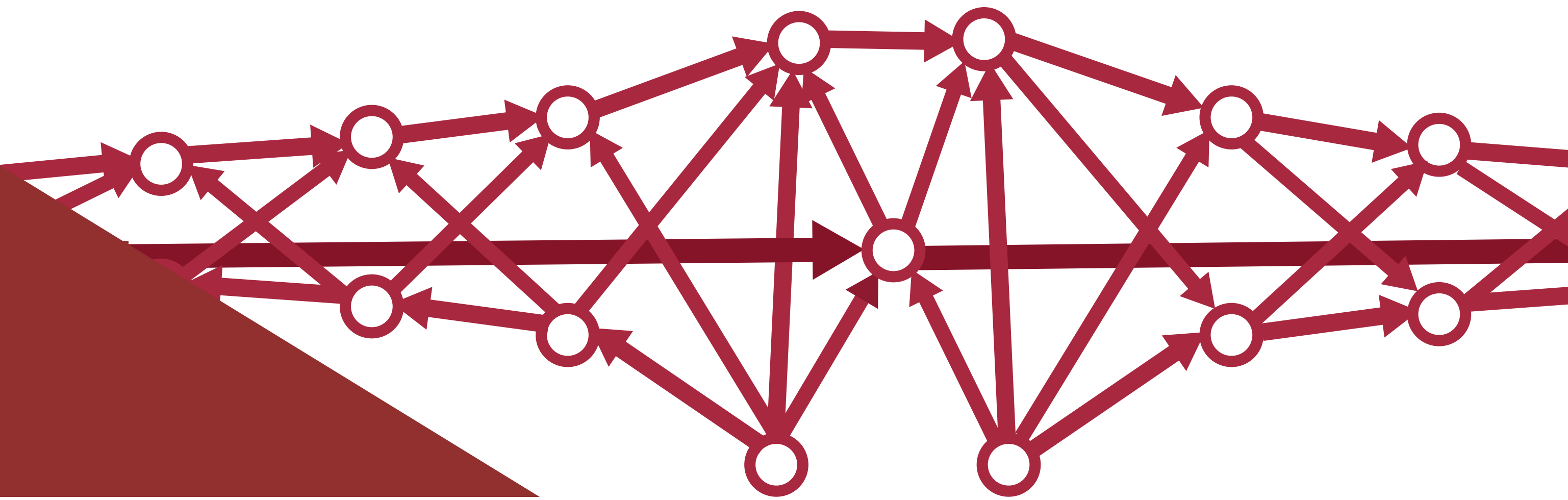


# Machine Learning & Data Analysis



## Week 3: Linear models for classification

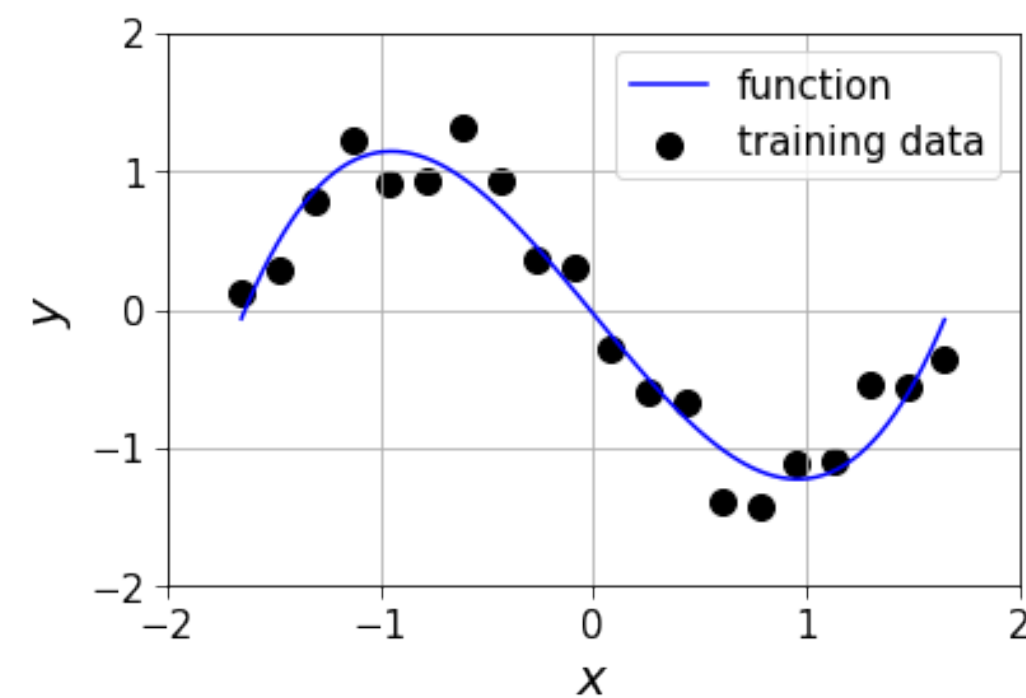
Elliot J. Crowley, 5/10/26



THE UNIVERSITY  
*of* EDINBURGH

# Recap

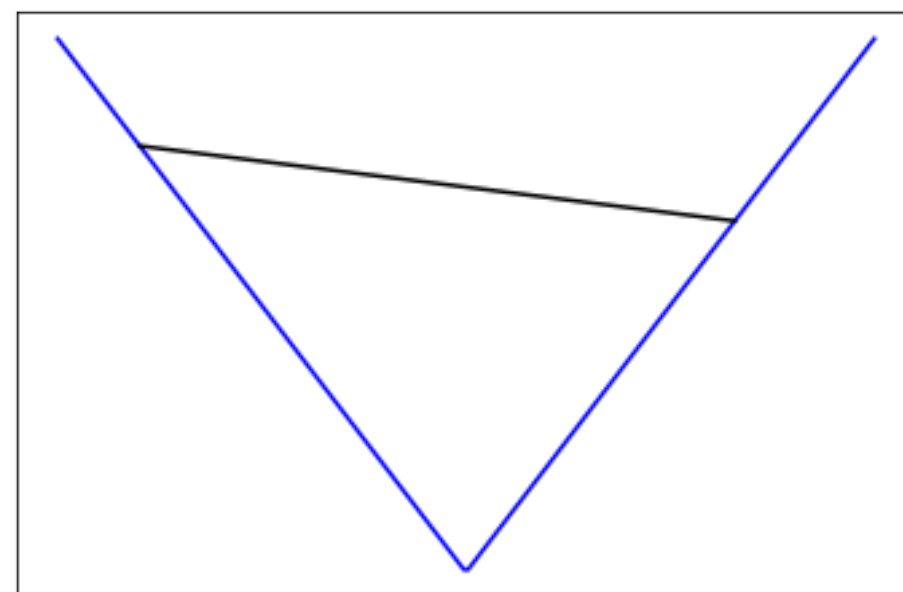
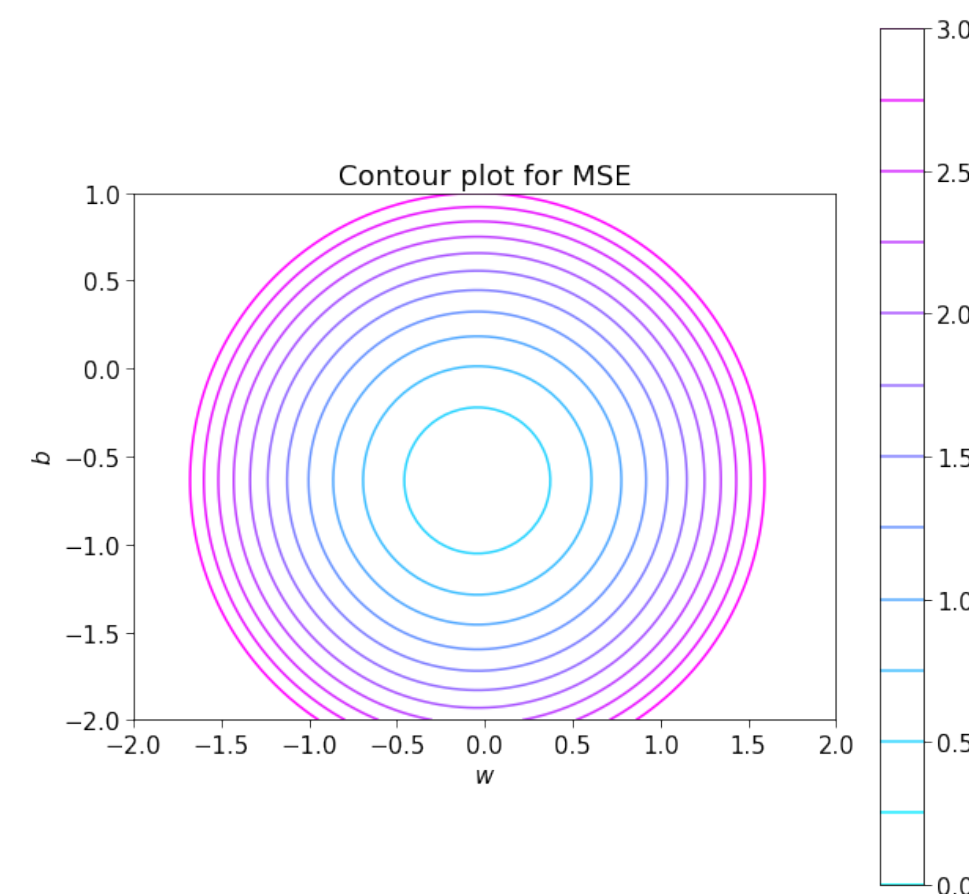
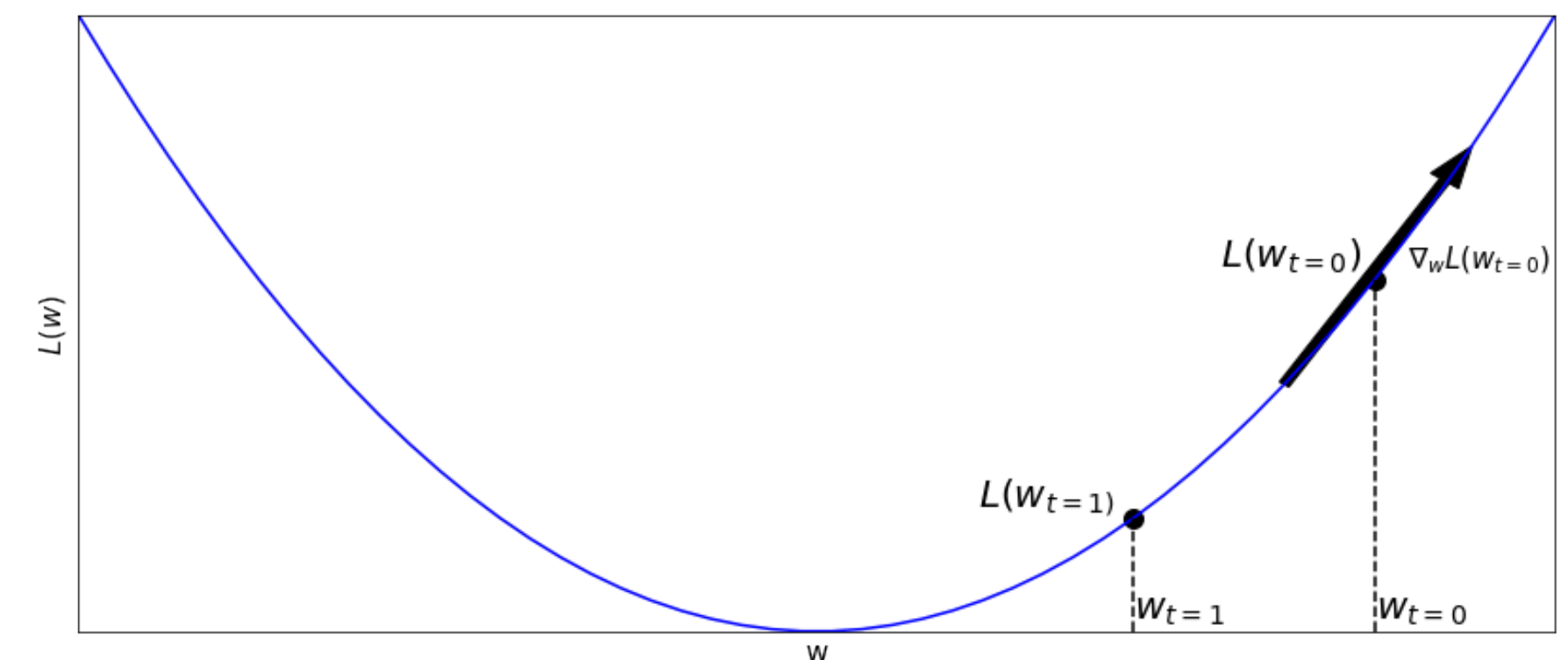
- We learned about different types of linear regression and regularisation



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

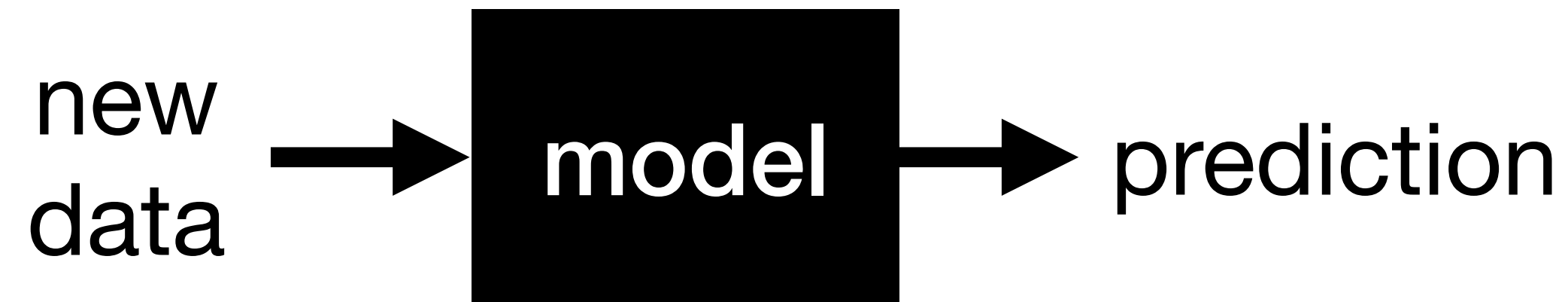
$$L_{\text{ridge}}(\mathbf{w}) = \underbrace{\|\mathbf{y} - \Phi \mathbf{w}\|^2}_{SE} + \underbrace{\lambda \|\mathbf{w}\|^2}_{\text{regularisation}}$$

- We revised convex functions and gradient descent



# Supervised Learning

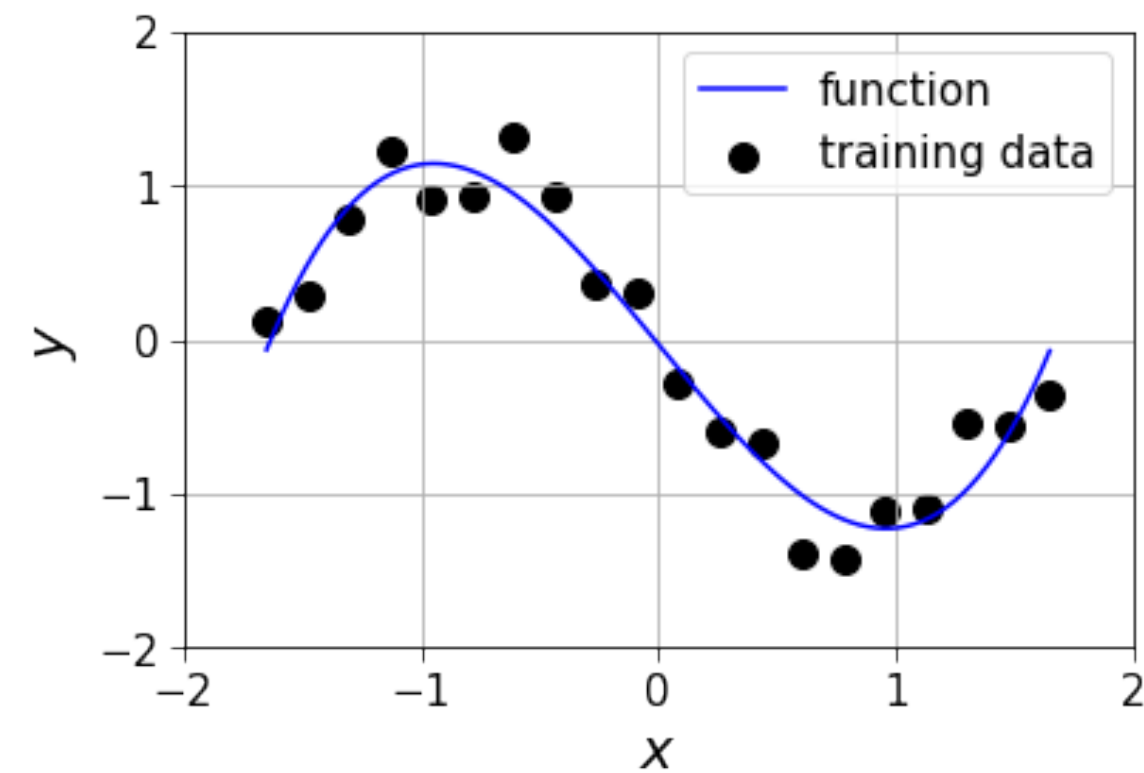
- We want a model that takes in a new data point and outputs a prediction



- 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

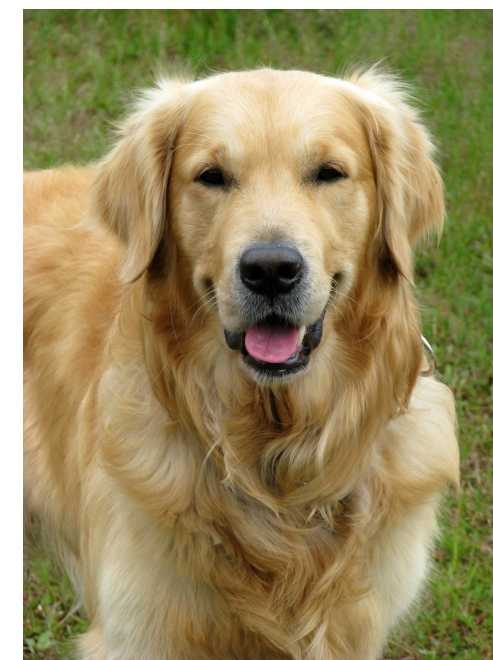
- **Regression:** Given input data, predict a continuous output



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



→ cat



→ dog

# Linear models for classification

# 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

# The classification problem

- Our training set consists of  $N$  data point-target pairs  $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$
- Data points  $\mathbf{x} \in \mathbb{R}^D$  are column vectors, targets are class labels  $y \in \mathbb{Z}_{<K}^+ = \{0, 1, \dots, K-1\}$
- 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 data correctly

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

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

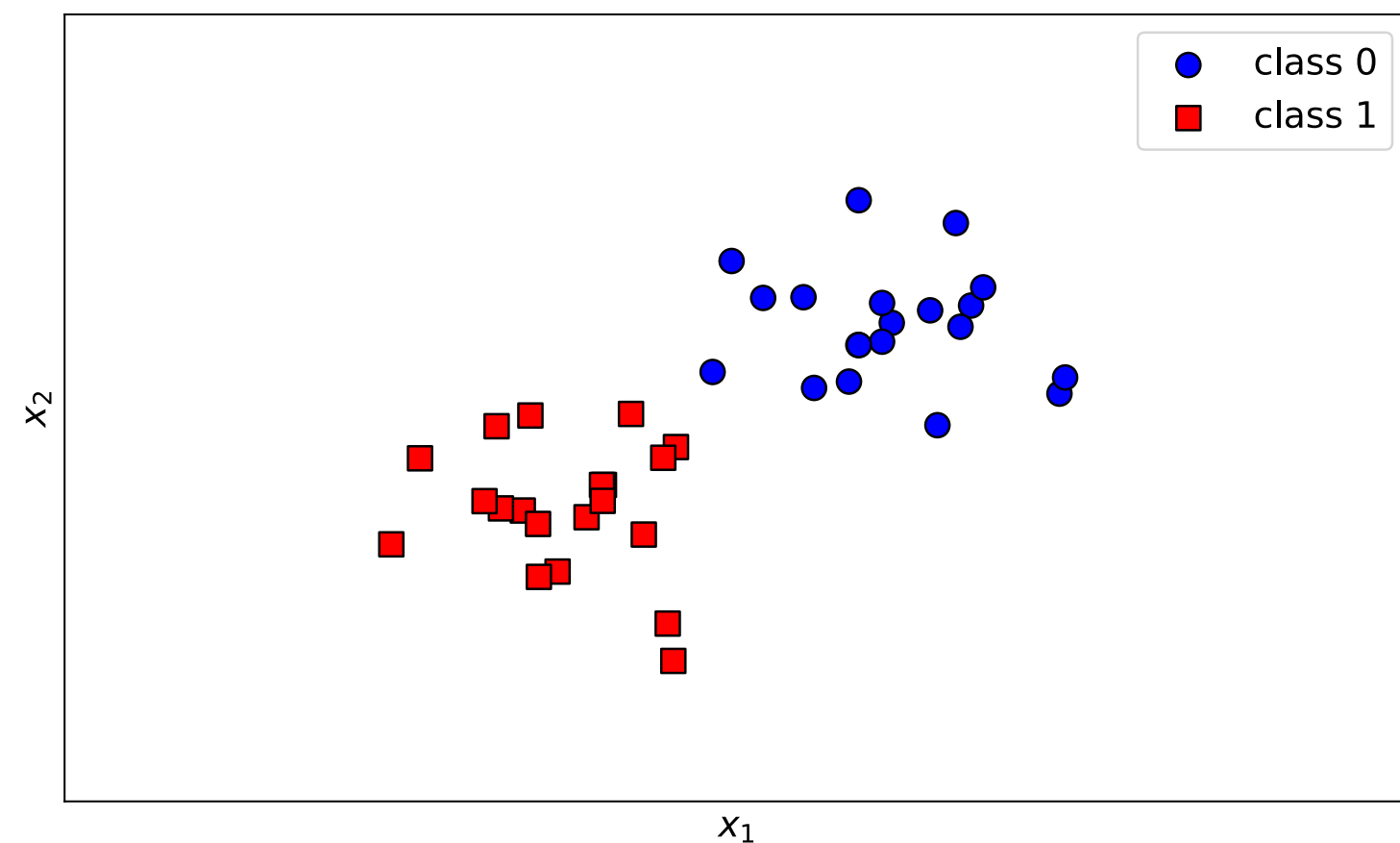
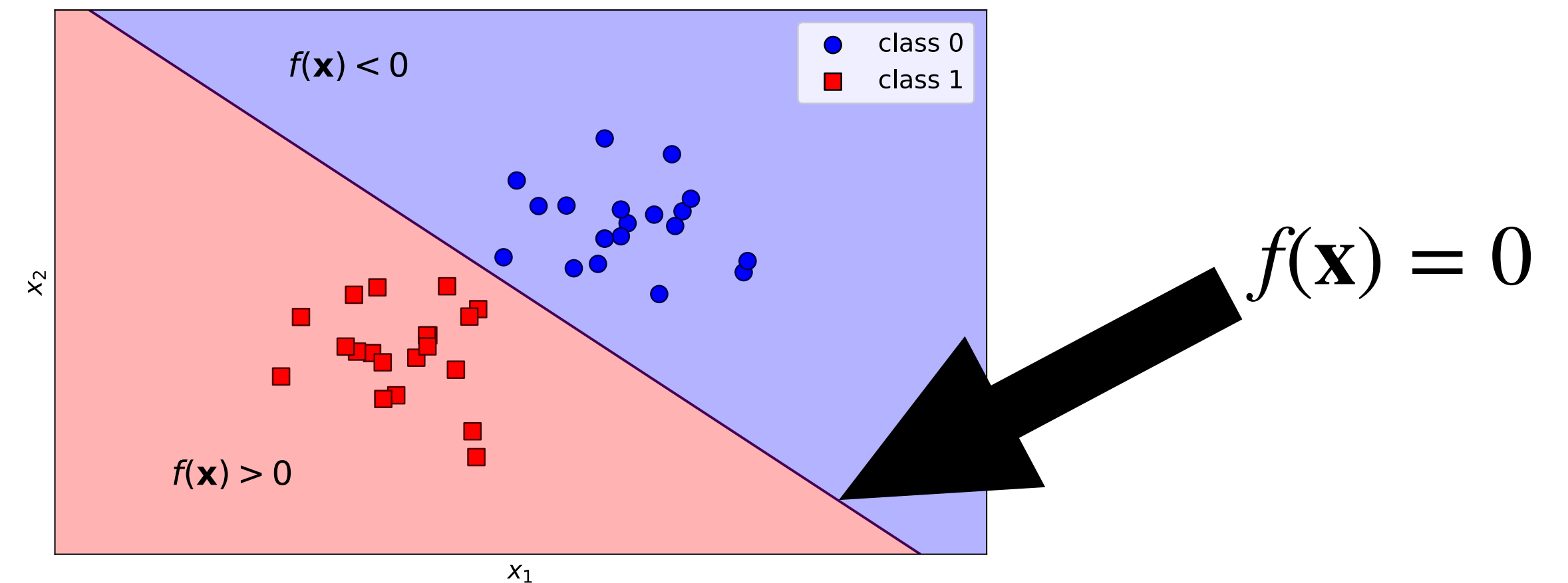
# A linear classifier is a linear model + a threshold function

