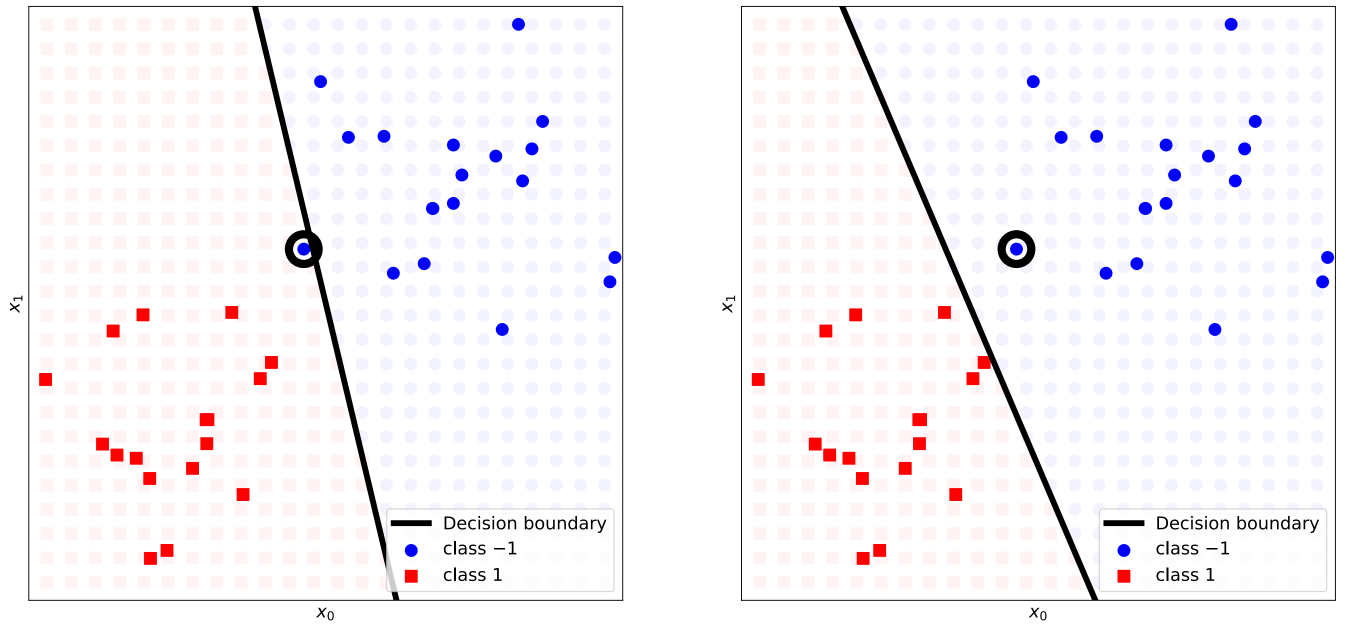
# Data Analysis and Machine Learning 4

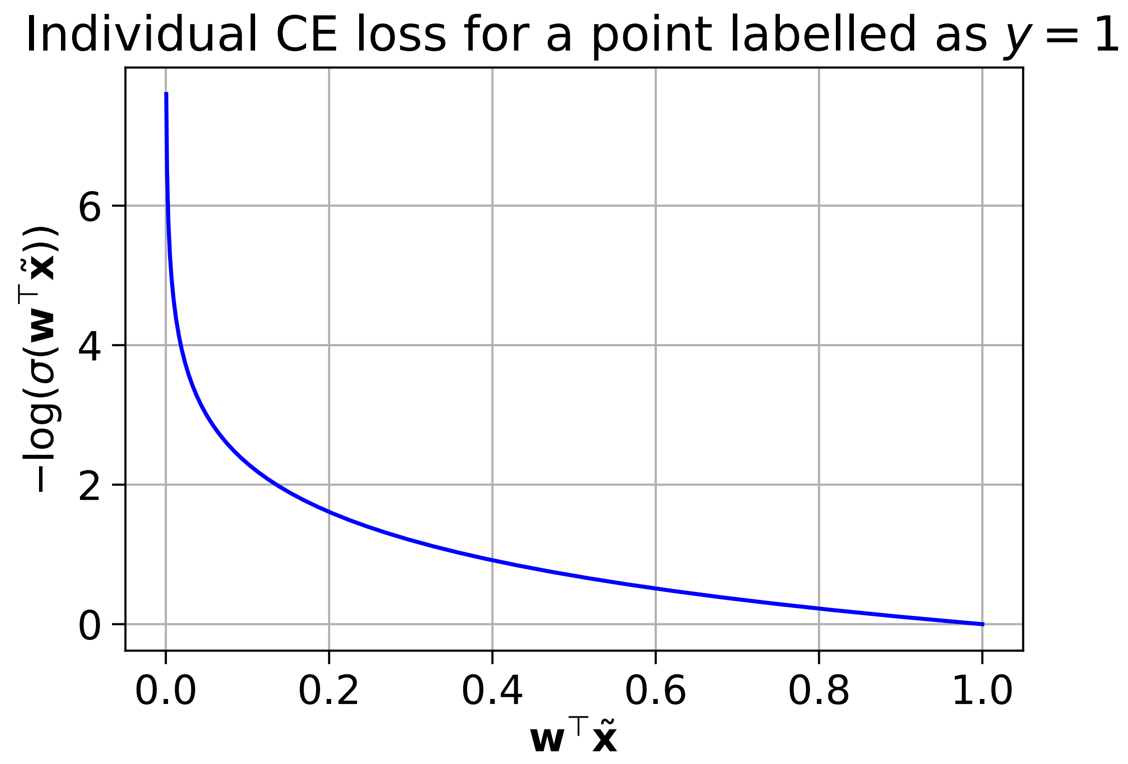
## Week 7: Support Vector Machines

Elliot J. Crowley, 6th March 2023

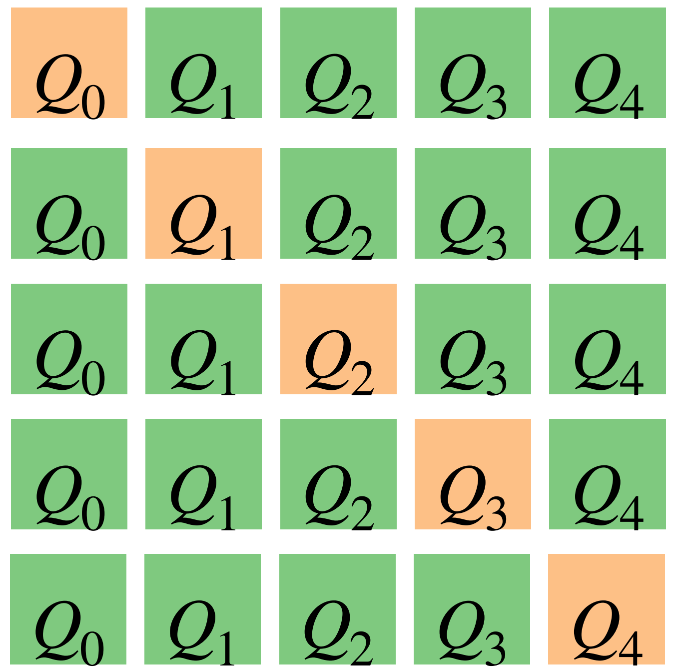
# Recap

- We considered the perceptron algorithm and logistic regression for learning the weights of linear classifiers



