Lecture 4: Linear Models Revisited

Matthew Caldwell

COMP0088 Introduction to Machine Learning · UCL Computer Science

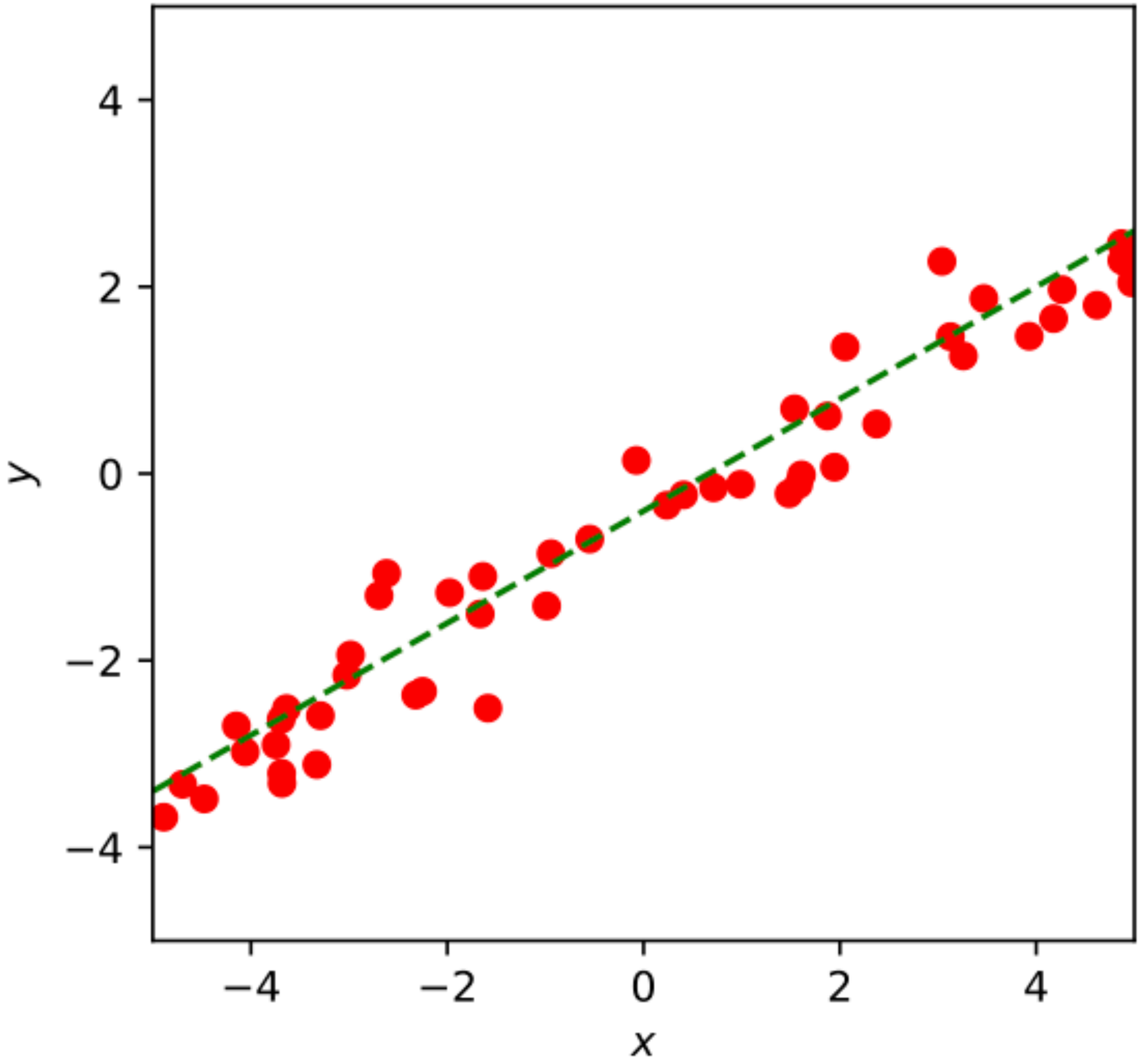
Contents

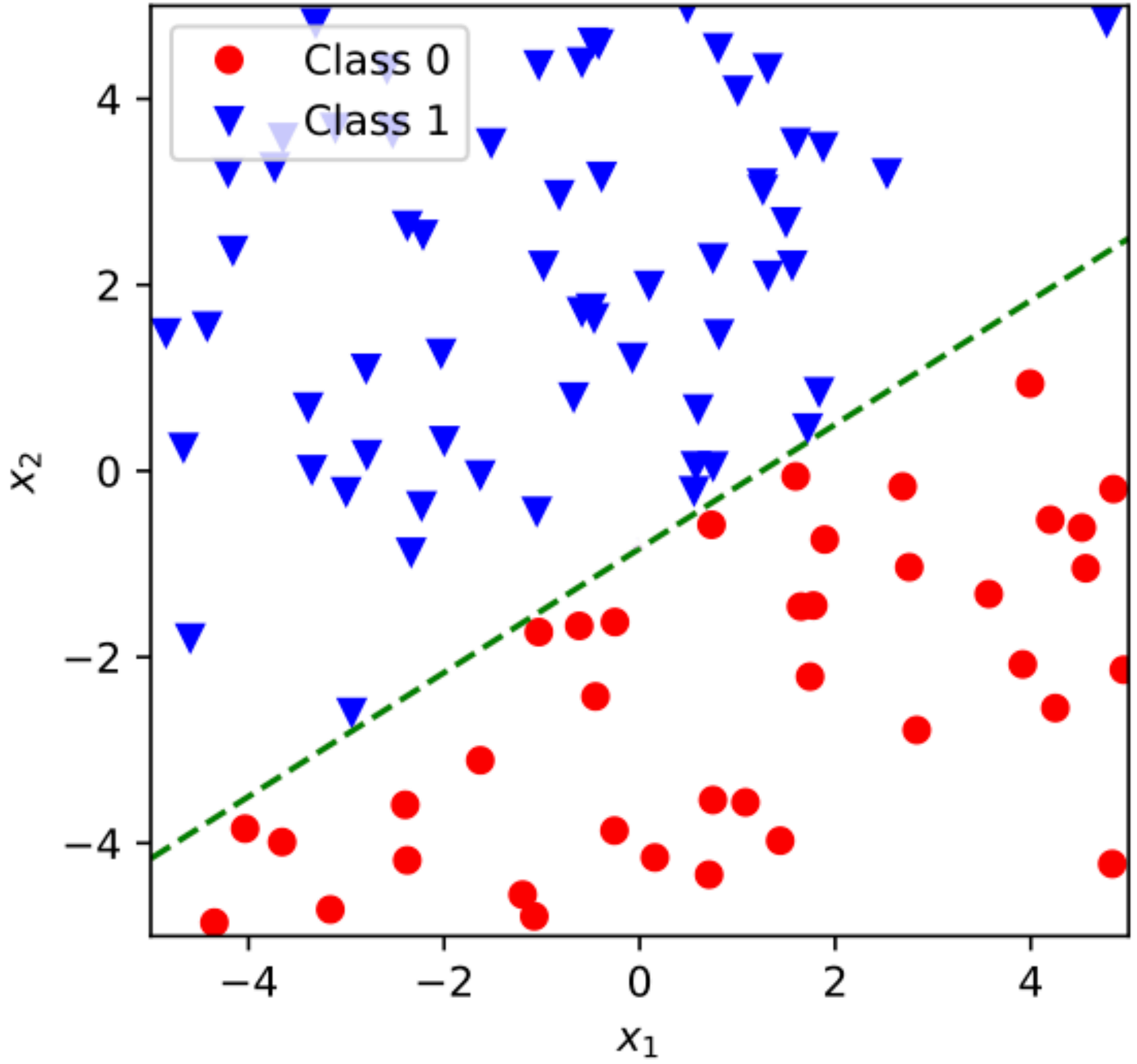
4.1: The Story So Far

COMP0088 Introduction to Machine Learning • UCL Computer Science

Linear models recap

- Linear models fit a hyperplane by taking a weighted sum of the input features
- For regression problems, the hyperplane traverses (feature+label) space and we'd like training samples to be on or near it
  - → A regression loss function captures this closeness to the hyperplane
- For binary classification, the hyperplane divides feature space into two half-spaces and training samples are separated by it
  - → A classification loss function captures the correctness of region occupancies
  - → Data that can be perfectly split by a hyperplane is known as linearly separable
- Multiclass problems require fitting of multiple hyperplanes and in some way adjudicating amongst them, either collectively or pairwise


4
2
0
−2
−4
y
−4
−2
0
2
4
x


Class 0
Class 1
4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1

Scoring & decision functions

- A convenient interpretation of linear model outputs for classification is as a score, estimating strength of conviction that a sample belongs to a class
- Scores are passed through a non-linear decision function to map them to discrete class predictions
  - → Often this is a zero threshold
- Other non-linearities may also be applied to impose constraints or structure on the score for calculating losses or estimating probabilities
  - → E.g. the logistic and softmax “squashing” functions ensure probabilities are non-negative and sum to 1

Margins

- The degree of correctness or incorrectness of a classification can be quantified by its margin
- Using {-1, 1} for *y* and scoring function *f(x)*, the margin is *y f(x)*
- Margin magnitude is confidence of prediction
- A negative margin means the prediction is incorrect

Basis expansion

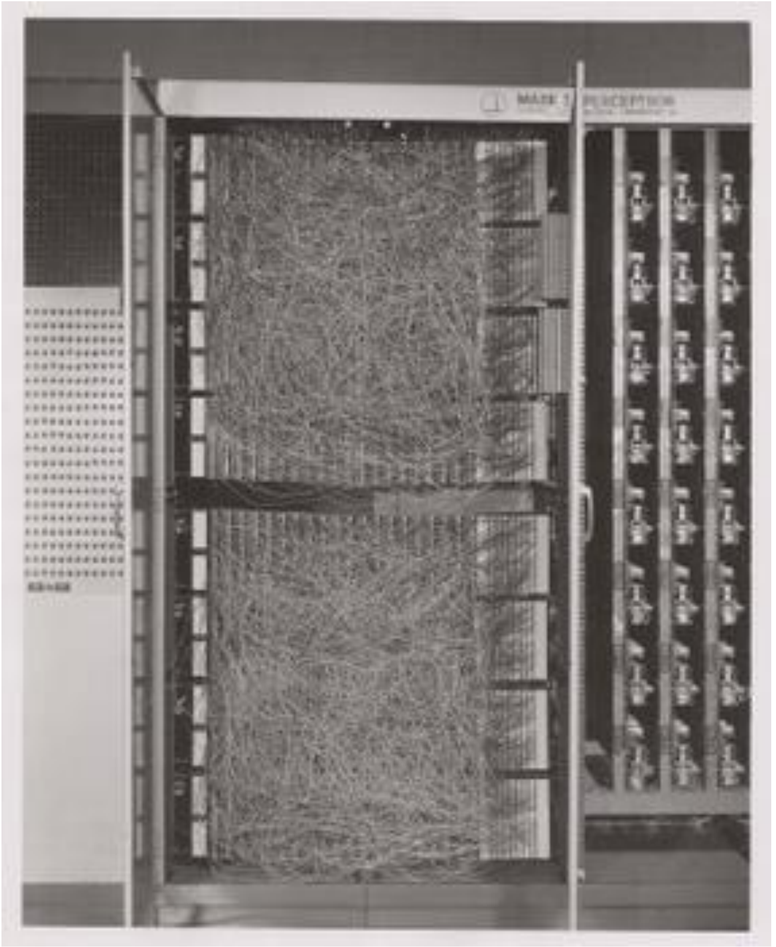
- Transformations applied to the inputs before fitting the linear model can project the data into a new feature space or basis
- Any set of functions can be taken as the basis within which to fit a linear model, provided the functions themselves have no parameters to learn
- A hyperplane in an expanded basis may represent an arbitrarily complex non-linear boundary when projected back into the original space
- But choosing an explicit basis can be problematic, and computing the transformed vectors expensive

4.2: Perceptrons

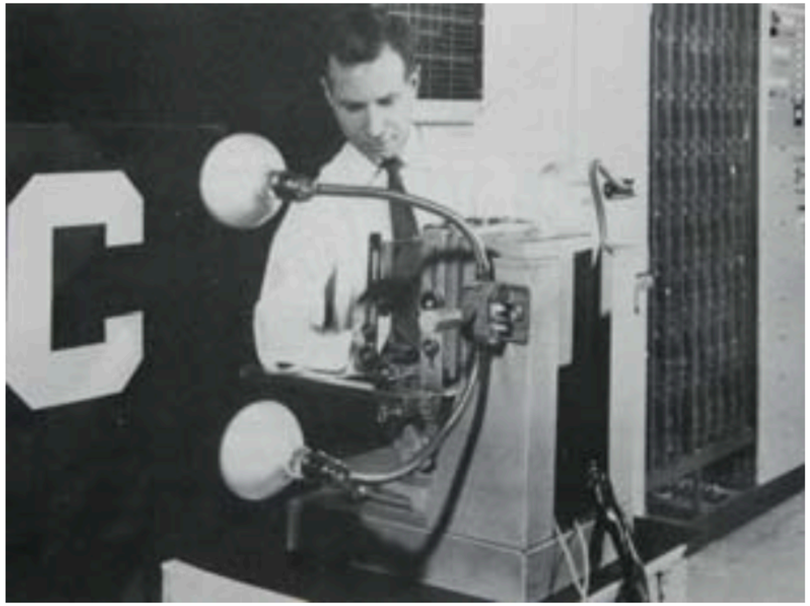
COMP0088 Introduction to Machine Learning · UCL Computer Science

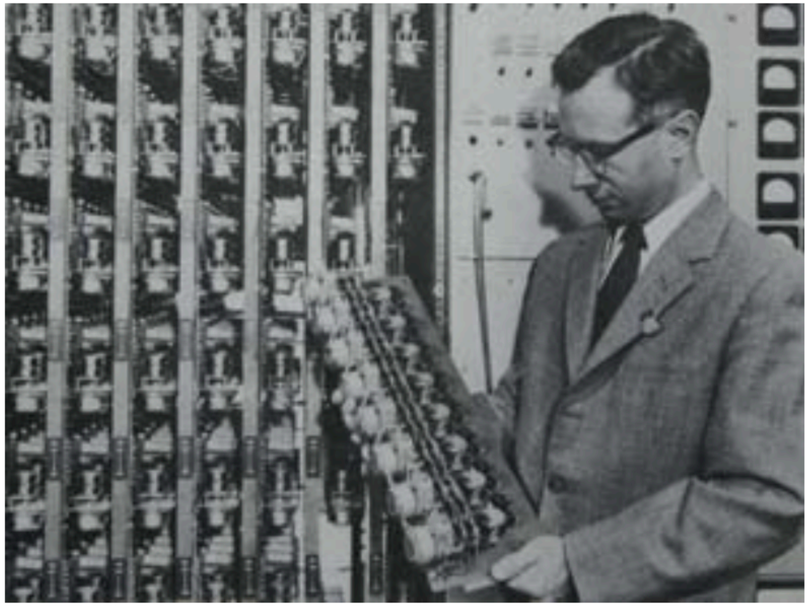
Tragical history tour

- Early “connectionist” (neural network) model, designed to be implemented in hardware
- Impressive ability to solve linearly-separable problems in a simple and elegant way
- Almost immediately promoted by overeager salespeople* in ill-advised hyperbolic fashion
- Failed to deliver on these wild promises because of limitations that were already understood
- Disappointment fuelled a backlash that stalled neural network research for years



* not actually salespeople





“Perceptron may eventually be able to learn, make decisions, and translate languages.”