- We will use a linear model as we did for regression  $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$
- For now we will consider **binary classification**:  $y \in \{0,1\}$  or  $y \in \{-1,1\}$
- For regression, we used  $f(\mathbf{x}) \in \mathbb{R}$  as our target prediction but we can't do this for classification because the class labels are discrete
- Instead we will supply a **threshold function** that maps  $f(\mathbf{x})$  to a discrete class prediction  $\hat{y}$
- This could be 
$$\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) > 0 \\ 0 & \text{if } f(\mathbf{x}) < 0 \end{cases}$$

We can call  $f(\mathbf{x})$  the classifier score  
for  $\mathbf{x}$

# Linear classifier decision boundary in 2D

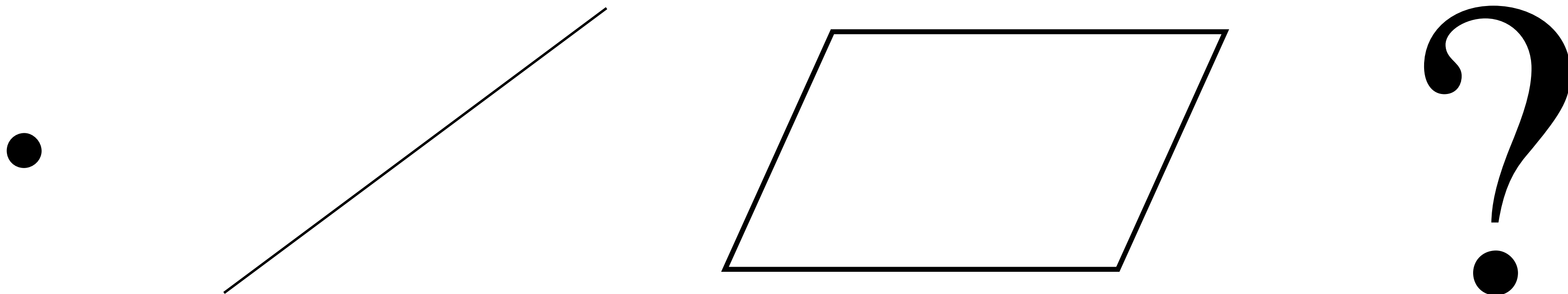
- Consider a training set  $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$  with  $\mathbf{x} \in \mathbb{R}^2$  and  $y \in \{0,1\}$
- We have  $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$  where  $\mathbf{x} = [x_1 \ x_2]^\top$  and  $\mathbf{w} = [w_1 \ w_2]^\top$
- Let's use the threshold function  $\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) > 0 \\ 0 & \text{if } f(\mathbf{x}) < 0 \end{cases}$
- The line  $f(\mathbf{x}) = w_1x_1 + w_2x_2 + b = 0$  forms the decision boundary of the classifier



# Decision boundaries are hyperplanes

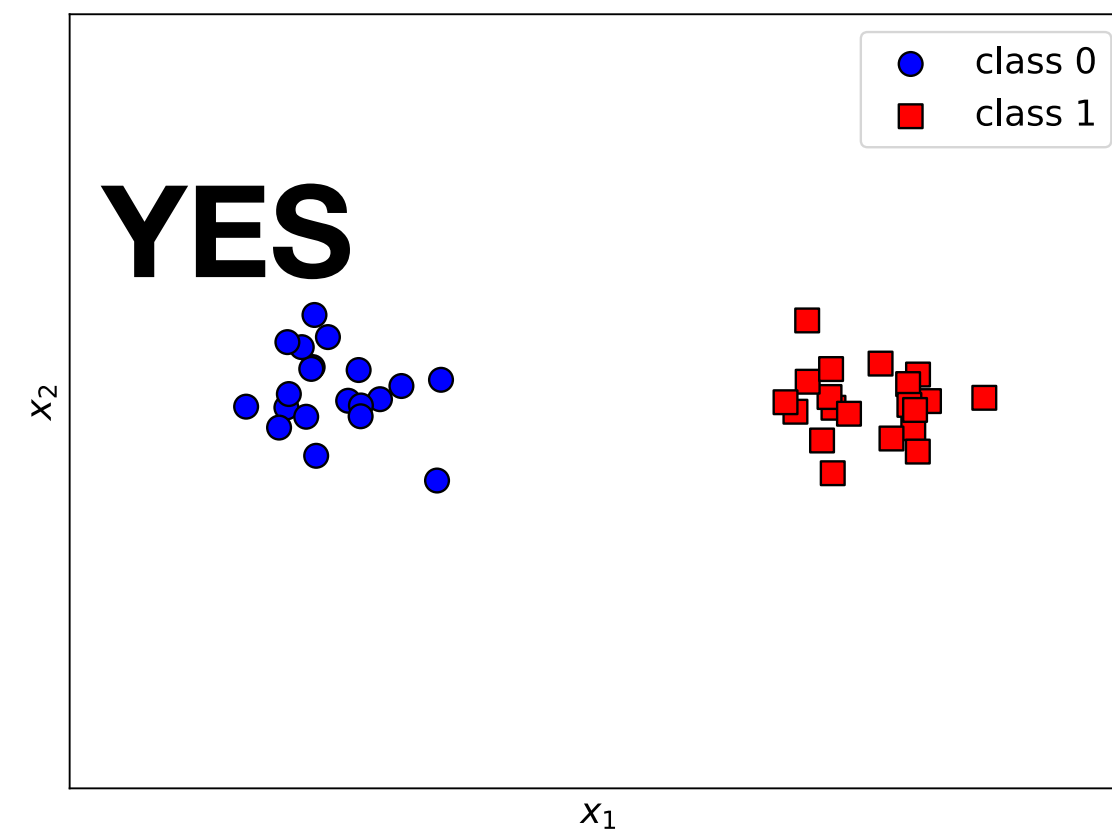
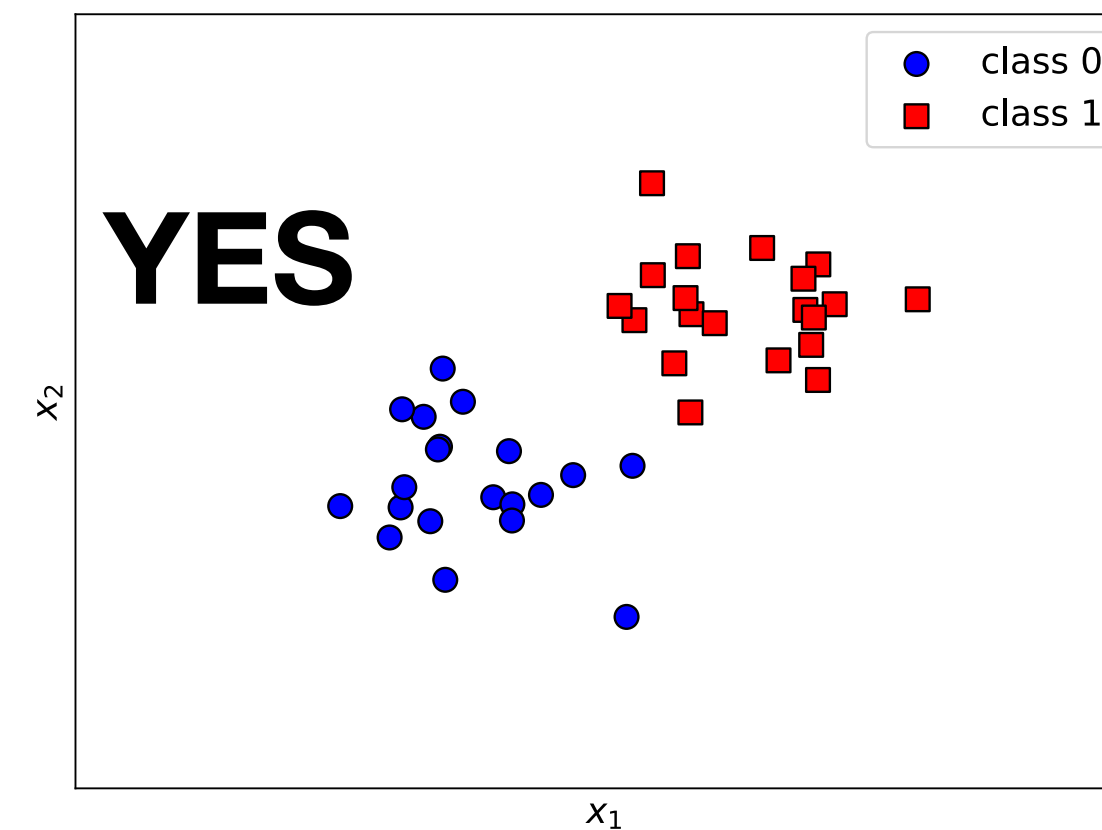
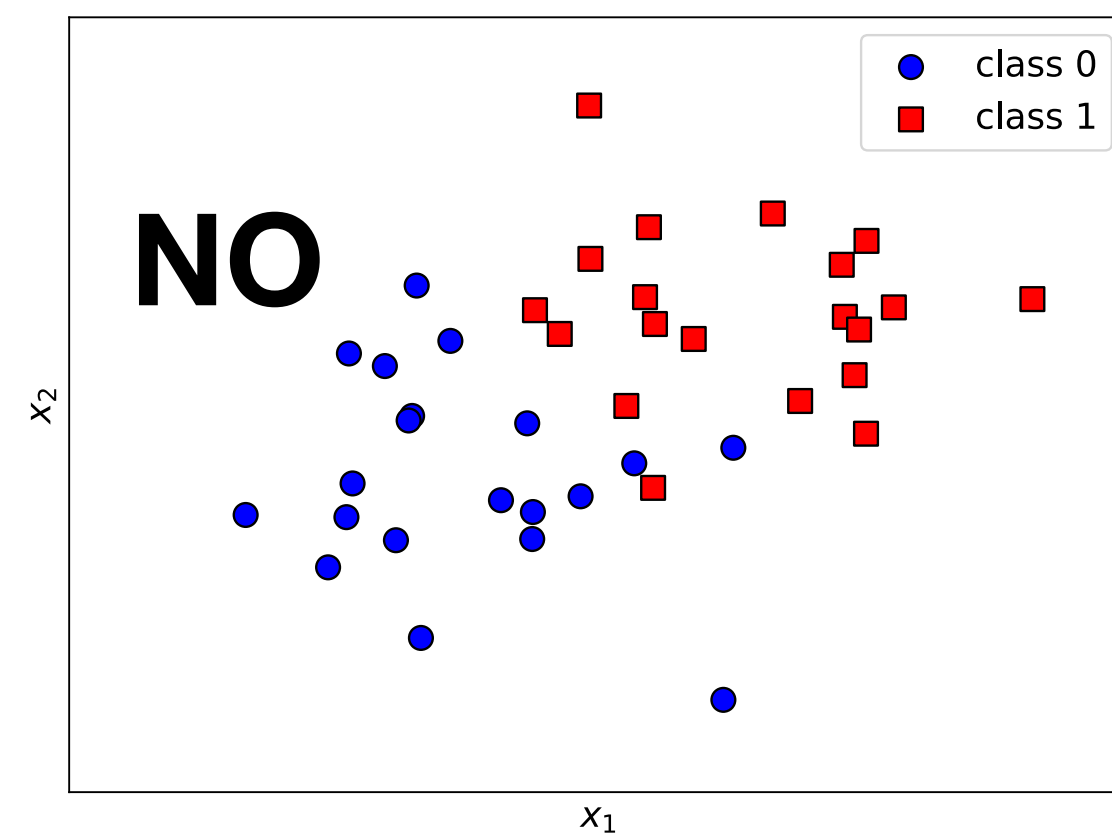
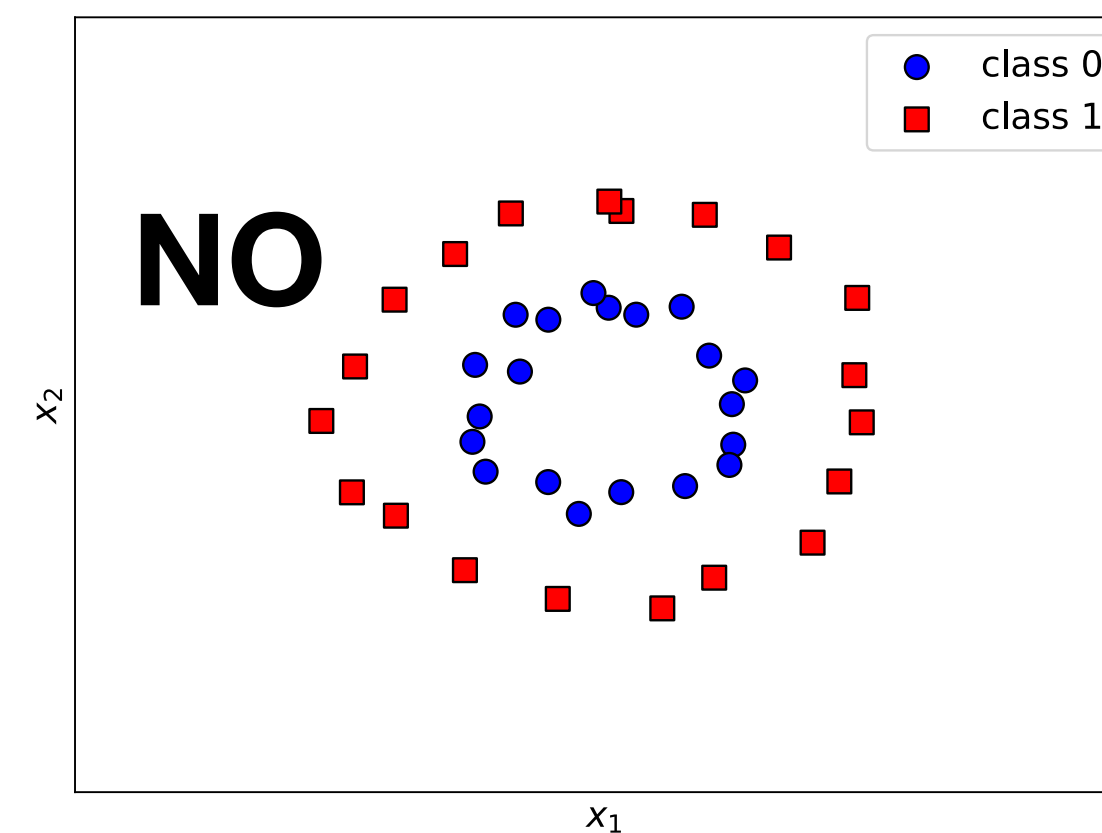
For  $\mathbf{x} \in \mathbb{R}^D$  the decision boundary of a linear classifier is in  $D - 1$

- 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 (:



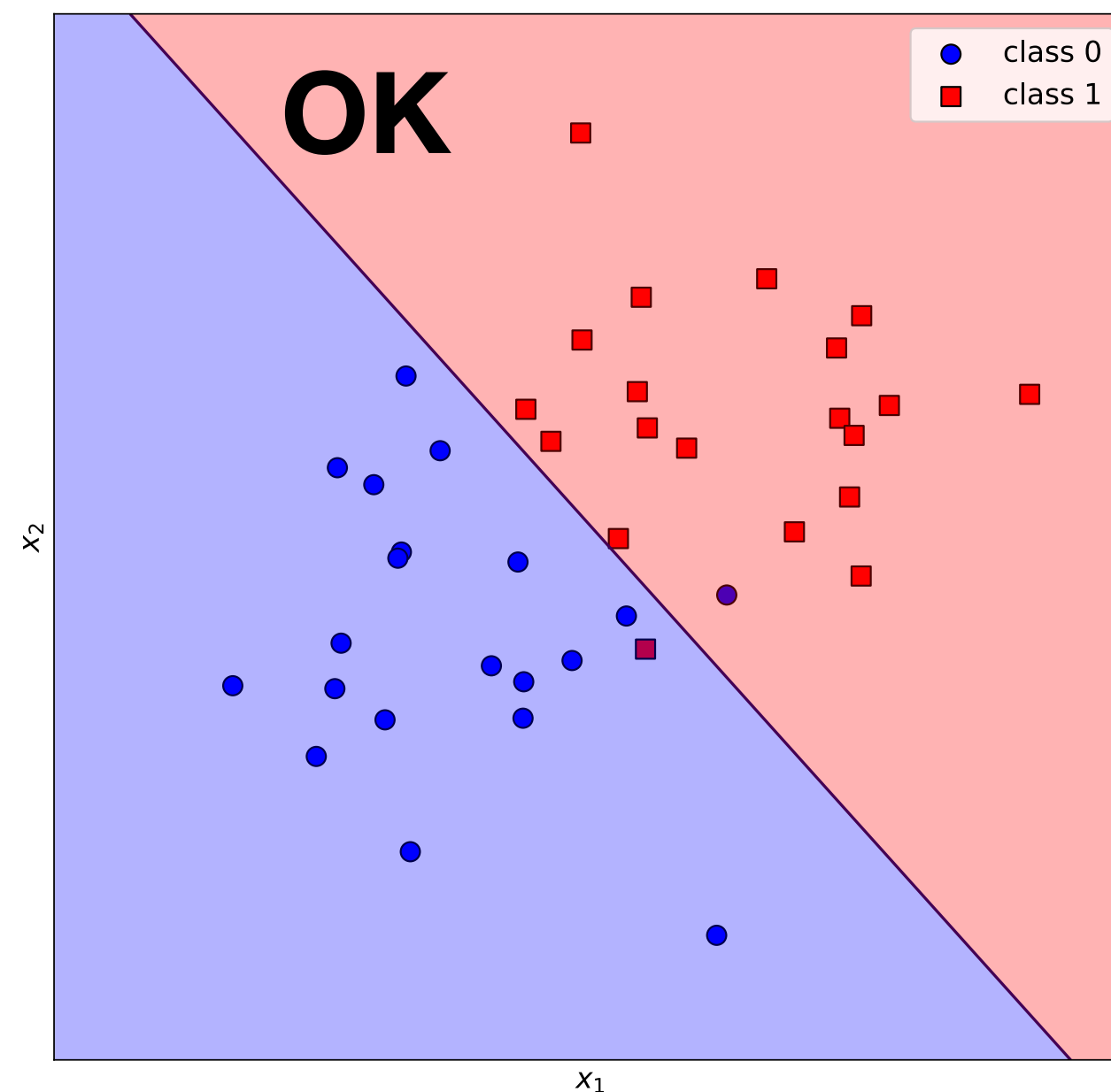
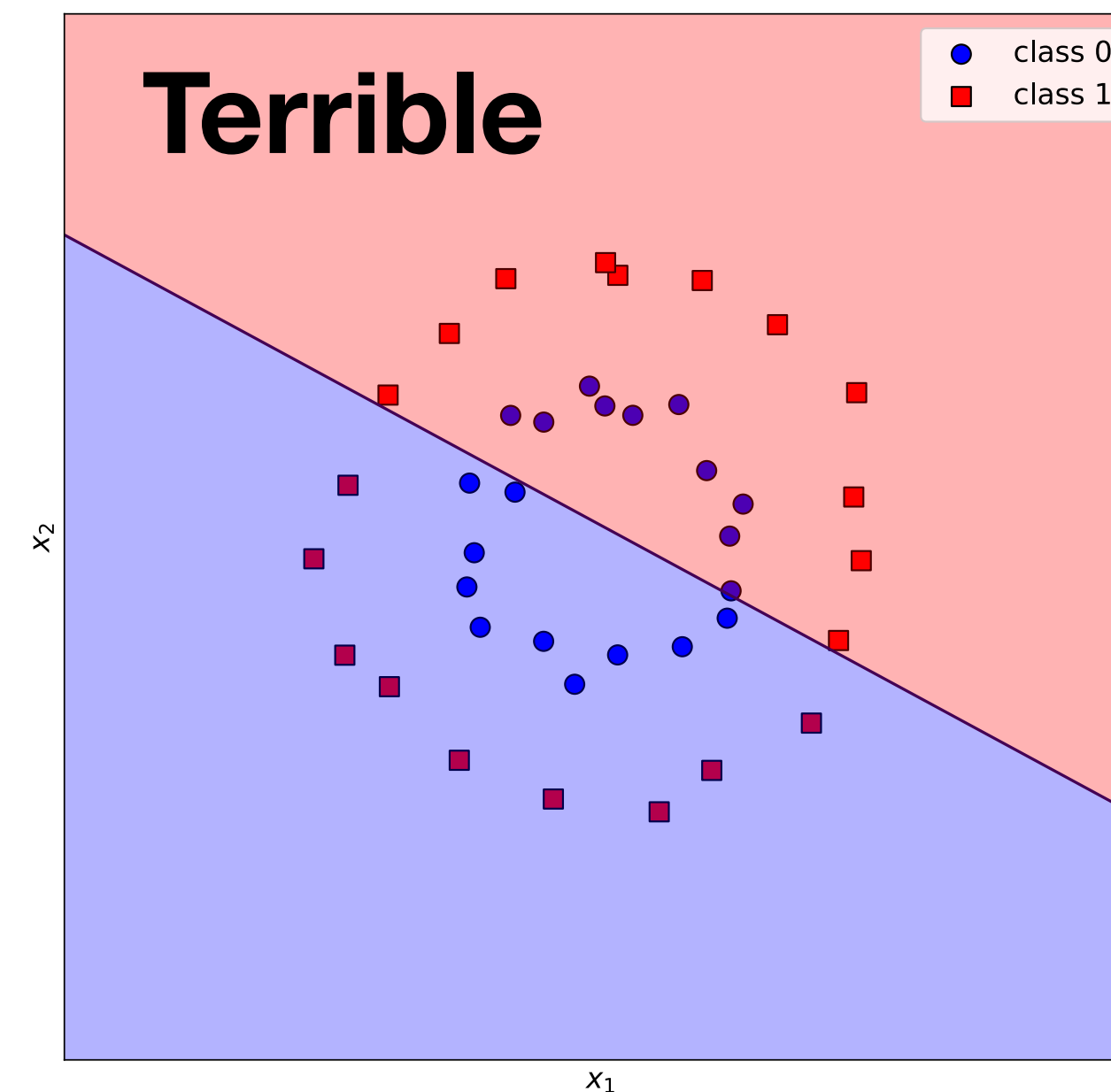
# Linear separability