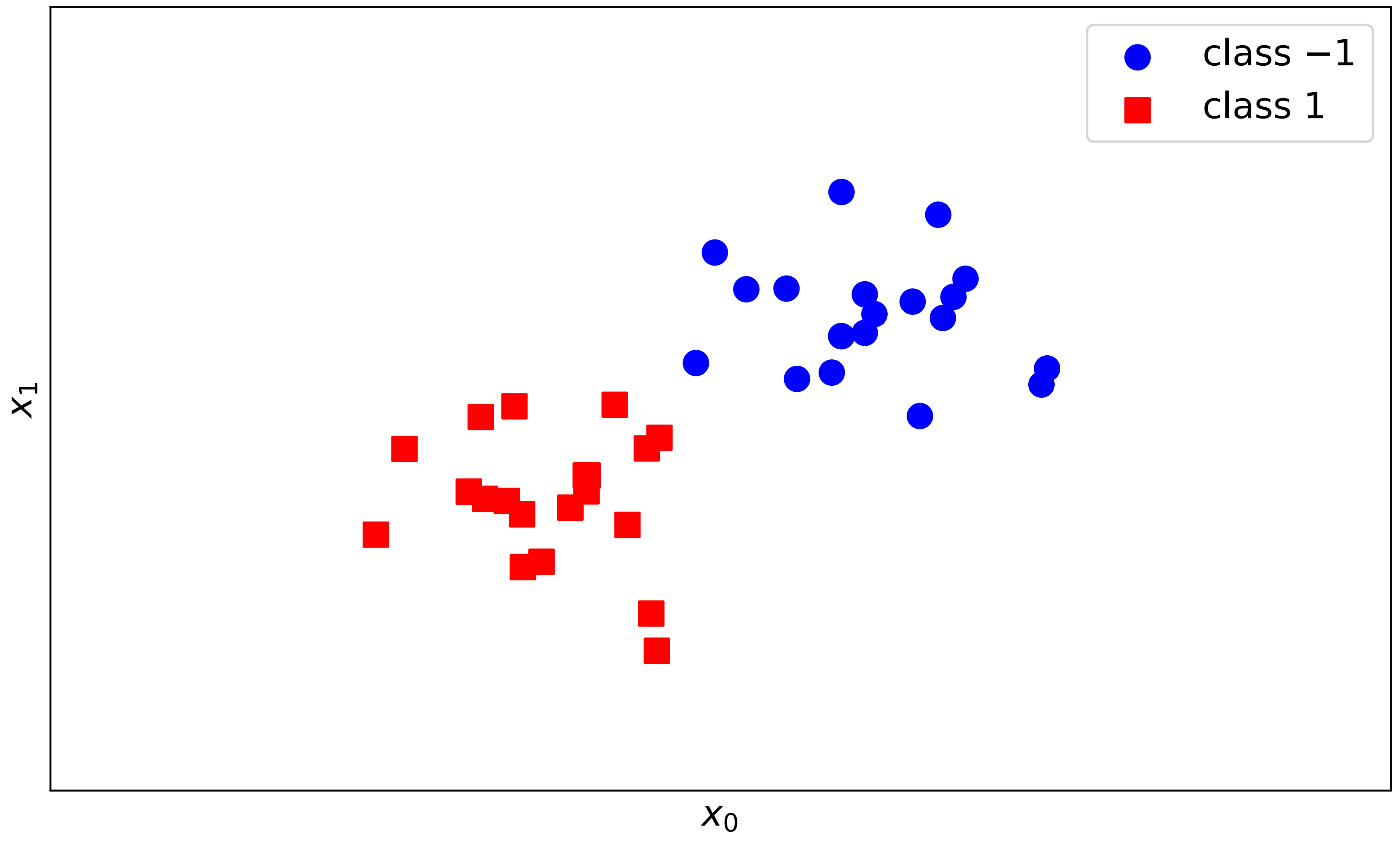

- We learnt about cross validation and how it can be combined with grid search

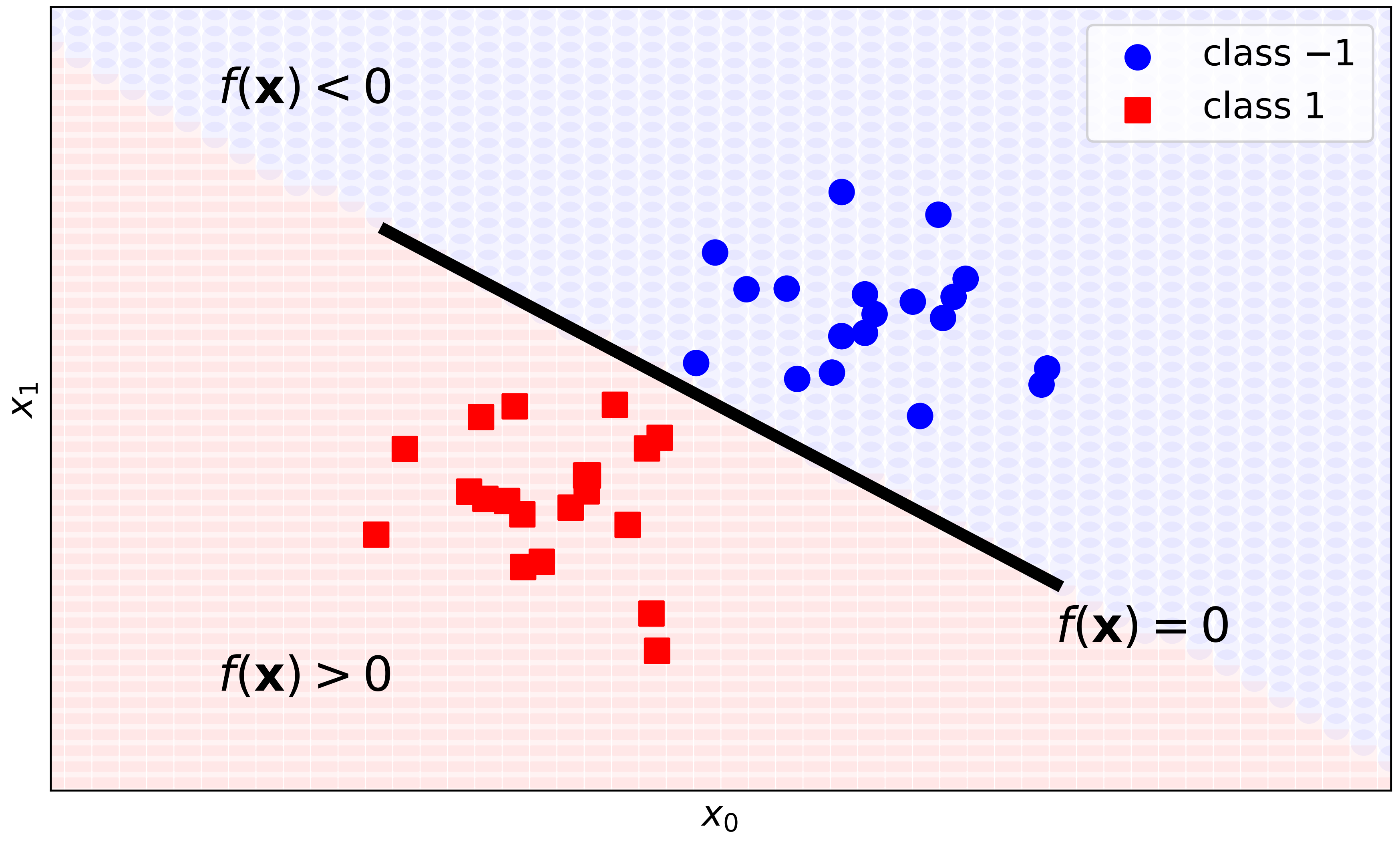


# Support Vector Machines (SVMs)

# Linear classifier decision boundary

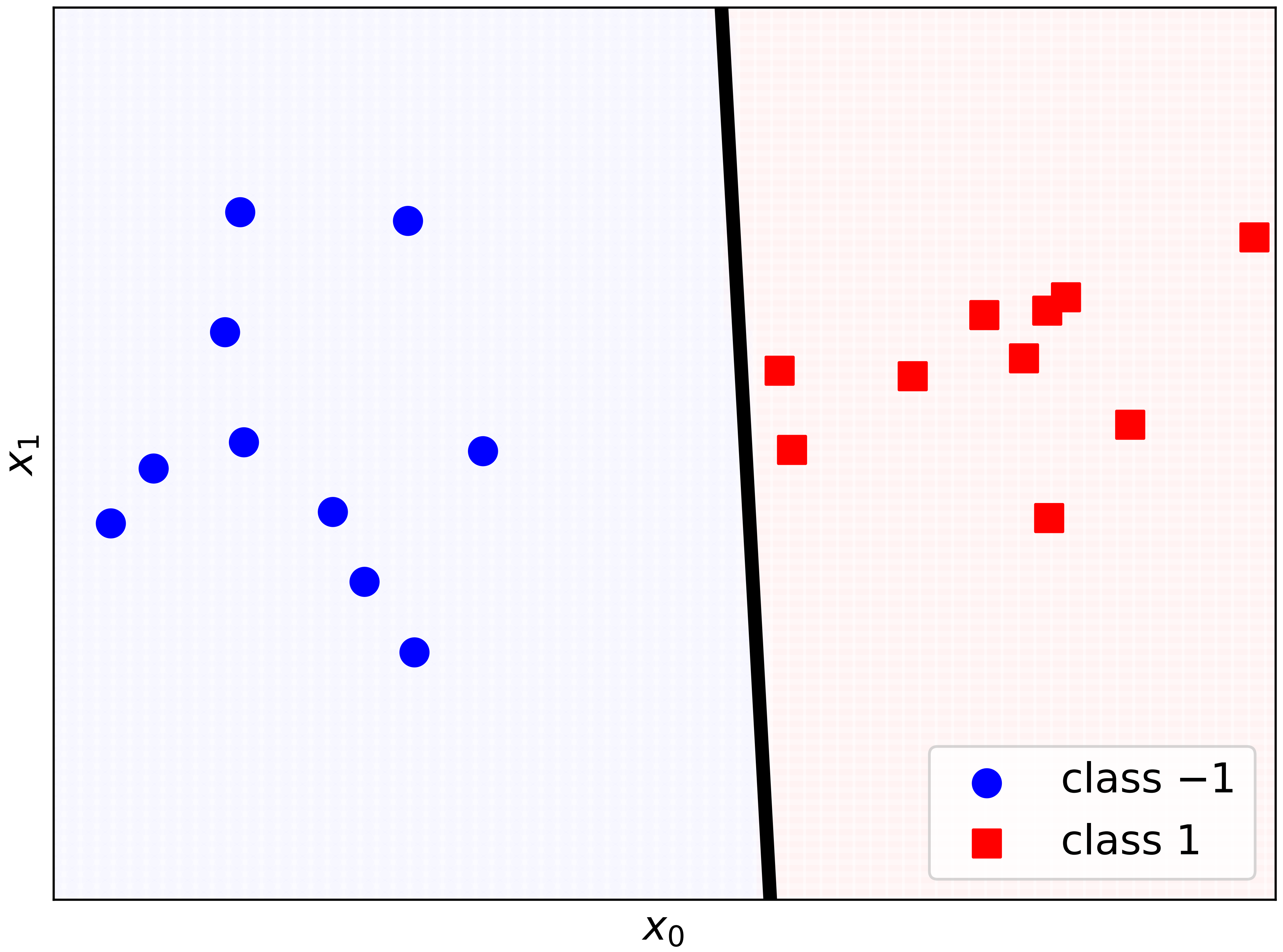
- Consider a training set $\{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$ with $\mathbf{x} \in \mathbb{R}^D$ and $y \in \{-1,1\}$
- We have $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ where $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}^1$
- Predictions are determined using $\hat{y} = \begin{cases}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



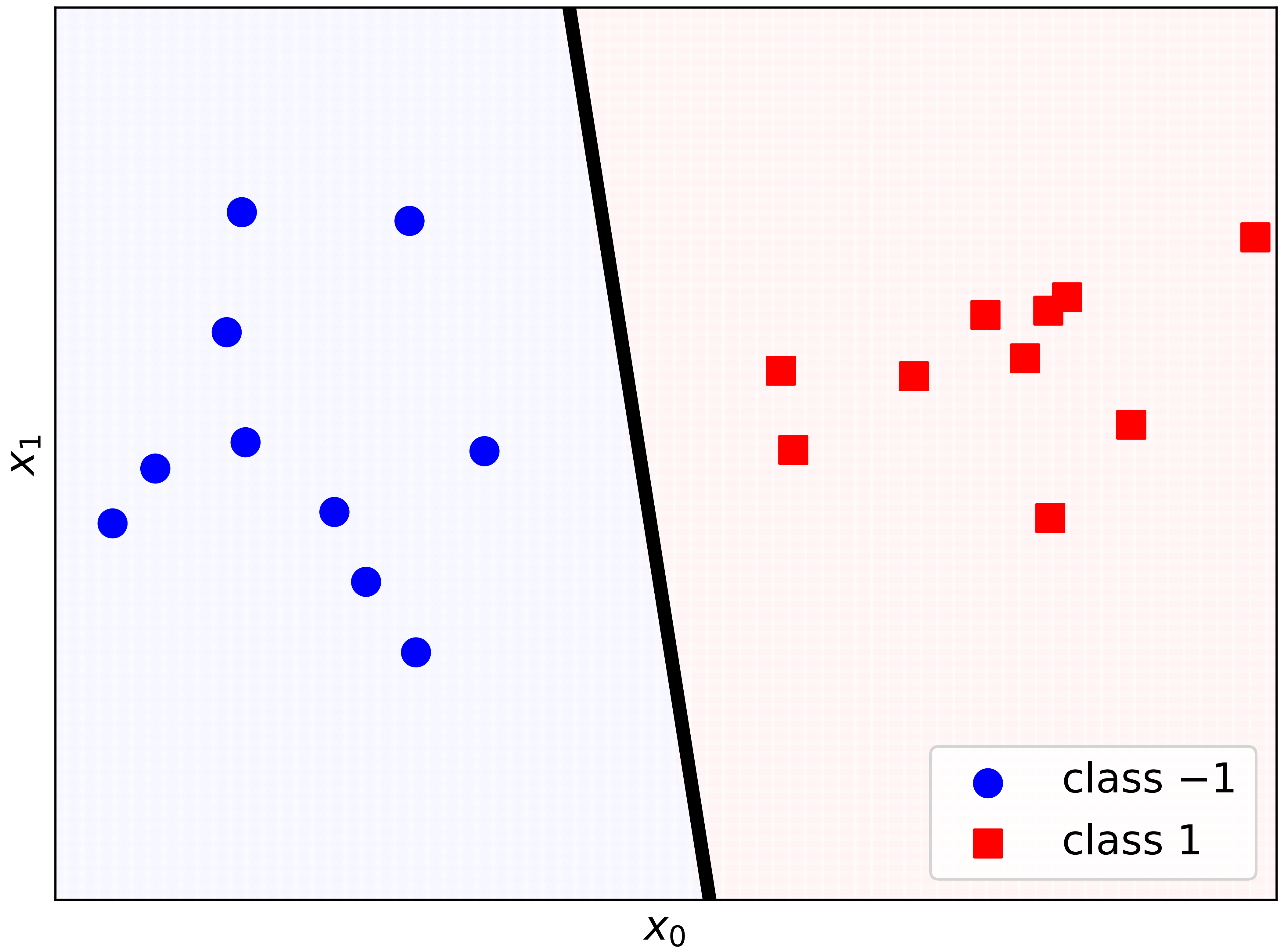



# Which classifier is better and why?

A

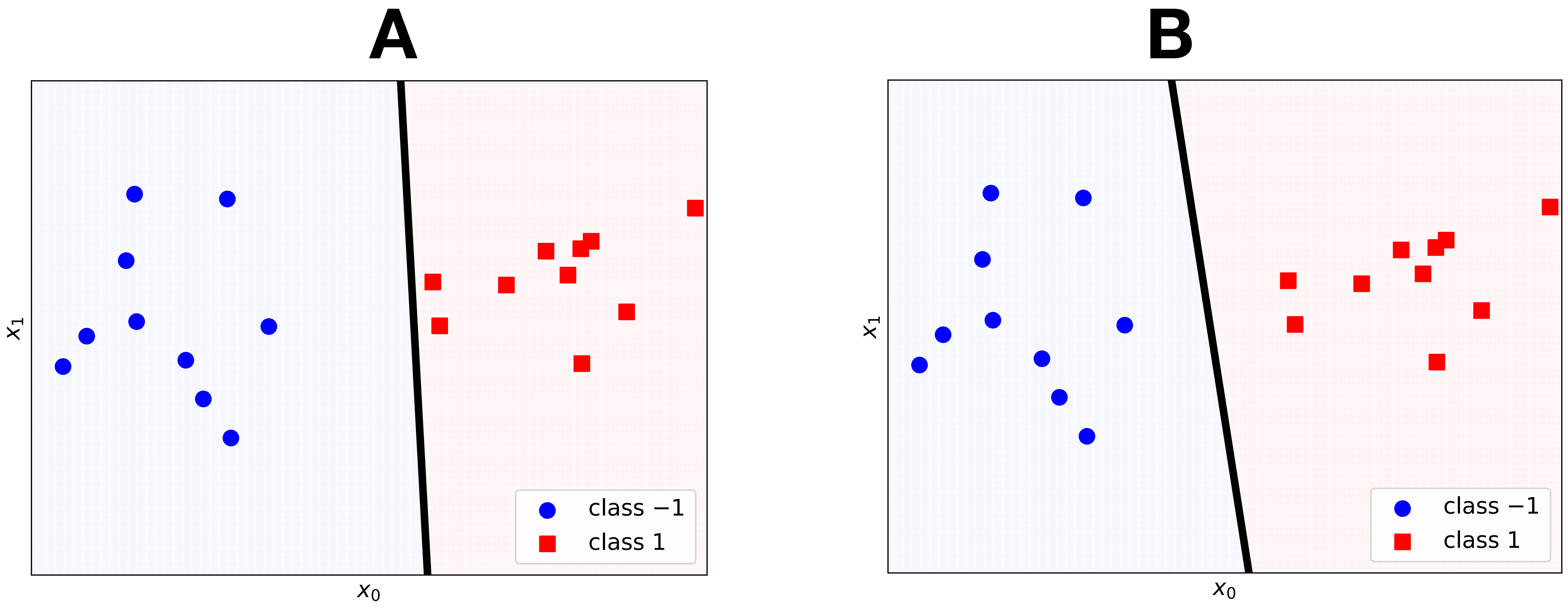



B



# Robustness

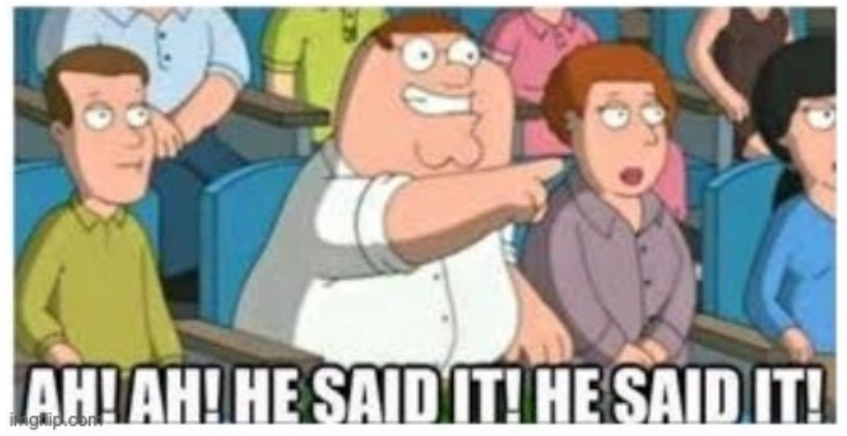
- The decision boundary of classifier A is very close to its nearest points
- The decision boundary of classifier B is far away from its nearest points
- Small perturbations shouldn't cause a point to be classified differently
- Classifier B should generalise to **new data** better



# Building in robustness

- **Assuming linearly separable data**, we want the decision boundary to be as far away as possible from the nearest training points

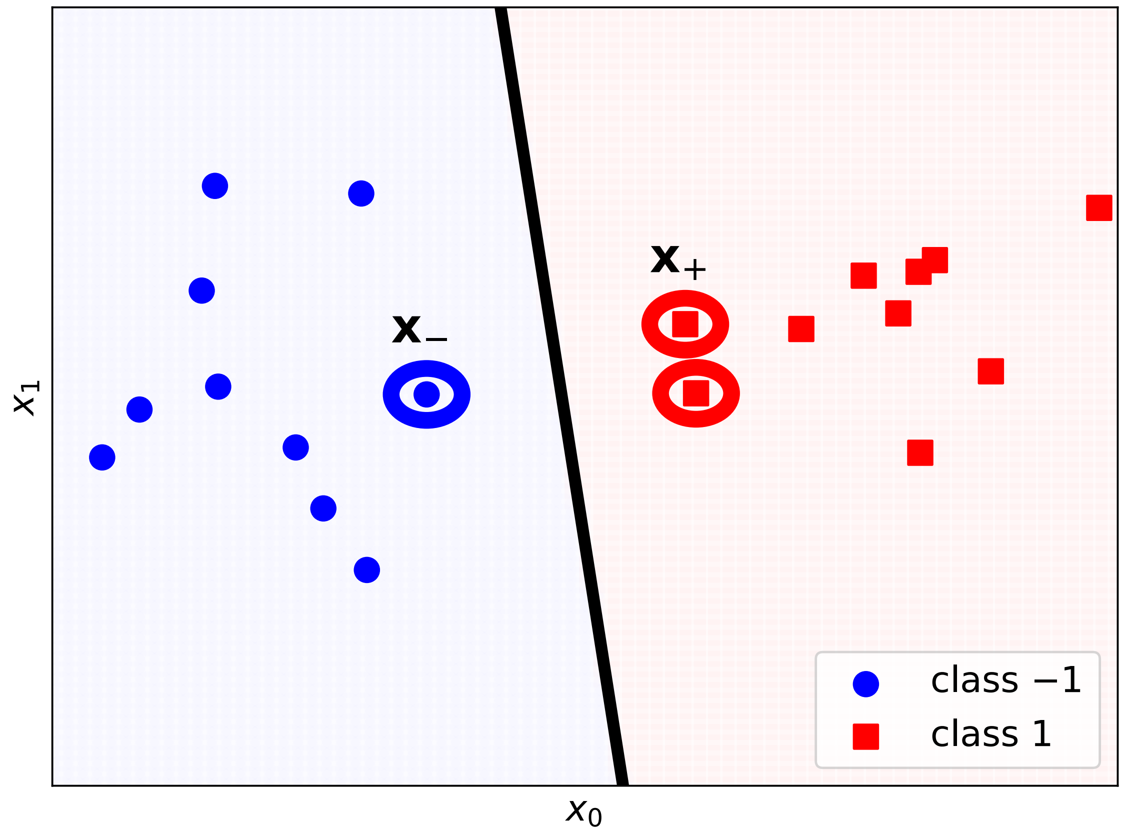
- This happens when it is equidistant from the nearest point(s) in class 1 $\mathbf{x}_+$ and the nearest point(s) in class -1 $\mathbf{x}_-$