— *Frank Rosenblatt*

“The embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”

— *New York Times*

A model that isn’t a model

- The perceptron is just a bog-standard linear classifier
- Linear decision boundary at $\mathbf{x} \cdot \mathbf{w} = 0$
- It (sort of) uses undifferentiable **0-1 loss** to determine updates
  - → But better understood as optimising margin (towards 0) for misclassified samples, which makes the loss piecewise linear
- The perceptron **algorithm** is a method for iteratively training such a model, one training sample at a time
- Effectively performs stochastic gradient descent

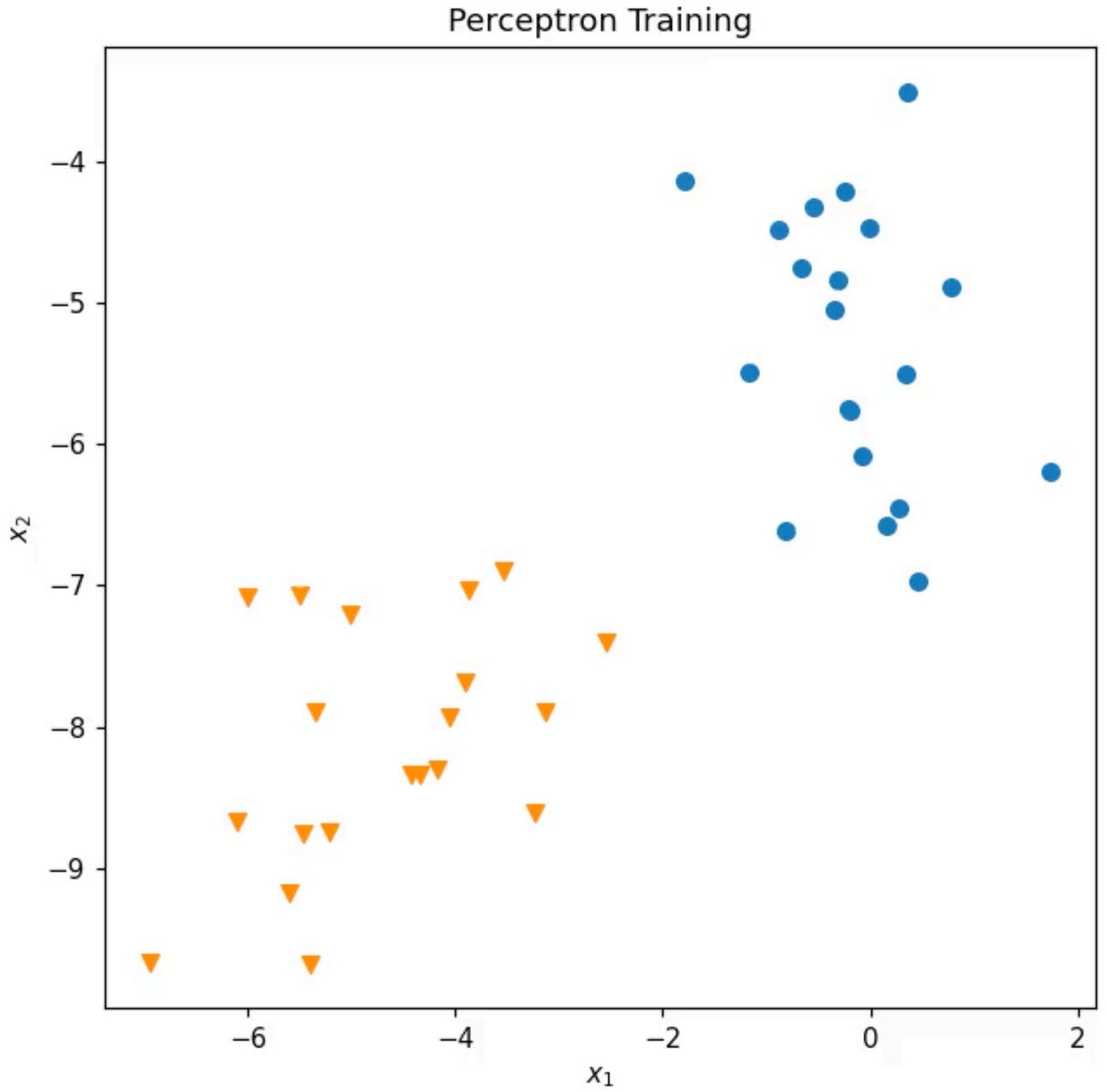
The Algorithm

- Initialise: $\mathbf{w} \leftarrow \mathbf{0}$
  - → (Could also be any other initial value)
- For each sample $\mathbf{x}$ in the training set:
  - Predict: $\hat{y} = \mathbf{1}(\mathbf{x} \cdot \mathbf{w} \geq 0)$
  - Update: $\mathbf{w} \leftarrow \mathbf{0} + \alpha(y - \hat{y})\mathbf{x}$
    - → NB: only updates weights when prediction is wrong
- Repeat until there are no errors

What is it doing?

- If a sample is correctly classified, do nothing
- If a sample is incorrectly classified, move the boundary towards it
  - → Easiest to consider without intercept (true in d+1 space anyway), so boundary passes through origin and only need to consider direction
  - → Direction to point **x** from origin is just the vector **x** (scaled by any positive amount)
  - → Adding or subtracting **x** makes **w** more like a scaled version of **x** — ie, rotates it in that direction

- Note that (with initialisation to 0) the eventual boundary vector **w** consists of a weighted sum of (some subset of) the training samples


Perceptron Training
−4
−5
−6
−7
−8
−9
x_2
−6
−4
−2
0
2
x_1

Properties

- Very easy to understand and implement
- There's a hyperparameter α, but it's not especially important
  - → Affects speed of convergence, but not the fact of it
- If the data is linearly separable, it's guaranteed to converge (eventually)
  - → Not necessarily to a "good" boundary — we'll come back to this shortly
- If not, the algorithm will never terminate
  - → and may loop in long cycles where there isn't an easily identified "best approximation" point to stop at

Legacy

- There are a number of variants that mitigate its limitations
  - → Maxover
  - → Voted perceptron
  - → α-perceptron
  - → etc
- But for our purposes it's mostly significant as
  - → a conceptual building block for modern neural nets
  - → an object lesson in the matter of AI overpromising and underdelivering

4.3: Optimal Hard Margin

COMP0088 Introduction to Machine Learning · UCL Computer Science

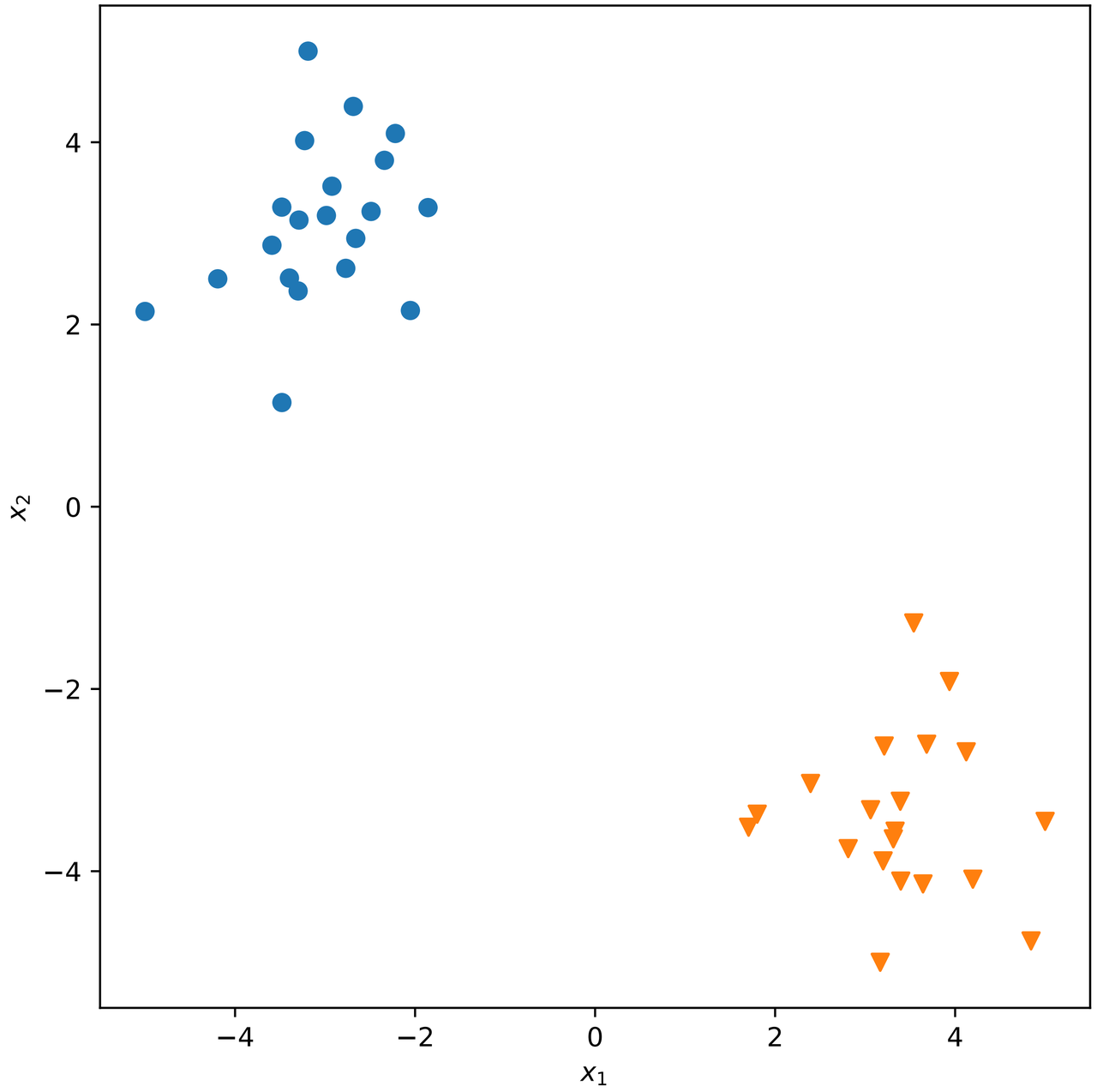
Sidling up to SVMs

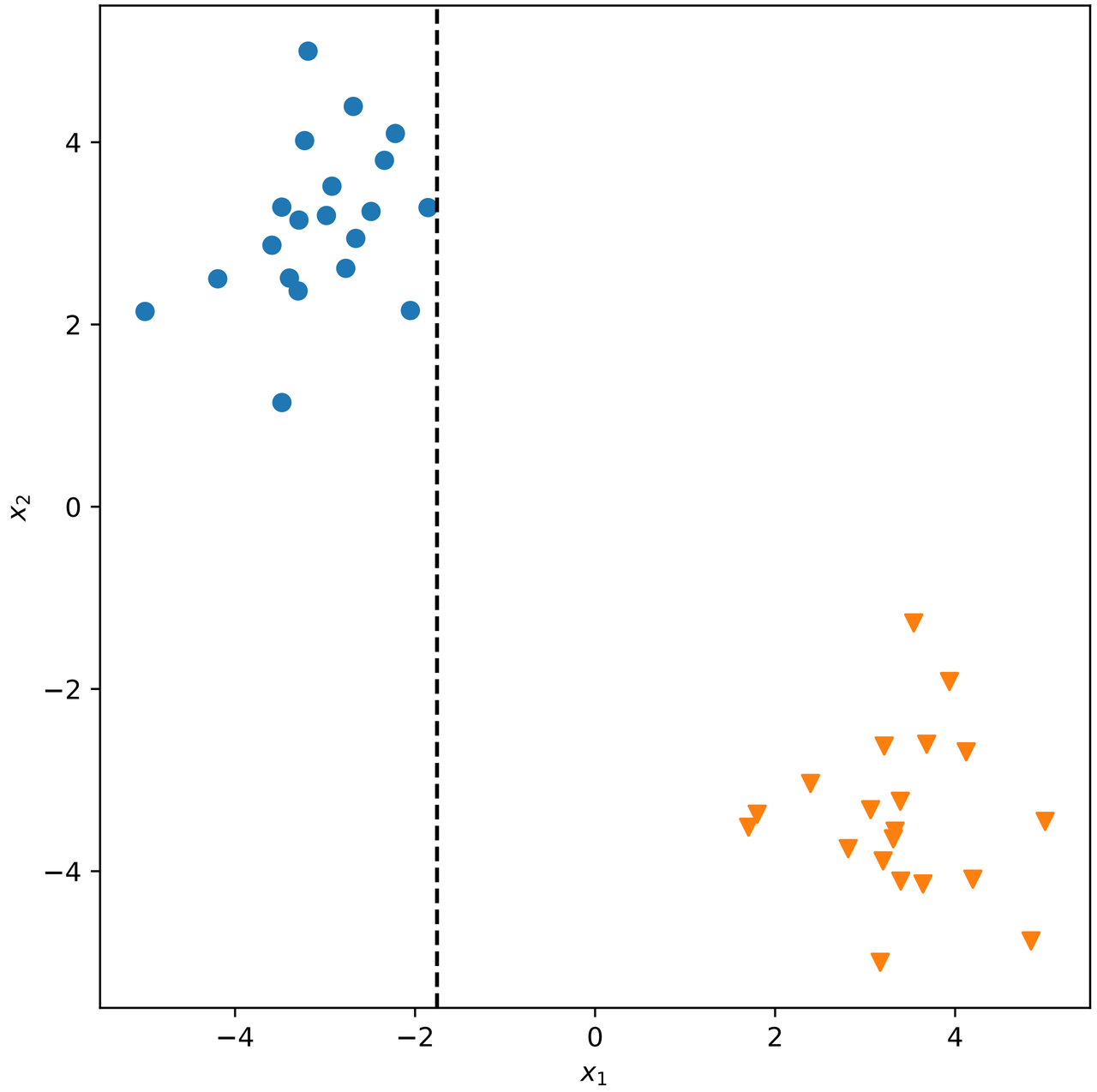
- All the rest of this week's lecture is about support vector machines
- But we're going to take it in stages
- For starters, consider the same kind of binary linear classifier and the same kind of linearly-separable binary problem as we've seen multiple times before, including for the perceptron, but with two notational differences
  - → Use the {-1, 1} labelling scheme we used for AdaBoost
  - → Separate out the intercept term *b* rather than merging as w_0