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



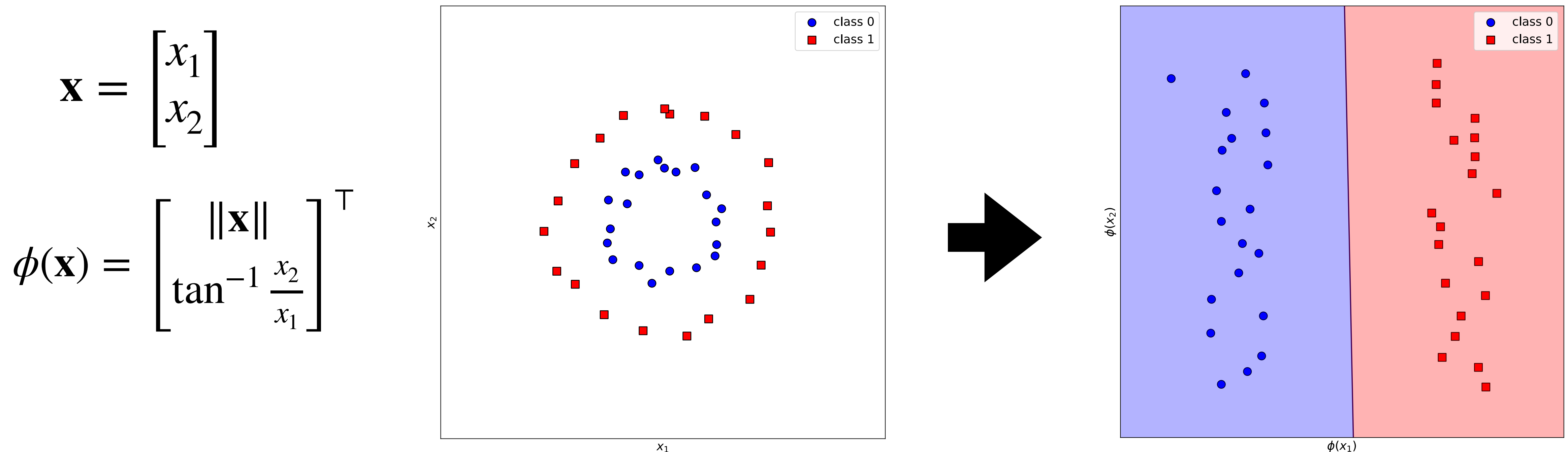
# Linear separability continued

- 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



# Feature transformations

- We can introduce some feature transformation  $\phi$  to map our data into a space where it is linearly separable. We can then use  $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$
- We'll come back to this later in the course



# Fitting a linear classifier

- For the classifier to be any good we learn the model parameters  $\mathbf{w}$ ,  $b$  using training data
- There are lots of ways to do this but they all largely boil down to minimising different loss functions that involve classifier scores  $f(\mathbf{x})$  and labels  $y$
- The loss functions rarely involve discrete predictions as the threshold function has a gradient of zero everywhere it is defined!
- We are going to cover logistic regression in detail and then look at some other approaches

# Logistic Regression

# First... treat classification as regression

- Consider a training set  $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$  where  $\mathbf{x} \in \mathbb{R}^D$  and  $y \in \{0,1\}$
- Let's treat  $y$  as continuous  $y \in \mathbb{R}$ : *it just happens to be 0/1 for training data*
- We can use  $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$  to predict this “continuous” label
- We could just minimise  $L_{MSE} = \frac{1}{N} \sum_n (y^{(n)} - f(\mathbf{x}^{(n)}))^2$
- We can just use e.g.  $\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) > 0.5 \\ 0 & \text{if } f(\mathbf{x}) < 0.5 \end{cases}$  as our threshold function
- This is known as label regression. Our  $f(\mathbf{x})$  isn't particularly 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

# The sigmoid function

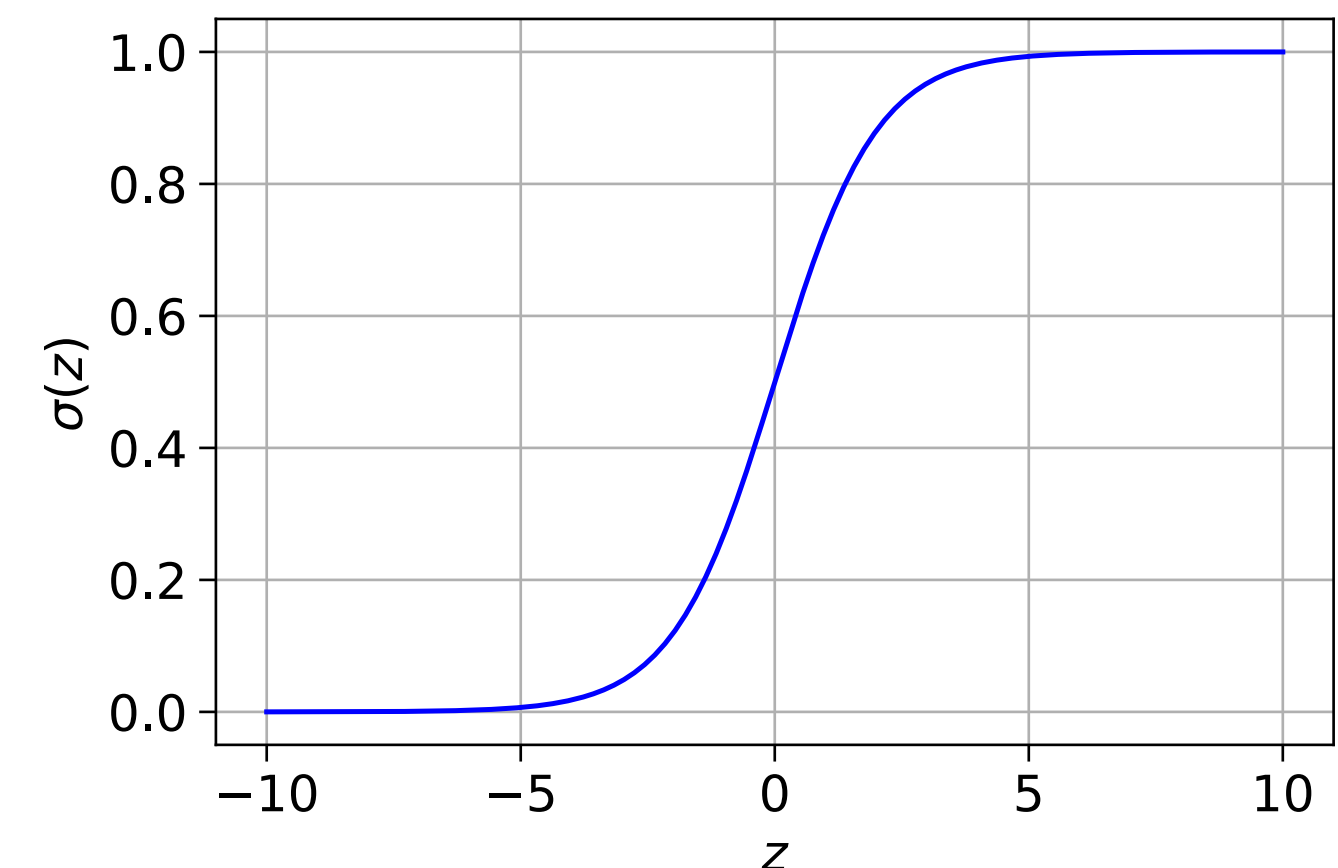
- In logistic regression our model predicts the log-odds for class 1

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

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

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

$\sigma$  is the sigmoid function.  
It squashes numbers to be between 0 and 1



# Discrete class predictions from log-odds

- We can convert log-odds to probabilities through  $p(y = 1 | \mathbf{x}) = \sigma(f(\mathbf{x}))$
- It follows that  $p(y = 0 | \mathbf{x}) = 1 - \sigma(f(\mathbf{x}))$  as there are only two classes
- What threshold function should we use to make a discrete class prediction  $\hat{y}$ ?
- The obvious approach is to use  $\hat{y} = \begin{cases} 1 & \text{if } p(y = 1 | \mathbf{x}) \geq 0.5 \\ 0 & \text{if } p(y = 1 | \mathbf{x}) < 0.5 \end{cases}$
- $\sigma(\mathbf{w}^\top \mathbf{x} + b) = 0.5$  when  $\mathbf{w}^\top \mathbf{x} + b = 0$  which is a **hyperplane**
- We can rewrite the above as  $\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) \geq 0 \\ 0 & \text{if } f(\mathbf{x}) < 0 \end{cases}$

# Maximum likelihood estimation

- We can write  $p(y | \mathbf{x}) = \sigma(f(\mathbf{x}))^y (1 - \sigma(f(\mathbf{x})))^{1-y}$
- We can then write an expression for the likelihood of our data

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

- Maximising likelihood is the same as minimising negative log-likelihood (divided by the number of data points)

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

# NLL is the log loss

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

- We can write  $p(y = 1 \mid \mathbf{x}) = \sigma(f(\mathbf{x}^{(n)}))$  as  $p^{(n)}$  to express this more succinctly:

$$L_{\log} = -\frac{1}{N} \sum_n \left[ y^{(n)} \log p^{(n)} + (1 - y^{(n)}) \log(1 - p^{(n)}) \right]$$

- It is also known as the logistic loss, or the cross-entropy loss. Minimising it is performing maximum likelihood estimation (MLE)
- 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)

# Log loss

$$L_{log} = -\frac{1}{N} \sum_n \left[ y^{(n)} \log p^{(n)} + (1 - y^{(n)}) \log(1 - p^{(n)}) \right]$$

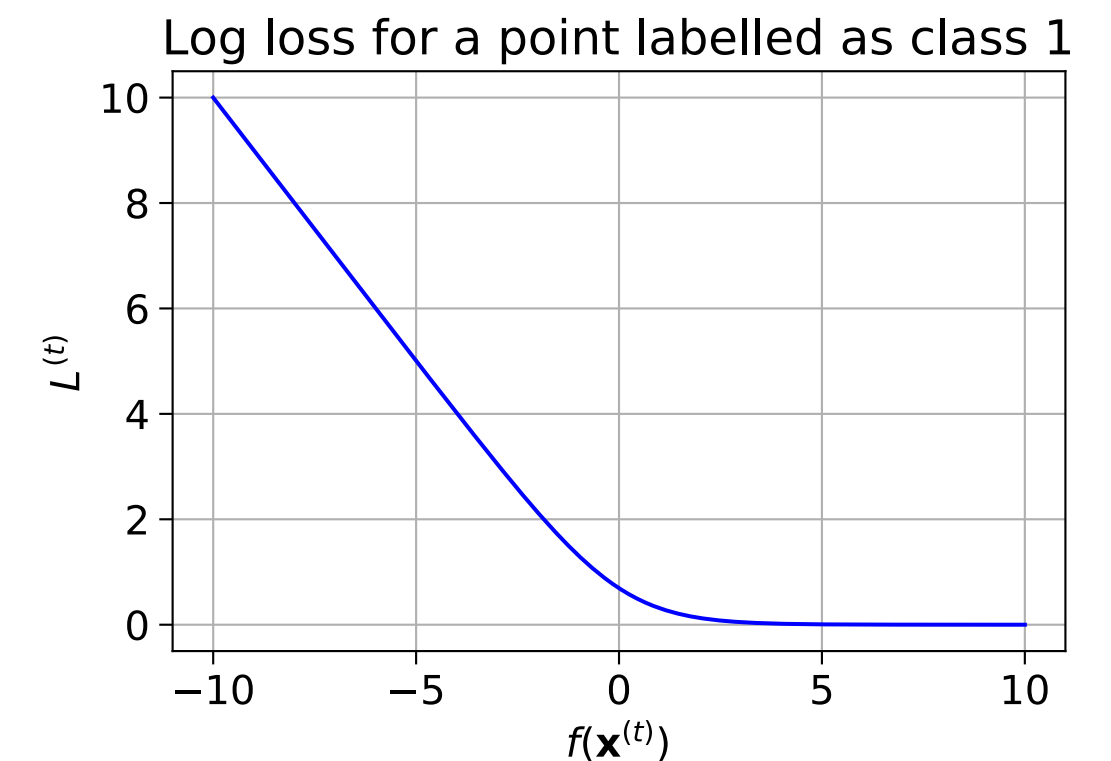
- This loss is convex for a linear model but doesn't have a closed-form minimiser
- We can use a gradient-based optimiser to solve minimise  $L_{log}$  using  $\mathbf{w}, b$

$$\nabla_{\mathbf{w}} L_{log} = -\frac{1}{N} \sum_n (y^{(n)} - p^{(n)}) \mathbf{x}^{(n)}$$

$$\nabla_b L_{log} = -\frac{1}{N} \sum_n (y^{(n)} - p^{(n)})$$

# Stochastic Gradient descent (SGD) algorithm

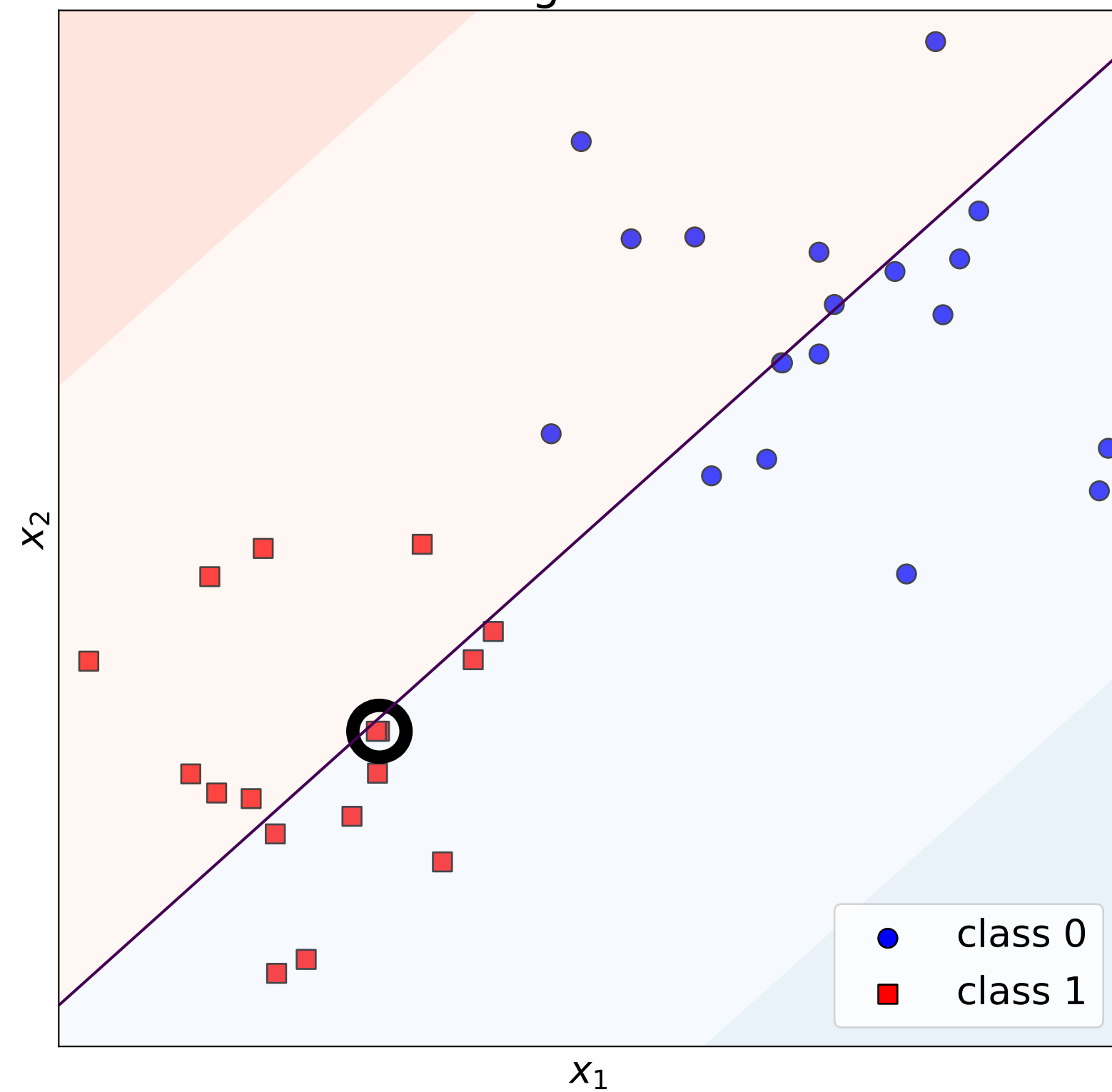
- Goal: We want to minimise the training loss of our model on  $\mathfrak{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$
- We can write the loss as an average of per example losses  $\frac{1}{N} \sum_n L(y^{(n)}, \mathbf{x}^{(n)}, \mathbf{w}, b) = \frac{1}{N} \sum_n L^{(n)}$
- Initialise  $\mathbf{w}$  and  $b$
- For epoch in range( $E$ ):
  - Shuffle  $\mathfrak{D}$
  - For  $n$  in range( $N$ ):
    - Compute  $\nabla_{\mathbf{w}} L^{(n)}$  and  $\nabla_b L^{(n)}$
    - Update  $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L^{(n)}$  and  $b \leftarrow b - \alpha \nabla_b L^{(n)}$



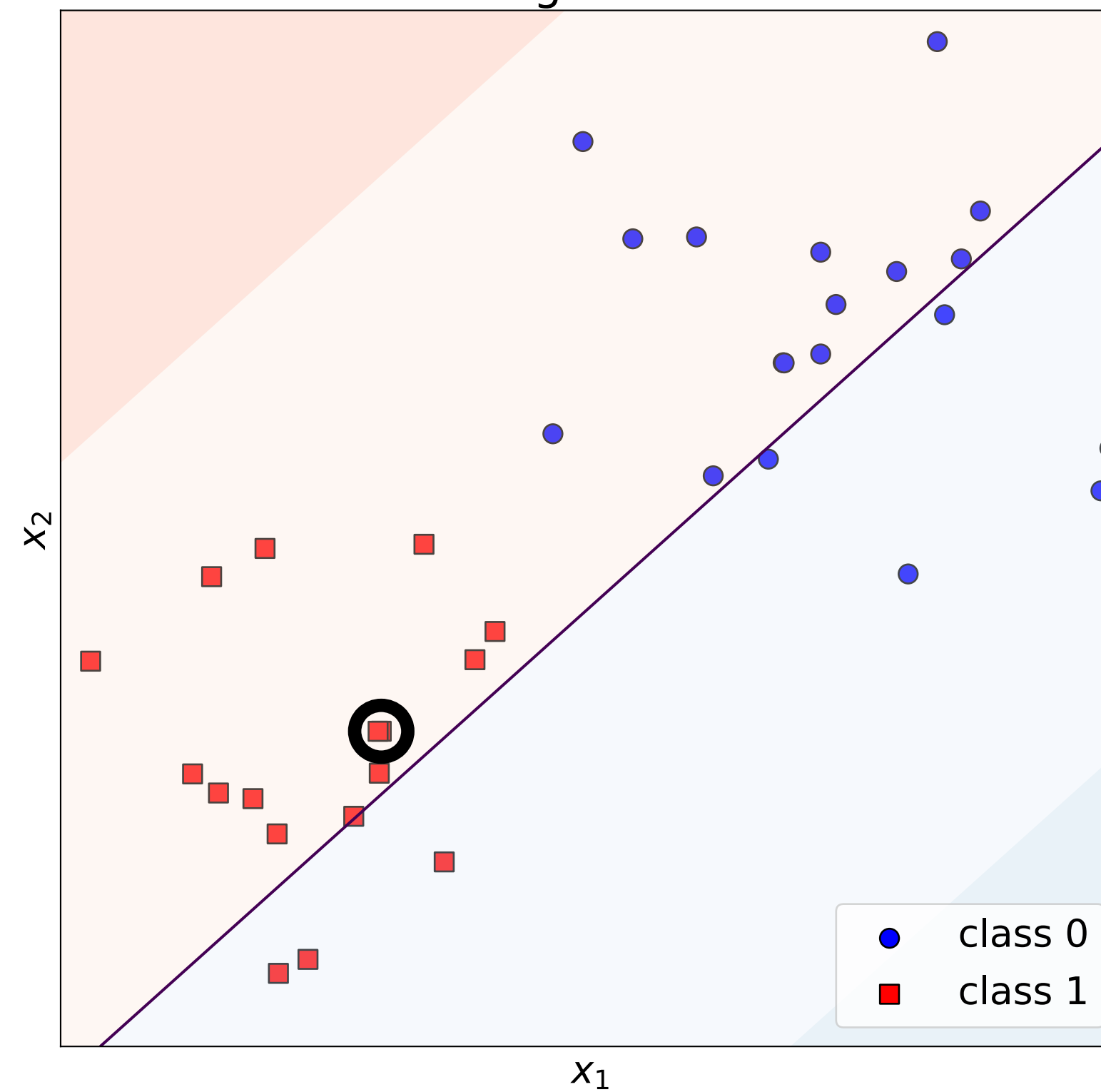
This is what sklearn does. There are lots of variants.  
See <https://sebastianraschka.com/faq/docs/sgd-methods.html>

# SGD Update 1

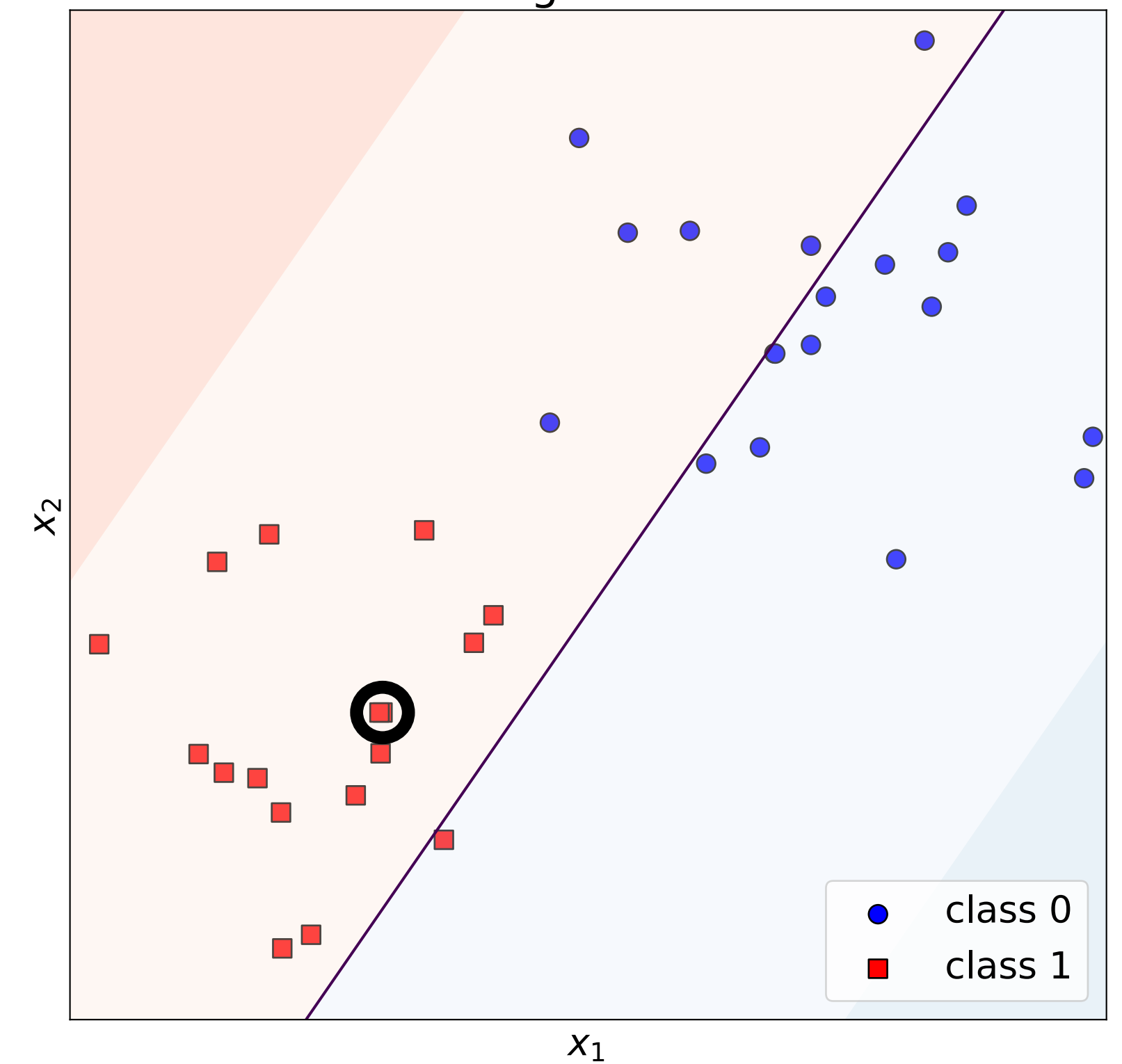
Step 1 before any update  
 $w_1 = -0.156, w_2 = 0.170, b = 0.000$   
total log loss: 0.700