- We will call $\mathbf{x}_+$ and $\mathbf{x}_-$ the support vectors

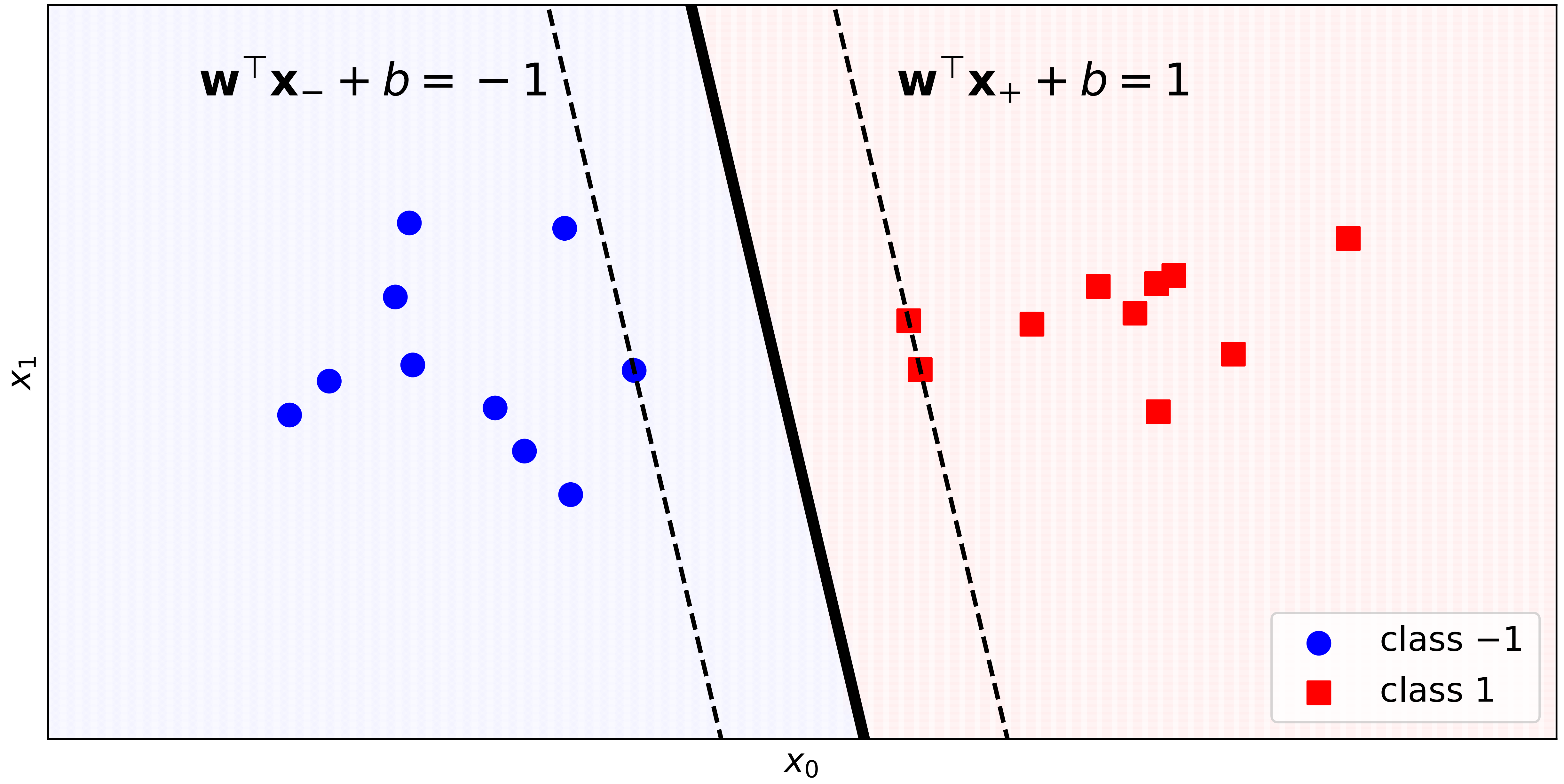
- The distance from the boundary to the support vectors should be the same

$$\frac{|\mathbf{w}^\top \mathbf{x}_+ + b|}{||\mathbf{w}||} = \frac{|\mathbf{w}^\top \mathbf{x}_- + b|}{||\mathbf{w}||}$$



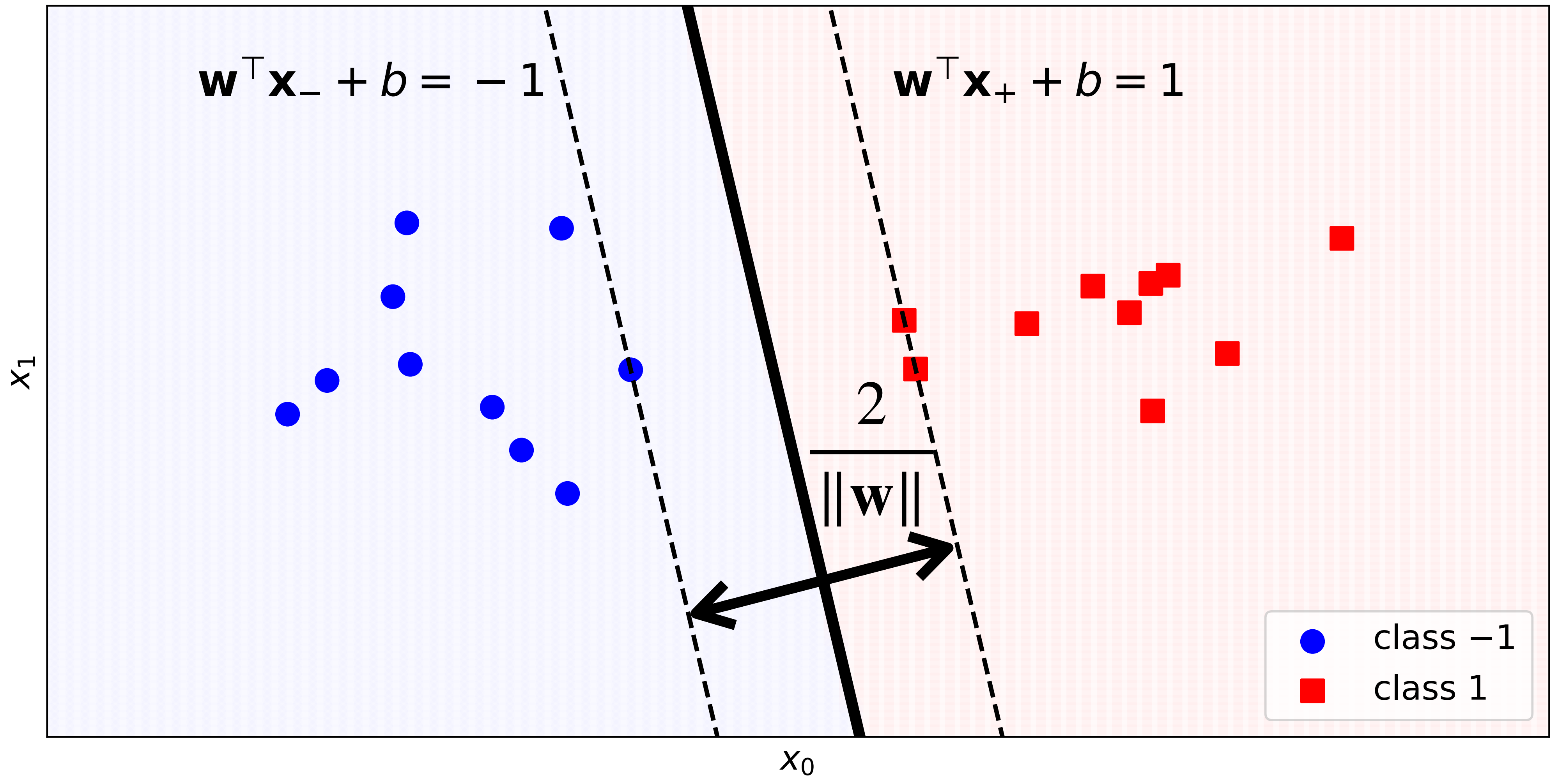
# Fixing scores

- We want $|\mathbf{w}^\top \mathbf{x}_+ + b| = |\mathbf{w}^\top \mathbf{x}_- + b|$
- The classifier scores for the support vectors should have the same magnitude
- We will choose 1 so we want $\mathbf{w}^\top \mathbf{x}_+ + b = 1$ and $\mathbf{w}^\top \mathbf{x}_- + b = -1$



# The margin

- We don't just want the decision boundary equidistant from $\mathbf{x}_+$ and $\mathbf{x}_-$
- We want the distance itself to be as large as possible
- This distance is given by $\dfrac{|\mathbf{w}^\top \mathbf{x}_+ + b|}{\|\mathbf{w}\|} = \dfrac{|\mathbf{w}^\top \mathbf{x}_- + b|}{\|\mathbf{w}\|} = \dfrac{1}{\|\mathbf{w}\|}$
- Twice this distance is the **margin** of the classifier



# Hard-margin SVM

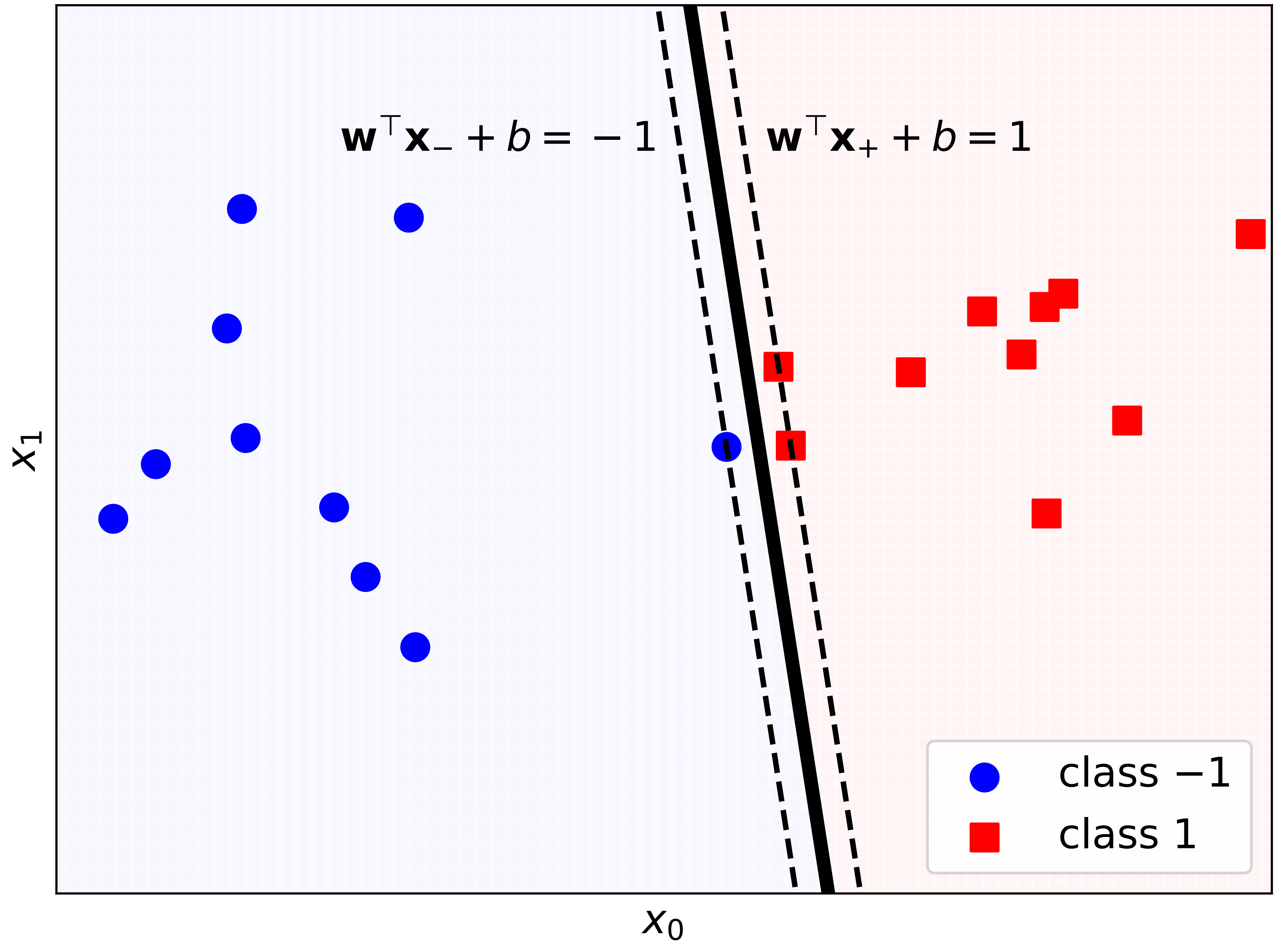
- We want to maximise the margin $\frac{2}{||\mathbf{w}||}$ which is the same as minimising $||\mathbf{w}||^2$
- If $\mathbf{w}^\top \mathbf{x}_+ + b = 1$ then we want $\mathbf{w}^\top \mathbf{x} + b > 1$ for other points in class 1
- If $\mathbf{w}^\top \mathbf{x}\_ + b = -1$ then we want $\mathbf{w}^\top \mathbf{x} + b < -1$ for other points in class -1
- Combining these, we can formulate a **constrained** optimisation problem

$$\underset{\mathbf{w},b}{\text{minimise}} ||\mathbf{w}||^2 \text{ subject to } y^{(n)}(\mathbf{w}^\top \mathbf{x}^{(n)} + b) \geq 1 \ \forall n$$

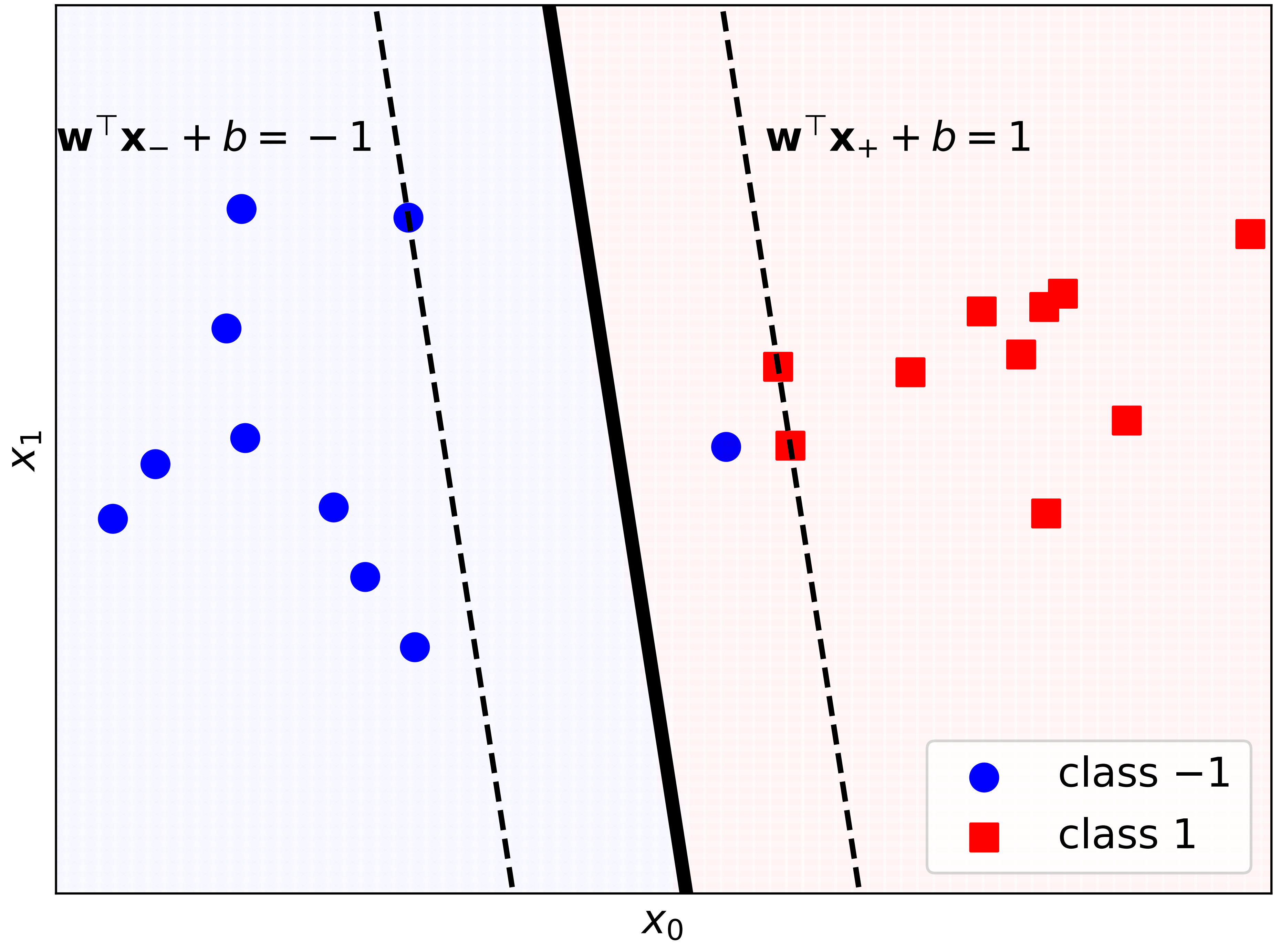
- Minimising a quadratic function subject to linear constraints can be solved using quadratic programming algorithms (which you don't need to know about for DAML4)