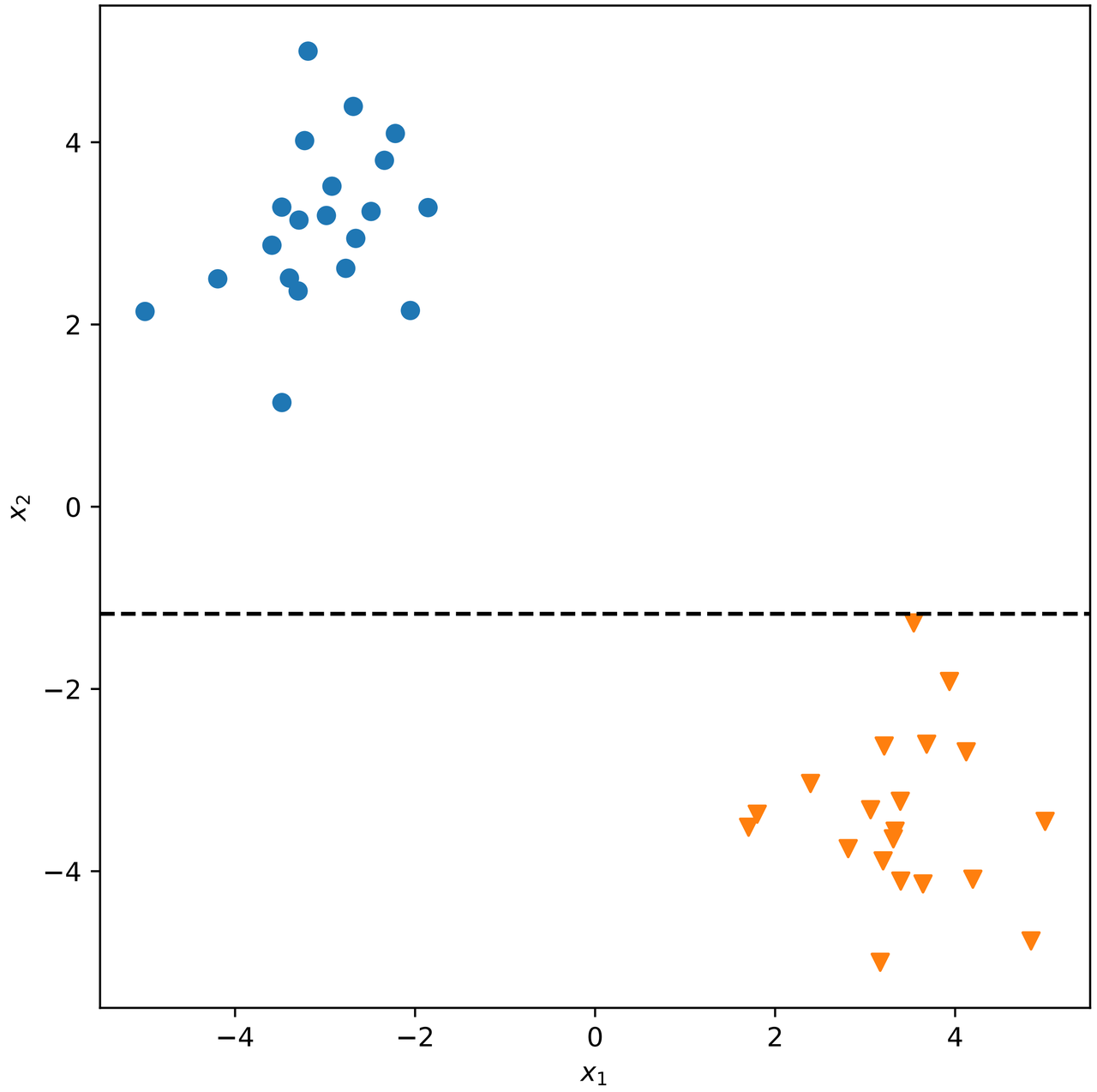
$$\hat{y} = \begin{cases} 1 & \text{if } \mathbf{x} \cdot \mathbf{w} + b \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

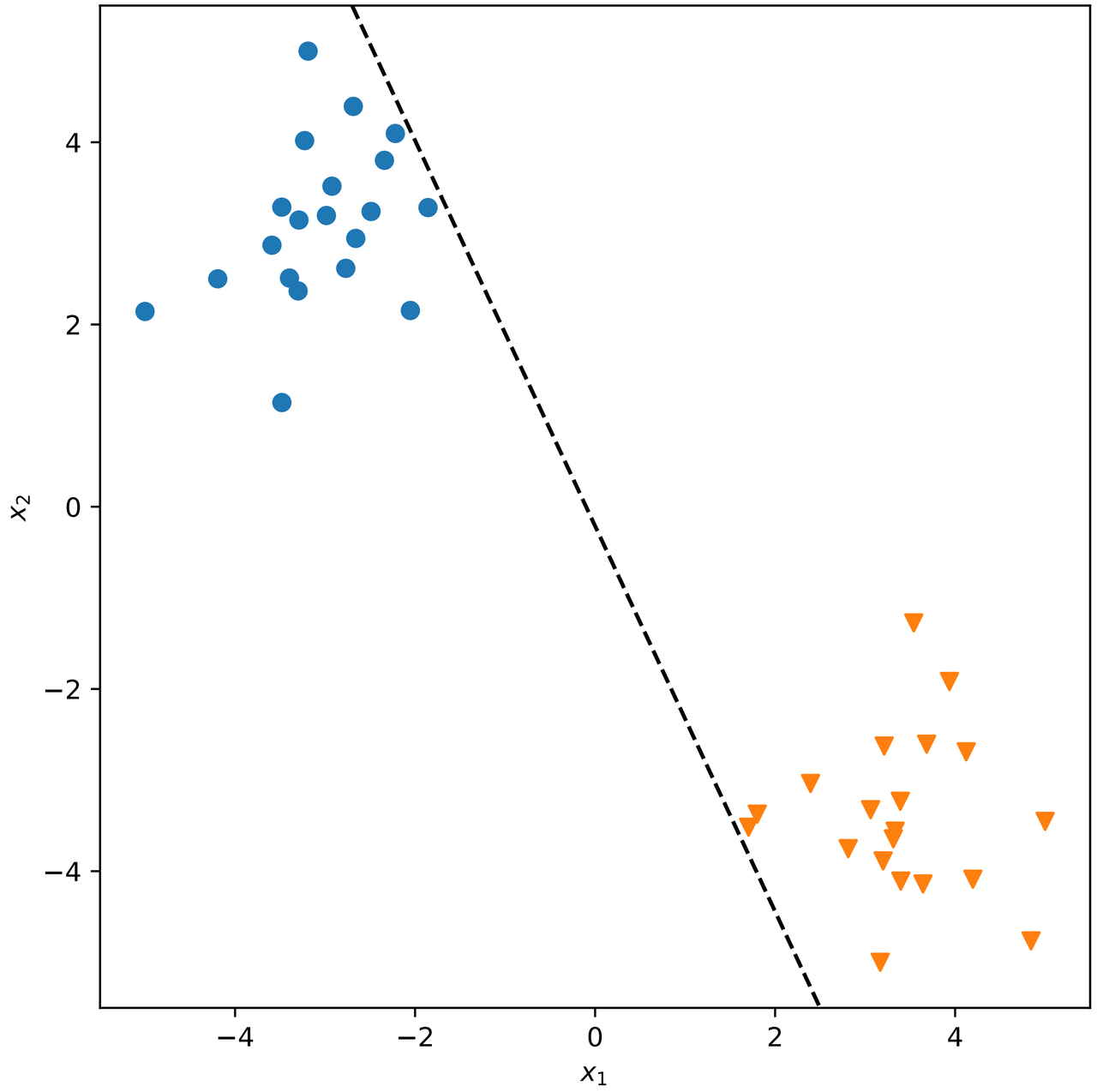
Where to draw the line?

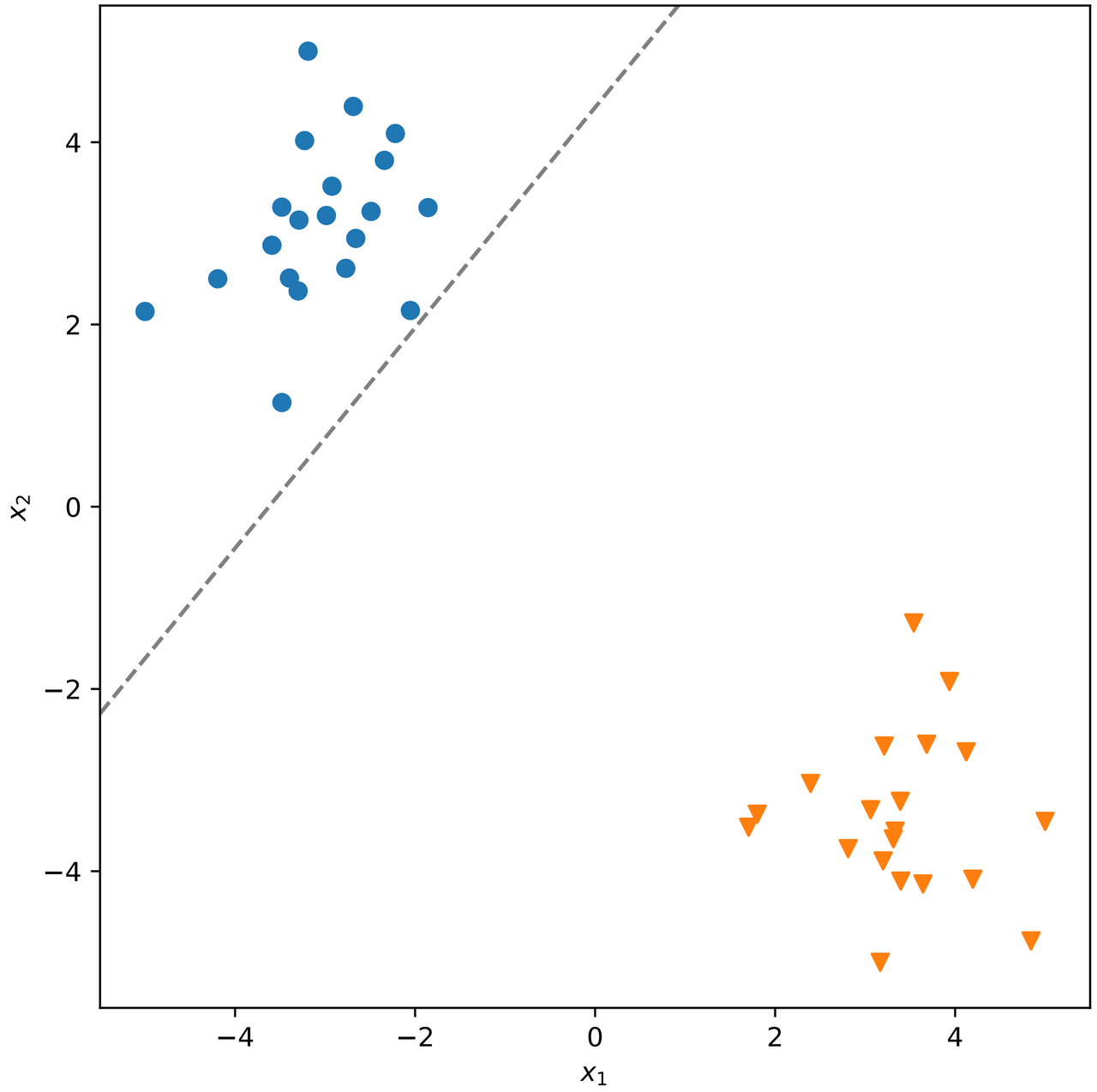
- So: we're fitting a *d*-dimensional hyperplane that separates all the positive class training samples from all the negative class ones
- Given that the classes are linearly separable, there must be an infinite number of possible decision boundaries (although they may be very similar in some cases)
- To an algorithm like the perceptron, any such boundary will do
- But we might want to consider other criteria