Step 1 after bias update  
 $w_1 = -0.156, w_2 = 0.170, b = 0.050$   
total log loss: 0.700

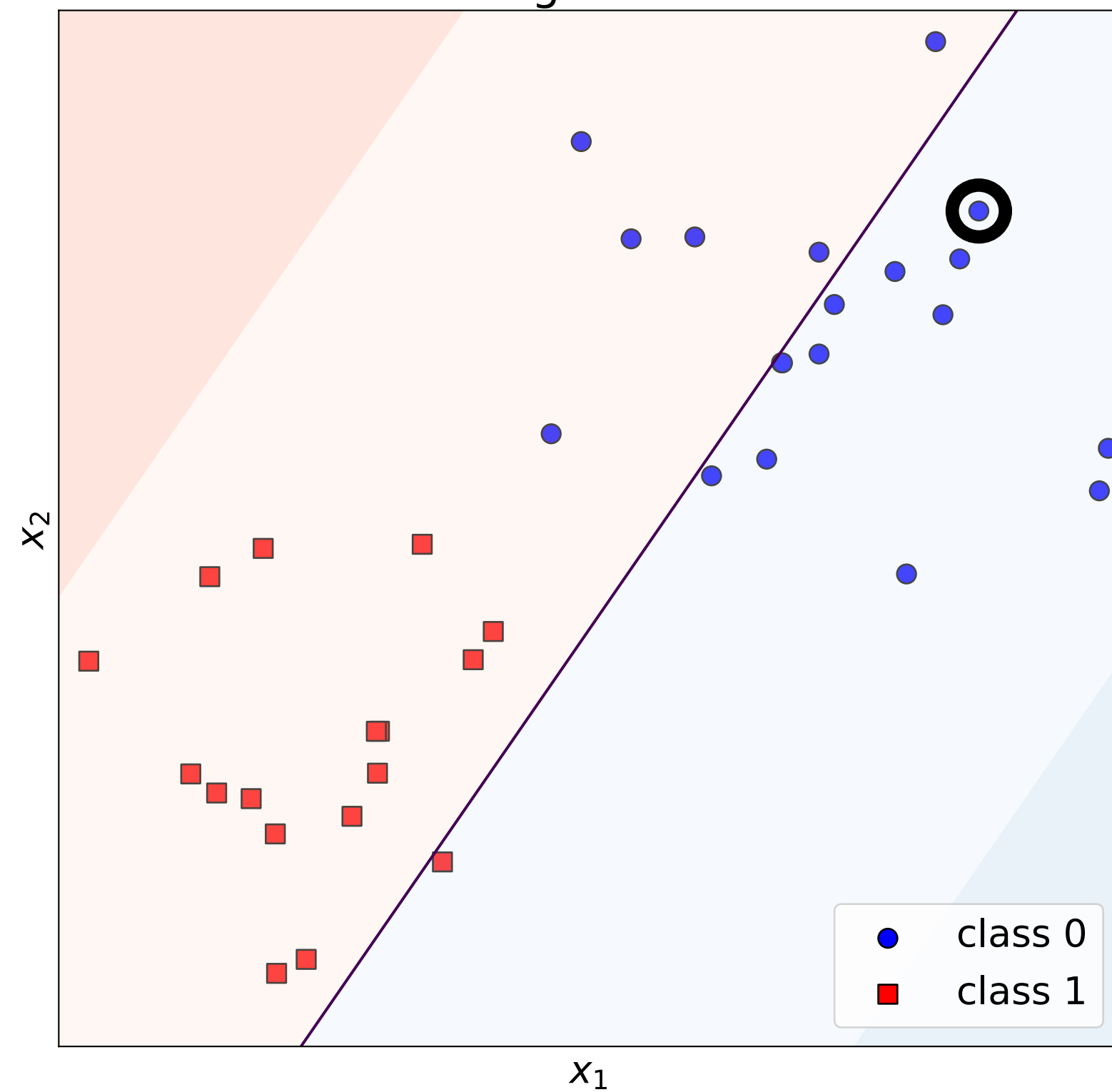


Step 1 after weight + bias update  
 $w_1 = -0.195, w_2 = 0.131, b = 0.050$   
total log loss: 0.664

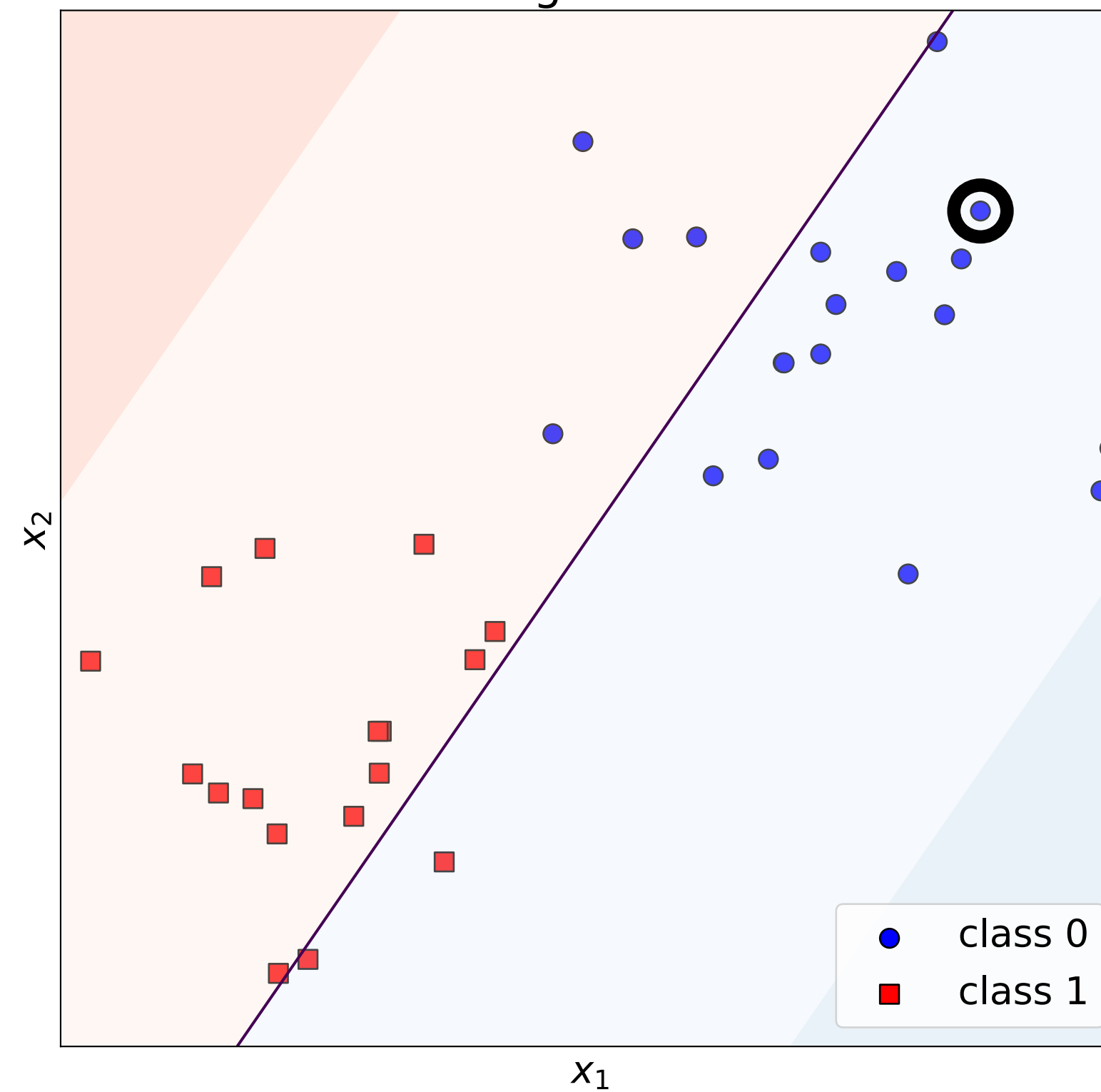


# SGD Update 2

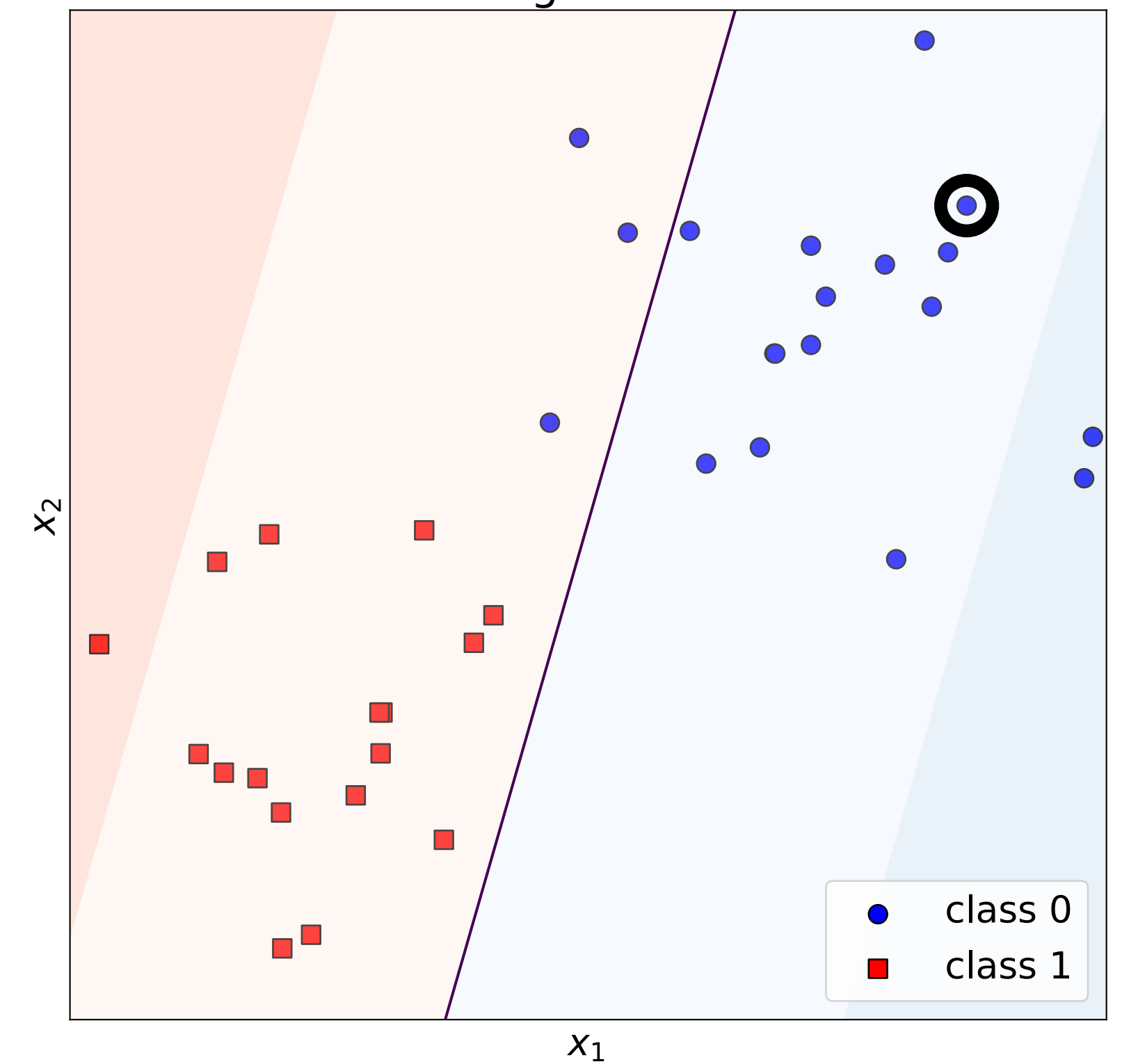
Step 2 before any update  
 $w_1 = -0.195, w_2 = 0.131, b = 0.050$   
total log loss: 0.664



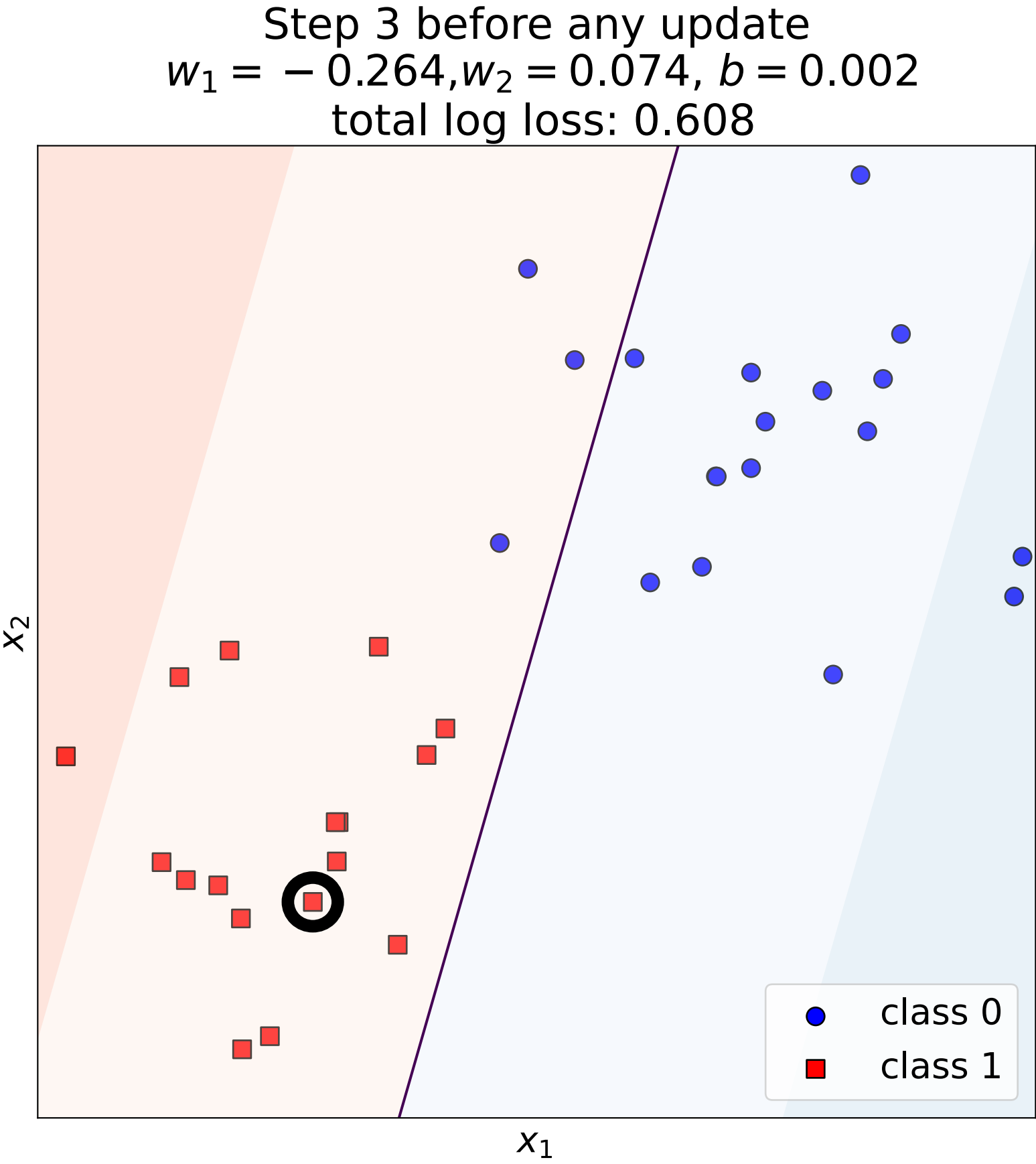
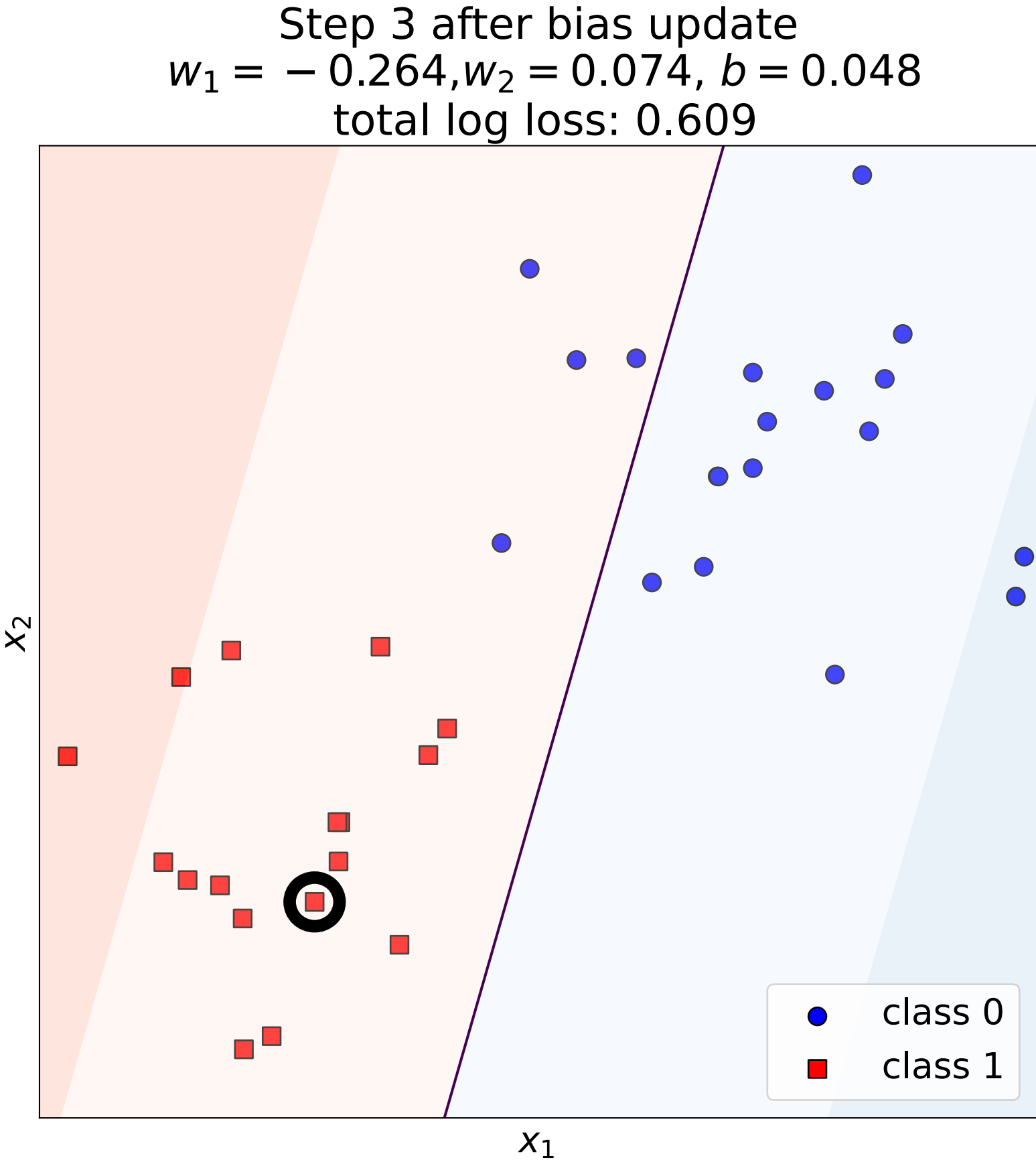
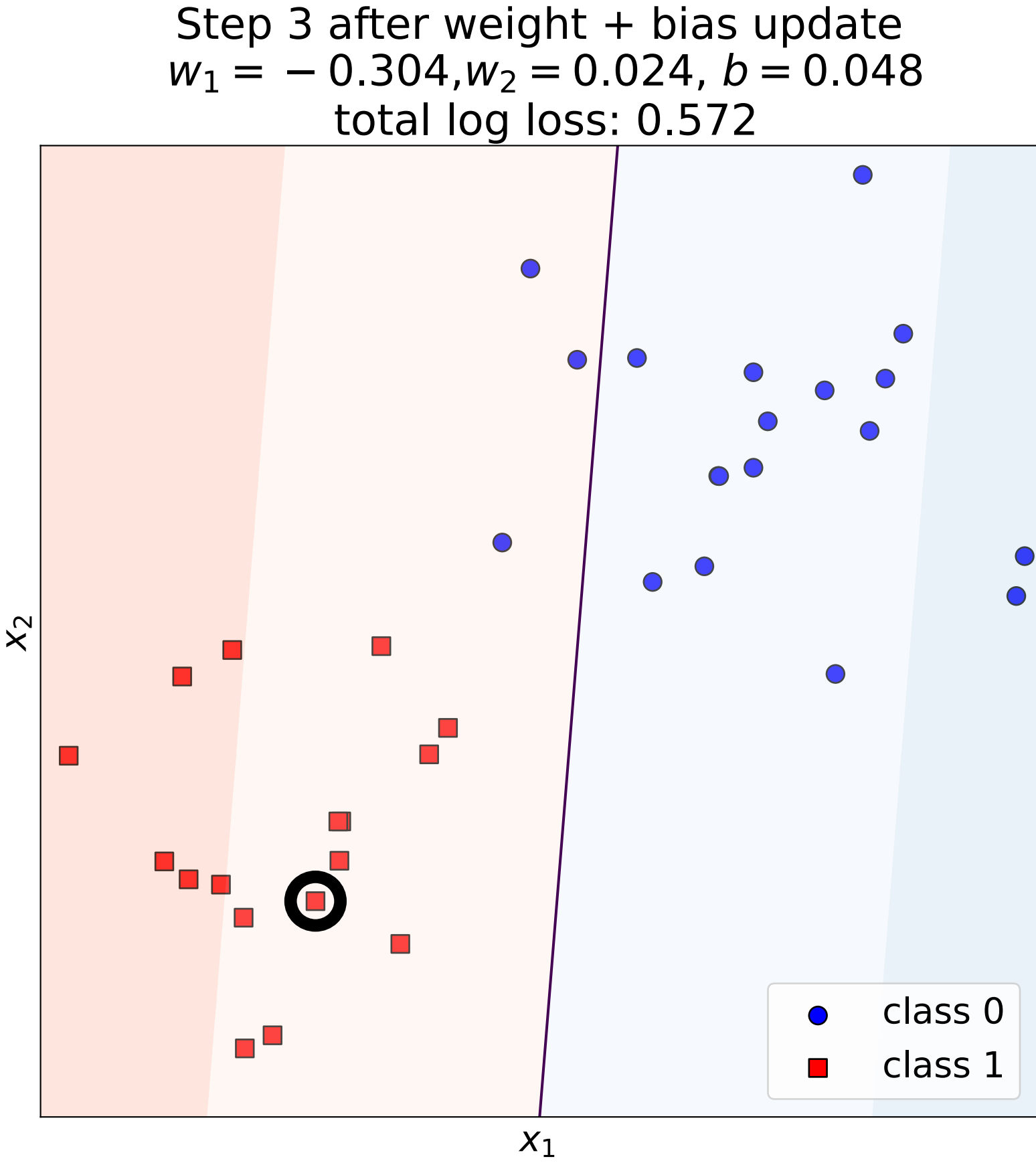
Step 2 after bias update  
 $w_1 = -0.195, w_2 = 0.131, b = 0.002$   
total log loss: 0.664