# Which classifier is better and why?

## A

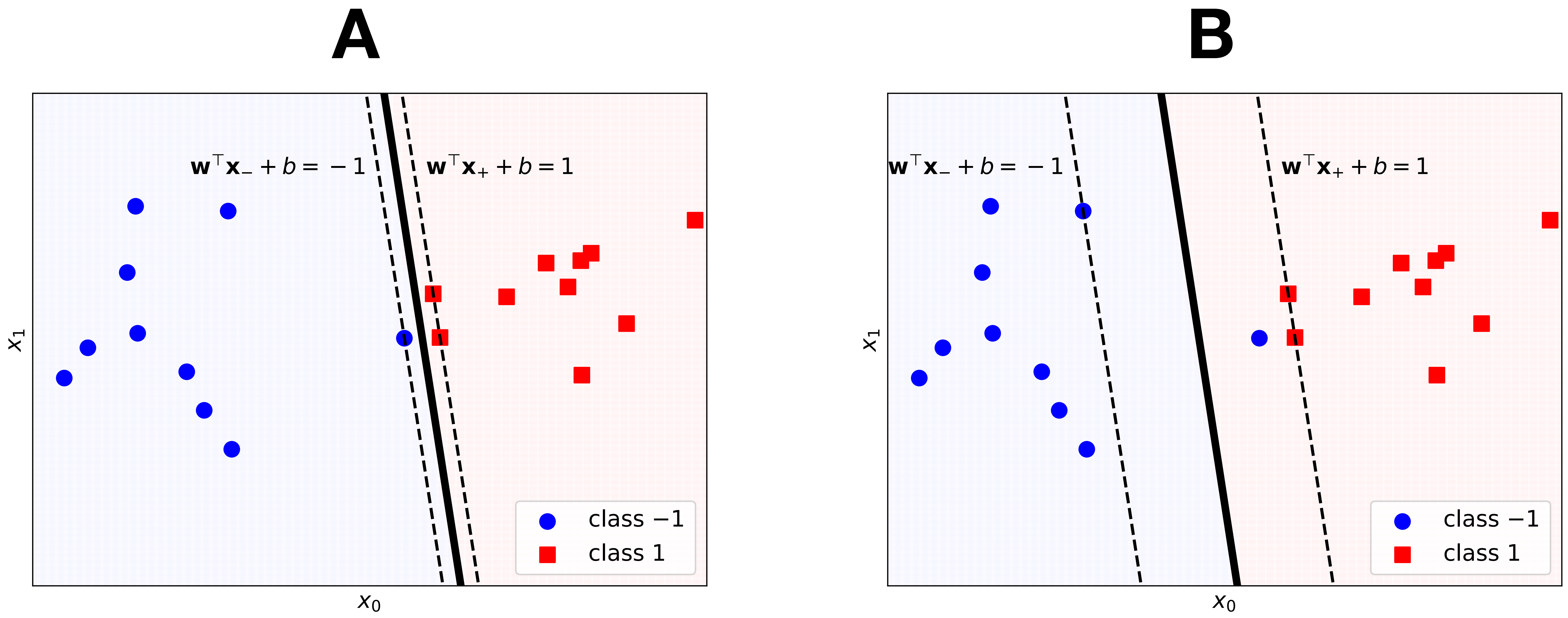



## B



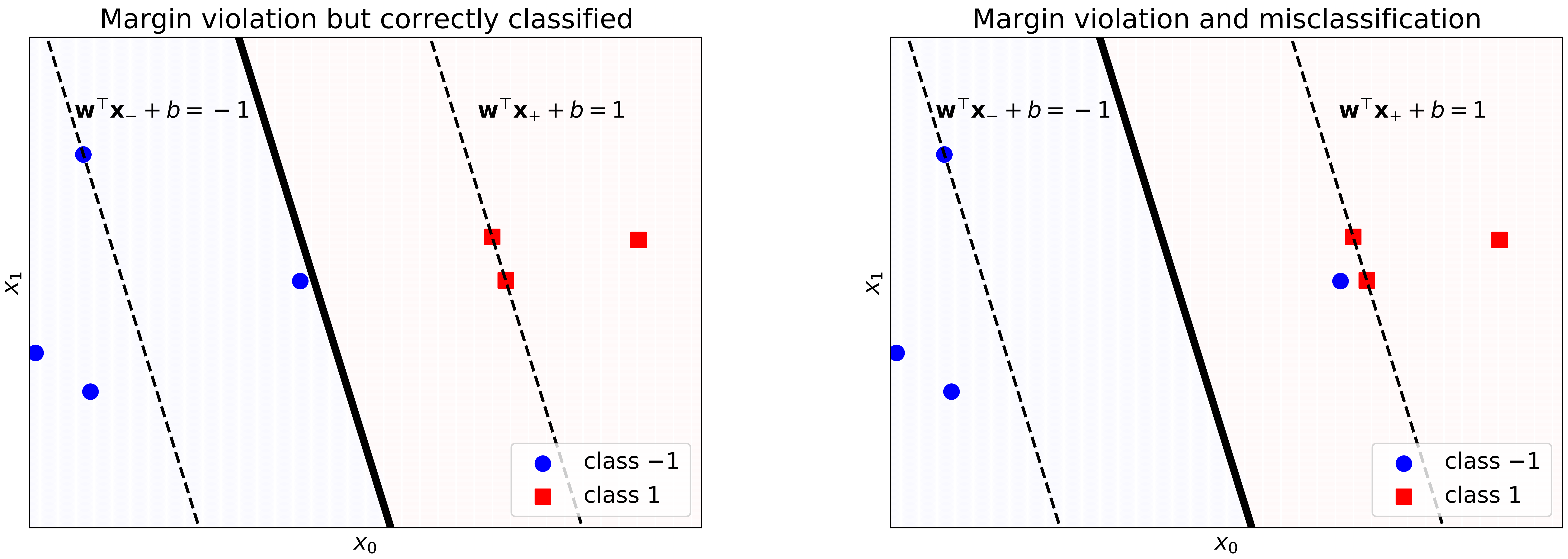
# A hard margin at what cost?

- Classifier A has a small margin
- Classifier B has a large margin but a single point is misclassified
- B likely generalises better but we can’t get it with a hard-margin SVM
- We should be able to tradeoff classifying points correctly against the margin size



# Allowing for margin violations

- For a hard margin SVM we have $\underset{\mathbf{w},b}{\text{minimise}}\ \|\mathbf{w}\|^2$ s.t. $y^{(n)}(\mathbf{w}^\top \mathbf{x}^{(n)} + b) \geq 1\, \forall n$
- This prevents points from being misclassified, or crossing into the margin
- Can we change our objective to facilitate this if it gives us a large margin?



# Soft-margin SVM

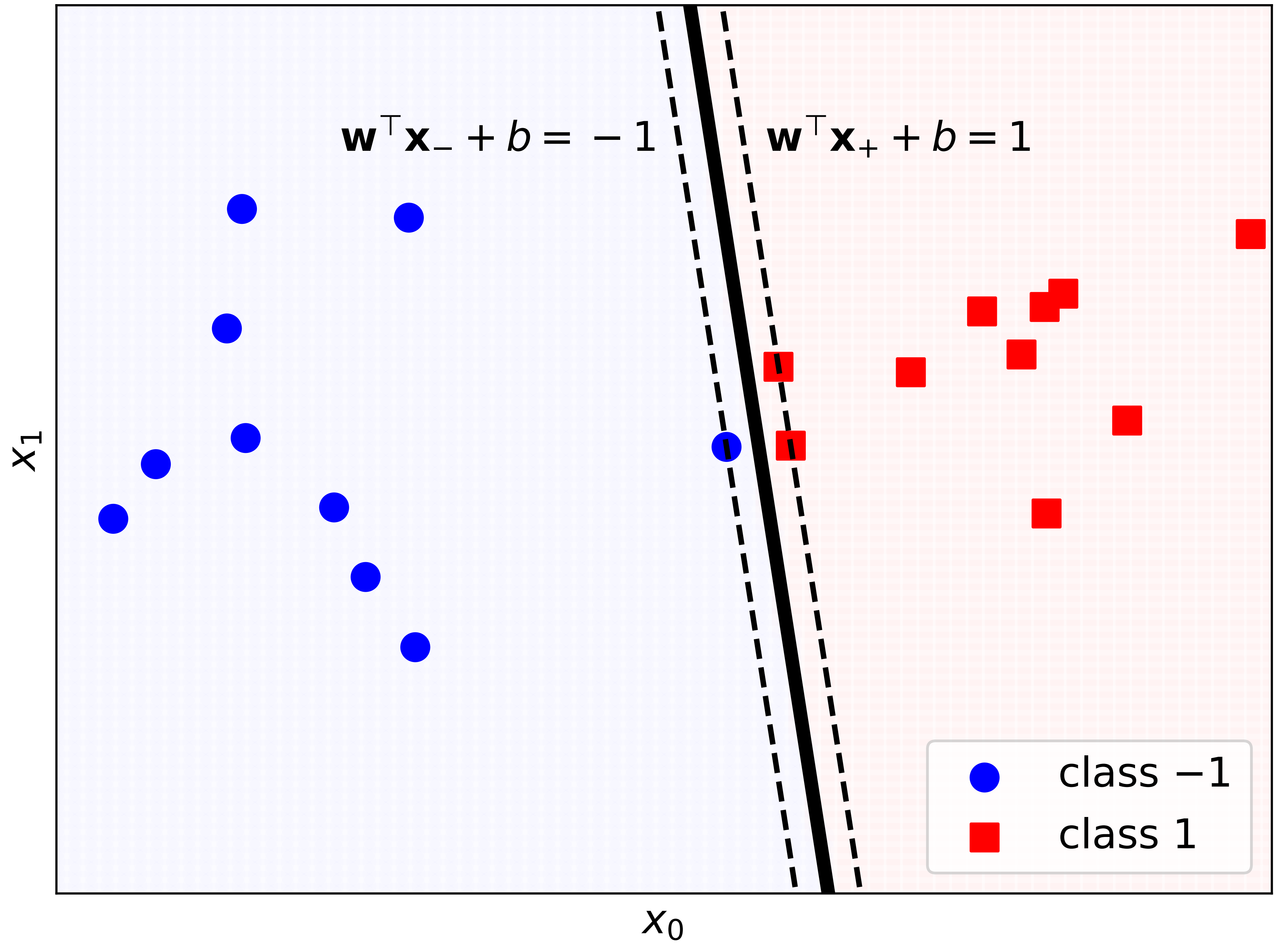
- Let's write a loss function that we intend to minimise consisting of two terms
- The first term should be small when the margin is big
- The second term should be small when there aren't many margin violations

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

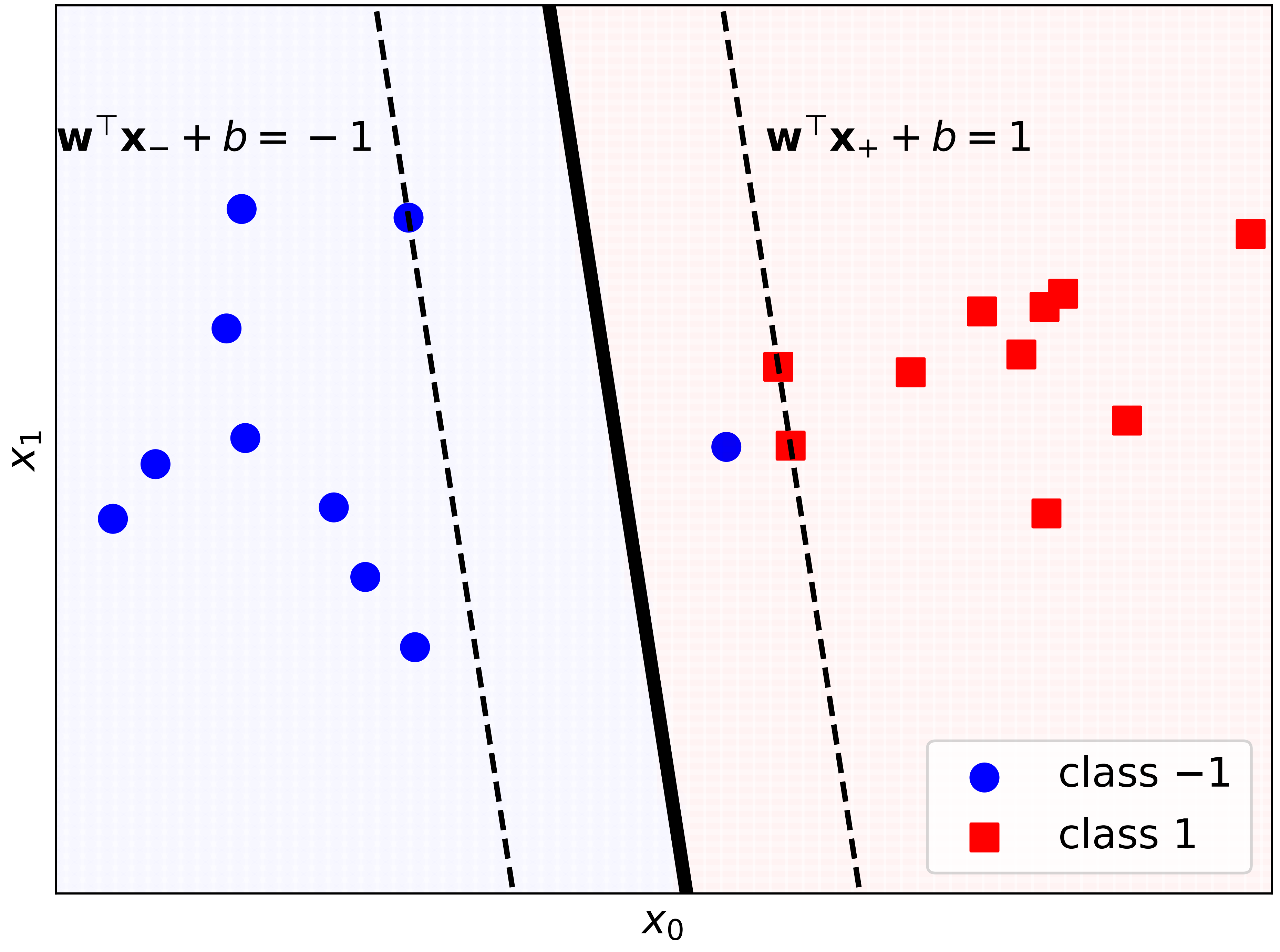
- $C$ is a hyperparameter that controls the penalty for margin violations
- $C = 0$ means there is no penalty and $C \rightarrow \infty$ is the hard-margin SVM
- We can now trade a large margin for some misclassifications