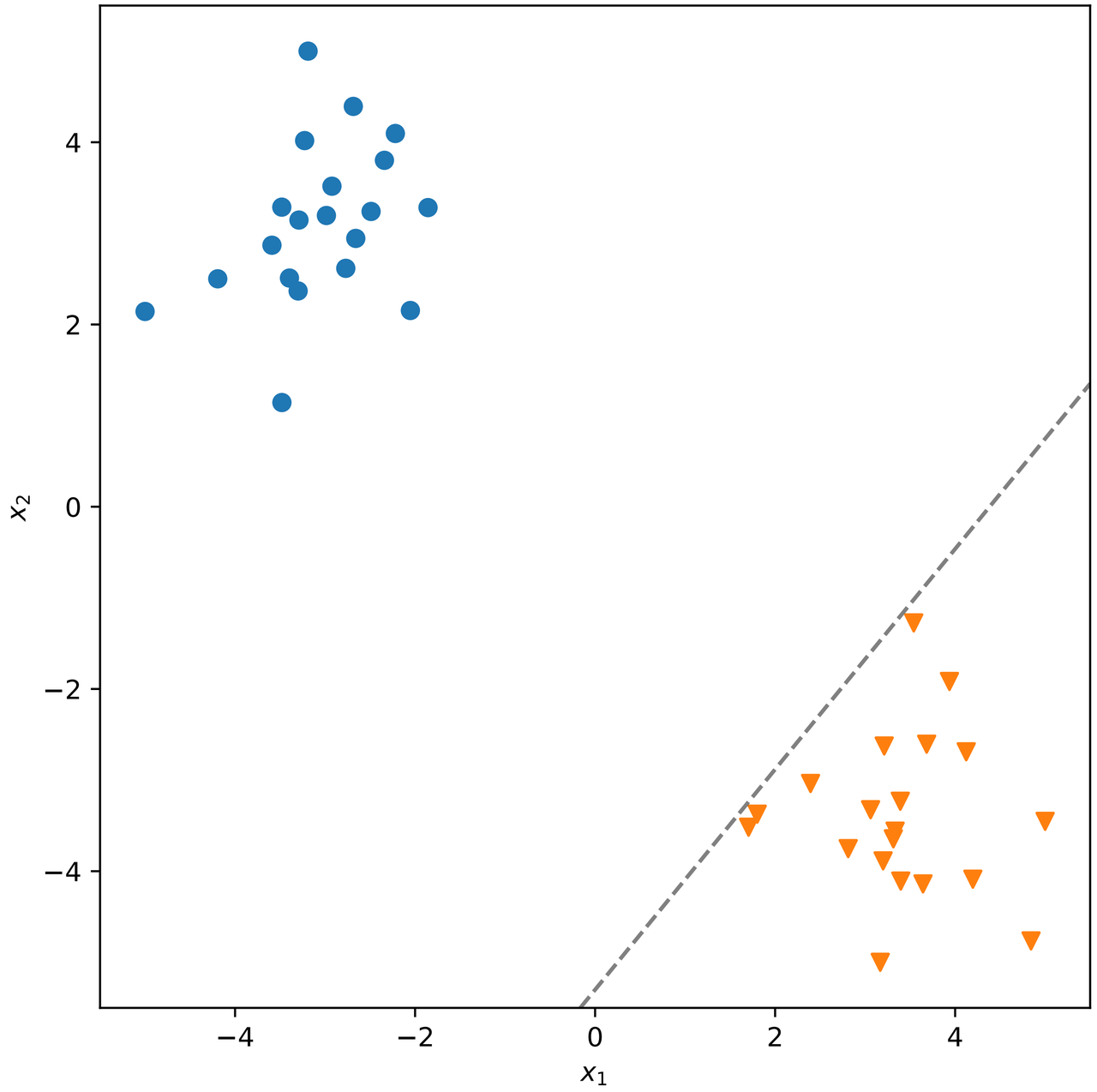

4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1


4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1


4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1

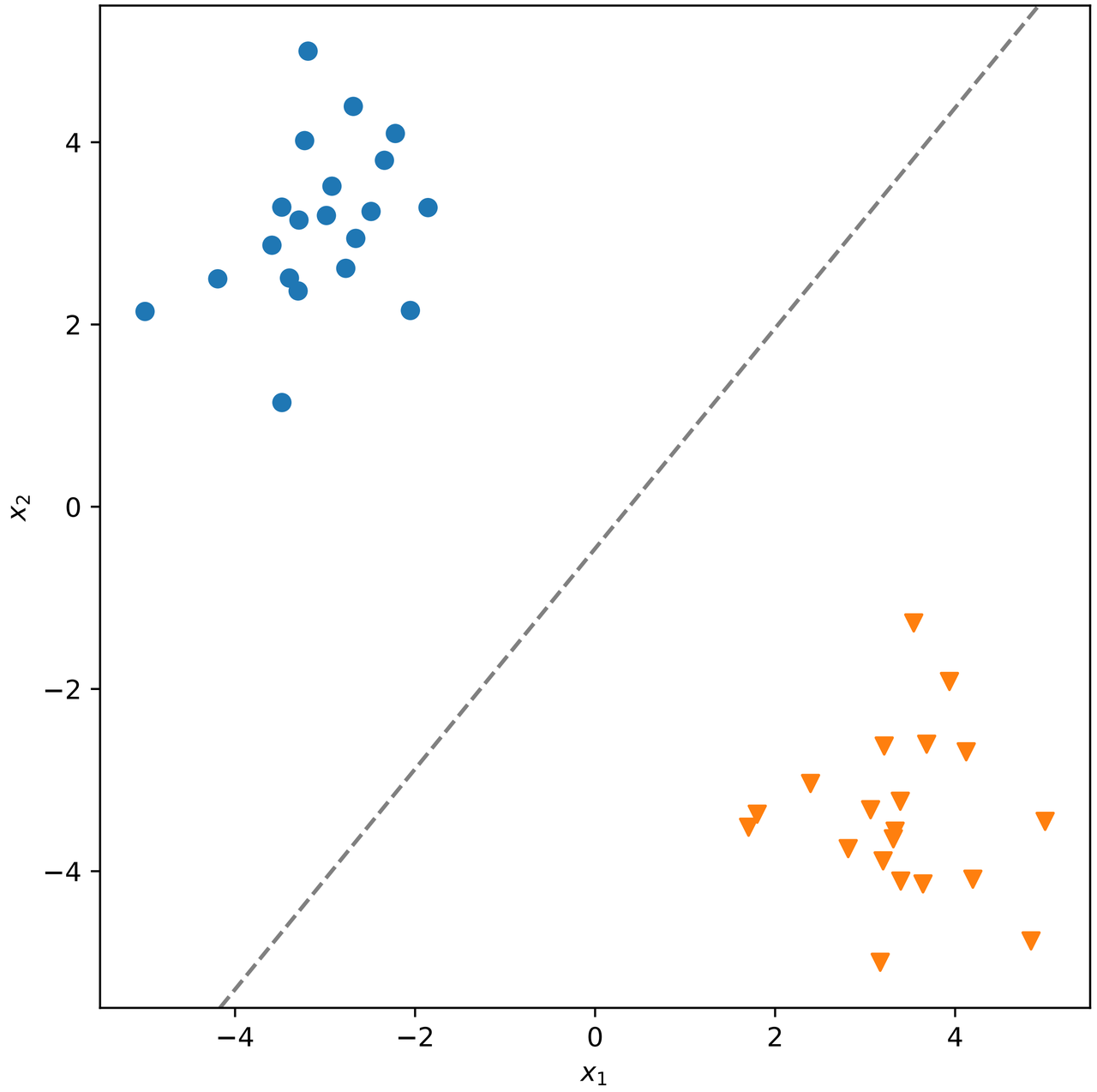

4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1

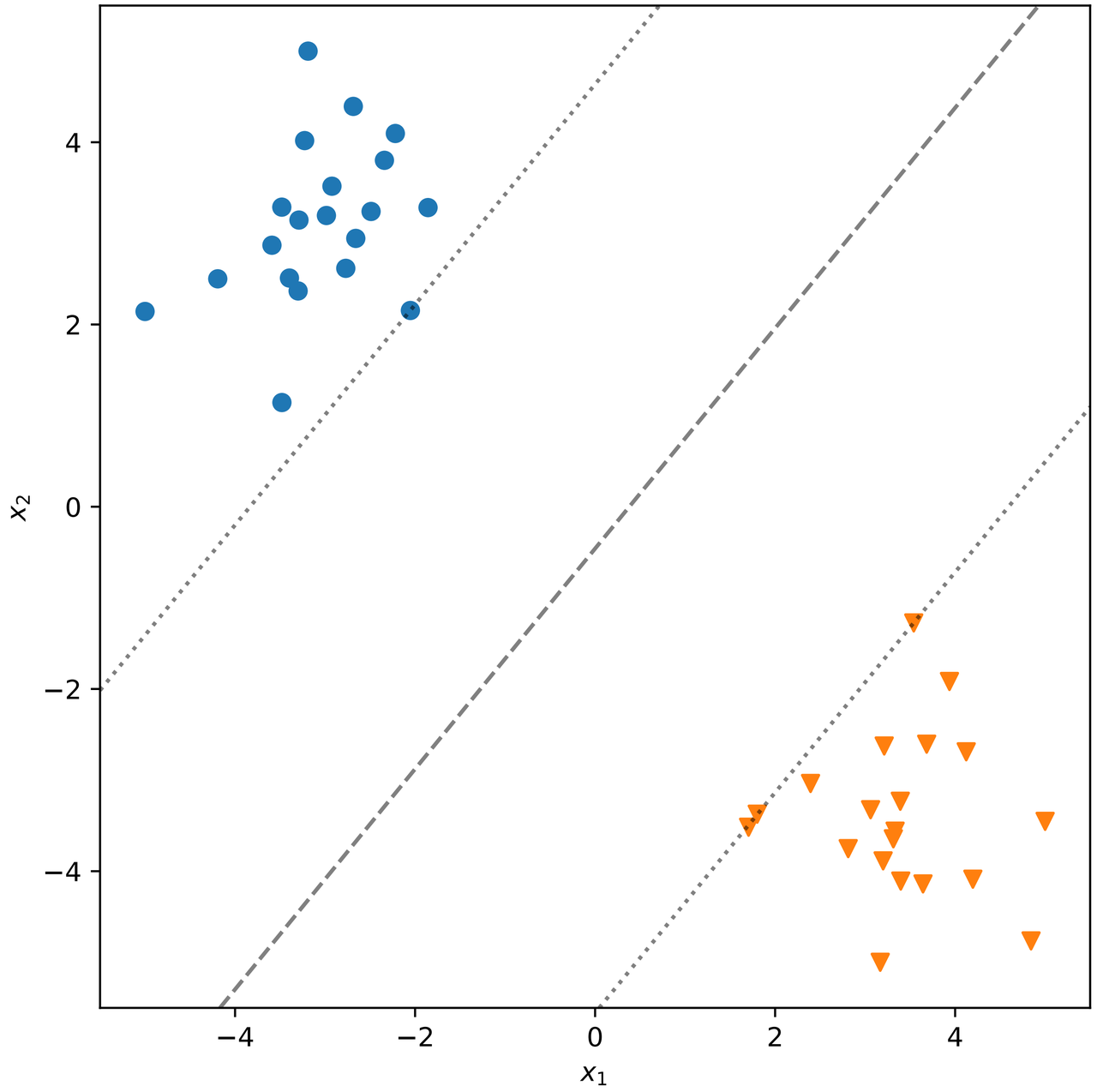

4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1

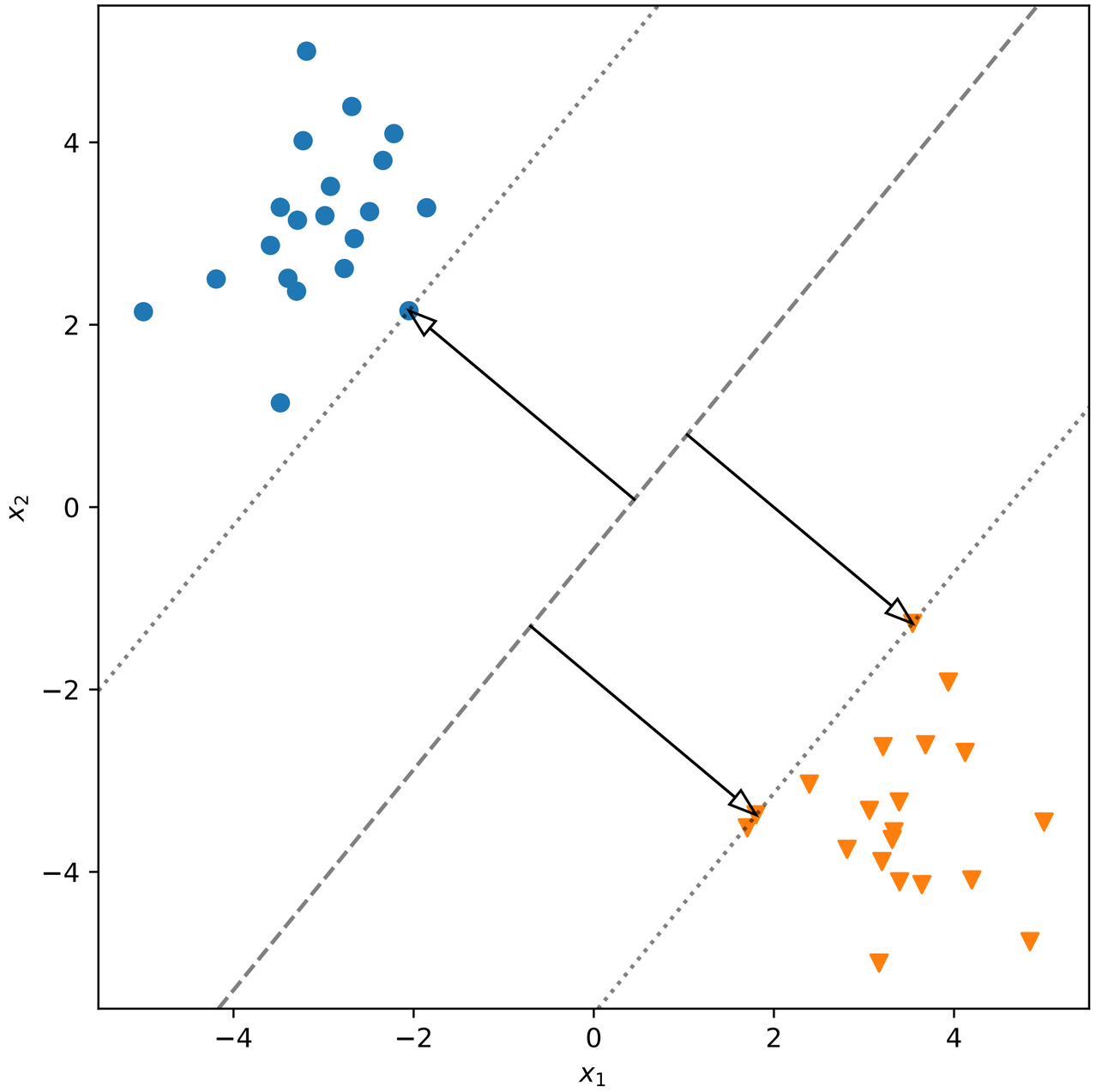

4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1

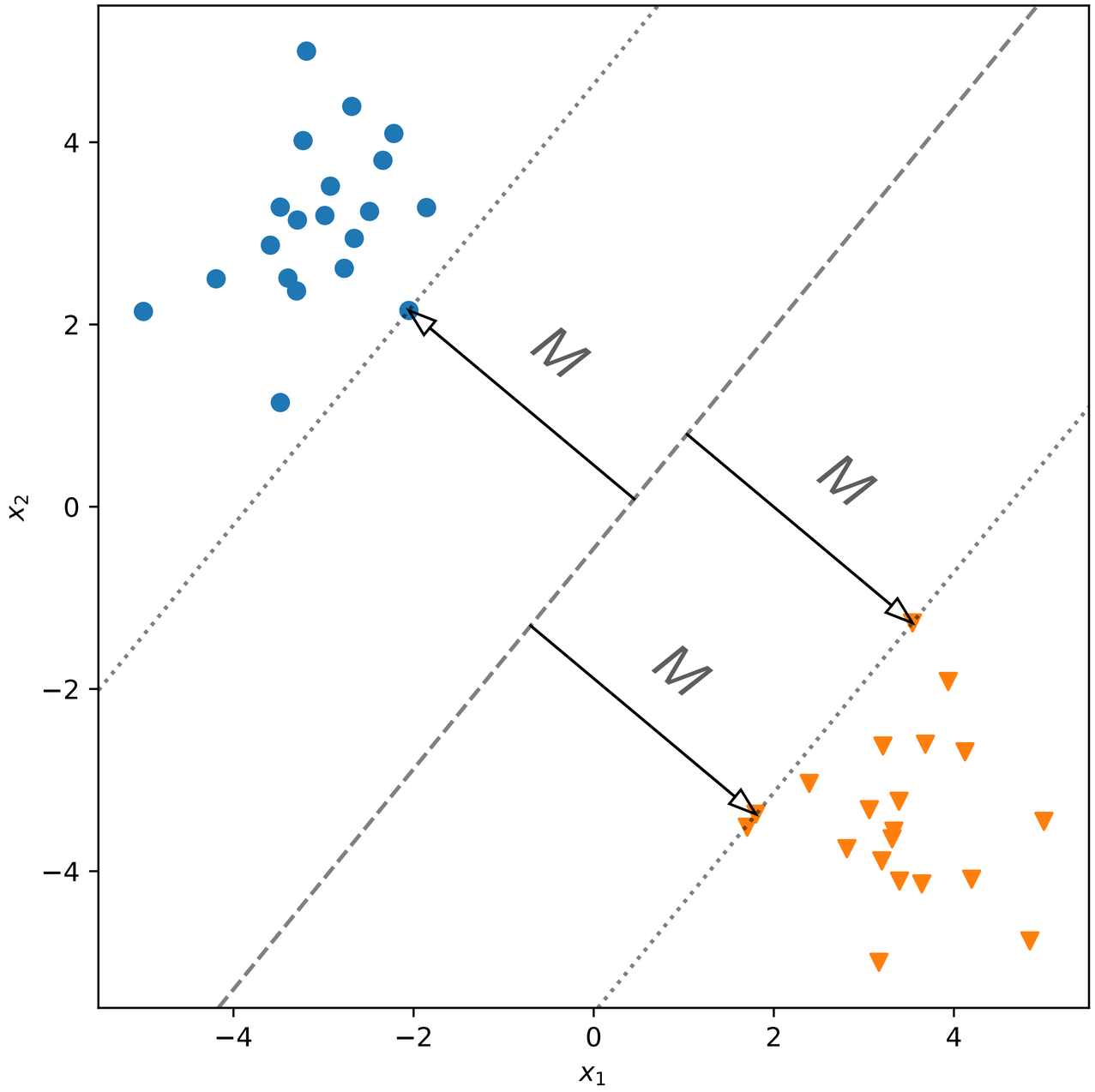
Intuition: classes should be well-separated

- If the boundary is very close to the samples of one class, that is putting a lot of faith in the accuracy and completeness of the training data
- Small amounts of natural variation or measurement noise in future test samples could easily slip over to the wrong side of the boundary
- With our scoring function interpretation, samples close to the boundary have a low score and hence high uncertainty
- We would like our predictions to be confident and robust


