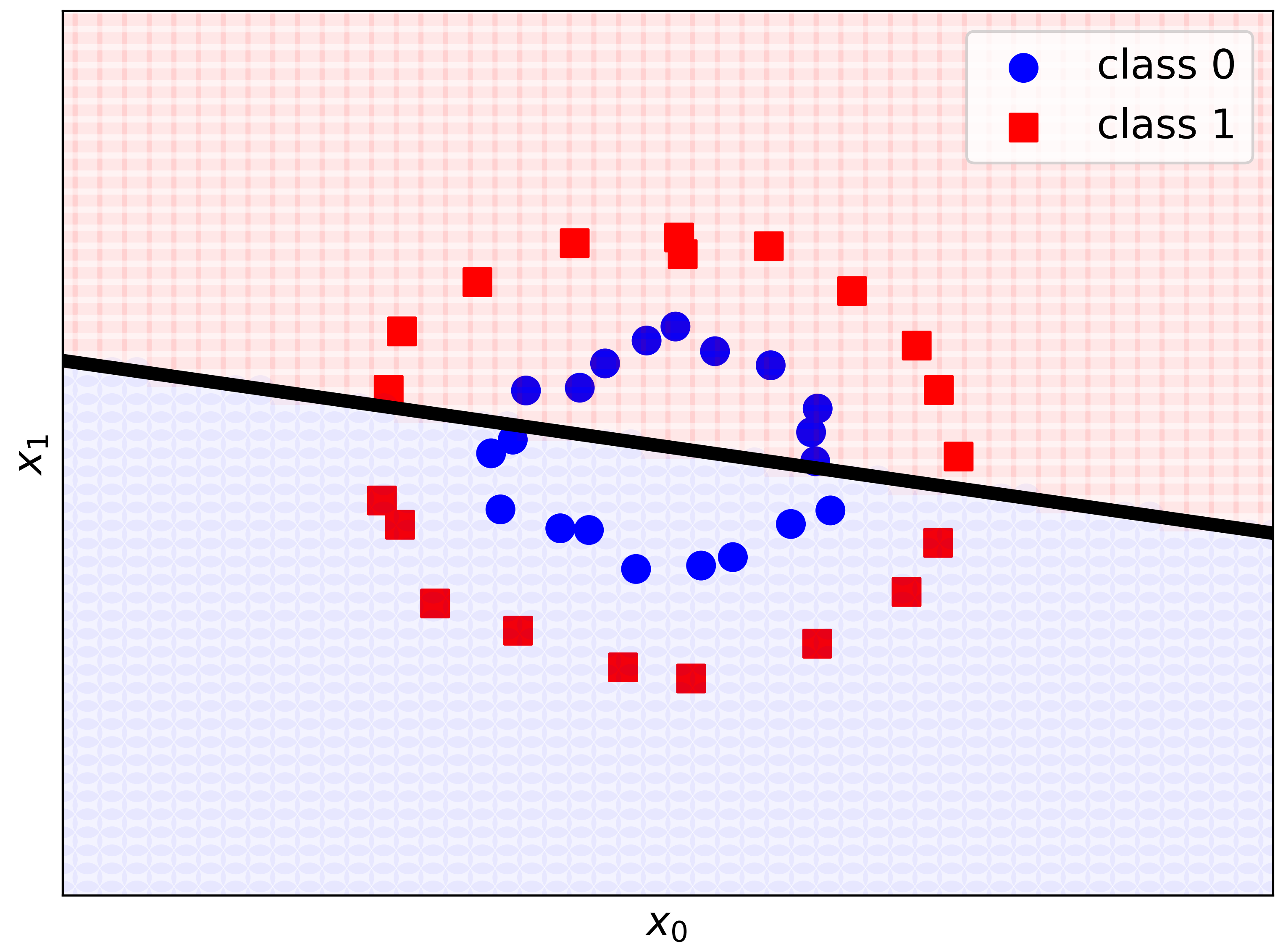
These are the columns of $\mathbf{W}$ displayed as images

Inspired by https://scikit-learn.org/stable/auto_examples/linear_model/plot_sparse_logistic_regression_mnist.html

A note on linear separability

- If training data isn't linearly separable, a linear classifier can't produce a decision boundary that perfectly classifies the training data
- You can still get good solutions if a hyperplane can separate most data
- If it can't then a linear classifier won't be any good :(