# Varying C

$C = 1000000$



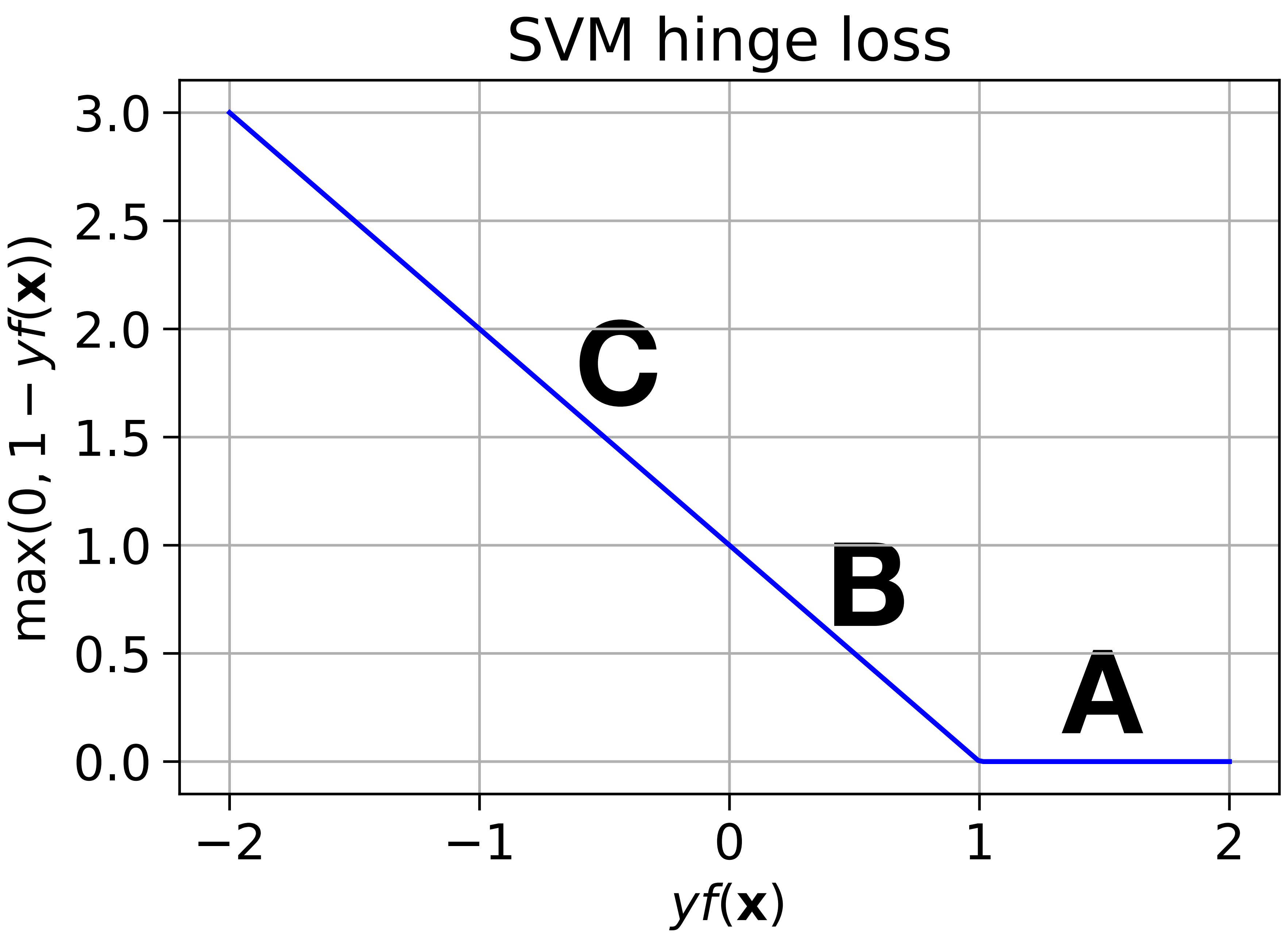

$C = 2$

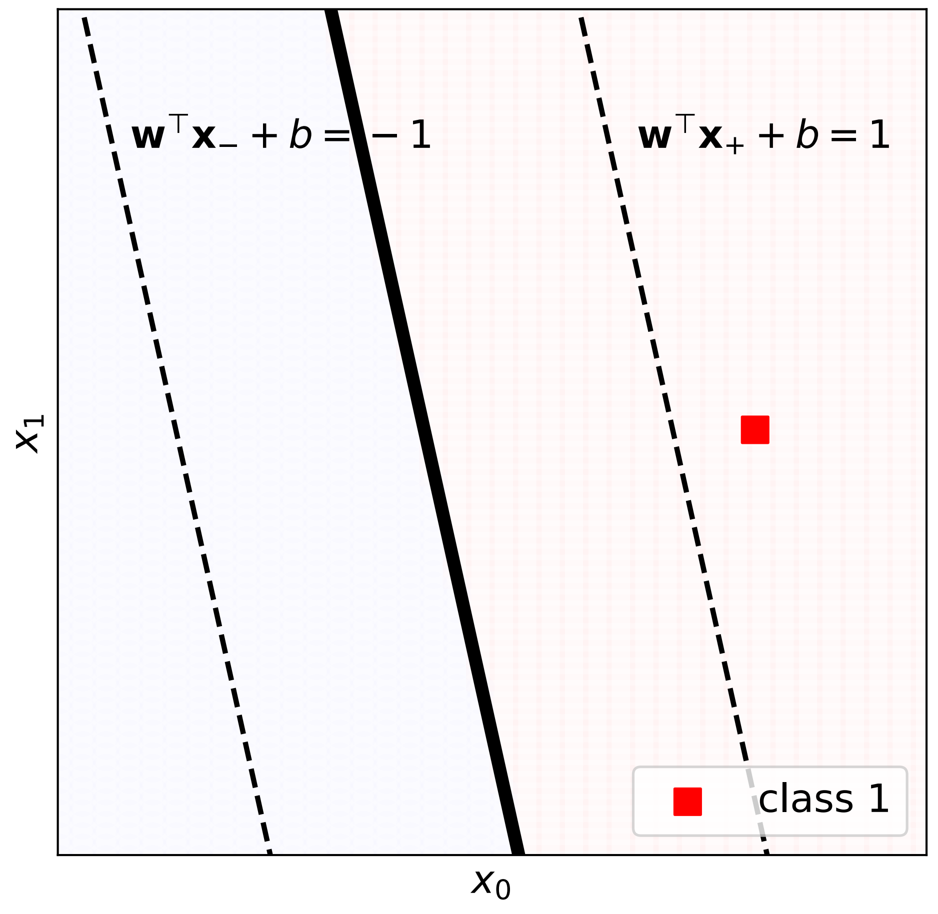



We get a large margin here at the expense of a single violation

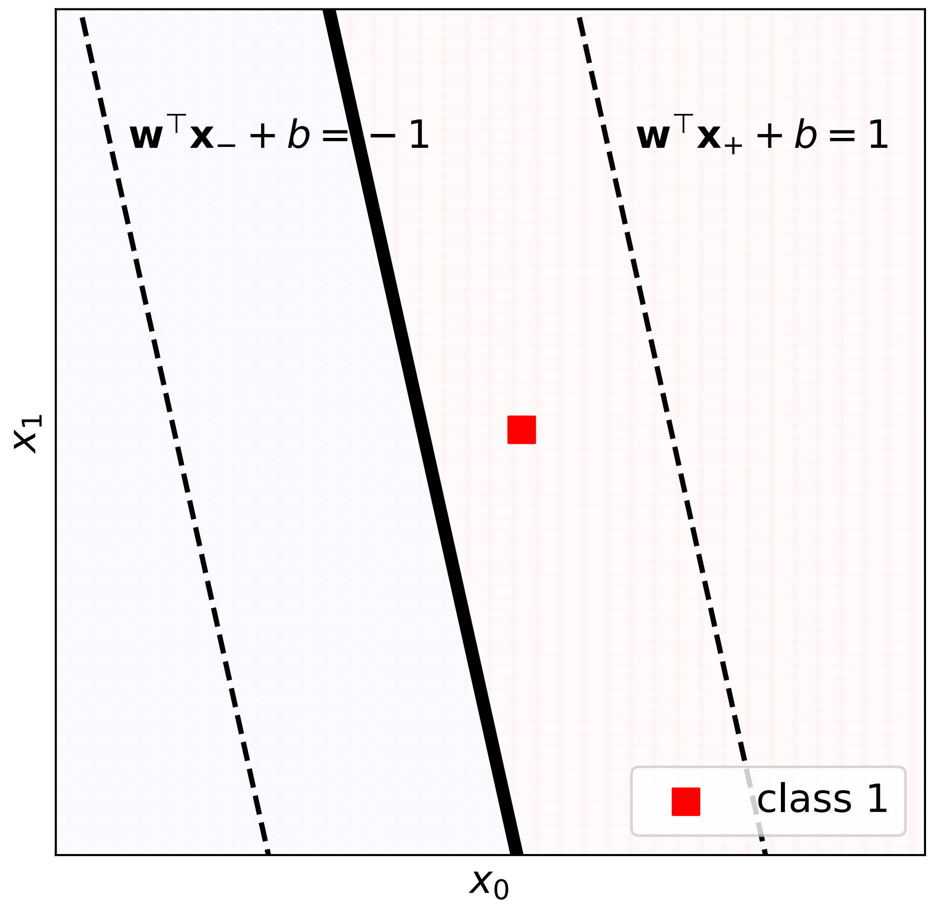
# Return of the hinge loss

- We last saw the hinge loss for the perceptron as $\max\left(0, -yf(\mathbf{x})\right)$
- For SVMs it is a bit different: $\max\left(0, 1 - yf(\mathbf{x})\right)$

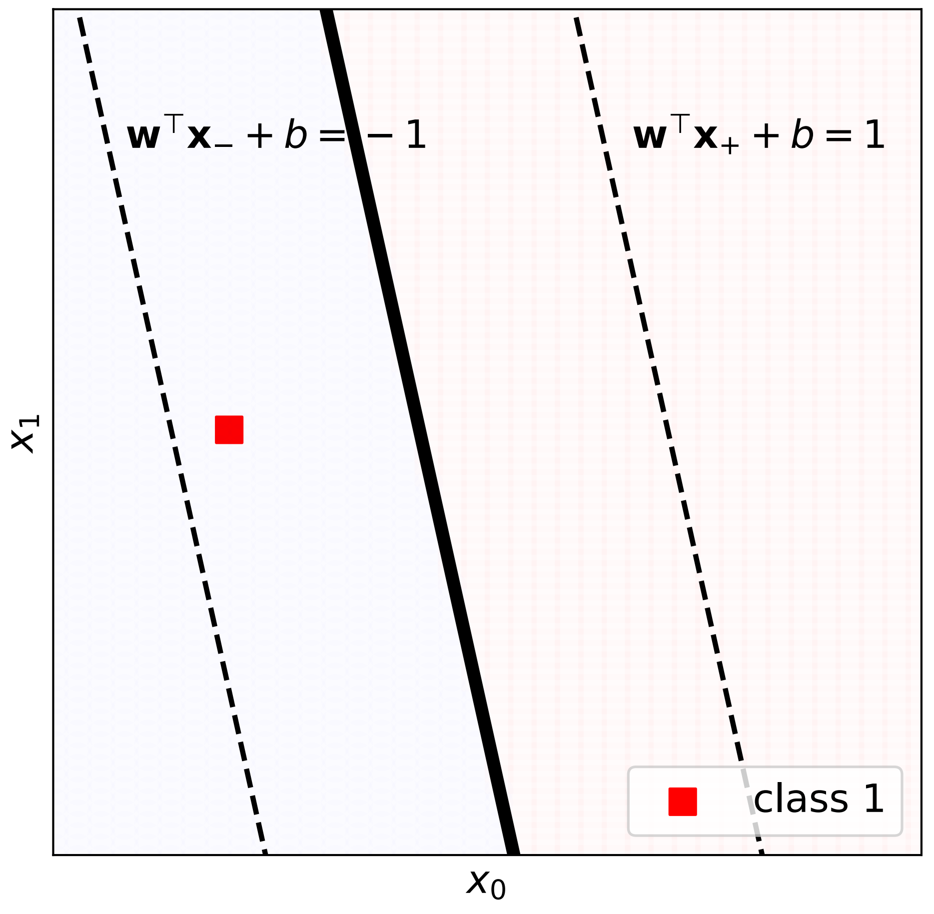



A




B



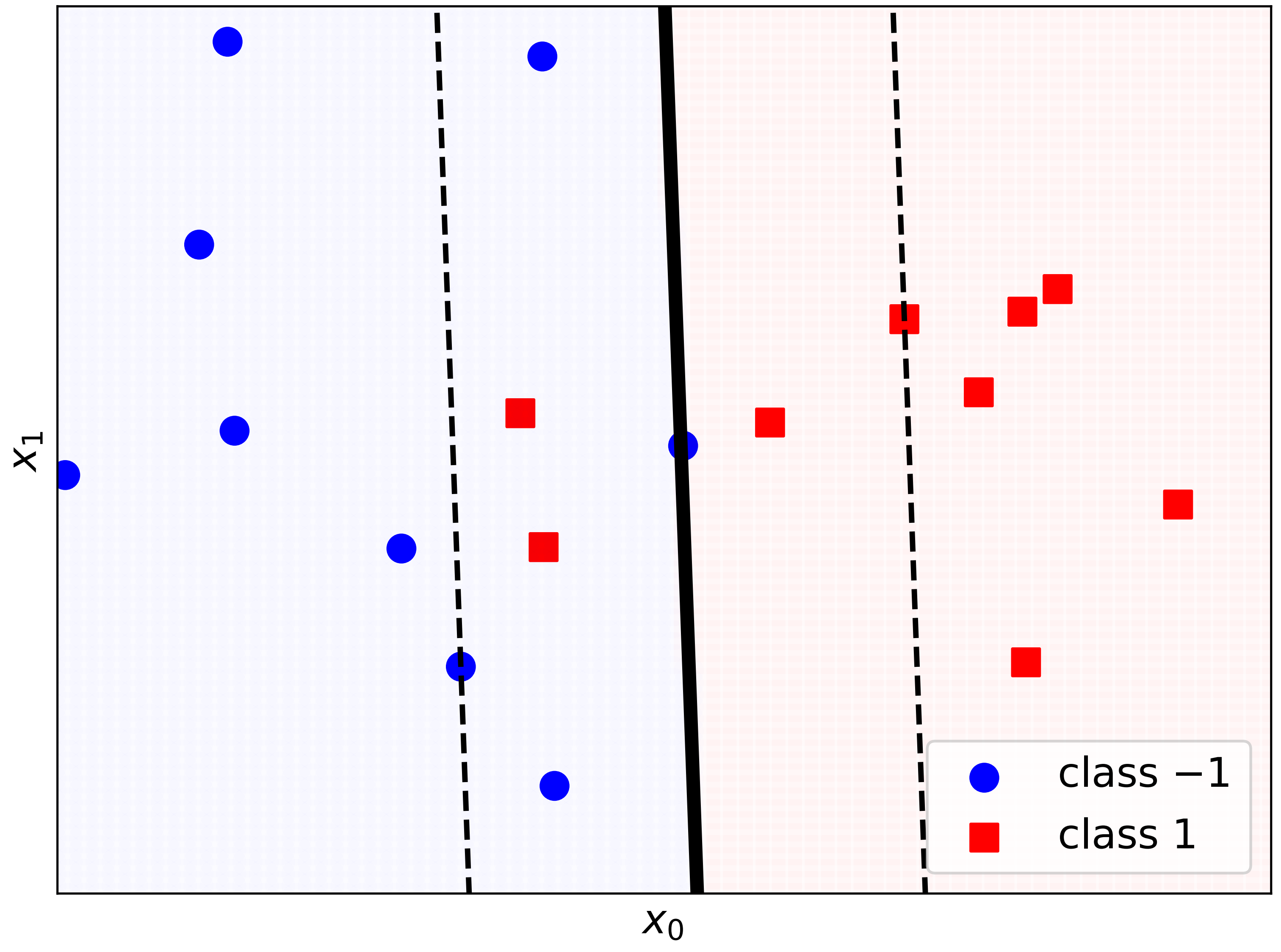

C

# Optimisation for SVMs

- We have a linear classifier $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$
- The soft-margin SVM loss is $L_{SVM} = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_n \max\left(0, 1 - y^{(n)} f(\mathbf{x}^{(n)})\right)$
- We want to solve $\underset{\mathbf{w},b}{\text{minimise}}\ L_{SVM}(\mathbf{w}, b)$
- $L_{SVM}$ is convex and the hinge loss is piecewise differentiable
- We can solve using stochastic gradient descent (SGD)

# Non-linearly separable data

- A soft-margin SVM can learn from non-linearly separable training data
- Margin violations are inevitable in this case
- Whenever there are margin violations the support vectors are defined as the points **on** and **in** the margin (even if the data is linearly separable)




Make sure you're happy that this classifier has 8 support vectors

# Multi-class SVMs

- The dominant approach is to train a binary SVM for each class in a one-versus-rest manner
- You will examine this in the lab

# The dual form of an SVM

We are no longer assuming linear separability

- To make a linear classifier $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ an SVM we solve

$$\underset{\mathbf{w},b}{\text{minimise}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_n \max\left(0, 1 - y^{(n)} f(\mathbf{x}^{(n)})\right)$$

- This is the primal problem. There is an equivalent dual problem

- For the dual we use the representer theorem to rewrite $f(\mathbf{x}) = \sum_n \alpha_n y^{(n)} \mathbf{x}^{(n)\top}\mathbf{x} + b$ and solve

$$\underset{\alpha_0,\ldots,\alpha_{N-1}}{\text{minimise}} \ \frac{1}{2}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1} \alpha_j \alpha_k y^{(j)} y^{(k)} (\mathbf{x}^{(j)\top}\mathbf{x}^{(k)}) - \sum_n \alpha_n \text{ subject to } 0 \le \alpha_n \le C \ \forall n \text{ and } \sum_n \alpha_n y^{(n)} = 0$$