Step 2 after weight + bias update  
 $w_1 = -0.264, w_2 = 0.074, b = 0.002$   
total log loss: 0.608

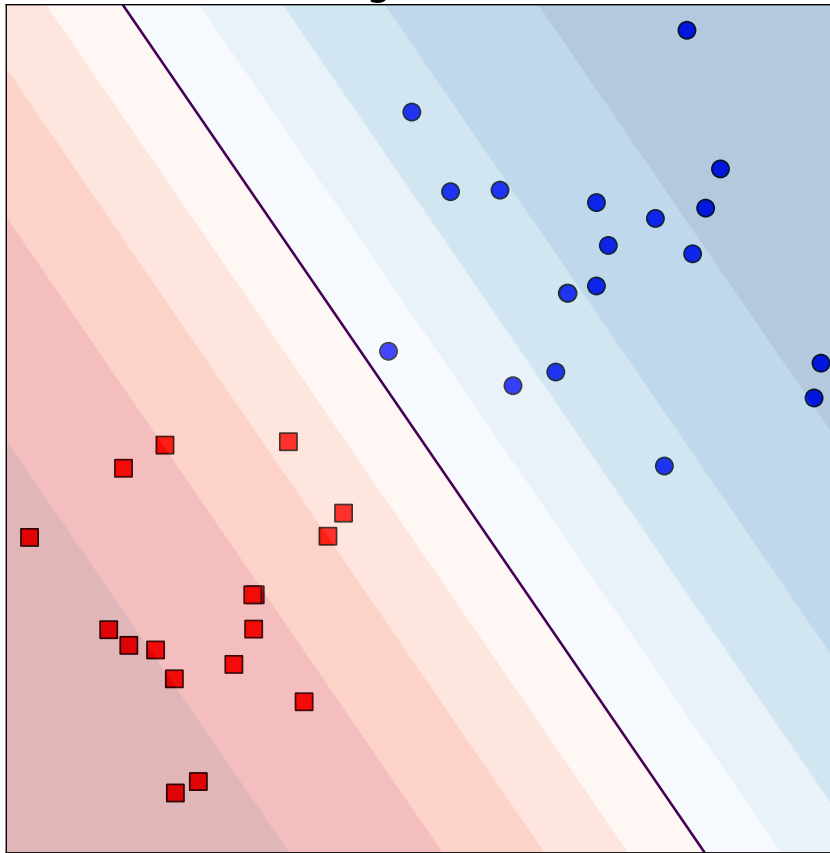


# SGD Update 3

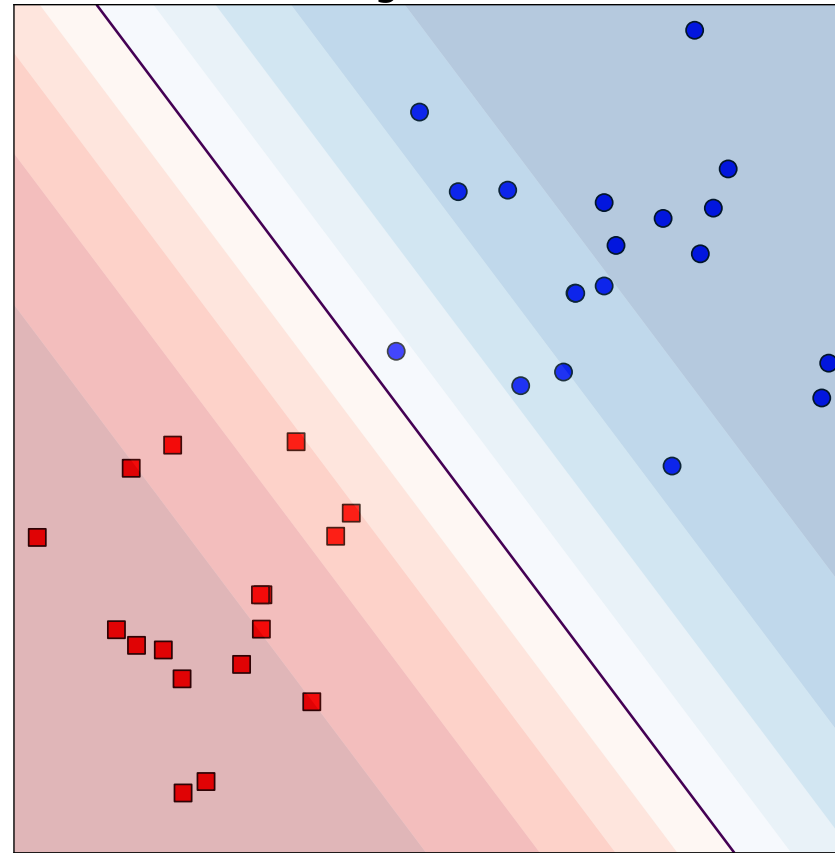


# SGD for 10 epochs

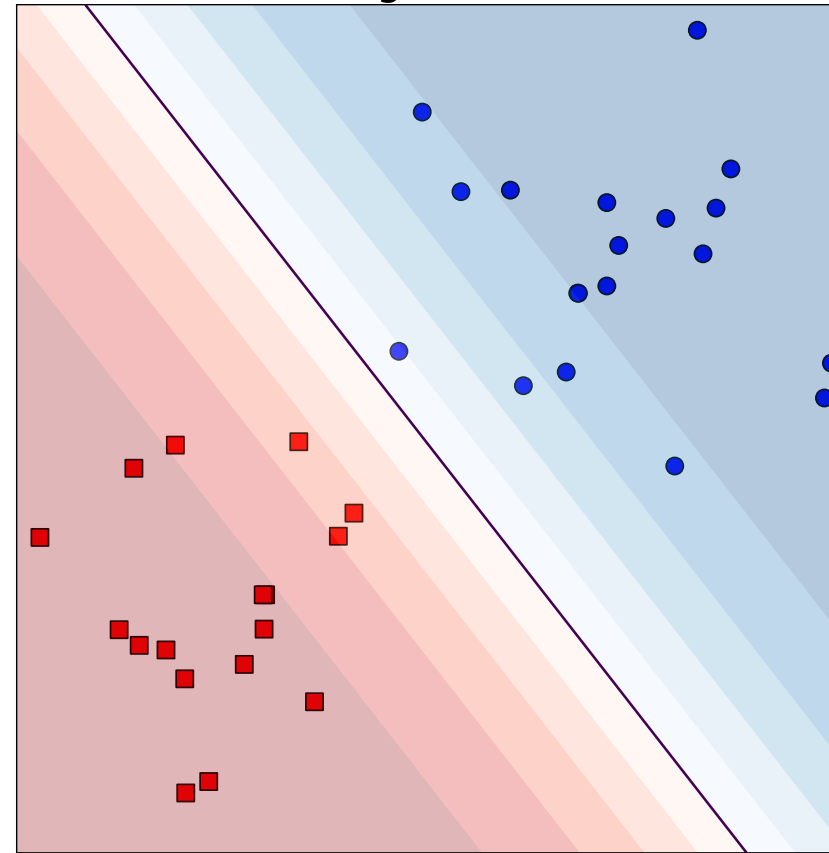
After Epoch 1:  
 $w_1 = -1.092, w_2 = -0.770, b = -0.028$   
total log loss: 0.189



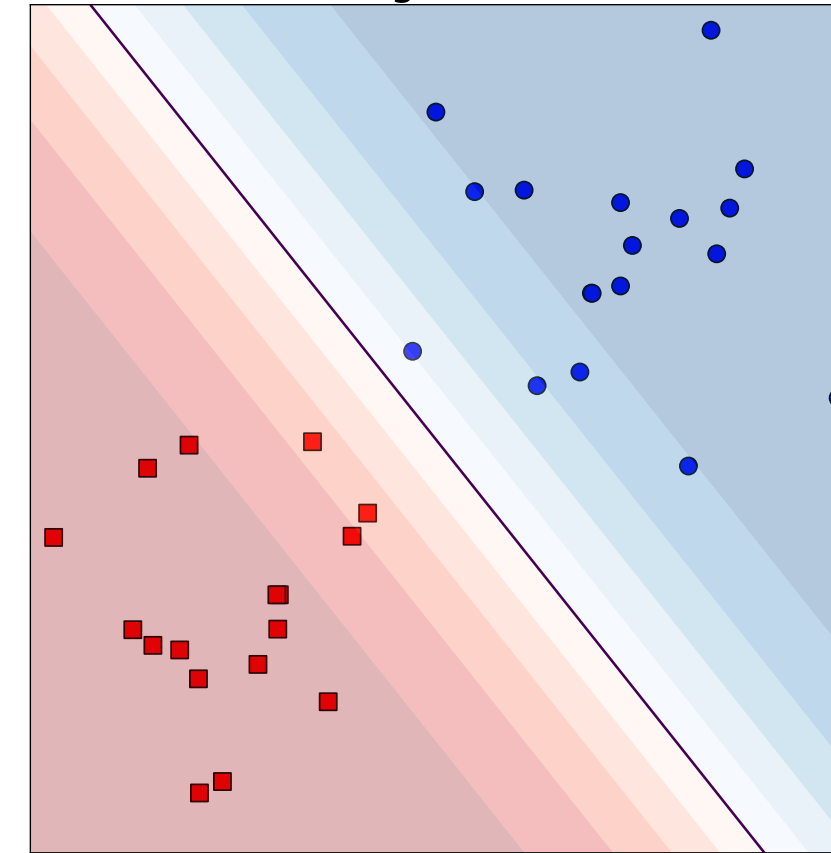
After Epoch 2:  
 $w_1 = -1.472, w_2 = -1.137, b = -0.080$   
total log loss: 0.119



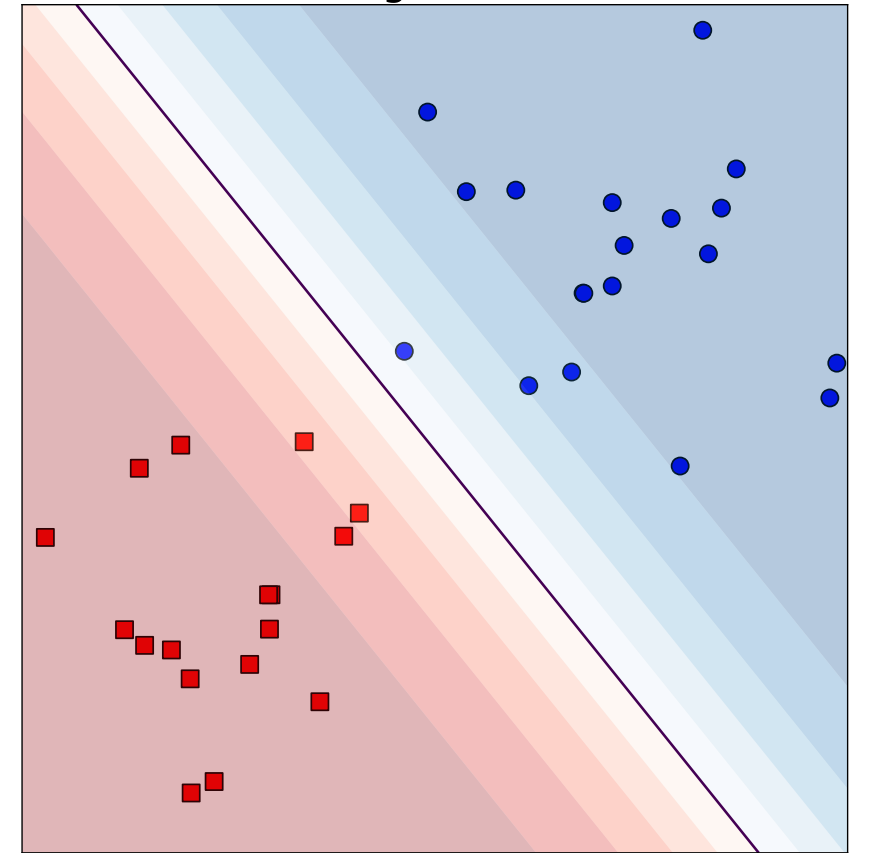
After Epoch 3:  
 $w_1 = -1.717, w_2 = -1.375, b = -0.112$   
total log loss: 0.090



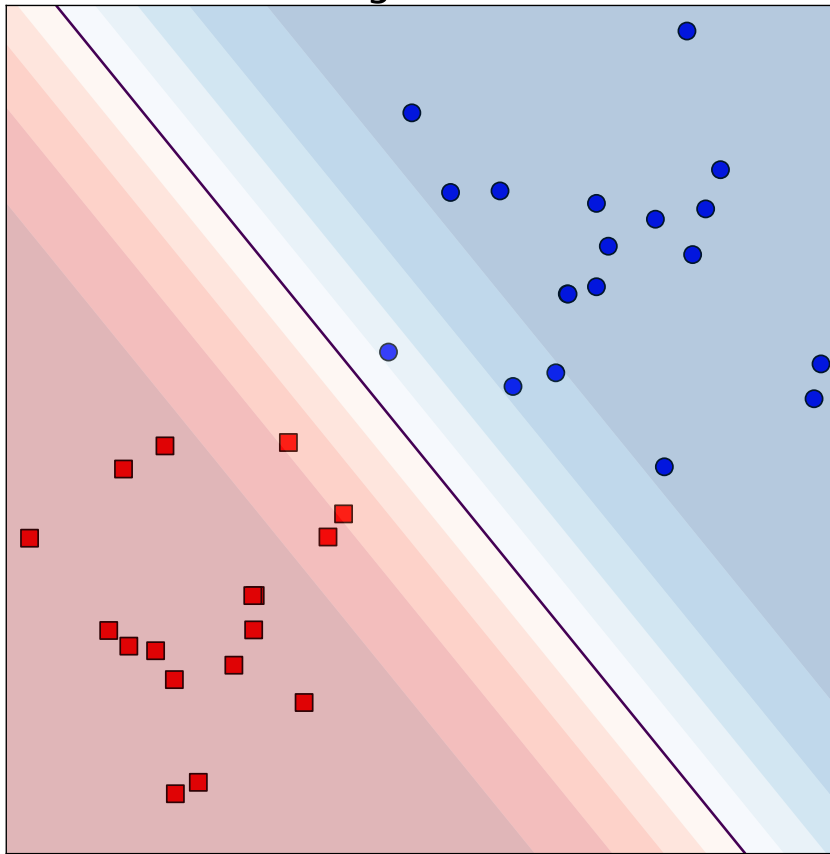
After Epoch 4:  
 $w_1 = -1.892, w_2 = -1.545, b = -0.144$   
total log loss: 0.075



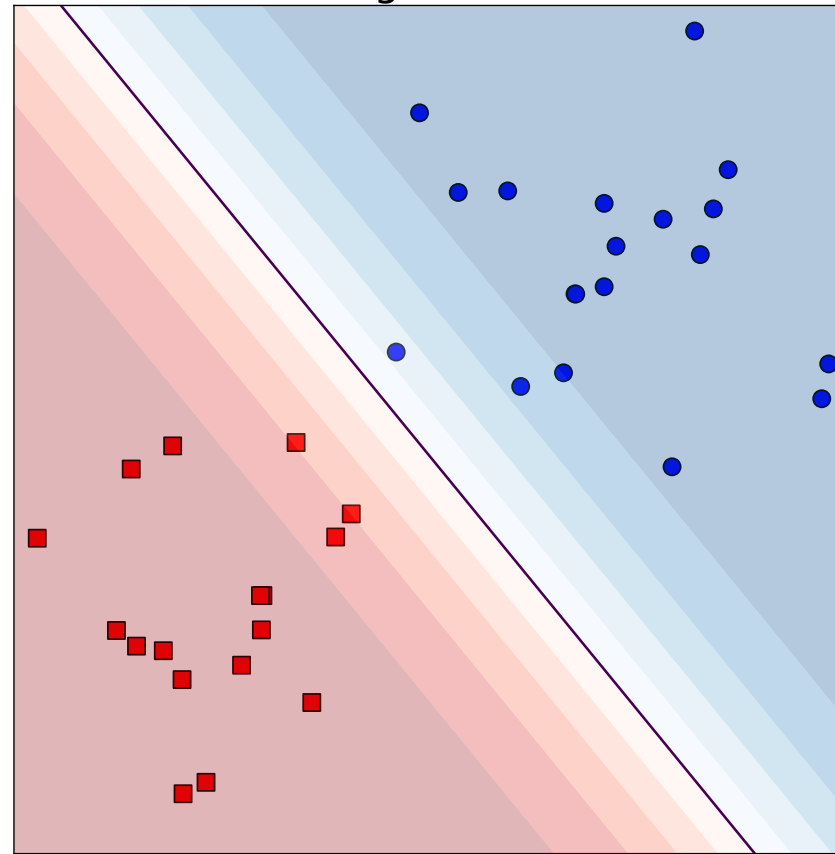
After Epoch 5:  
 $w_1 = -2.038, w_2 = -1.684, b = -0.170$   
total log loss: 0.065



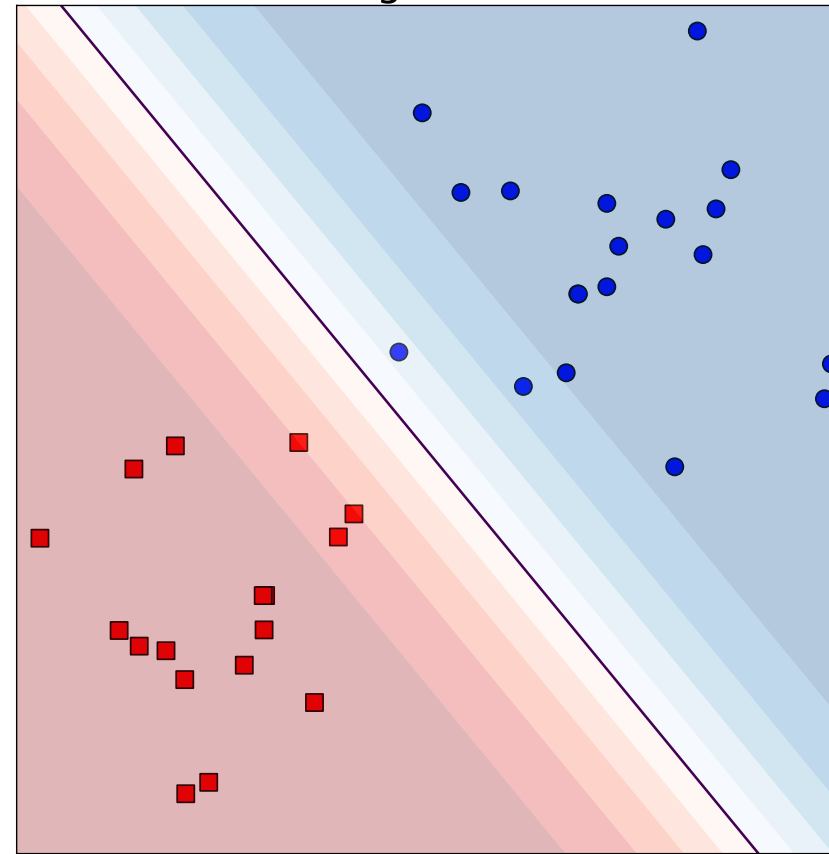
After Epoch 6:  
 $w_1 = -2.157, w_2 = -1.800, b = -0.191$   
total log loss: 0.058



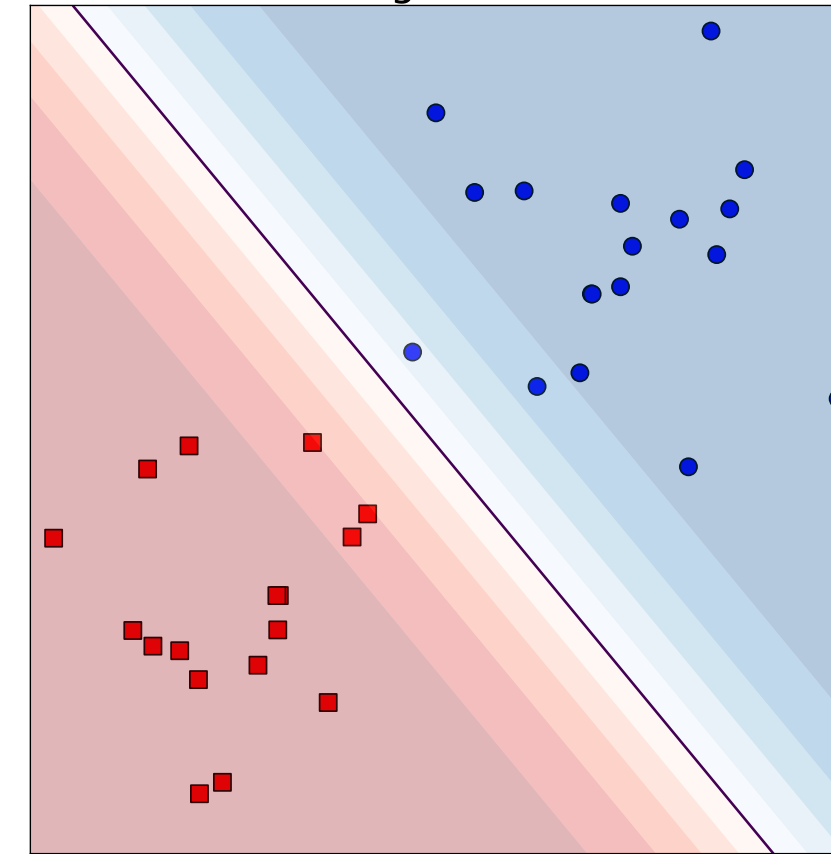
After Epoch 7:  
 $w_1 = -2.261, w_2 = -1.900, b = -0.206$   
total log loss: 0.053