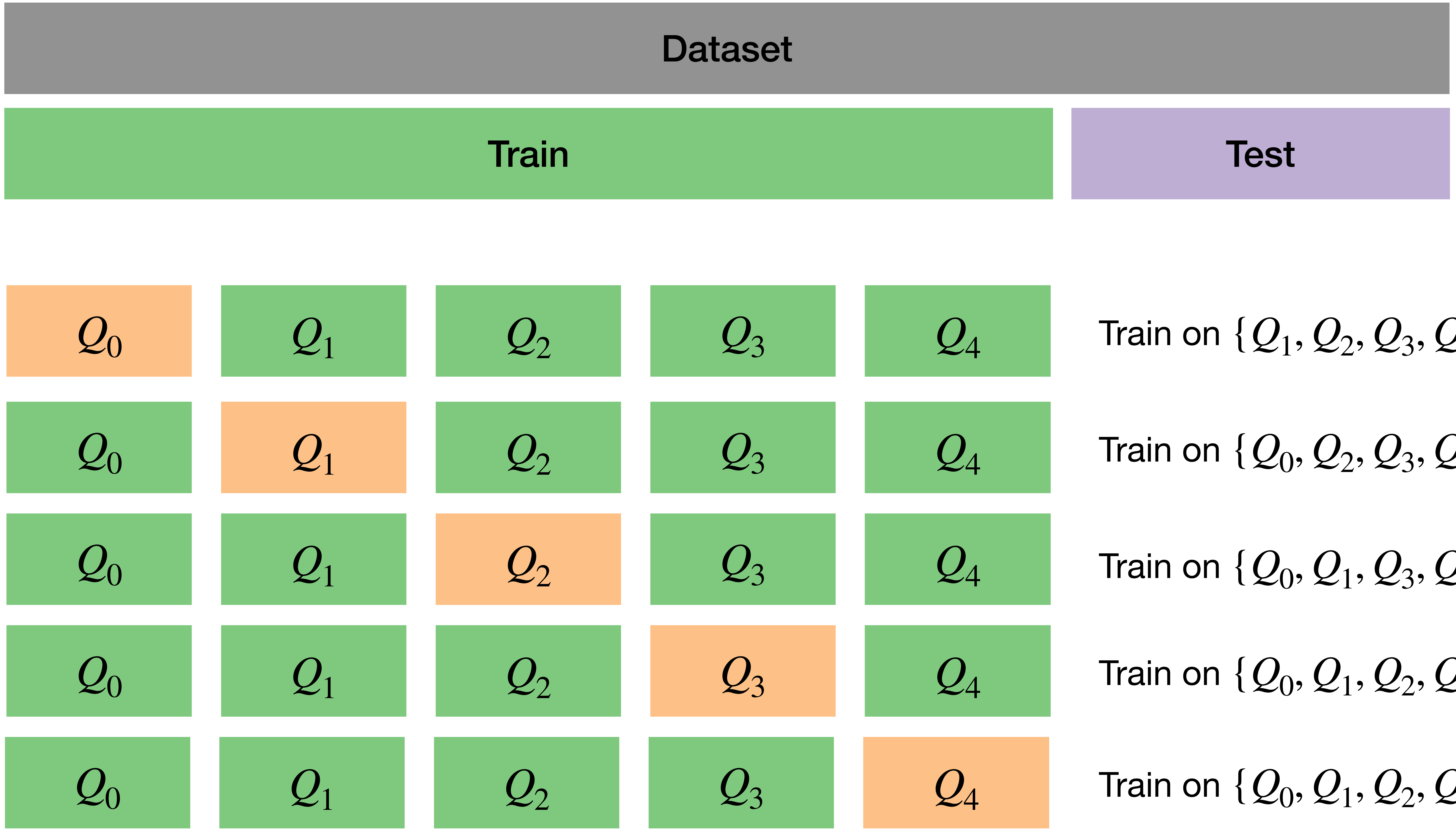
Hyperparameters again!

- We’ve seen learning rates, regularisation parameters… there will be more!
- We can tune these by:
    1. Creating a dedicated validation set, separate from train and test
    2. Grid searching across hyperparameters to maximise validation performance
- **But** this reduces the amount of data we have for actual training
- Also different train/val splits might give us noticeably different models

k-fold cross-validation

- We have been using the validation set for **model selection**
- Specifically, we have been training models with different hyperparameters and picking the one that maximises some score on validation
- We can instead perform model selection by looking at cross-validation performance. **This does not require us to have a dedicated validation set**