**This objective is given without proof and you are not required to understand it for this course.**
**It can be solved using quadratic programming and $b$ can then be calculated using the data and $\alpha$ values**

https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation

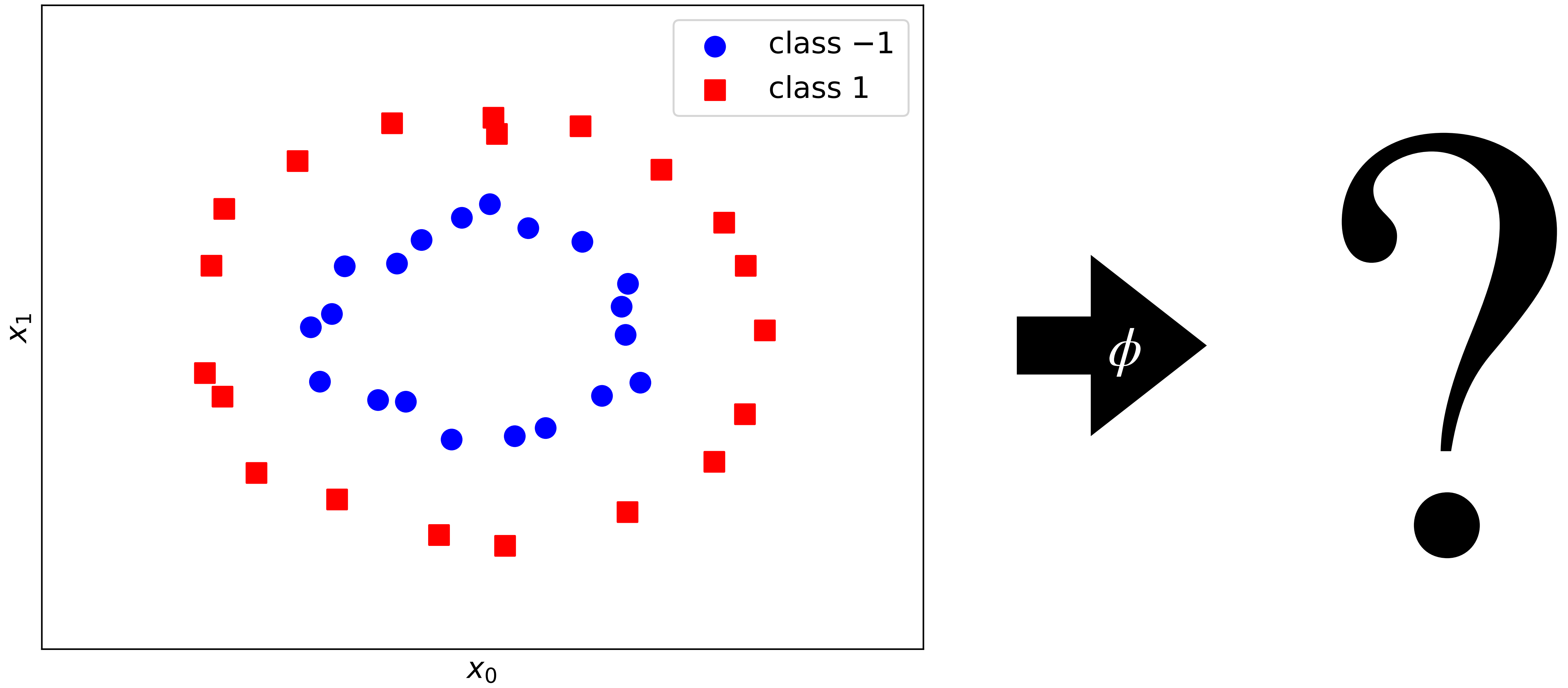
# SVM primal and dual forms

- We have primal form $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ and dual $f(\mathbf{x}) = \sum_n \alpha_n y^{(n)} \mathbf{x}^{(n)\top} \mathbf{x} + b$
- $\mathbf{w} \in \mathbb{R}^D$ can be constructed from the $\alpha$s using $\mathbf{w} = \sum_n \alpha_n y^{(n)} \mathbf{x}^{(n)}$
- When $D \gg N$ its more efficient to solve for the vector of $\alpha$s: $\boldsymbol{\alpha} \in \mathbb{R}^N$
- It then looks like we have to retain lots of data points but $\boldsymbol{\alpha}$ is very sparse
- **Its elements are only non-zero for training points that are support vectors**

# Kernels

# Feature maps for linear separability

- We have been using $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ but we could use $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$
- $\phi$ maps $\mathbf{x} \in \mathbb{R}^D$ to a feature vector $\phi(\mathbf{x}) \in \mathbb{R}^Z$ that lives in feature space
- The issue of linearly inseparability keeps cropping up
- Let's deal with this by using a $\phi$ that makes data separable in feature space

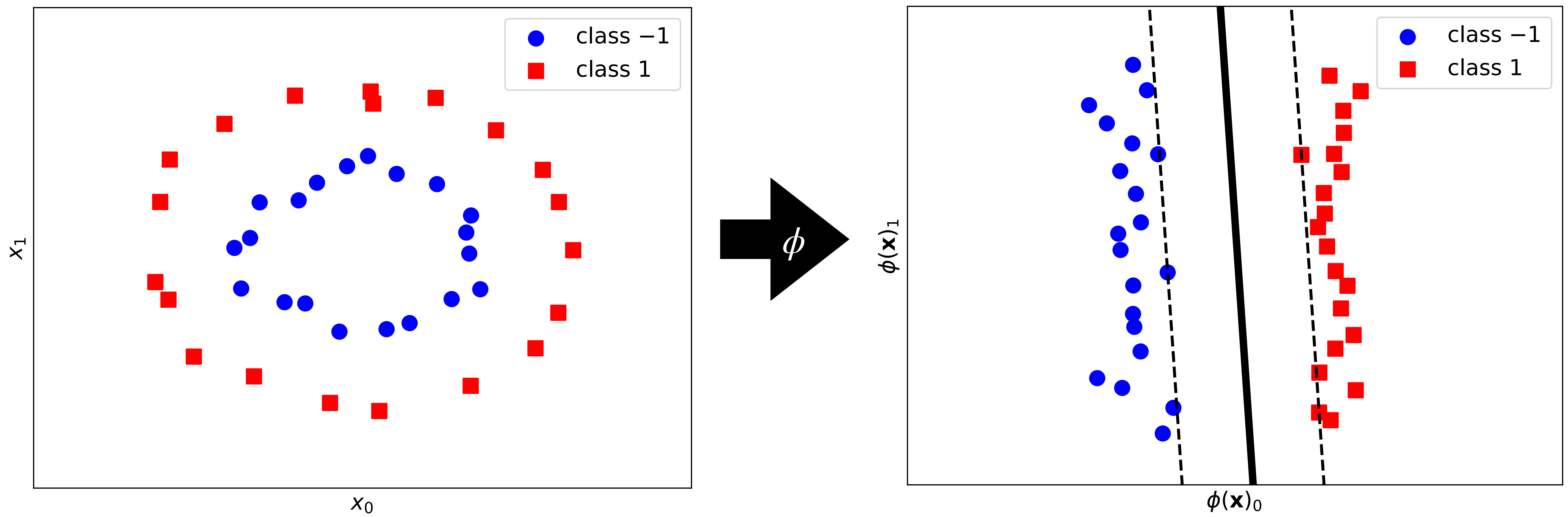


# Features as polar coordinates

- In this contrived example, data from each classes lies on a circle (with noise)
- Let's use a $\phi$ that maps to polar coordinates to separate these
- We can then learn the weights for $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$ e.g. with an SVM loss

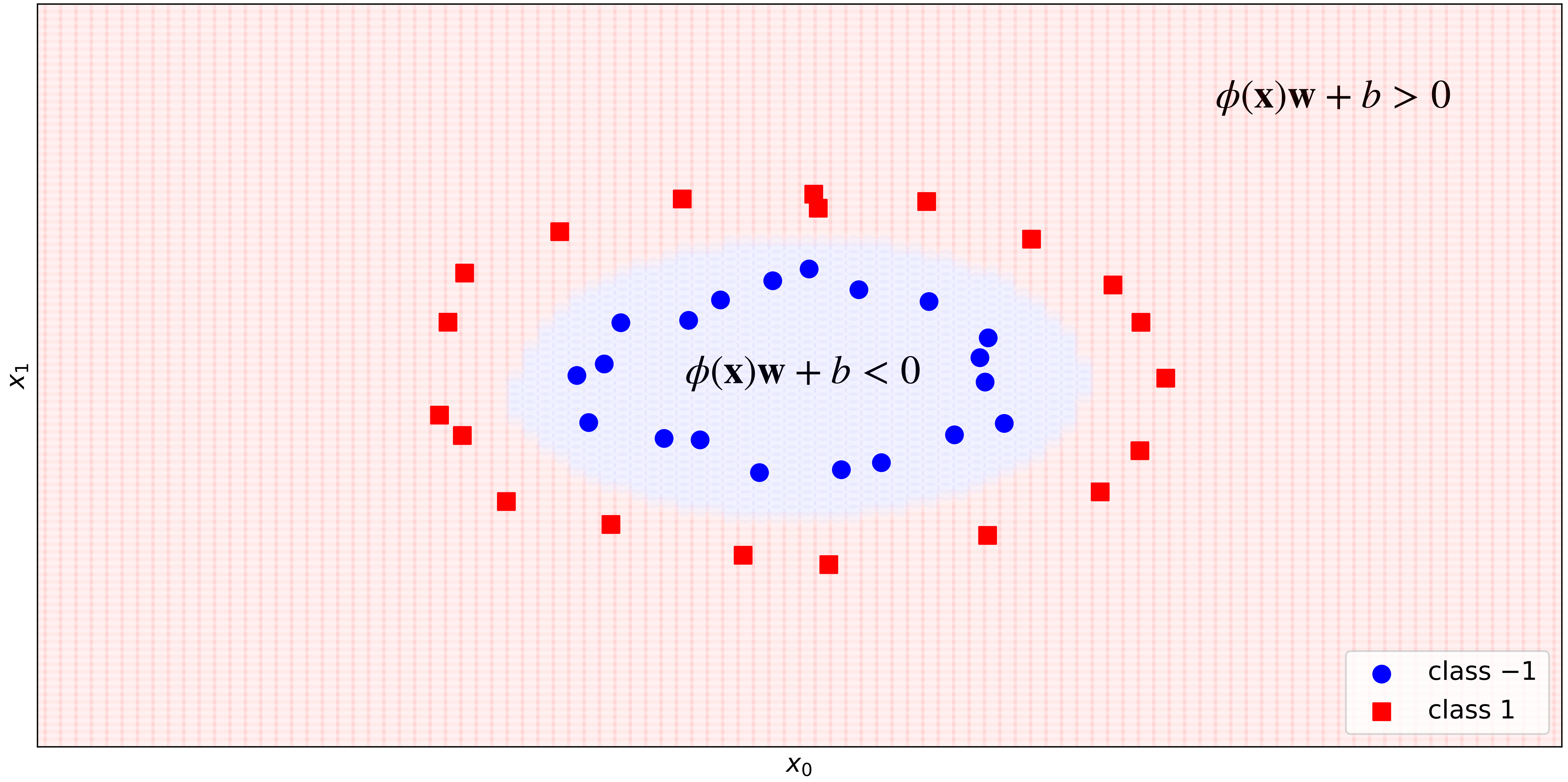
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

$$\phi(\mathbf{x}) = \begin{bmatrix} \|\mathbf{x}\| \\ \tan^{-1} \dfrac{x_1}{x_0} \end{bmatrix}^\top$$



# Non-linear decision boundary

- We can see how dummy points in the original space will be classified
- We can see our linear classifier in feature space has given us a non-linear decision boundary in the original space

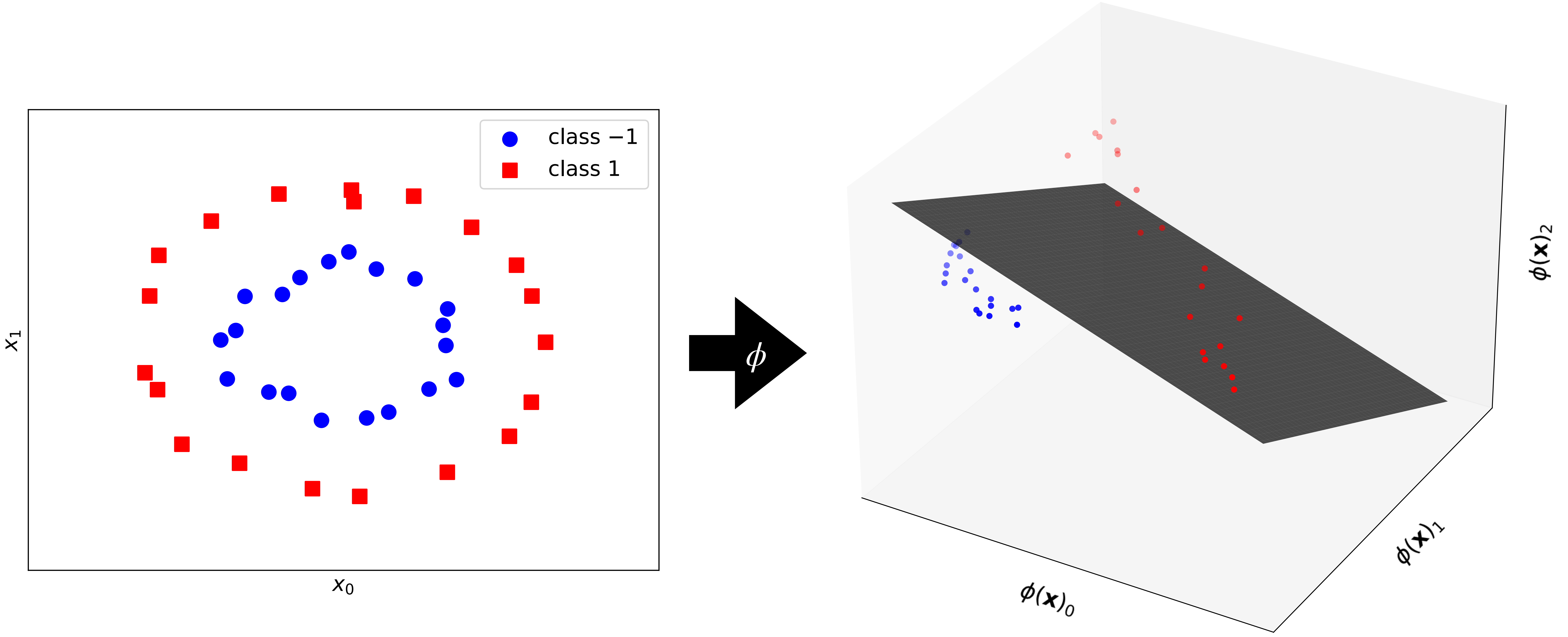


# Mapping to higher dimensions

- $\phi$ maps $\mathbf{x} \in \mathbb{R}^D$ to a feature vector $\phi(\mathbf{x}) \in \mathbb{R}^Z$ that lives in feature space
- Data that isn't linearly separable in $D$ dimension can be in higher dimensions

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

$$\phi(\mathbf{x}) = \begin{bmatrix} {x_0}^2 \\ {x_1}^2 \\ x_0 x_1 \end{bmatrix}$$



# Dot products of features

- Consider the primal form of an SVM linear classifier $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$
- $\phi$ maps $\mathbf{x} \in \mathbb{R}^D$ to a feature vector $\phi(\mathbf{x}) \in \mathbb{R}^Z$ that lives in feature space
- This classifier has $Z + 1$ parameters so is expensive to train for large $Z$
- Can we solve the dual to learn $N$ parameters instead?
- The equivalent dual form of the classifier is $f(\mathbf{x}) = \sum_n \alpha_n y^{(n)} \phi(\mathbf{x}^{(n)})^\top \phi(\mathbf{x}) + b$
- We just substitute $\mathbf{x}$ for $\phi(\mathbf{x})$ in the dual problem formulation