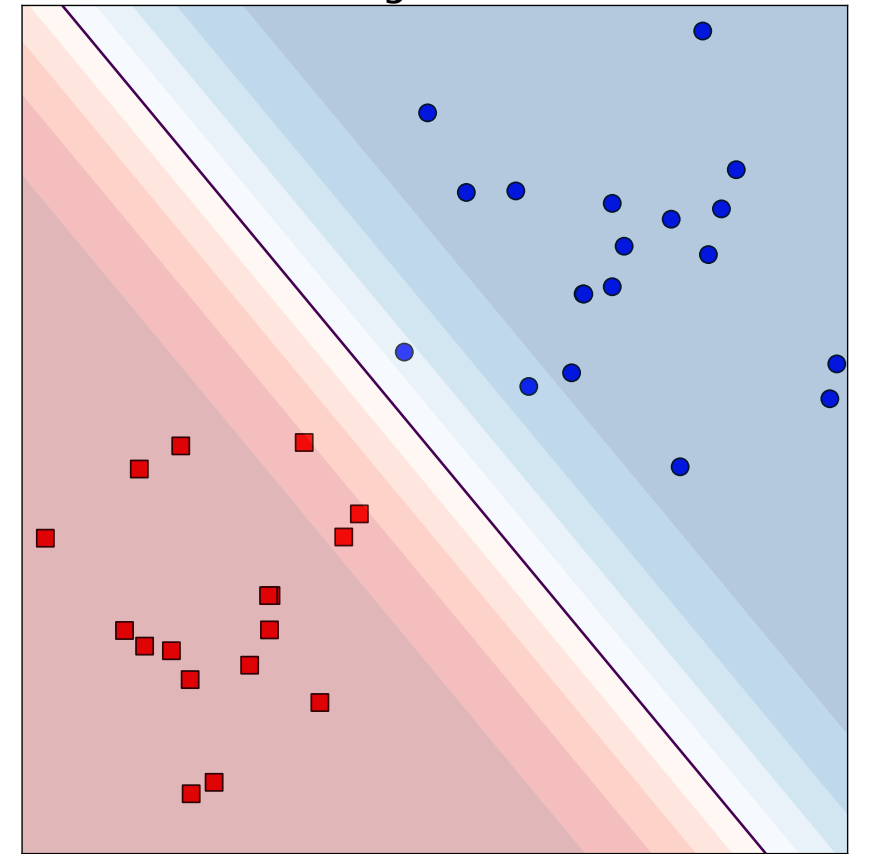
After Epoch 8:  
 $w_1 = -2.353, w_2 = -1.988, b = -0.224$   
total log loss: 0.049



After Epoch 9:  
 $w_1 = -2.439, w_2 = -2.070, b = -0.237$   
total log loss: 0.045



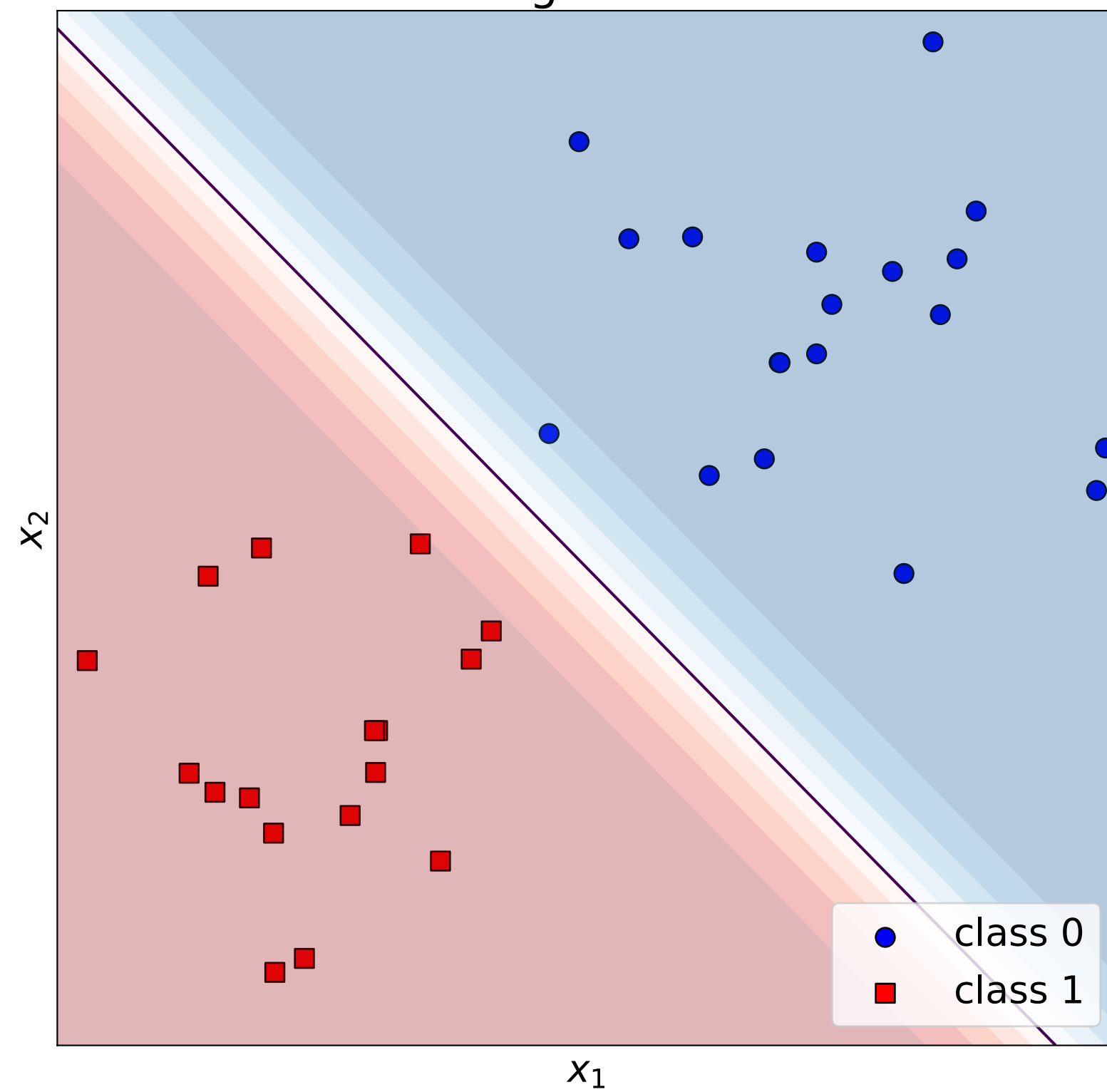
After Epoch 10:  
 $w_1 = -2.514, w_2 = -2.142, b = -0.253$   
total log loss: 0.042



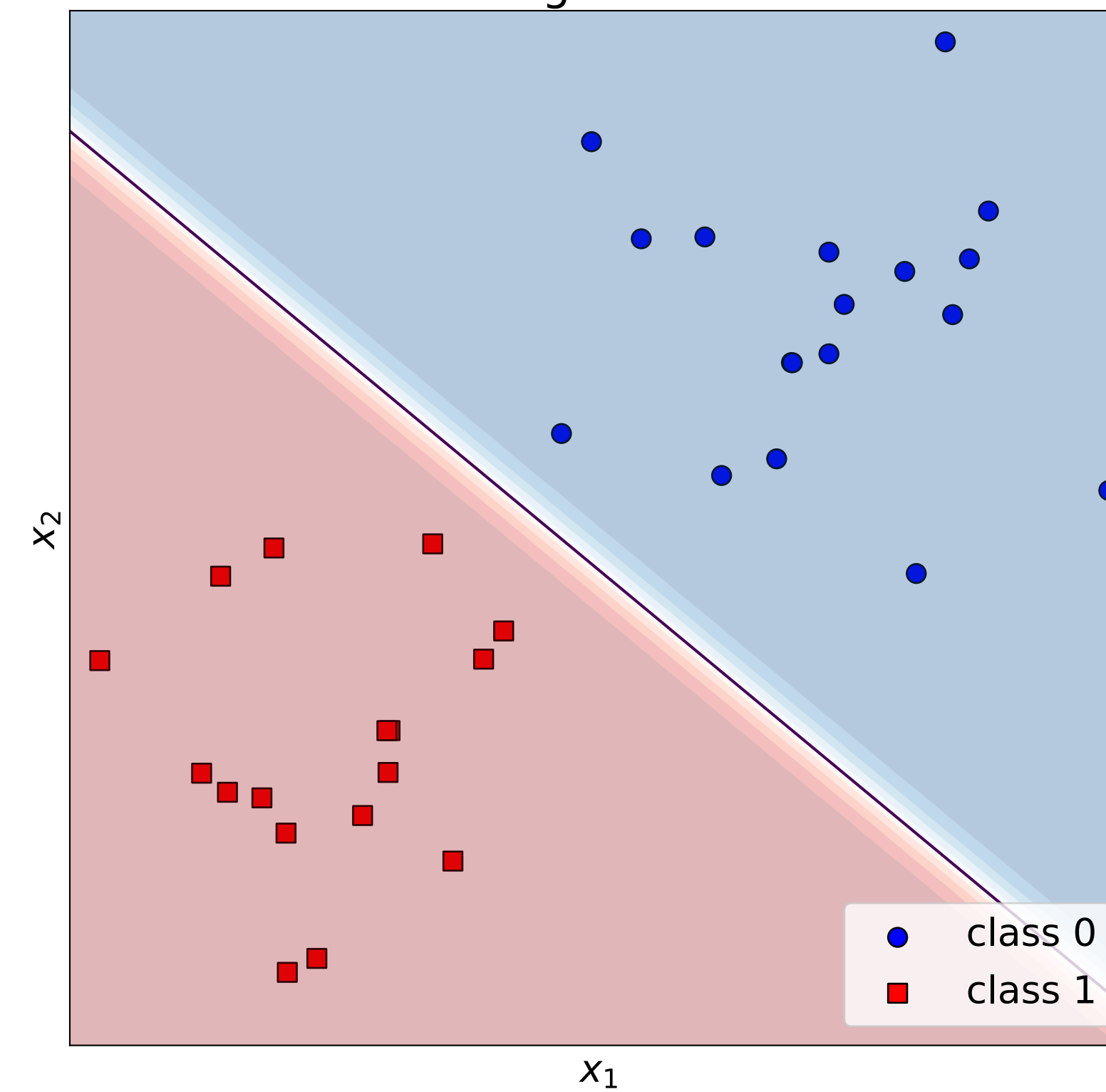
# “Don’t get cocky!”



After Epoch 100:  
 $w_1 = -4.496, w_2 = -4.300, b = -0.688$   
total log loss: 0.011



After Epoch 1000:  
 $w_1 = -11.223, w_2 = -13.181, b = -2.222$   
total log loss: 0.000



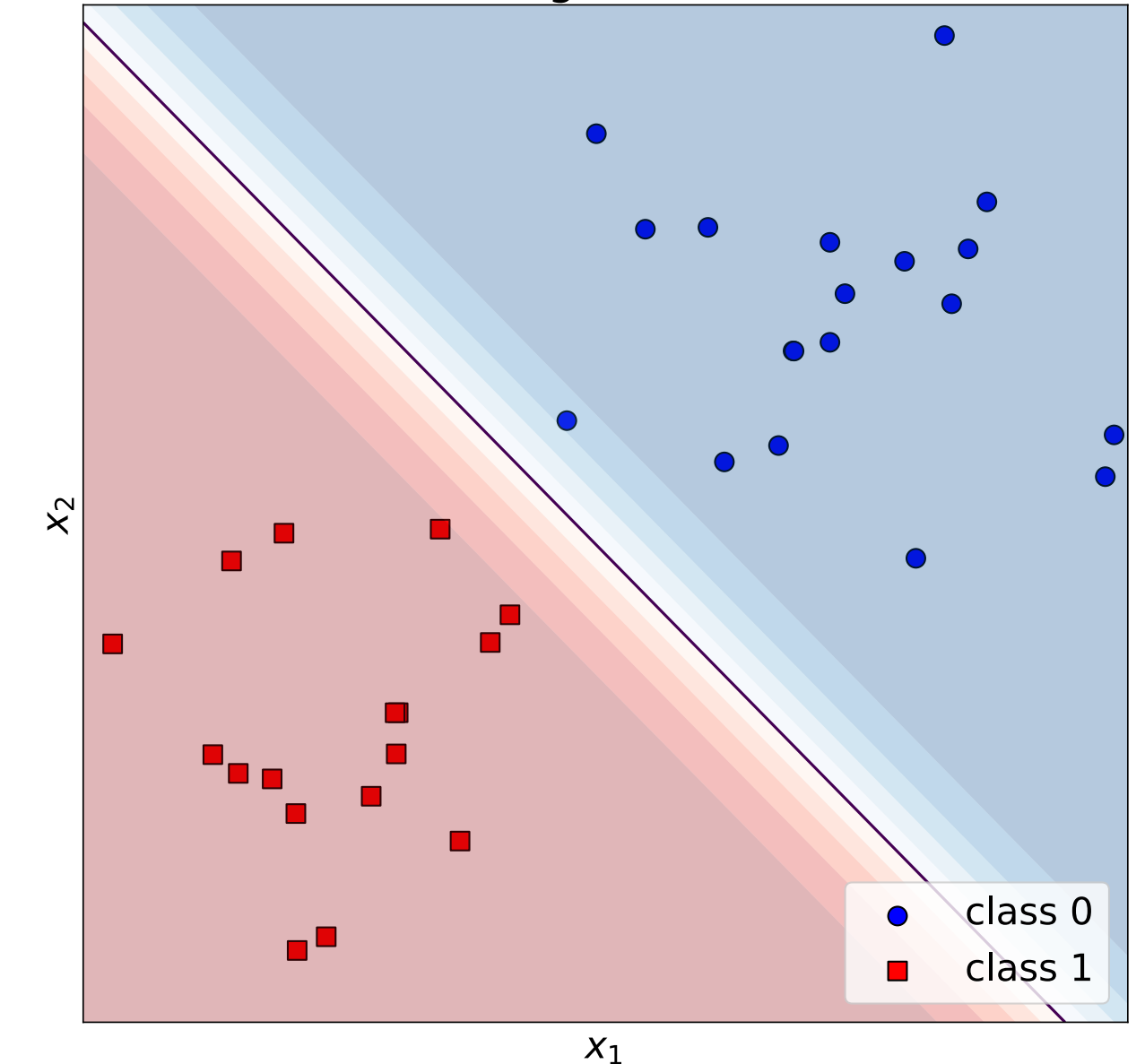
# Regularisation

- We are seeing a model that is far too confident near the boundary. It is overfitting to the training data
- And look... its weights are large!
- We can add regularisation as we did for regression

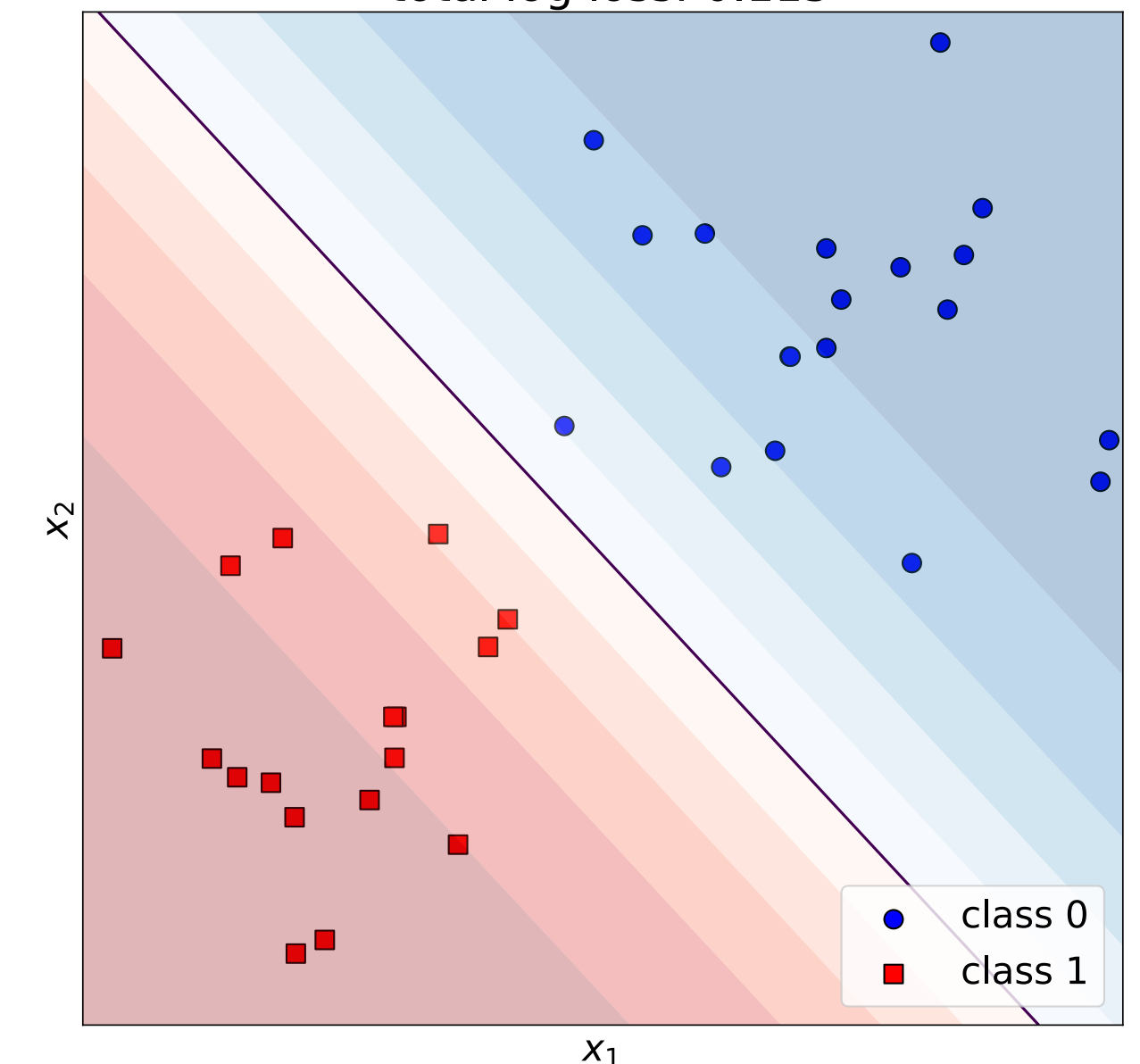
$$L_{total} = \underbrace{L_{log}}_{\text{classification}} + \underbrace{\frac{\lambda}{2} \|\mathbf{w}\|^2}_{\text{regularisation}}$$

We can use the validation set to find the optimal  $\lambda$

After Epoch 100:  
 $w_1 = -4.496, w_2 = -4.300, b = -0.688$   
total log loss: 0.011



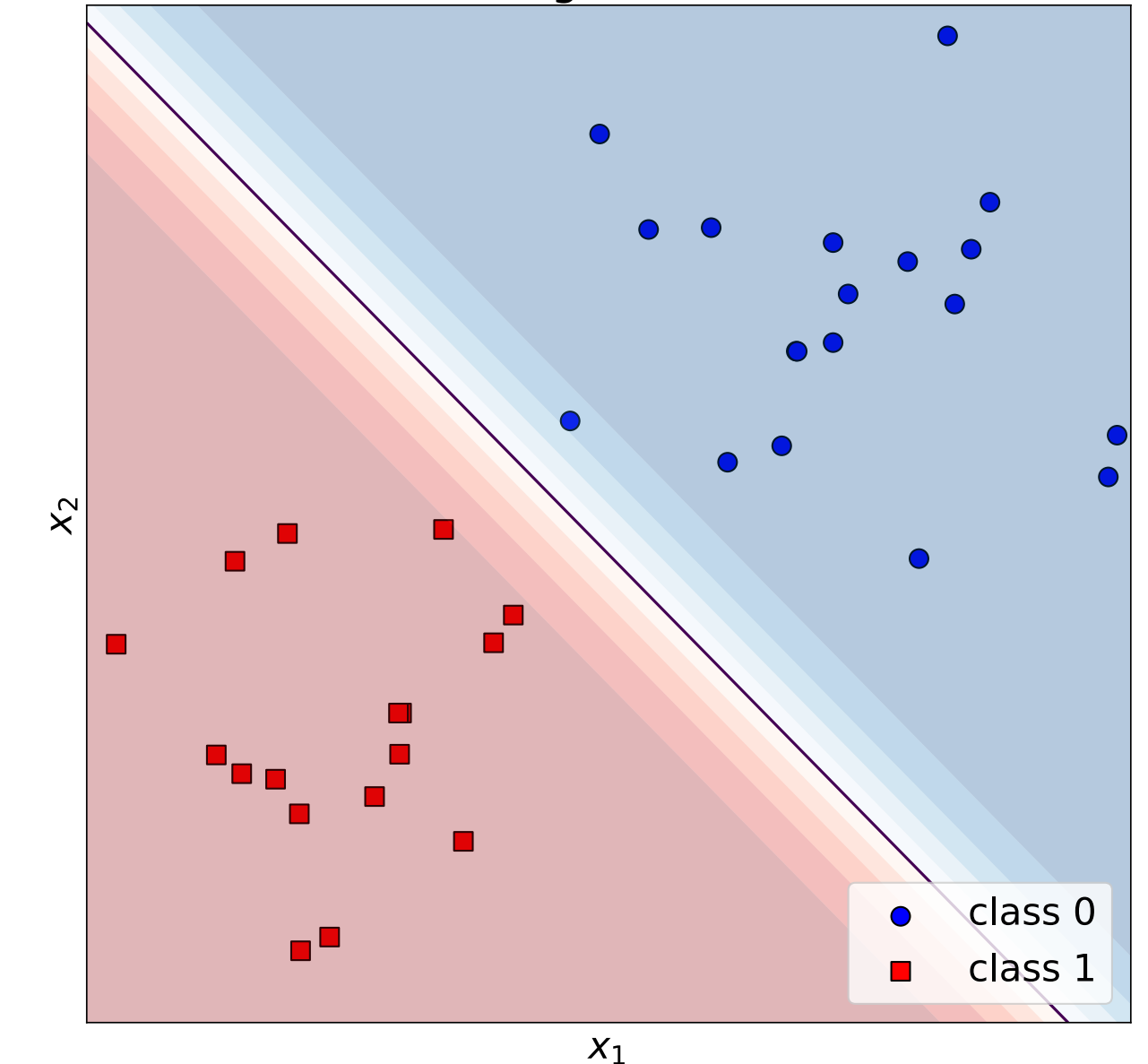
After Epoch 100 with regularisation:  
 $w_1 = -1.398, w_2 = -1.264, b = -0.184$   
total log loss: 0.115



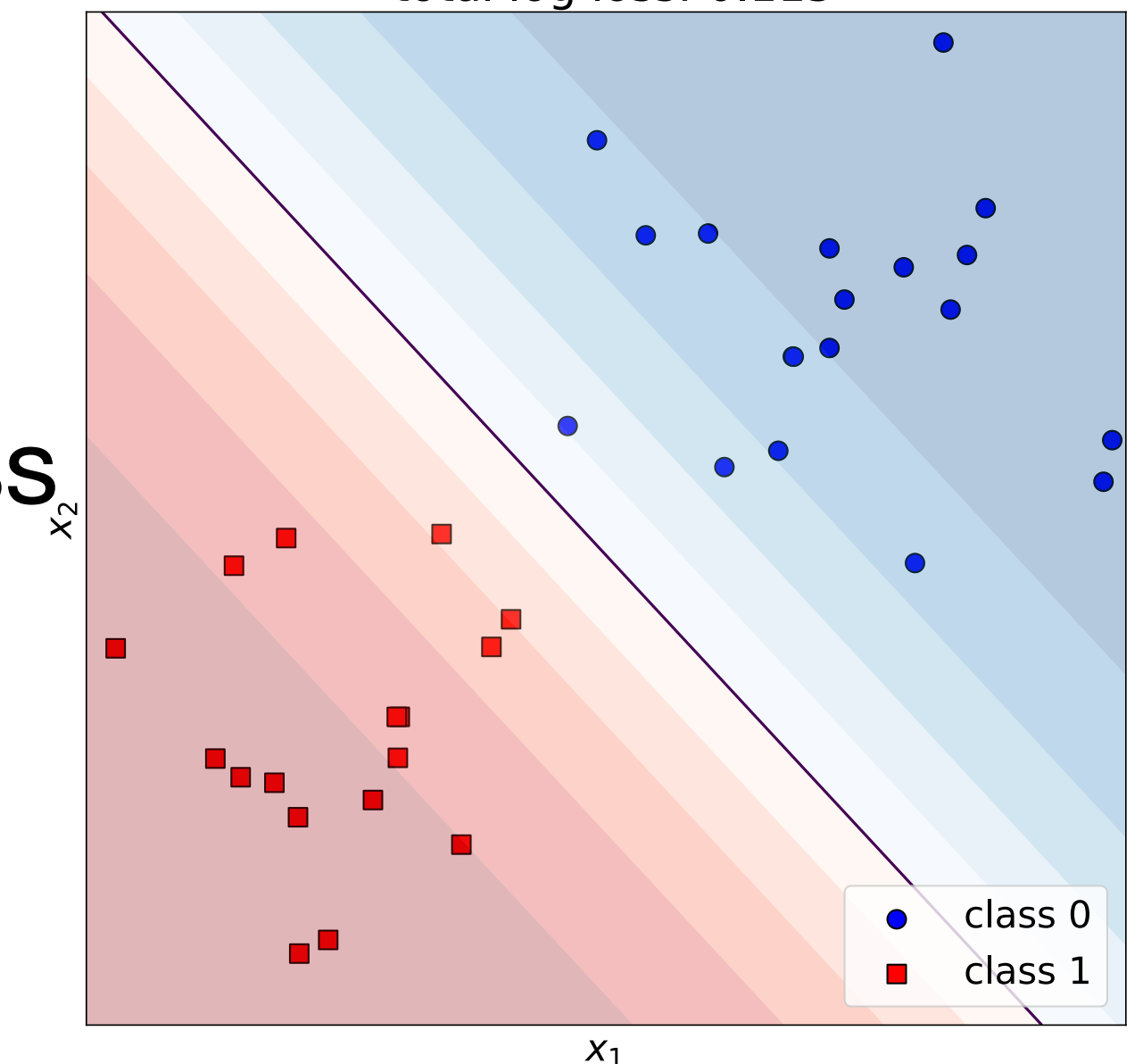
# Wait, why did that help?