4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1


4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1


4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1


4
2
0
−2
−4
x_2
−4
−2
0
2
4
x_1
M
M
M

Large margin criterion

- One intuitive way of choosing split is that it should have the largest margin
- We've previously defined the margin like this:

$$M = yf(\mathbf{x}) = y(\mathbf{x} \cdot \mathbf{w} + b)$$

- This is sometimes referred to as the functional margin. The arrows on the figure show what is known as the geometric margin — the actual distance.
- Both margins are the same if **w** is a unit vector, i.e. under the constraint

$$\|\mathbf{w}\| = 1$$

How do we fit this?

- *M* is the worst case margin — all points should have margin at least *M*
- Might express this as:

$$\underset{\mathbf{w},b}{\text{maximise}} \;\; M$$

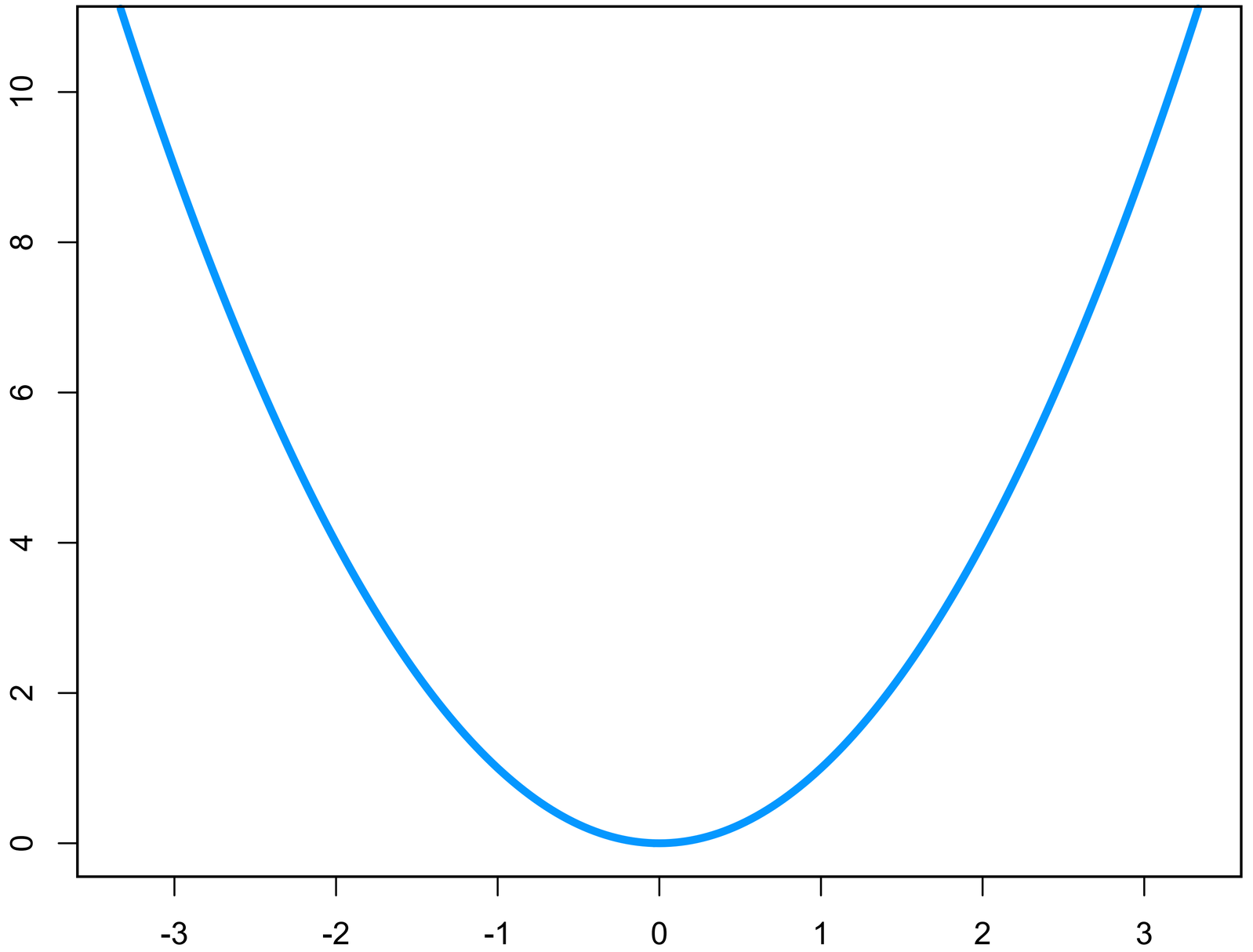
$$\text{subject to:}$$

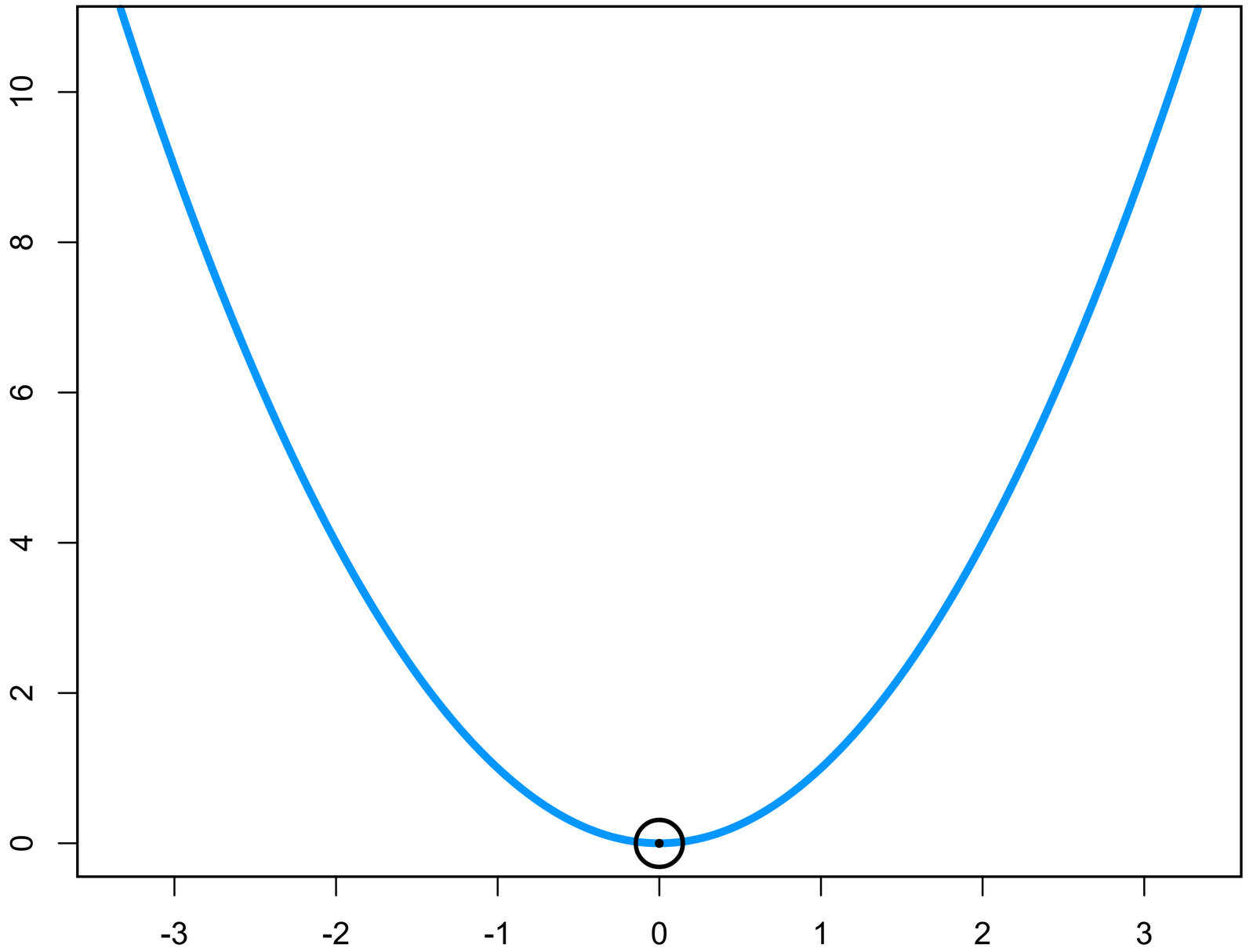
$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq M$$