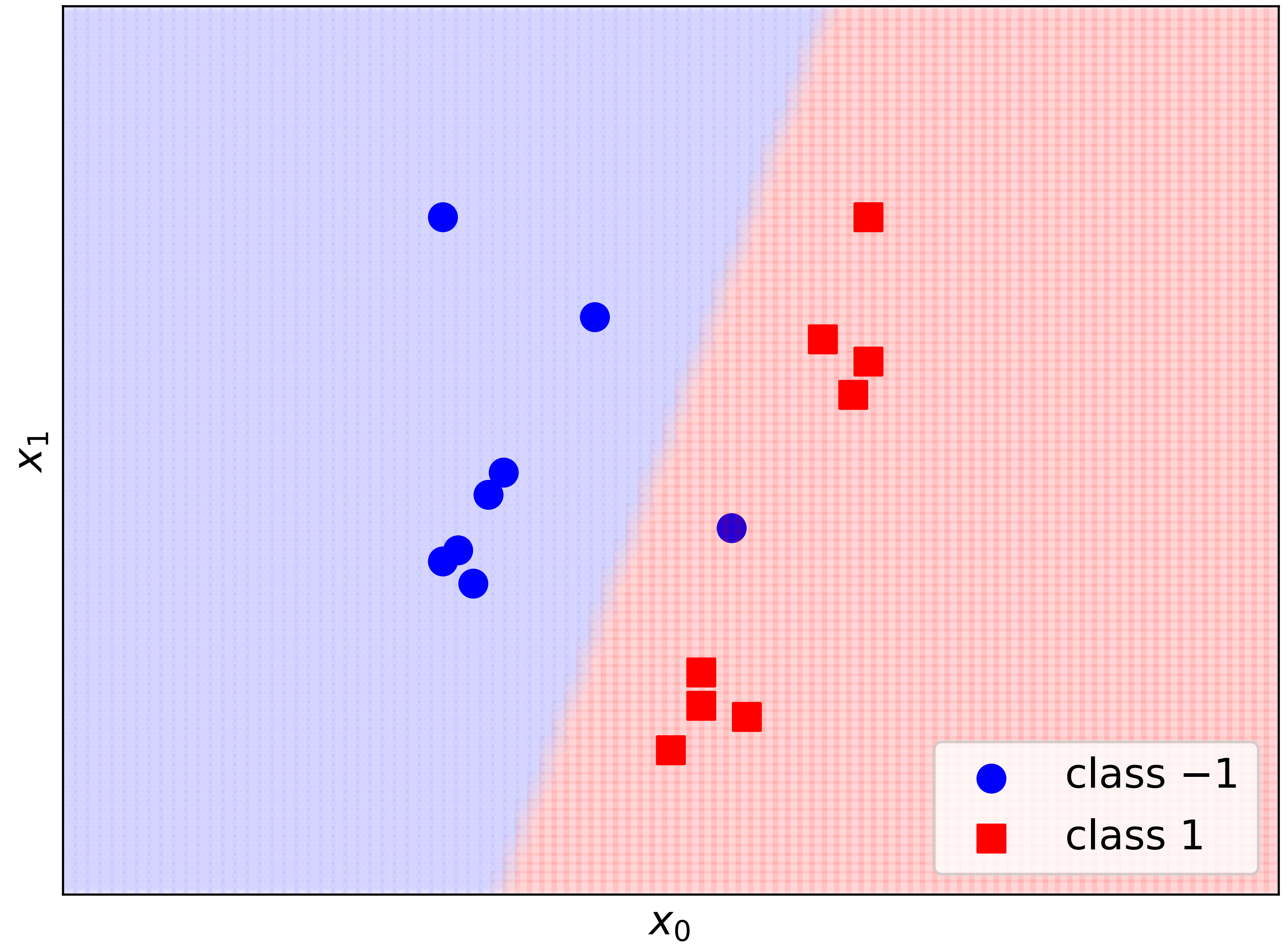
# The kernel trick

- In the dual $\phi$ only appears in dot products: $\phi(\mathbf{x}^{(j)})^\top \phi(\mathbf{x}^{(k)})$
- Consider for some $\phi$ a function $k(\mathbf{x}^{(j)}, \mathbf{x}^{(k)}) = \phi(\mathbf{x}^{(j)})^\top \phi(\mathbf{x}^{(k)})$
- This let's us compute this dot product without actually computing features
- The classifier becomes $f(\mathbf{x}) = \sum_n \alpha_n y^{(n)} k(\mathbf{x}^{(n)}, \mathbf{x}) + b$
- If we know the kernel $k(\mathbf{x}^{(j)}, \mathbf{x}^{(k)})$ for $\phi$ then we can project data to high dimensions implicitly

# Kernel SVM

Linear kernel

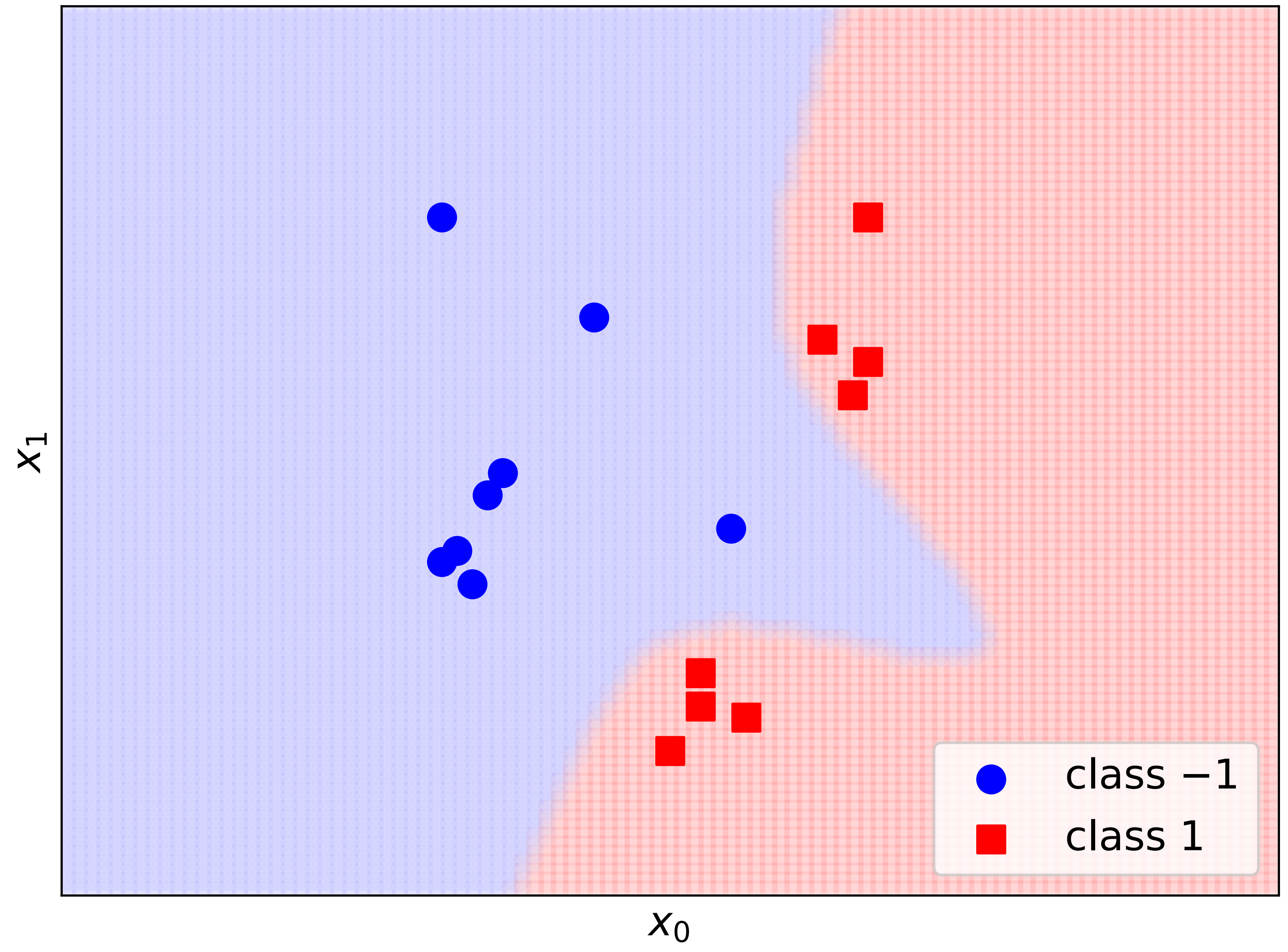
$$k(\mathbf{x}^{(j)}, \mathbf{x}^{(k)}) = \mathbf{x}^{(j)\top}\mathbf{x}^{(k)}$$




$\phi(\mathbf{x}) = \mathbf{x}$. This is what we have been using
Most of the time

Polynomial kernel

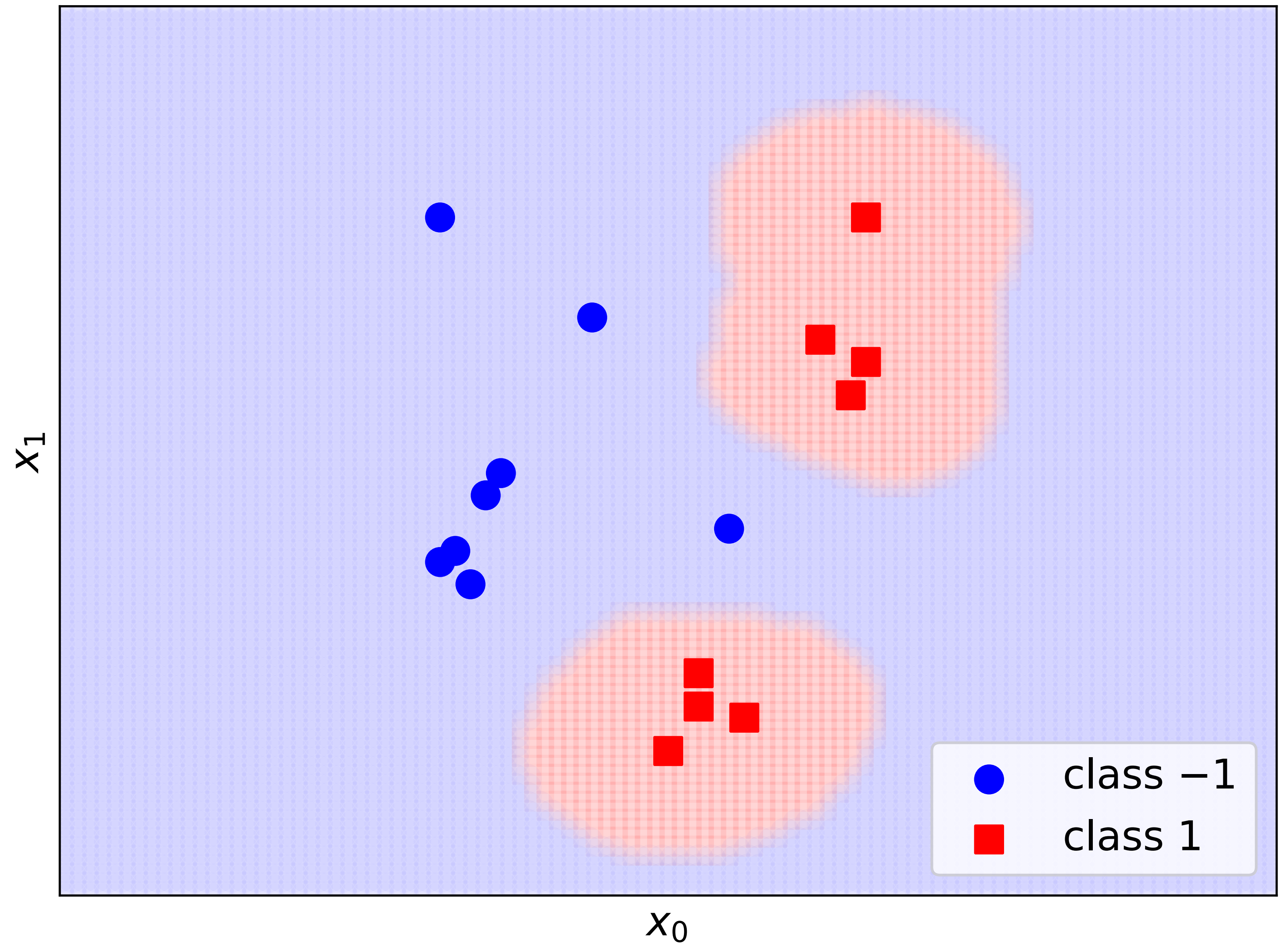
$$k(\mathbf{x}^{(j)}, \mathbf{x}^{(k)}) = (\gamma\mathbf{x}^{(j)\top}\mathbf{x}^{(k)} + r)^d$$




$\phi(\mathbf{x})$ contains polynomial terms up to the
$d^{th}$ degree

RBF kernel

$$k(\mathbf{x}^{(j)}, \mathbf{x}^{(k)}) = e^{(-\gamma\|\mathbf{x}^{(j)}-\mathbf{x}^{(k)}\|^2)}$$




$\phi(\mathbf{x})$ for this kernel is in infinite dimensions
Don't think about this too much :D

Dataset credit: https://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html

# A note on kernels

- Kernels are often associated with SVMs but are not bound to that framework
- They feature prominently in Gaussian processes (not covered in DAML4)
- Several algorithms we have covered thus far can be combined with kernels to form e.g. kernel PCA, kernel ridge regression

# Classifier selection and evaluation

# No free lunch

- You now know about perceptrons, logistic regression, and SVMs
- Perceptrons are terrible, so you can forget about using those in practice
- But should you use an SVM or logistic regression?
- If you use an SVM, which kernel do you pick?
- The answer to both of the above questions are **it depends on the problem**
- **There is no universal best model! There is no free lunch!**

# Model selection

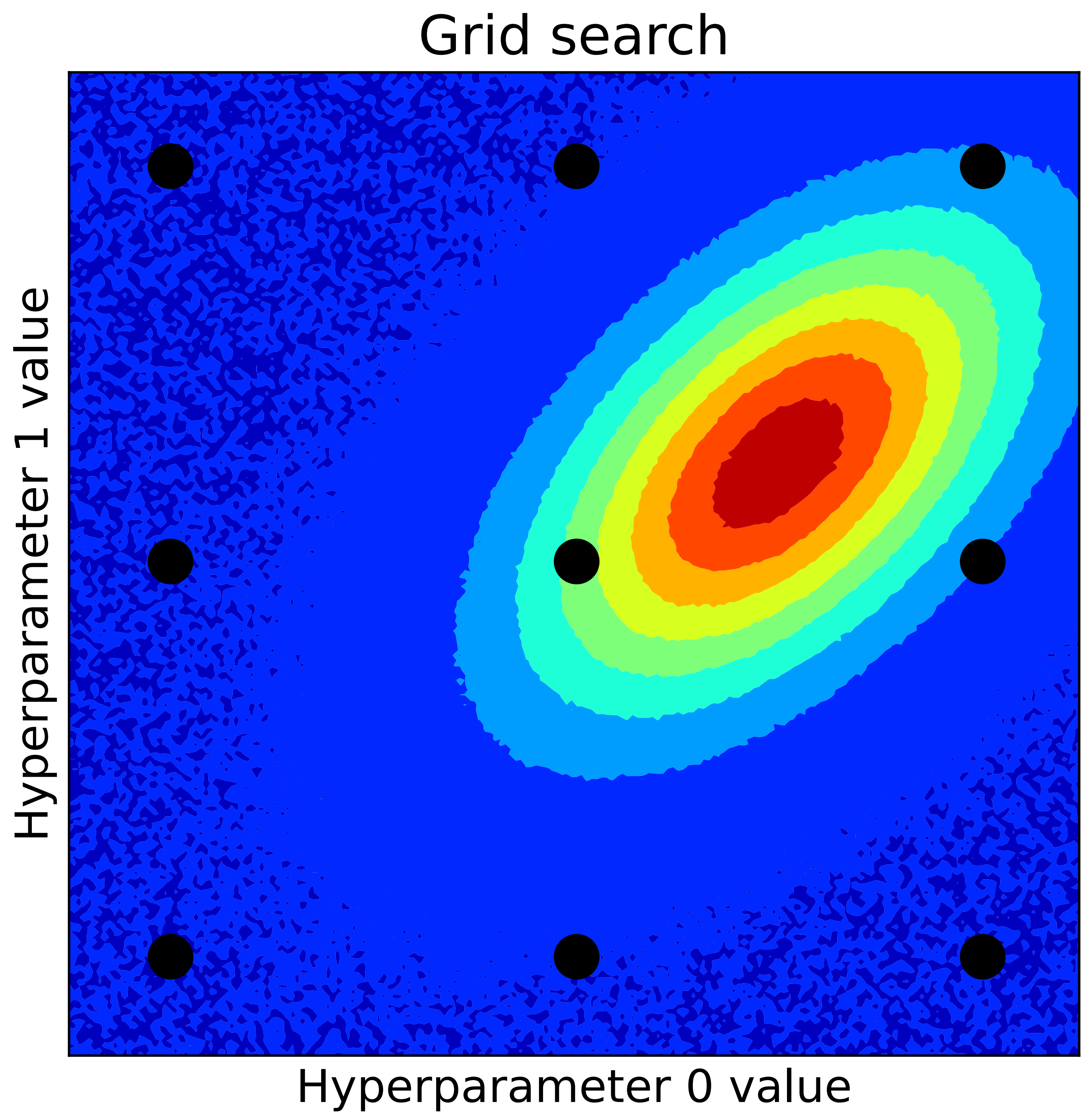
- Choosing between SVMs and logistic regression is a model selection problem
- Choosing which kernel to use is a model selection problem
- Use validation (or cross-validation) performance for model selection
- You can view e.g. kernel type as another hyperparameter to be tuned