5-fold cross validation



Train on $\{Q_1, Q_2, Q_3, Q_4\}$, evaluate on Q_0

Train on $\{Q_0, Q_2, Q_3, Q_4\}$, evaluate on Q_1

Train on $\{Q_0, Q_1, Q_3, Q_4\}$, evaluate on Q_2

Train on $\{Q_0, Q_1, Q_2, Q_4\}$, evaluate on Q_3

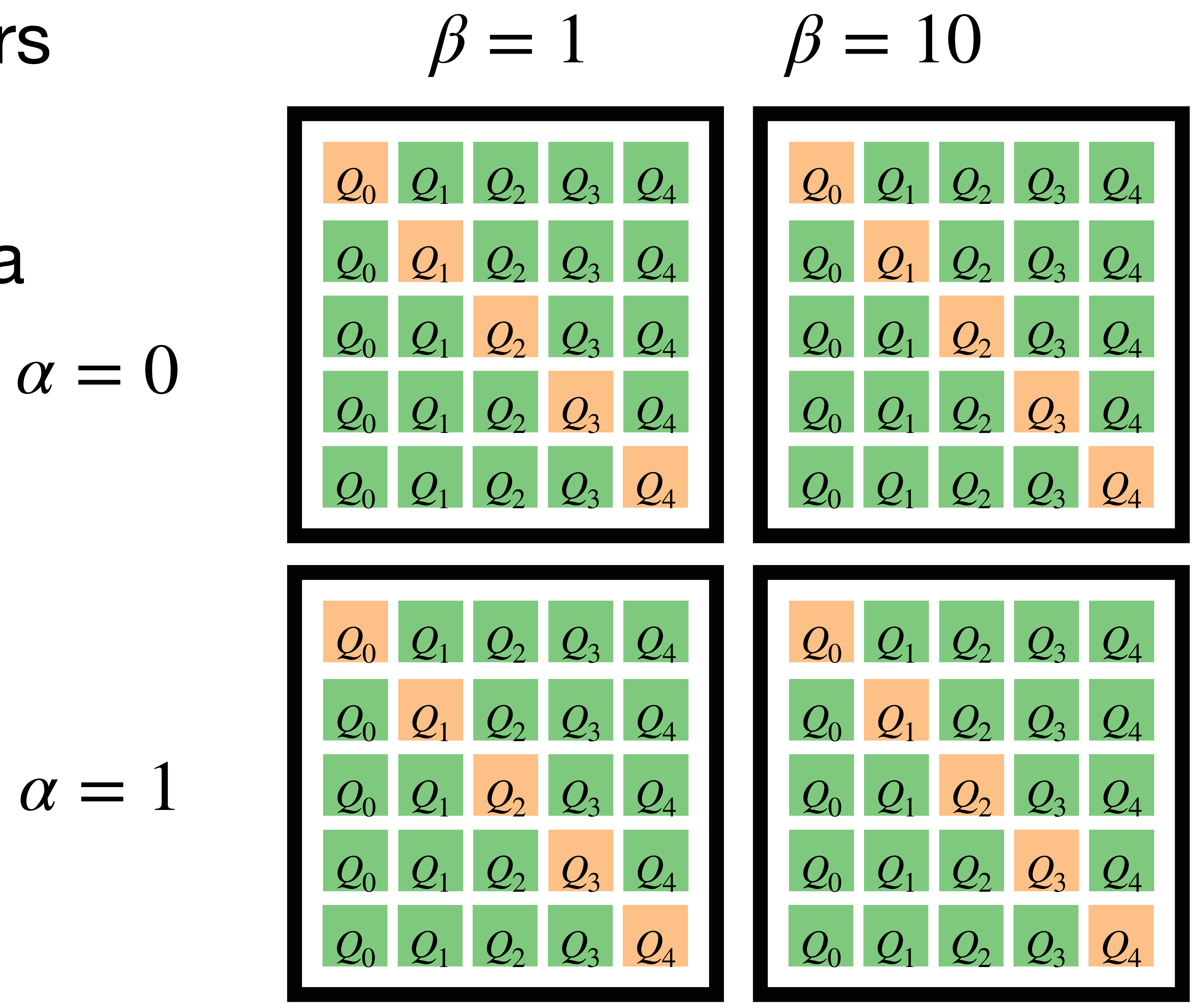
Train on $\{Q_0, Q_1, Q_2, Q_3\}$, evaluate on Q_4

Then take average performance across $Q_0, Q_1, Q_2, Q_3, Q_4, Q_5$

Inspiration: https://scikit-learn.org/stable/modules/cross_validation.html#multimetric-cross-validation

Grid search with k-fold cross validation

- Perform k-fold cross-validation for each element in the grid
- This gives you your tuned hyperparameters
- Then train a final model with these hyperparameters on all of the training data



Summary

- We have found out how to optimise the weights of linear classifiers for binary classification using the perceptron algorithm, and through logistic regression
- We have learnt how to modify linear classifiers for multi-class classification
- We have seen some failure modes of linear classifiers applied directly to data
- We have looked at cross-validation as an alternative to having a dedicated validation set, and how we can combine it with grid search for tuning