- Consider the line of points that would correspond to some log-odds  $k$
- This is just the line  $k = \mathbf{w}^\top \mathbf{x} + b$
- The decision boundary is the line  $0 = \mathbf{w}^\top \mathbf{x} + b$
- The distance between these two parallel lines is  $\frac{k}{\|\mathbf{w}\|}$
- Regularising increases this distance, making the model less confident near the boundary

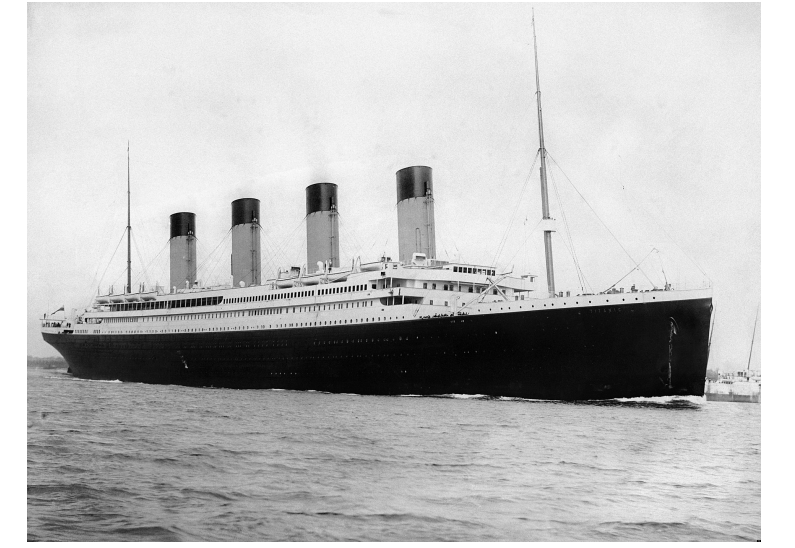
After Epoch 100:  
 $w_1 = -4.496, w_2 = -4.300, b = -0.688$   
total log loss: 0.011



After Epoch 100 with regularisation:  
 $w_1 = -1.398, w_2 = -1.264, b = -0.184$   
total log loss: 0.115



# 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

$$\mathbf{w} = \begin{bmatrix} \text{Pclass} & \text{Sex} & \text{Age} & \text{SibSp} & \text{Parch} & \text{Fare} \end{bmatrix}^T = [-0.97 \quad 1.27 \quad -0.52 \quad -0.27 \quad -0.03 \quad 0.16]^T$$

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

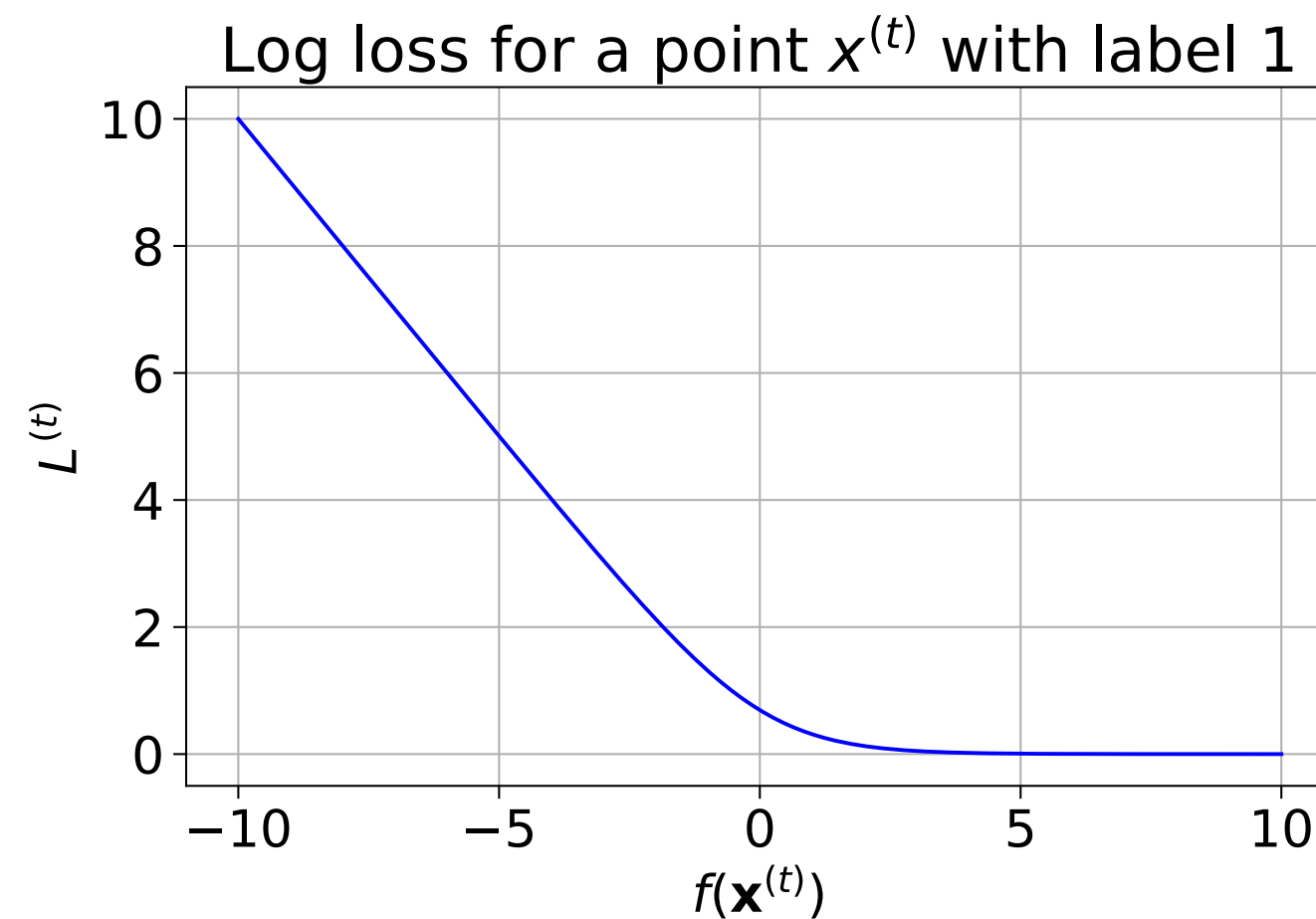
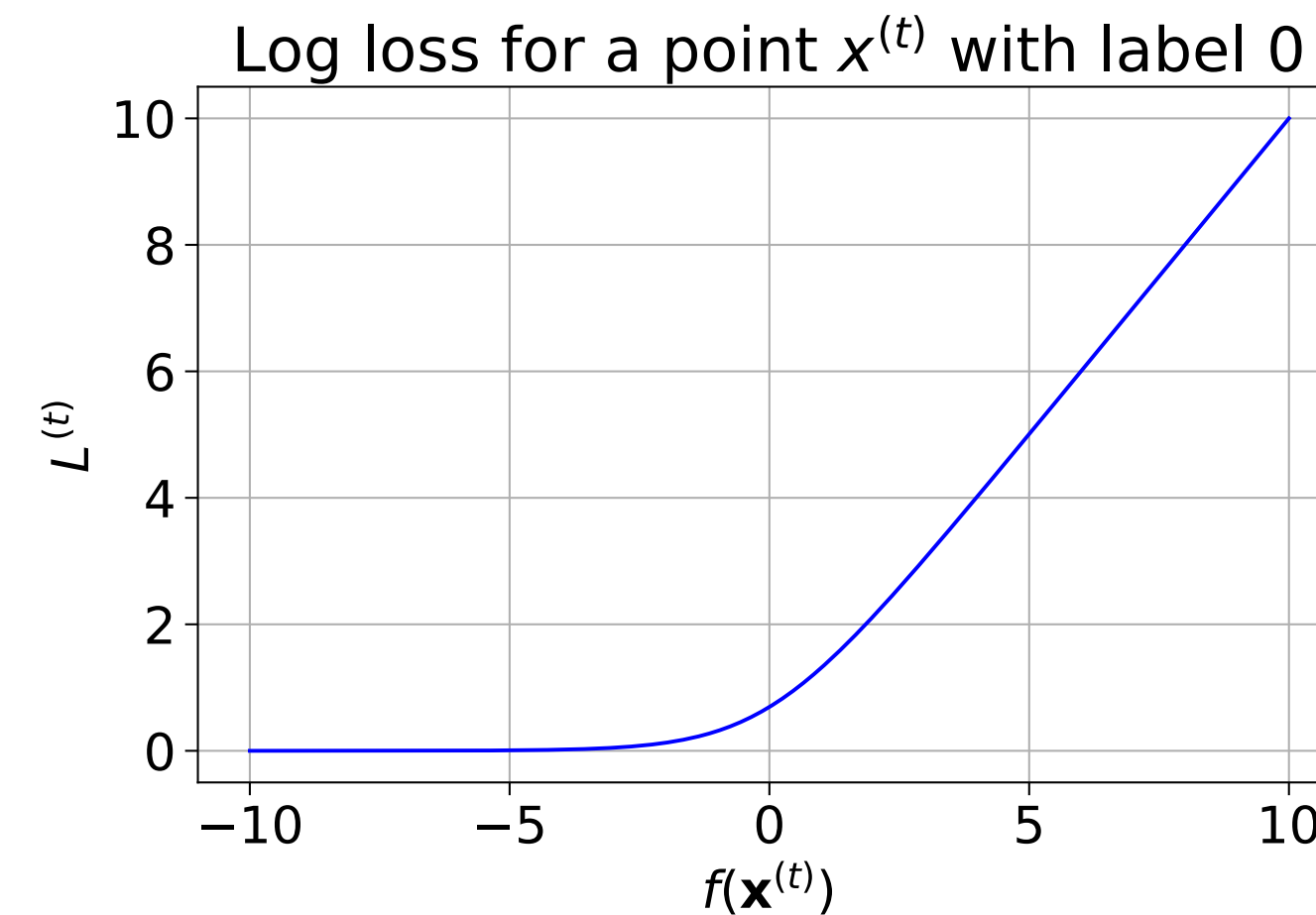


Gets 80% on held-out data so is a reasonable model

# Logistic regression in short

- Given a linear model  $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ , logistic regression is solving minimise  $L_{log}$  over  $\mathbf{w}, b$

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



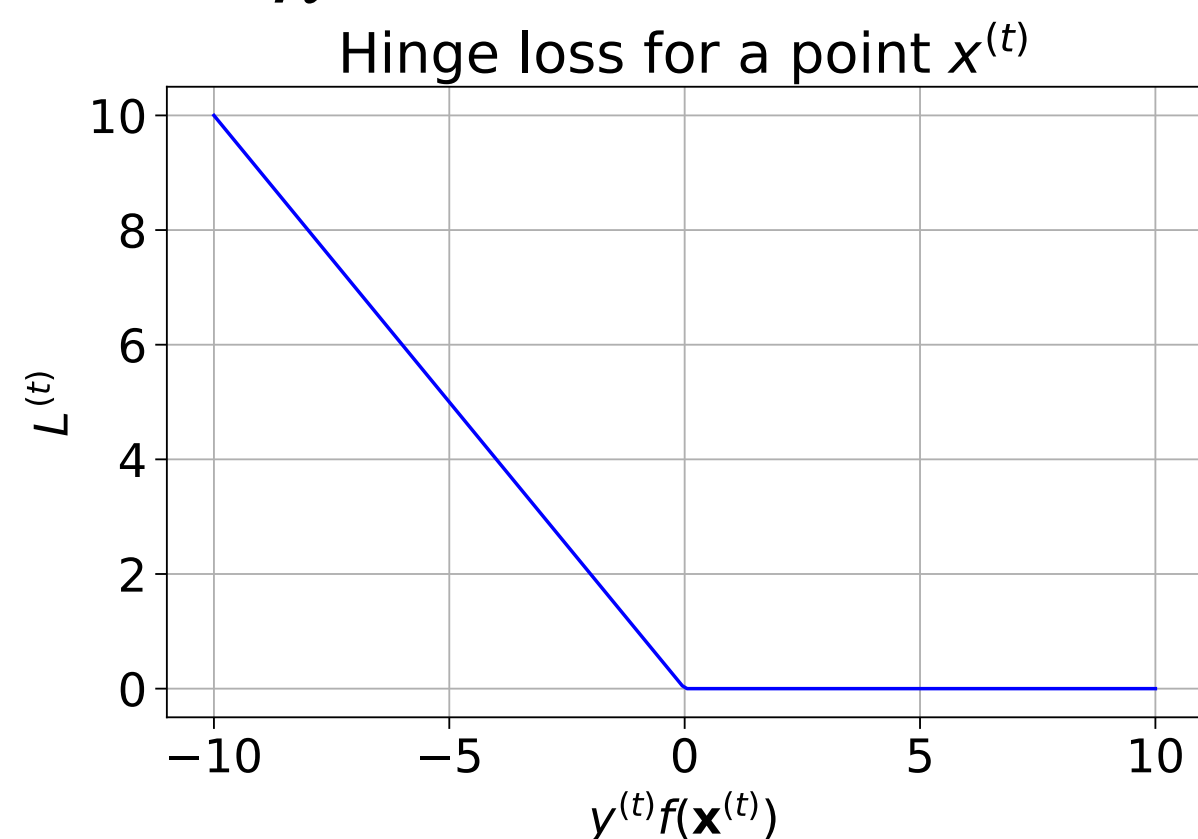
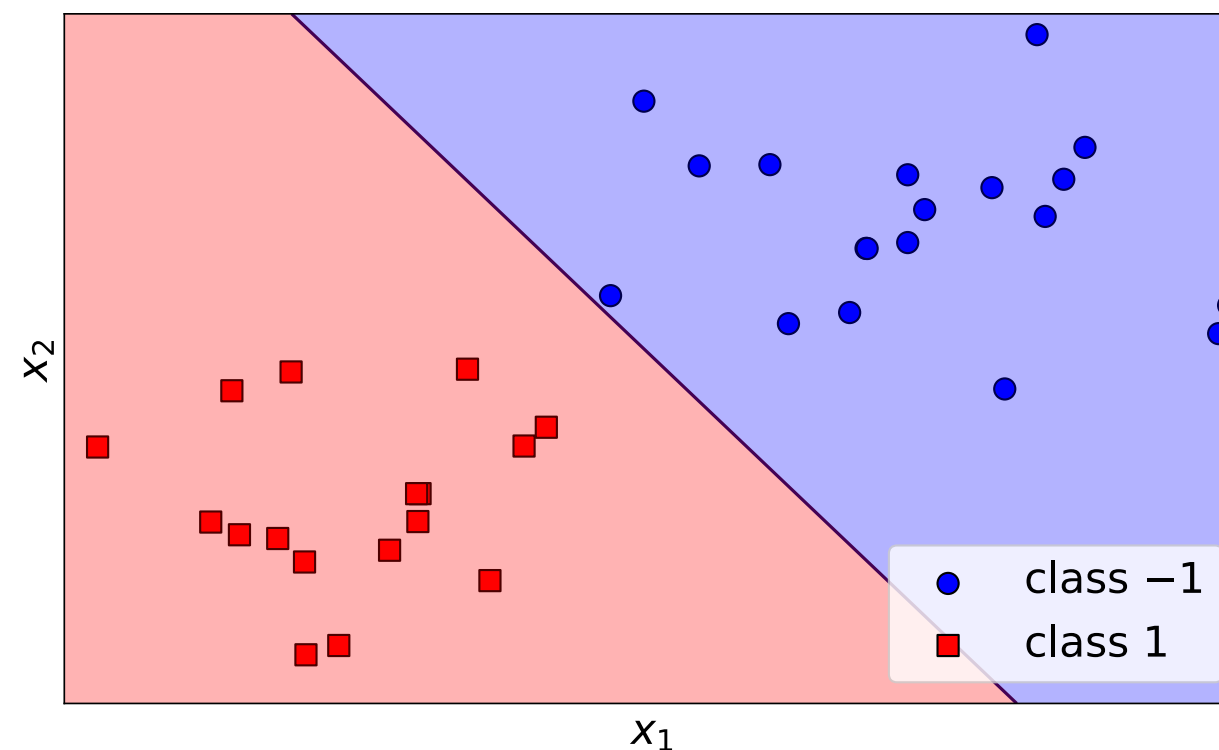
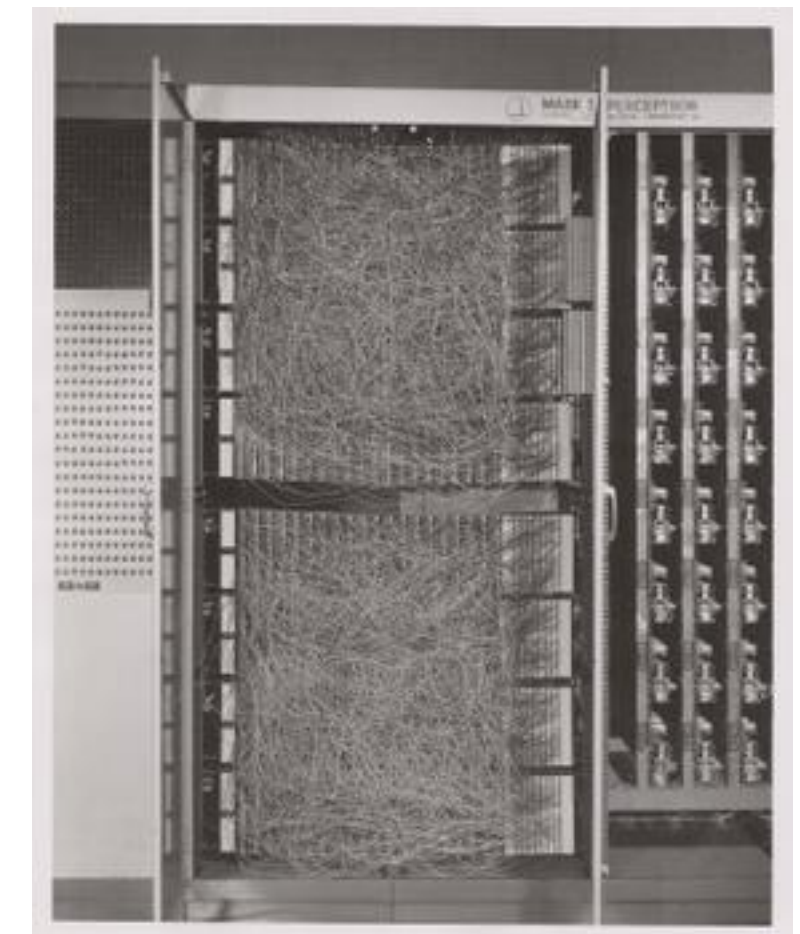
- There are other methods that boil down to minimising different losses



# Perceptron learning

- Consider a training set  $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$  where  $\mathbf{x} \in \mathbb{R}^D$  and  $y \in \{-1, 1\}$
- Given a linear model  $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$  and threshold function  $\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) \geq 0 \\ -1 & \text{if } f(\mathbf{x}) < 0 \end{cases}$  the perceptron learning algorithm is equivalent to solving minimise  $L_{\text{hinge}}$  over  $\mathbf{w}, b$

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

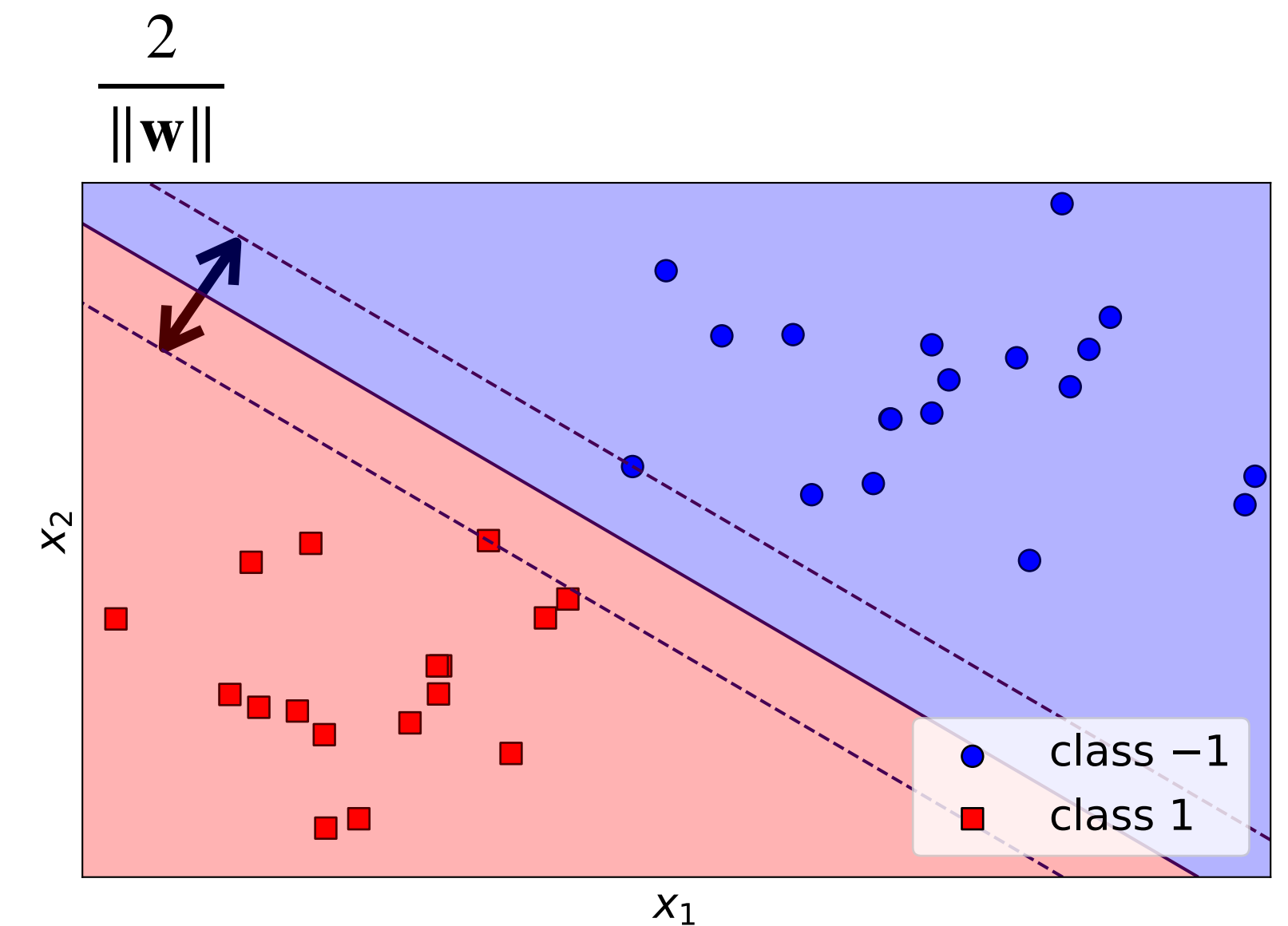


<https://en.wikipedia.org/wiki/Perceptron>

# Support Vector Machines (SVMs)

- Consider a training set  $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$  where  $\mathbf{x} \in \mathbb{R}^D$  and  $y \in \{-1, 1\}$
- Given a linear model  $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$  and threshold function  $\hat{y} = \begin{cases} 1 & \text{if } f(\mathbf{x}) \geq 0 \\ -1 & \text{if } f(\mathbf{x}) < 0 \end{cases}$   
(linear) SVM learning is equivalent to solving minimise  $L_{SVM}$   
 $\mathbf{w}, b$

$$L_{SVM} = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_n \max\left(0, 1 - y^{(n)} f(\mathbf{x}^{(n)})\right)$$