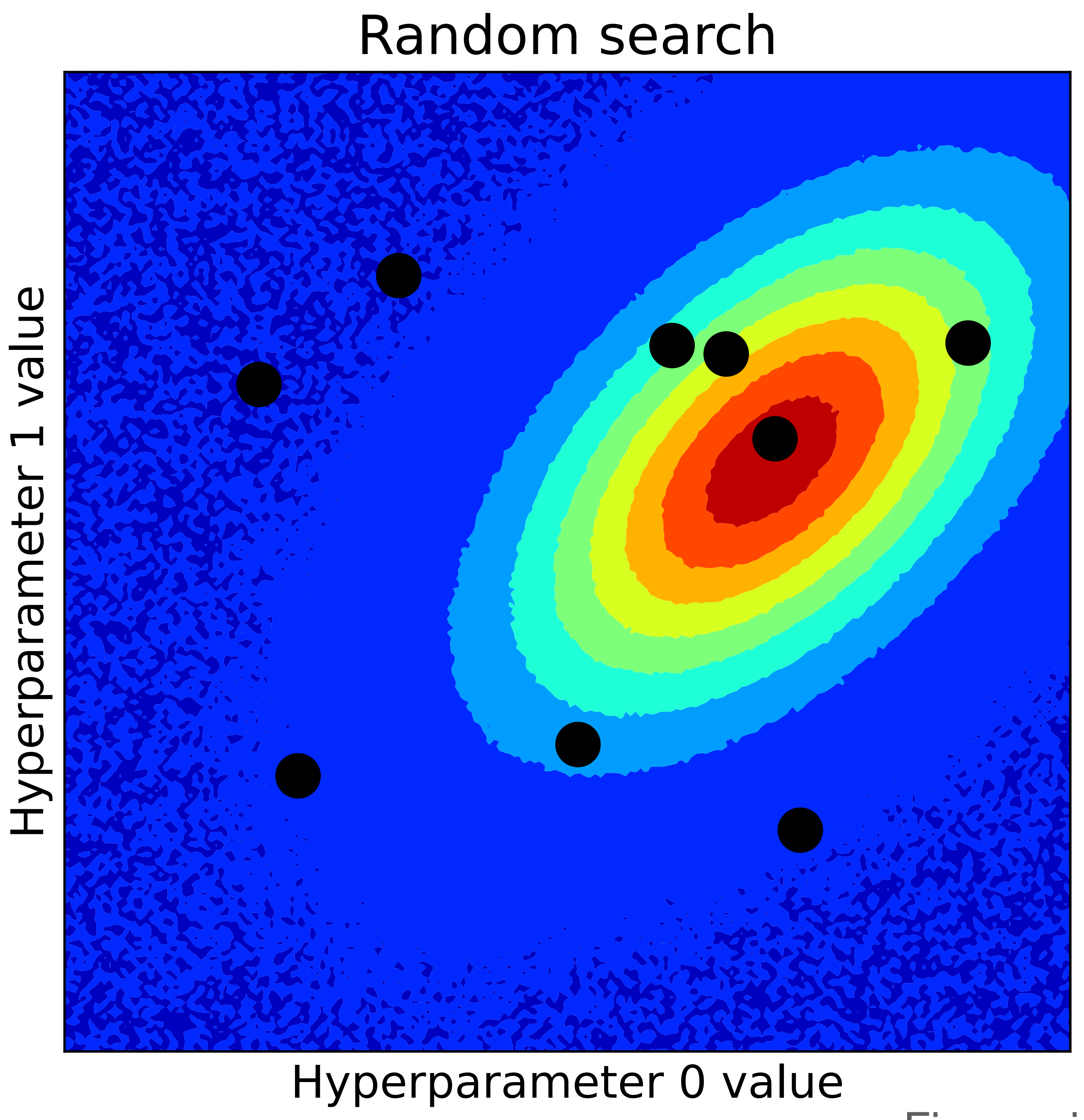
| | $\beta = 0.1$ | $\beta = 1$ | $\beta = 10$ |
|---|---|---|---|
| rbf kernel | 95% | 80% | 78% |
| poly kernel | 56% | 99% | 80% |

Evaluate on the test set as little as possible or you will overfit to it!

# A note on grid search

- Grid search is an intuitive starting point for hyperparameter tuning
- But random search (and other schemes) work better in practice!




Figures inspired by Raschka et al.'s book

# Evaluating classifiers

- So far we have used accuracy as the de facto means to evaluate a classifier
- This is simply the fraction of correct classifications overall
- There are other ways to evaluate classifiers, as accuracy isn’t always the most important thing

# Not all (binary) classifications are equal

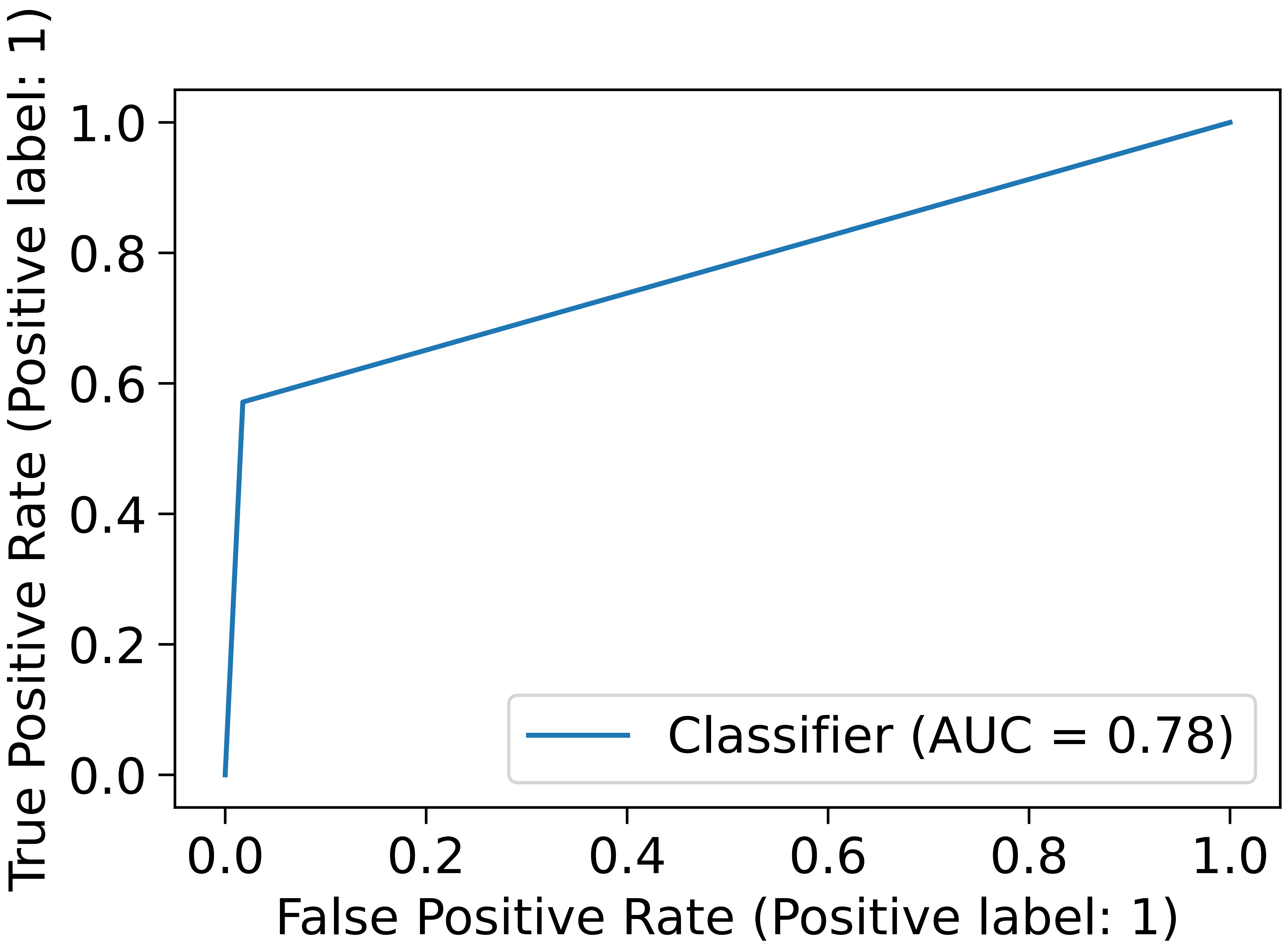
- A patient with cancer is classified as having cancer (**True positive**)
- A patient with cancer is classified as not having cancer (**False negative**)
- A patient without cancer is classified as having cancer (**False positive**)
- A patient without cancer is classified as not having cancer (**True negative**)

We can summarise these possibilities across a dataset using a confusion matrix

| | | Predicted class 0 | Predicted class 1 |
|---|---|---|---|
| True class | 0 | TN | FP |
| | 1 | FN | TP |

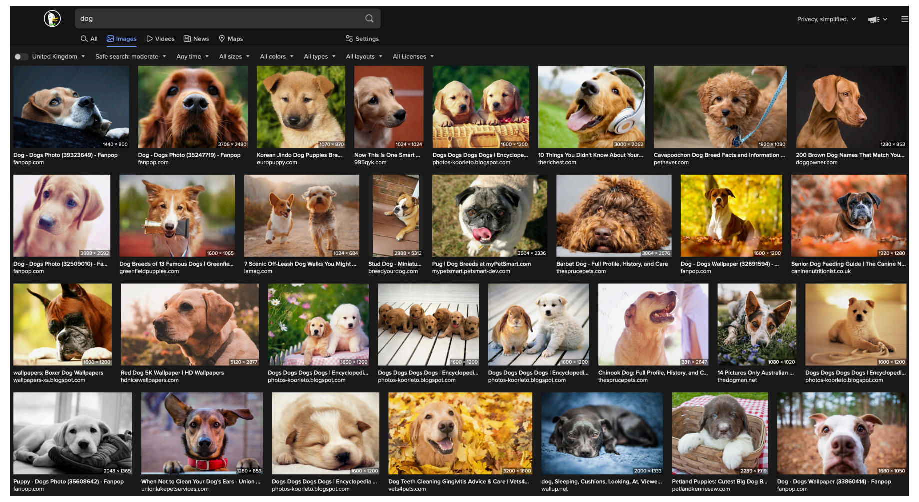
# Receiver operating characteristic (ROC) curves

- These compare true positive rates against true negative rates using different classifier scores as thresholds for binary classifiers
- The area under the curve (AUC) can be used to summarise this
- Ideally this would be 1



# Retrieval

- In a retrieval task we are interested in extracting some data class (e.g. images of dogs) from a larger corpus (e.g. all the images on the internet)
- We can sort data in our corpus according to classification score for the class we want (from highest to lowest score)
- We can then evaluate how good our retrieval system is by looking at:
  - **Precision:** The fraction of top-$k$ scoring points that are in the class we want
  - **Recall:** The number of top-$k$ scoring points that are the in the class we want divided by the total number of data points in that class



# Retrieving dogs

- Let's say we have a corpus of 200 images where 100 are of dogs
- We apply our dog-vs-not-dog classifier to this corpus, and retrieve the top scoring images one by one

$k = 1$
Precision @ $k = 1$
Recall @ $k = 1/100$

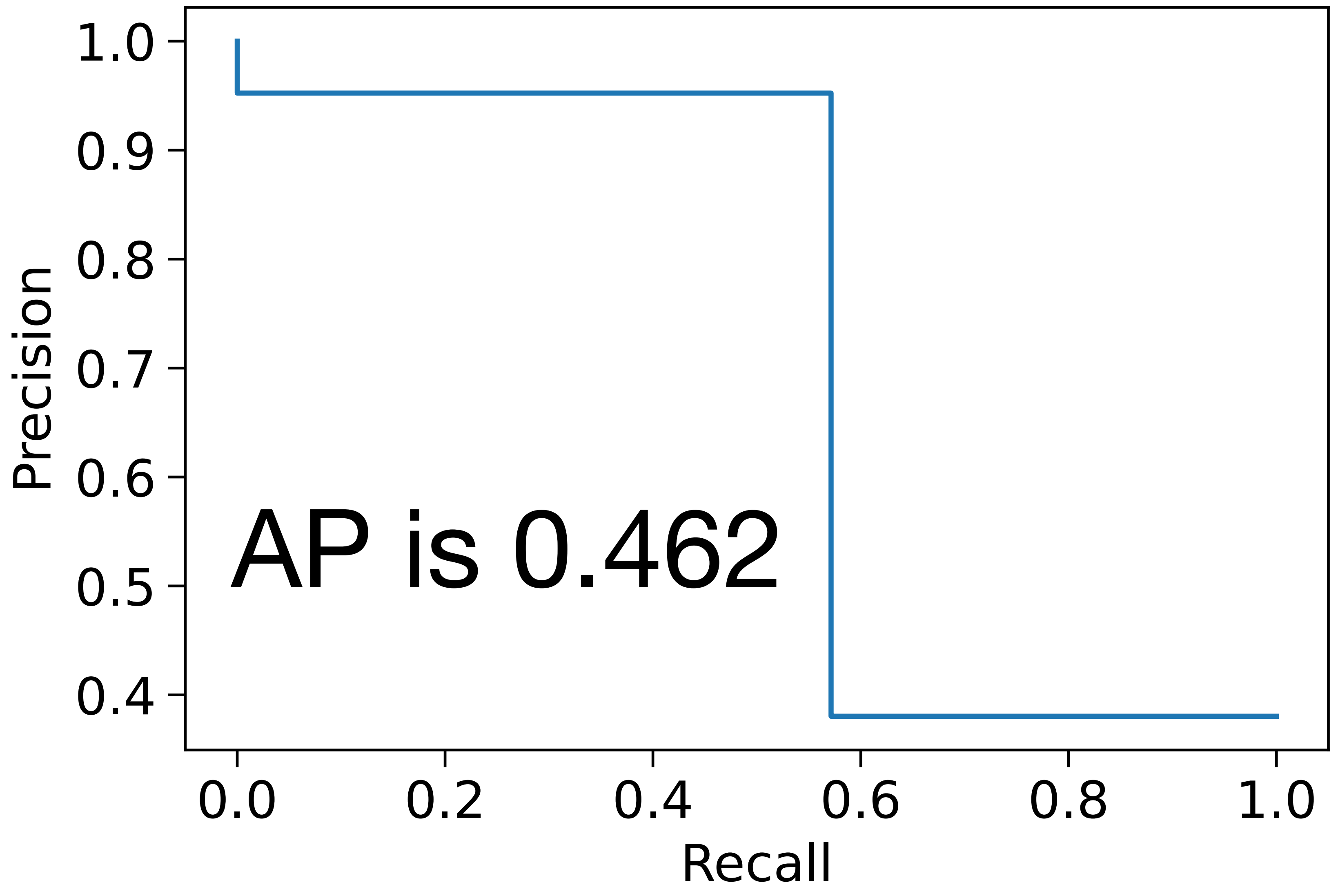
$k = 2$
Precision @ $k = 1$
Recall @ $k = 2/100$

$k = 3$
Precision @ $k = 2/3$
Recall @ $k = 2/100$

$k = 4$
Precision @ $k = 3/4$
Recall @ $k = 3/100$

# Precision-Recall curves

- Precision and recall can be plotted against each other
- The area under this curve is called **average precision (AP)** and is commonly used to summarise retrieval performance (especially for object detection)
- If we are retrieving multiple classes separately we can take the mean of the AP for each class to get mean average precision (mAP)



# Summary

- We have learnt how we can maximise the margin of a linear classifier to form a hard-margin support vector machine for linearly separable data
- We have seen how we can relax the margin constraint to form a soft-margin support vector machine that allows for margin violations
- We have considered the dual form of an SVM expressed in terms of support vectors
- We have considered feature maps for dealing with linear inseparability
- We have seen how the kernel trick can implicitly perform feature mapping
- We have looked at different way to evaluate classifiers