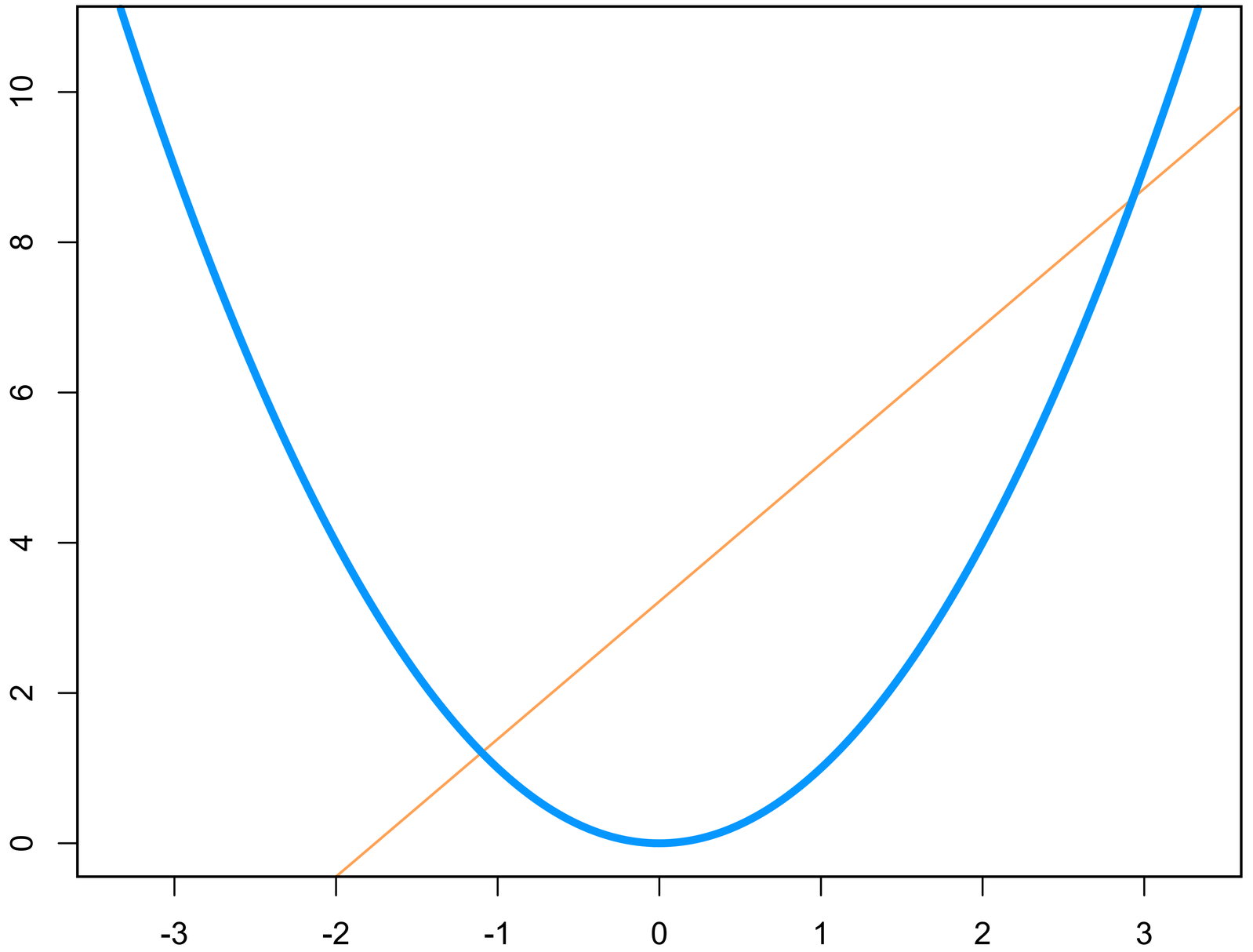
$$\|\mathbf{w}\| = 1$$

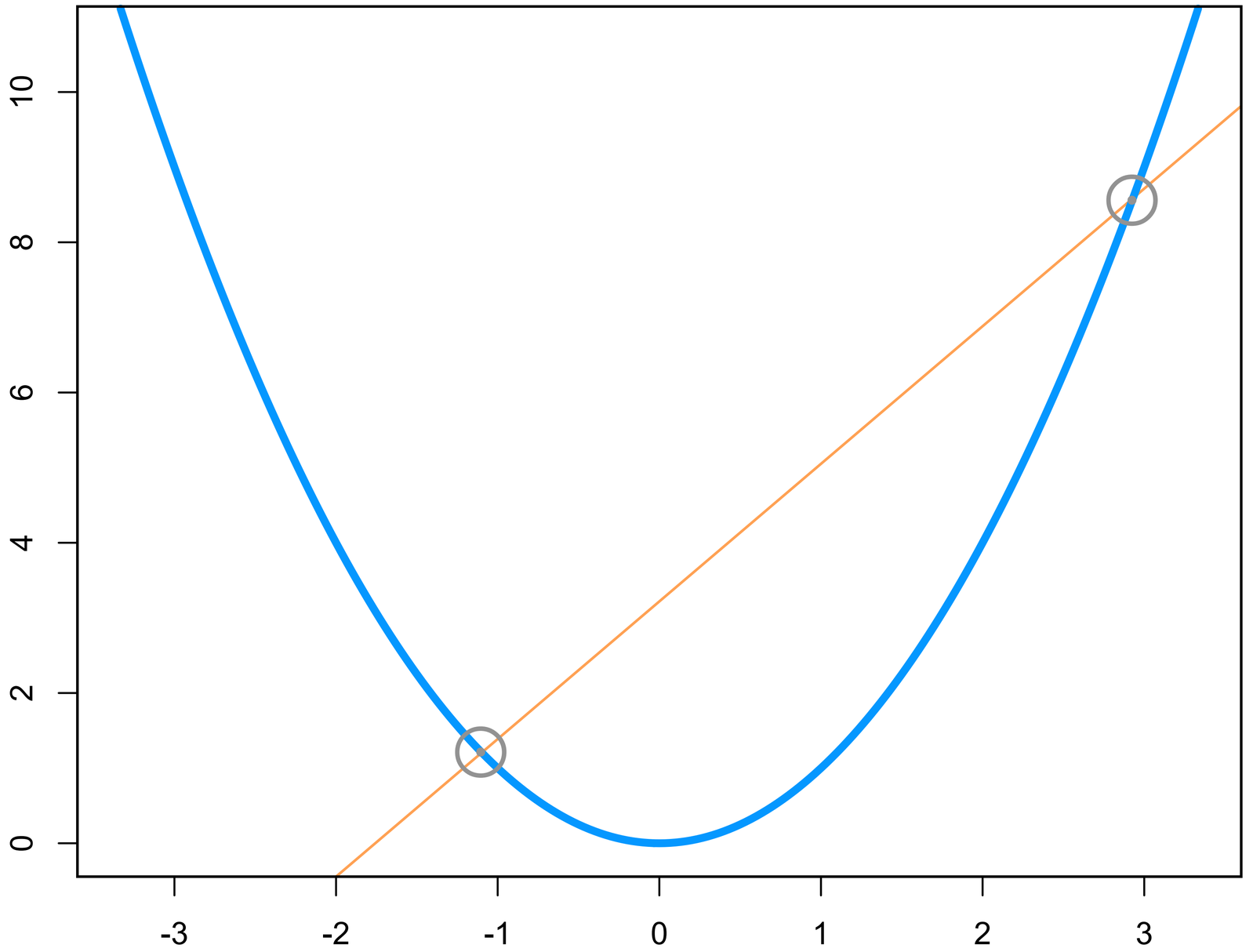
Sidebar: constrained optimisation

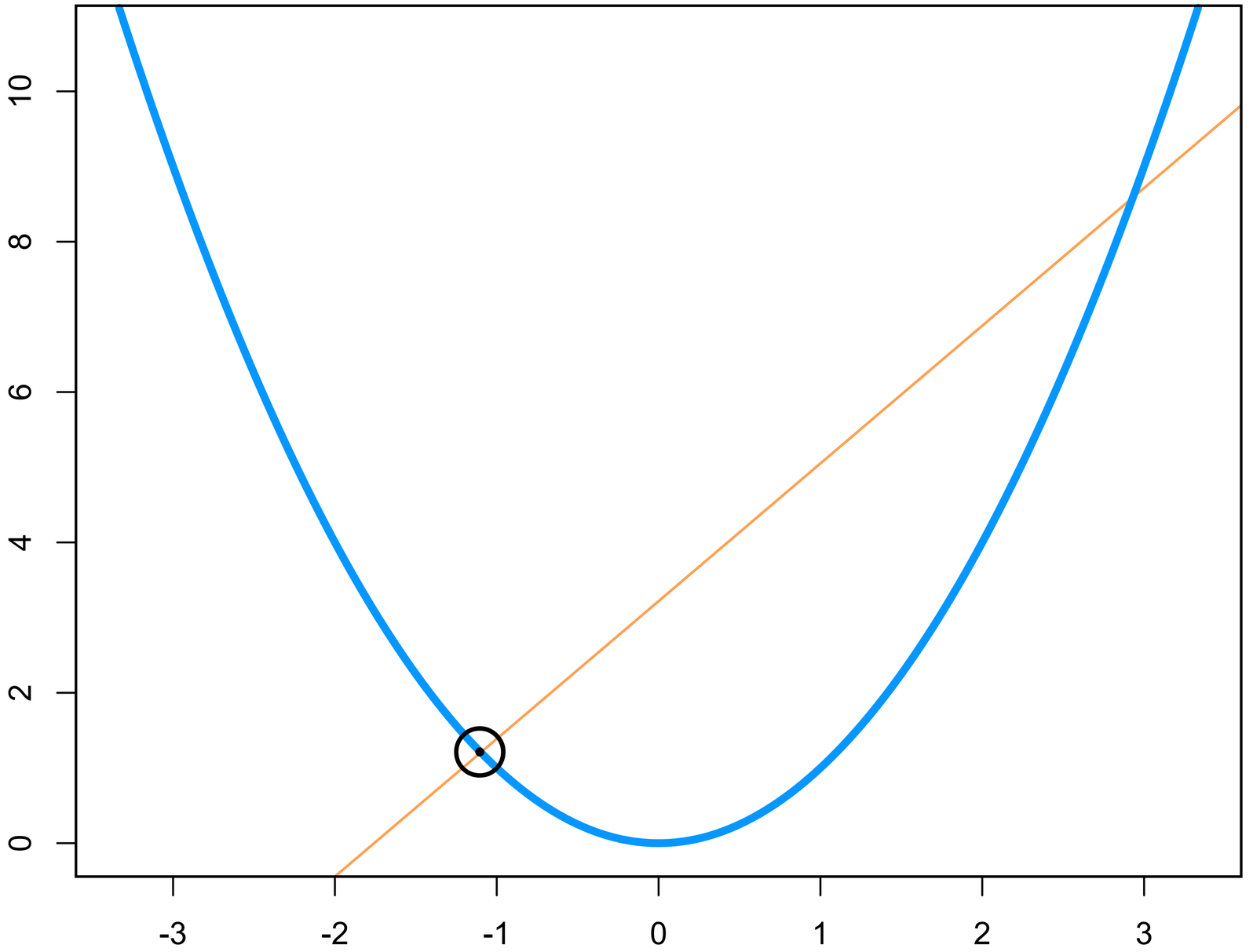
- Margin maximisation differs from others we've seen so far
- Adding constraints changes what you're optimising
- It's possible that the constraints don't turn out to matter — optimum may wind up being in the same place — but also possible they change everything