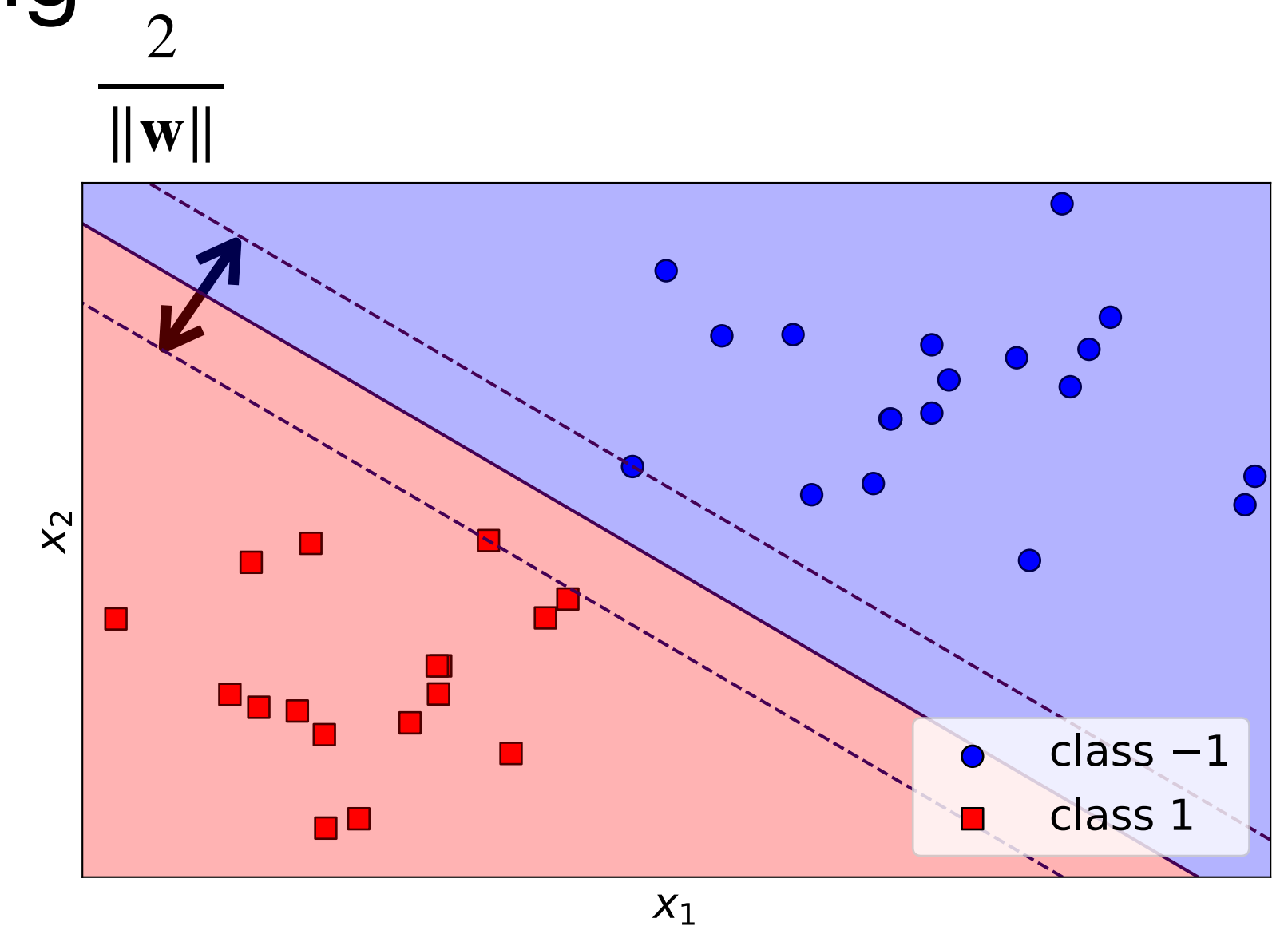
# SVMs are maximum margin classifiers

$$L_{SVM} = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_n \max\left(0, 1 - y^{(n)} f(\mathbf{x}^{(n)})\right)$$

If we define the margin as the region where  $|f(\mathbf{x})| < 1$  then:

- The first term in  $L_{SVM}$  is small when the margin is big
- The second term in  $L_{SVM}$  is small when points don't live in the margin
- $C$  is a hyperparameter that controls the relative importance of these terms

Logistic regression +  
sufficient regularisation also  
gives a large margin



# Multinomial Logistic Regression

# Multi-class classification with linear classifiers

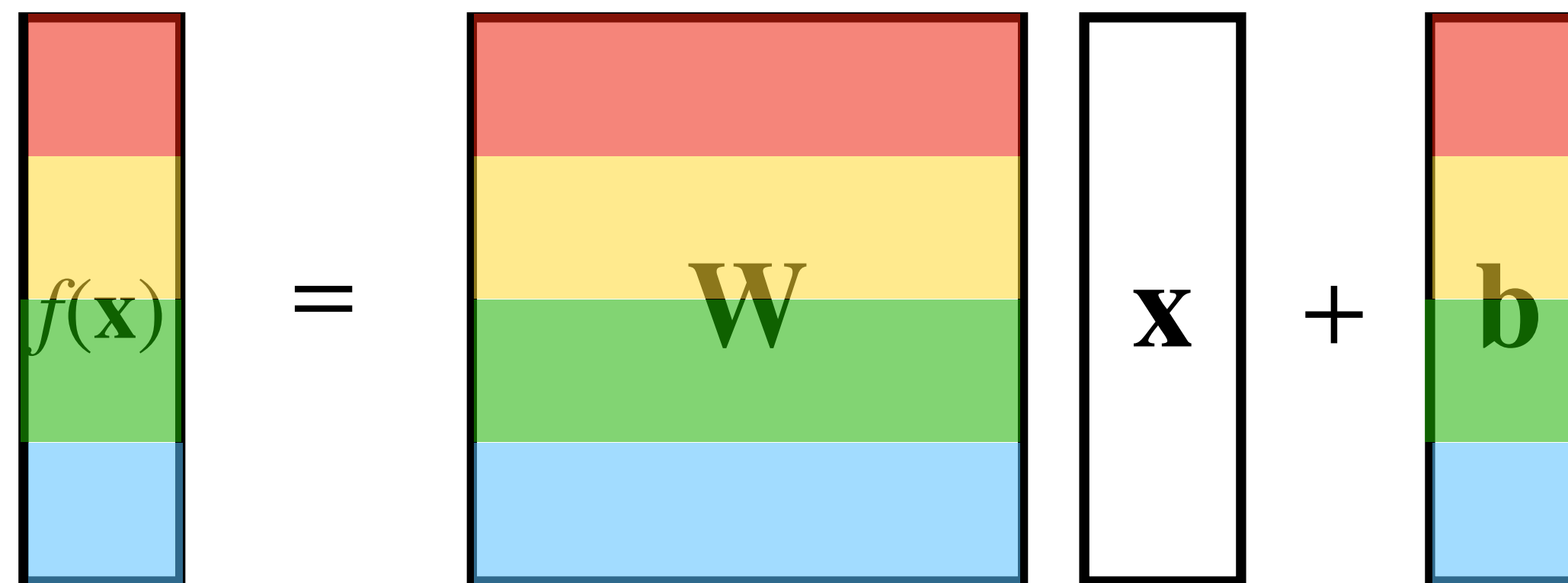
Now consider the multi-class scenario  $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$  with  $y \in \mathbb{Z}_{<K}^+ = \{0, 1, \dots, K-1\}$ .

There are three different approaches to solving this:

1. We could learn  $K$  one-vs-rest classifiers:  $f_0(\mathbf{x}), f_1(\mathbf{x}), \dots, f_{K-1}(\mathbf{x})$  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 linear model output a **vector** where each element is a score for a different class and select the class with the highest score

# A multi-class linear model

- In the binary case  $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$  where  $\mathbf{x} \in \mathbb{R}^D$  and  $f(\mathbf{x}) \in \mathbb{R}$
- For our model 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  $b \in \mathbb{R}$  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  
 $K$  models  
side-by-side

# Multinomial logistic regression

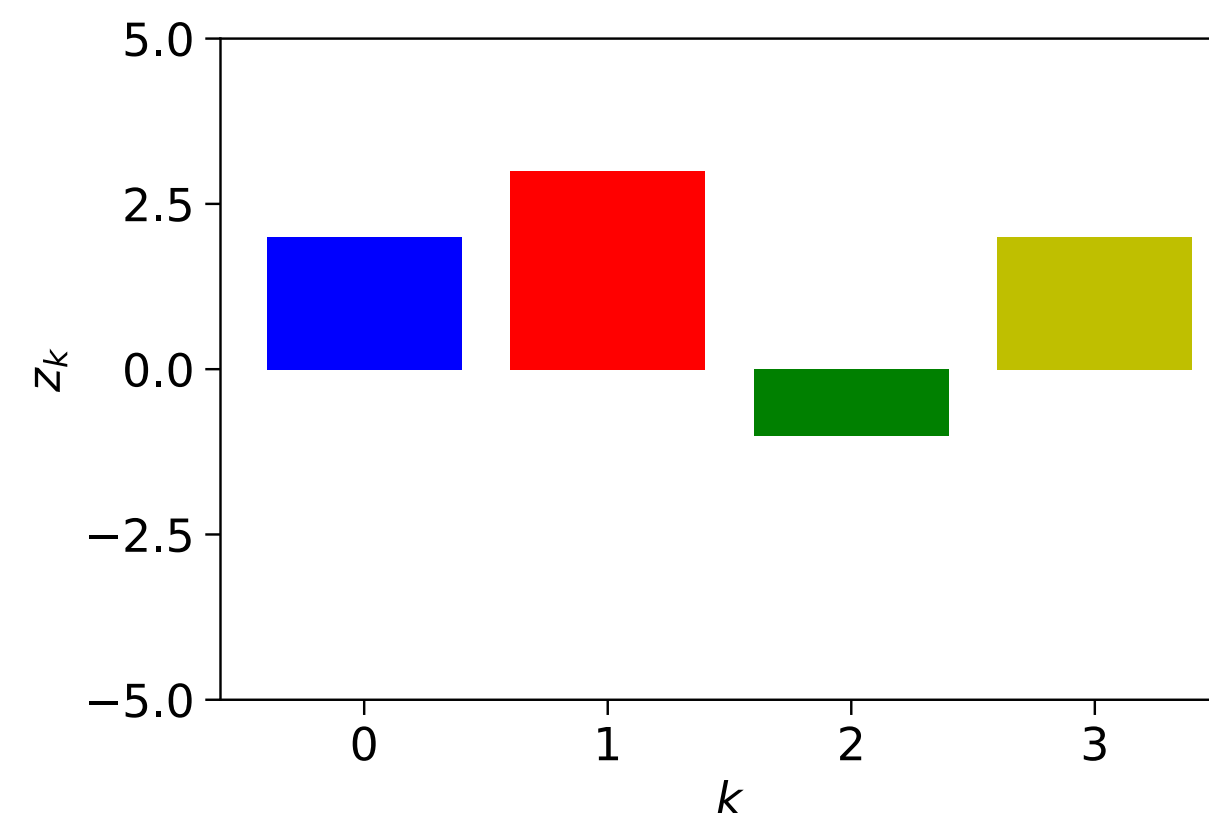
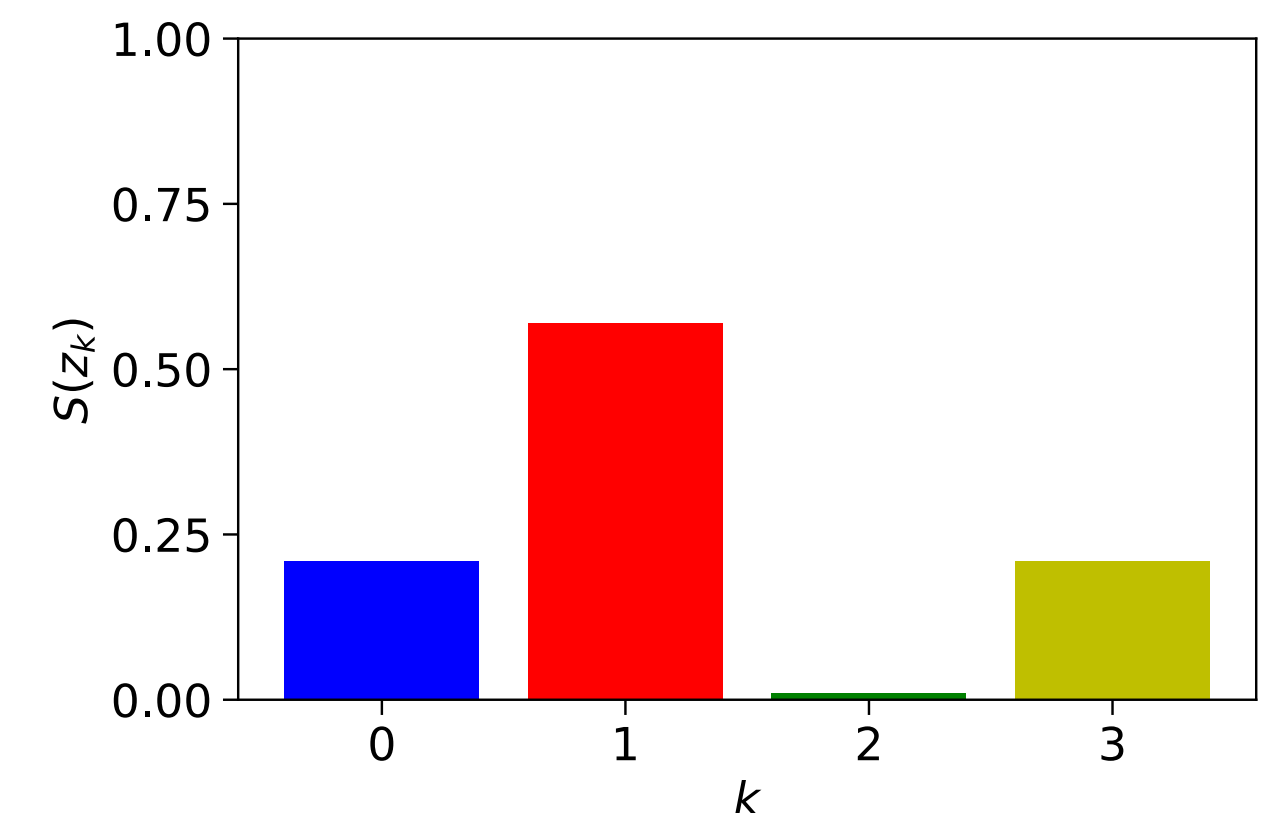
- Logistic regression naturally extends to multi-class problems
- In the binary setting, we just had  $f(\mathbf{x}) \in \mathbb{R}$  as the log-odds for class 1
- We now have  $f(\mathbf{x}) \in \mathbb{R}^K$ . These are the **logits** for each class
- They are unnormalised log-probabilities; a generalisation of log-odds
- Let's store the actual probabilities in a vector  $\mathbf{p}$  and relate these to the logits via some function  $S$

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

# 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 and all values are between 0 and 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 multinomial logistic regression

- We have  $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$  and  $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$  and want to solve minimise  $L_{log}$  over  $\mathbf{W}, \mathbf{b}$
- If we **one-hot encode** our labels as vectors then we can write the log loss as
$$L_{log} = \frac{1}{N} \sum -\mathbf{y}^{(n)\top} \log \mathbf{p}^{(n)}$$
- $\mathbf{y} \in \mathbb{R}^K$  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  $\mathbf{y}^{(t)} = [0 \ 0 \ 1 \ 0 \ 0 \ 0]^\top$
- We can use a gradient-based optimiser with  $\nabla_{\mathbf{W}} L_{log}$  and  $\nabla_{\mathbf{b}} L_{log}$

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

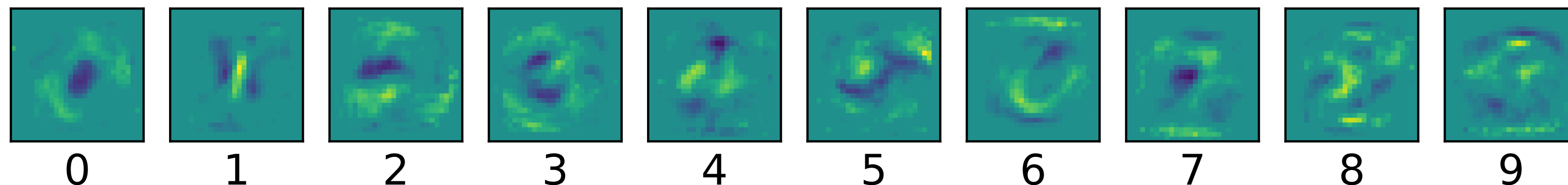
See Murphy p346 for a derivation, noting differences in notation.

# Digit classification on MNIST

- MNIST dataset has 70k images (split 60k/10k train/test)
- Images are  $28 \times 28$  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
- Predict according to most probable class:  $\hat{y} = \underset{k}{\operatorname{argmax}} \mathbf{p} = \underset{k}{\operatorname{argmax}} f(\mathbf{x})$
- Test accuracy: 89.4% (or test error: 10.6%)



[https://en.wikipedia.org/wiki/MNIST\\_database#/media/File:MnistExamples.png](https://en.wikipedia.org/wiki/MNIST_database#/media/File:MnistExamples.png)



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

# Summary

- We have seen that a linear classifier is a linear model plus a threshold function and looked at its decision boundary
- We have found out how to perform logistic regression for binary classification
- We have briefly looked at perceptrons and SVMs
- We have seen how to adapt logistic regression for multi-class classification

# Questions

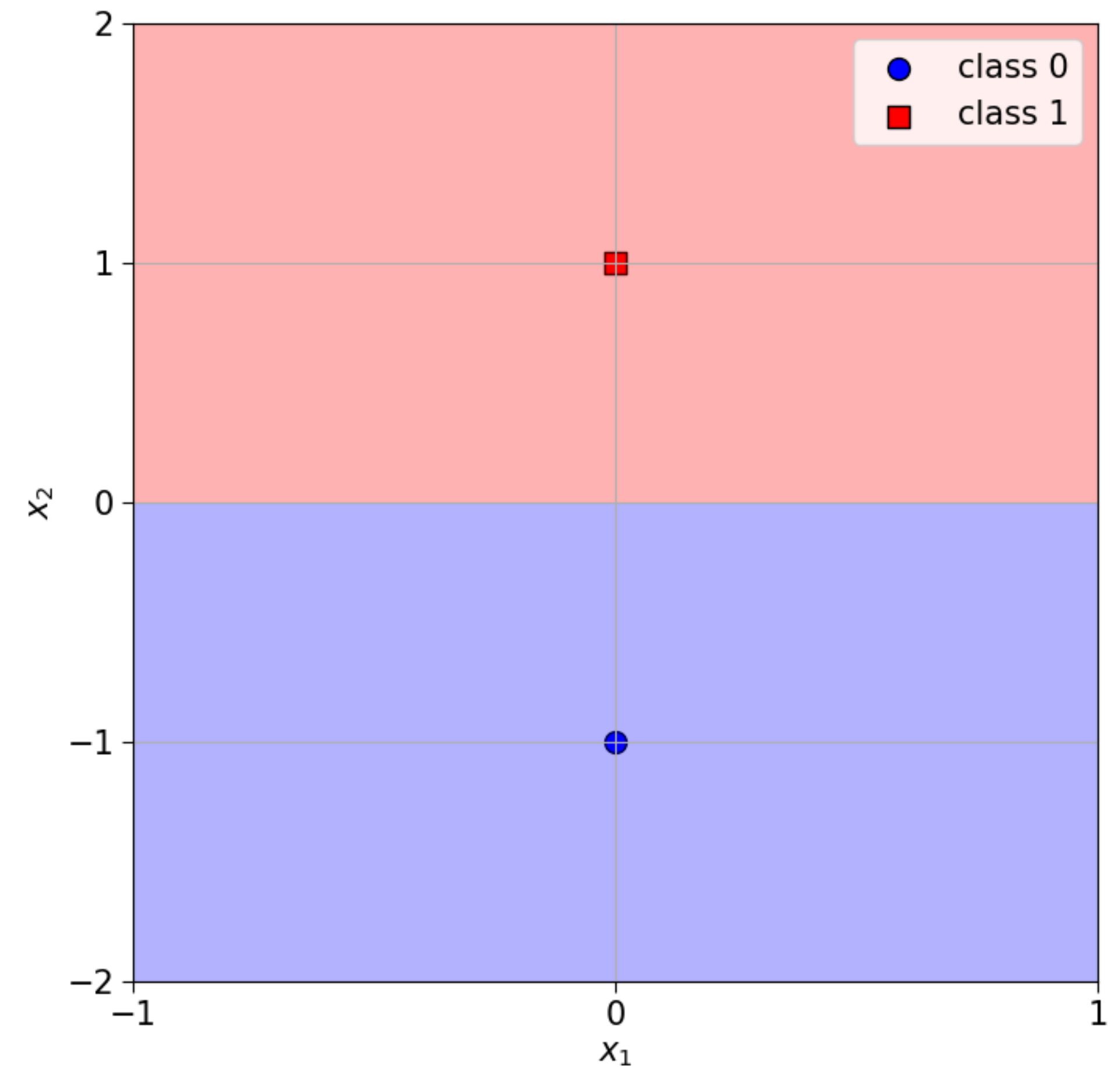
# Q1

You are given a linear model  $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$  where  $\mathbf{w} = [1 \quad -1 \quad 2]^\top$  and  $b = 1$ . This has been trained using logistic regression for a binary classification problem ( $y \in \{0,1\}$ )

Compute  $p(y = 0 \mid \mathbf{x}^{(t)})$  for  $\mathbf{x}^{(t)} = [0 \quad 2 \quad 1]^\top$

# Q2

Compute the log loss, averaged across training points, for the model in the figure given that  $\|\mathbf{w}\|=1$  and the model was fit using logistic regression.



# Q3

Consider the models in the figure which are both trained through logistic regression on the same training set.

1. Which model has a higher log loss?

2. Which model has a higher  $||\mathbf{w}||$ ?