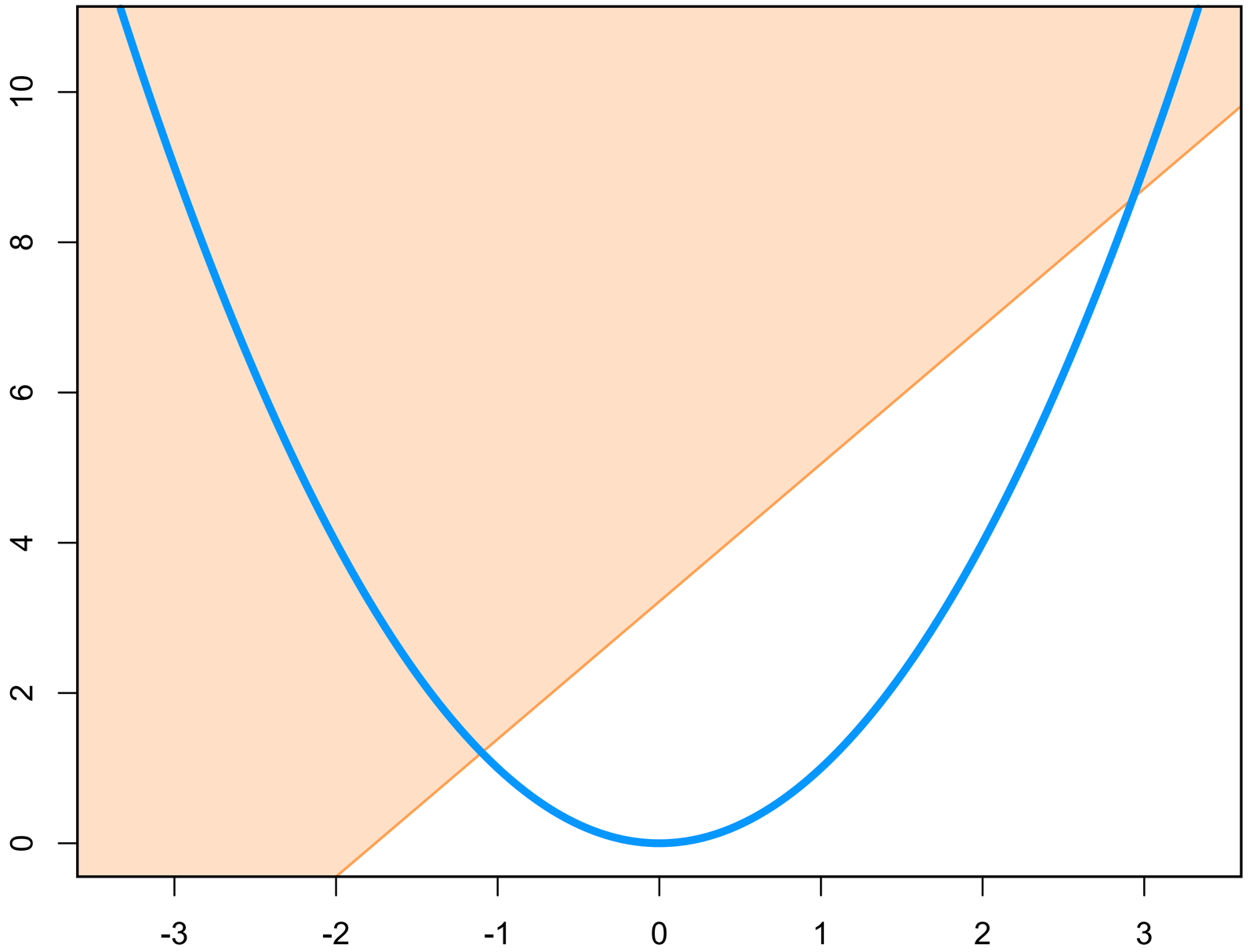

0
2
4
6
8
10
-3
-2
-1
0
1
2
3

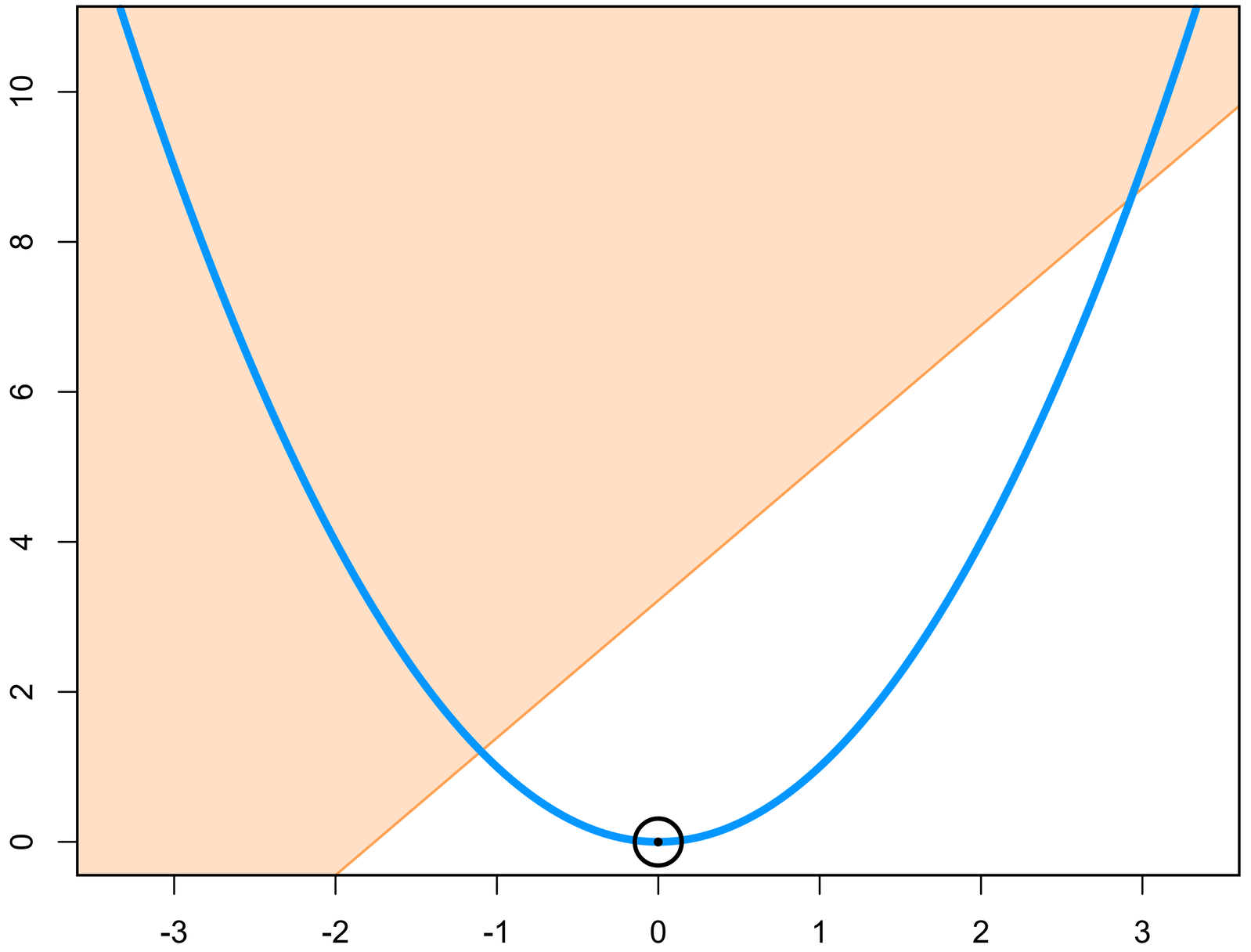

0
2
4
6
8
10
-3
-2
-1
0
1
2
3

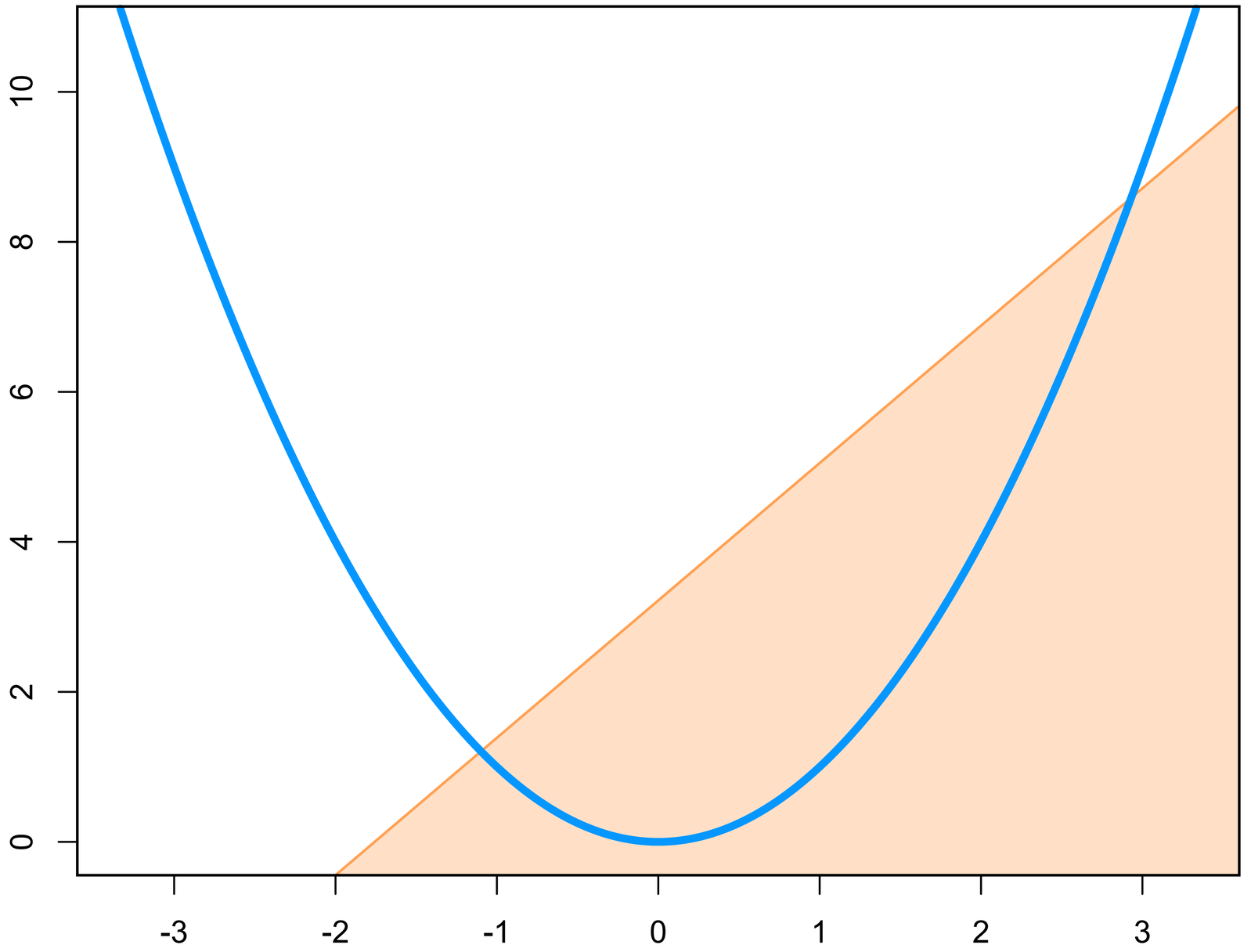

0
2
4
6
8
10
-3
-2
-1
0
1
2
3

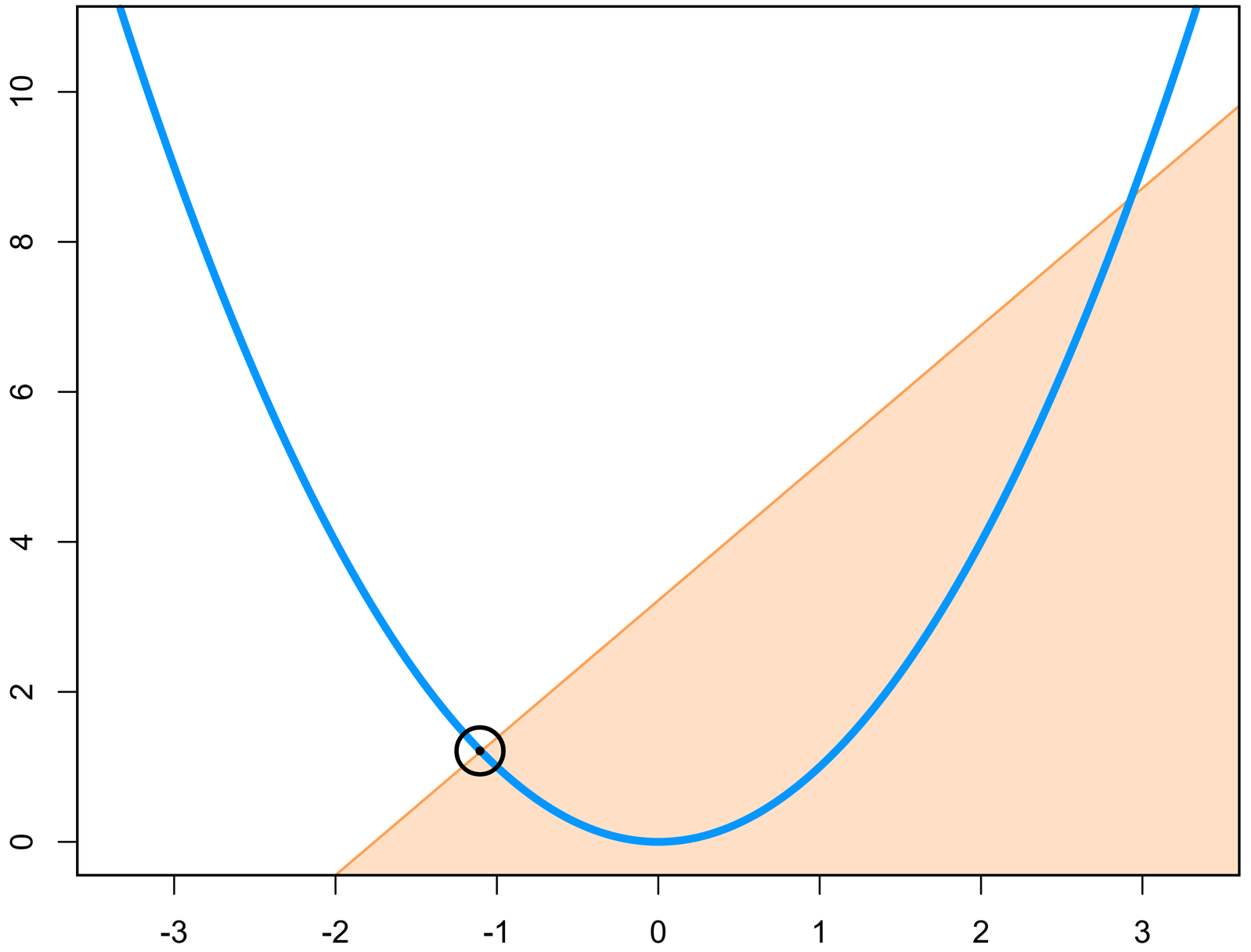

0
2
4
6
8
10
-3
-2
-1
0
1
2
3


0
2
4
6
8
10
-3
-2
-1
0
1
2
3


0
2
4
6
8
10
-3
-2
-1
0
1
2
3


10
8
6
4
2
0
-3
-2
-1
0
1
2
3


0
2
4
6
8
10
-3
-2
-1
0
1
2
3


10
8
6
4
2
0
-3
-2
-1
0
1
2
3

Lagrange multipliers

- Classical approach to constrained optimisation rolls equality constraints into objective via Lagrange multipliers, solving for stationary points of combined expression:

$$\begin{aligned} &\text{maximise} \quad f(x) \\ &\text{subject to:} \quad g(x) = 0 \end{aligned} \qquad \longrightarrow \qquad \mathcal{L}(x, \alpha) = f(x) - \alpha g(x)$$

- Extends to inequalities via Karush-Kuhn-Tucker approach
- One notable condition (as already seen in our trivial example): inequality constraints are only active at equality

Constrained optimisation special cases

- There are specific classes of (convex) constrained optimisation for which efficient off-the-peg numerical solvers exist, including:
  - → Linear Programming: linear objective, linear constraints
    - ▸ see eg scipy.optimize.linprog
  - → Quadratic Programming: quadratic objective, linear constraints
    - ▸ mentioned off-hand in connection with lasso
    - ▸ the classic approach to SVMs
    - ▸ see eg cvxpy
- We are not going to go into details (see e.g. Boyd & Vandenberghe), we're just going to take for granted that this works

Back to the margins

- Unfortunately, this optimisation is not in a tractable form

$$\underset{\mathbf{w},b}{\text{maximise}} \;\; M$$

$$\text{subject to:}$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq M$$

$$\|\mathbf{w}\| = 1$$

- We can merge the two constraints into one by explicitly dividing out the magnitude of the weights:

$$\frac{y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right)}{\|\mathbf{w}\|} \geq M$$

- Equivalently:

$$y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) \geq M \|\mathbf{w}\|$$

- This must hold for any magnitude of **w**, so we can arbitrarily require that

$$\|\mathbf{w}\| = \frac{1}{M}$$

  → This also entails a rescaling of *b*, but we don't need to care about that

- In which case, we can eliminate *M* entirely and reframe the maximisation purely in terms of **w**

$$\underset{\mathbf{w},b}{\text{maximise}} \quad \frac{1}{\|\mathbf{w}\|}$$

$$\text{subject to:} \quad y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) \geq 1$$

$$\underset{\mathbf{w},b}{\text{maximise}} \quad \frac{1}{\|\mathbf{w}\|}$$

$$\text{subject to:} \quad y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) \geq 1$$

Magnitude can't be negative, so maximising this reciprocal is equivalent to *minimising* $\|\mathbf{w}\|$

$$\underset{\mathbf{w},b}{\text{minimise}} \quad \|\mathbf{w}\|$$

$$\text{subject to:} \quad y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) \geq 1$$

$$\underset{\mathbf{w},b}{\text{minimise}} \quad ||\mathbf{w}||$$

As in previous cases, we can also arbitrarily rescale and apply other transformations

$$\text{subject to:} \quad y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) \geq 1$$

$$\underset{\mathbf{w},b}{\text{minimise}} \quad \frac{1}{2} \|\mathbf{w}\|^2$$

$$\text{subject to:} \quad y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) \geq 1$$

Are we there yet?

- This is a quadratic objective with linear constraints, ie a quadratic program
- It would be possible at this point to pass it to a QP solver and forget it
- Indeed we will eventually do that, but we’re going to go a bit further in order to discover some important properties of the solution

- Combining constraints into objective with Lagrange multipliers gives the primal optimisation objective, to be minimised with respect to **w** and *b*:

$$L_p = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_i^n \alpha_i \left(y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \right)$$

- Differentiate with respect to **w**:

$$\nabla_{\mathbf{w}} L_p = \mathbf{w} - \sum_{i}^{n} \alpha_i y_i \mathbf{x}_i$$

- Set to zero and rearrange:

$$\mathbf{w} = \sum_{i}^{n} \alpha_i y_i \mathbf{x}_i$$

- Differentiate with respect to *b*:

$$\frac{\partial L_p}{\partial b} = -\sum_i^n \alpha_i y_i$$

- Again, set to zero and rearrange:

$$\sum_i^n \alpha_i y_i = 0$$

- Substitute these conditions back into the primal objective:

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i^n \alpha_i \left(y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \right)$$

- Substitute these conditions back into the primal objective:

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i^n \alpha_i \left(y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \right)$$

$$= \frac{1}{2} \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right)$$

- Substitute these conditions back into the primal objective:

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i^n \alpha_i \left(y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \right)$$

$$= \frac{1}{2} \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right)$$

- Substitute these conditions back into the primal objective:

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i^n \alpha_i \left(y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \right)$$

$$= \frac{1}{2} \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - b \sum_i^n \alpha_i y_i$$

- Substitute these conditions back into the primal objective:

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i^n \alpha_i \left(y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \right)$$

$$= \frac{1}{2} \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - b \sum_i^n \alpha_i y_i + \sum_i^n \alpha_i$$

- Substitute these conditions back into the primal objective:

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i^n \alpha_i \left(y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \right)$$

$$= \frac{1}{2} \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - b \sum_i^n \alpha_i y_i + \sum_i^n \alpha_i$$

$$= \sum_i^n \alpha_i$$

- Substitute these conditions back into the primal objective:

$$
\begin{aligned}
L_p &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \alpha_i \left(y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \right) \\
&= \frac{1}{2} \left(\sum_i \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j \alpha_j y_j \mathbf{x}_j \right) - \left(\sum_i \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j \alpha_j y_j \mathbf{x}_j \right) - b \boxed{\sum_i^n \alpha_i y_i} + \sum_i^n \alpha_i \\
&= \sum_i^n \alpha_i
\end{aligned}
$$

This sum is 0, so this term disappears

- Substitute these conditions back into the primal objective:

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i^n \alpha_i \left(y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \right)$$

$$= \frac{1}{2} \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - b \sum_i^n \alpha_i y_i + \sum_i^n \alpha_i$$

$$= \sum_i^n \alpha_i - \frac{1}{2} \sum_i^n \sum_j^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

- Substitute these conditions back into the primal objective:

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i^n \alpha_i \left(y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \right)$$

$$= \frac{1}{2} \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - \left(\sum_i^n \alpha_i y_i \mathbf{x}_i \right) \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - b \sum_i^n \alpha_i y_i + \sum_i^n \alpha_i$$

$$L_d = \sum_i^n \alpha_i - \frac{1}{2} \sum_i^n \sum_j^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

- This is the dual objective, in terms of only the multipliers α
- This reciprocal problem is maximised at our desired solution point

$$\underset{\alpha}{\text{maximise}} \sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

But why?

- After all that, we still have a quadratic programming problem to solve in order to find the coefficients α so that we can then go on to compute the **w** and b, which is what we actually want
- However, if we look back at the elements of the solution, we can identify three important properties

Important property #1

- Optimal weights vector is in the span of the training set
- It's a weighted sum of (some of) the input samples:

$$\mathbf{w} = \sum_{i}^{n} \alpha_i y_i \mathbf{x}_i$$

- This is an instance of something known as the Representer Theorem

Important property #2

- Fitting depends only on the inputs in form of pairwise inner products

$$\underset{\alpha}{\text{maximise}} \sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} \alpha_i \alpha_j y_i y_j \boxed{(\mathbf{x}_i \cdot \mathbf{x}_j)}$$

- Substituting our weighted sum for **w** back into the prediction function, we find that it too depends only on these pairwise inner products

$$\hat{y} = \begin{cases} 1 & \text{if } \sum_i^n \alpha_i y_i \boxed{(\mathbf{x}_i \cdot \mathbf{x})} + b \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Important property #3

- Not so obvious, but one of the Karush-Kuhn-Tucker conditions for solution of the dual optimisation problem is:

$$\alpha_i \left[y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \right] = 0 \quad \forall i$$

- Inequality constraints are only active at equality
- Coefficients α are only non-zero when **x** is exactly on the margin
- So solution depends only on a small subset of the training samples
- These are known as support vectors

4.4: Soft Margin

COMP0088 Introduction to Machine Learning · UCL Computer Science

Slacking off

- Not all that useful if linear separability demanded
- We can relax that requirement by introducing slack variables
- These give the optimiser an option to trade off too-small (including negative) margins for better overall fitting
- Each training sample $\mathbf{x}_i$ has its own slack variable ξ_i
- There is a limited ξ budget that the optimiser can spend to increase margin:

$$\sum_i \xi_i \leq \text{ constant}$$

- Can be posed as a maximisation of M, just like before:

$$\underset{\mathbf{w},b,\xi}{\text{maximise}} \;\; M$$

$$\text{subject to:}$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq M(1 - \xi_i)$$

$$\|\mathbf{w}\| = 1$$

$$\xi_i \geq 0 \;\; \forall i$$

$$\sum_i \xi_i \leq \;\text{constant}$$

- And similarly reframed as a minimisation w.r.t. **w**:

$$\underset{\mathbf{w},b,\xi}{\text{minimise}}\ \frac{1}{2}\left\|\mathbf{w}\right\|^2 + C\sum_{i}^{n}\xi_i$$

$$\text{subject to:}$$

$$y_i\left(\mathbf{x}_i \cdot \mathbf{w} + b\right) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

- Lagrange multiplier formulation of primal problem is similar, albeit longer
  - → We now have two sets of multipliers for the two sets of constraints, α for the margins and μ for the slack budget

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i^n \xi_i - \sum_i^n \alpha_i \left[y_i\left(\mathbf{x}_i \cdot \mathbf{w} + b\right) - \left(1 - \xi_i\right)\right] - \sum_i^n \mu_i \xi_i$$

- When we differentiate and substitute to get the dual problem in terms of α, the objective turns out to look exactly the same:

$$L_d = \sum_i^n \alpha_i - \frac{1}{2} \sum_i^n \sum_j^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

- But there's an additional constraint on the optimisation:

$$0 \leq \alpha_i \leq C$$

- The KKT conditions now also encompass the slack variables ξ_i

$$\alpha_i \left[y_i(\mathbf{x}_i \cdot \mathbf{w}) - (1 - \xi_i) \right] = 0$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w}) - (1 - \xi_i) \geq 0$$

What does this mean?

- Hard margin properties continue to apply
  - → **w** is a weighted sum of training samples
  - → inputs only enter as inner products
  - → many coefficients may be zero
- But:
  - → support vectors include those within margins as well as those exactly on them

Relaxation as regularisation

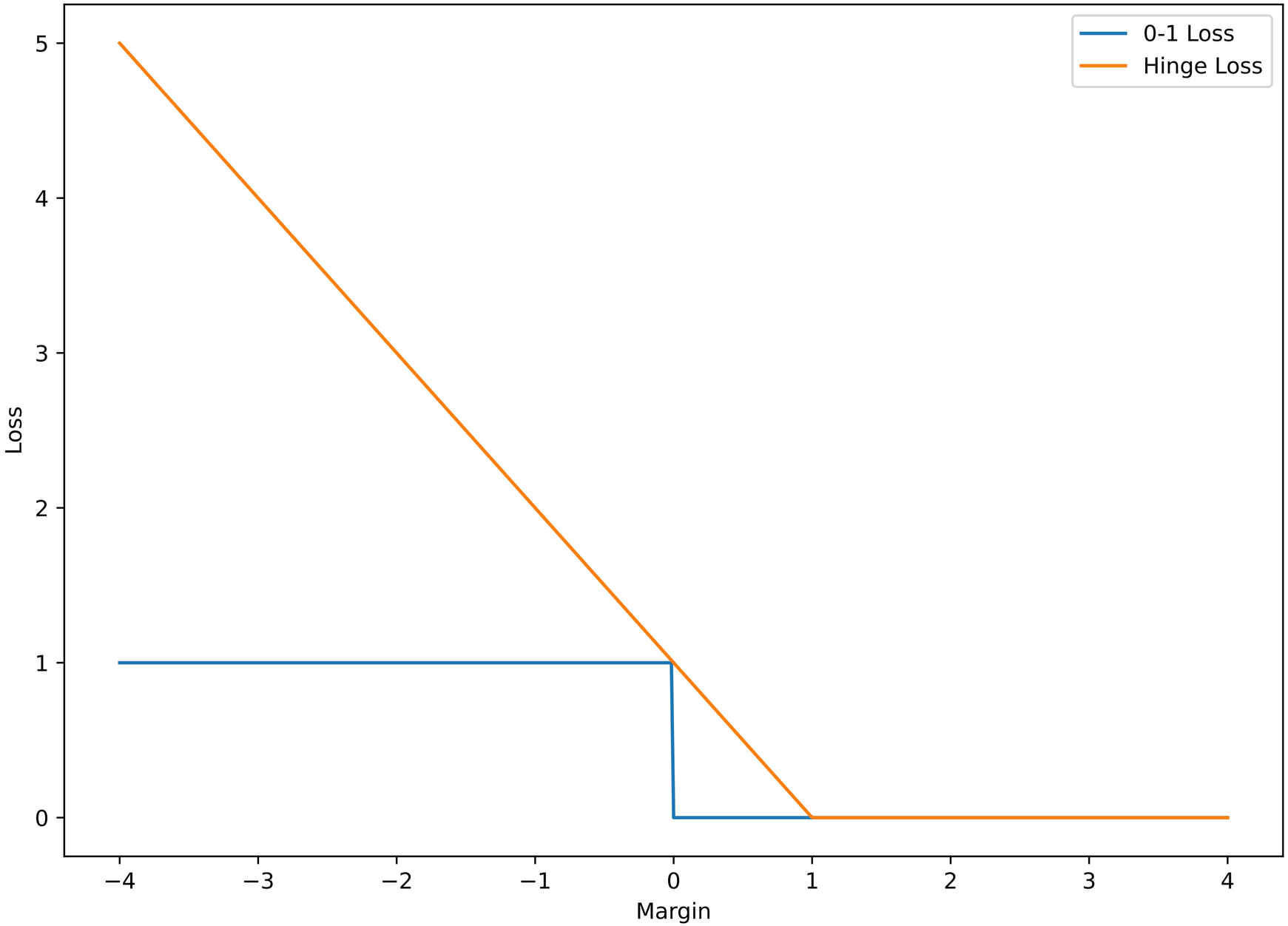
- The cost parameter *C* looks a lot like the regularisation parameter λ in ridge & lasso…
- …and indeed it has a similar regularising effect
- *C* imposes an upper bound on α, thereby preventing the model from paying too much attention to any individual training sample
- So it reduces susceptibility to outliers
  - → If *C* is very high, each individual has more influence and there's scope for overfitting
  - → If *C* is low, a more consensus boundary is fitted, potentially underfitting
- Note that *C* and λ act in opposite (reciprocal) directions

Hinge loss & gradient descent

- Margin optimisation can also be expressed as a loss minimisation problem
- The loss function is known as the hinge loss:

$$\begin{aligned} L(y, \mathbf{x}) &= \max(0, 1 - yf(\mathbf{x})) \\ &= \max(0, 1 - y(\mathbf{x} \cdot \mathbf{w} + b)) \end{aligned}$$

- Not differentiable at 1, but it is easy to define a subgradient
- Convex and optimisable by gradient descent


0-1 Loss
Hinge Loss
Loss
Margin
5
4
3
2
1
0
−4
−3
−2
−1
0
1
2
3
4

Soft margin QP ≈ hinge optimisation

- Minimisation of hinge loss is automatically “soft margin” — it does not assume loss can be driven to 0
- Combination of hinge loss and ridge regularisation is really just a rearrangement of the soft margin problem:
  - → Slack variables correspond to hinge loss
  - → L_2 penalty corresponds to $\|\mathbf{w}\|$ minimisation
- But gradient descent optimisation of hinge loss does not have the property that inputs can be reduced to pairwise inner products

4.5: The Kernel Trick

COMP0088 Introduction to Machine Learning · UCL Computer Science

Inner products compare samples

- For both the hard and soft margin classifiers, and for both fitting and prediction, the input samples appear only in the form of inner products between pairs
- These serve as a kind of similarity metric, looking at how close together the samples are
- Loosely: the classification boundary depends only on the subset of samples that are most similar given their opposing labels
- The same criteria applies under any basis expansion: fitting depends on similarity in the transformed space, as captured by the inner product

Kernel functions

- Suppose we have some basis expansion

$$\mathbf{x}' = \phi(\mathbf{x})$$

- If we can find some function *K* (known as a kernel) such that for any **u**, **v**

$$K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u}) \cdot \phi(\mathbf{v})$$

- Then we can use that instead of the dot product terms in the fitting & prediction functions

Kernelised versions

- Fitting:

$$L_d = \sum_i^n \alpha_i - \frac{1}{2} \sum_i^n \sum_j^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

- Prediction:

$$\hat{y} = \begin{cases} 1 & \text{if } \sum_i^n \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

But why?

- *K* is a scalar function

$$K : (\mathbb{R}^d, \mathbb{R}^d) \mapsto \mathbb{R}$$

- phi is a vector function

$$\phi : \mathbb{R}^d \mapsto \mathbb{R}^m$$

- Potentially $m \gg d$ plus you still have to compute the inner product
- So there could be a big computational saving

It turns out…

- Given we're discussing this at all, it's probably no surprise that there do turn out to be useful kernels
- Direct calculation of inner product bypasses very difficult or impossible transformation into huge (even infinite) dimensions
- This kernel trick makes it possible to fit classification boundary efficiently in a space that would otherwise be computationally intractable
- We'll look at a couple of popular and important cases

Polynomial kernel

- The kernel function

$$K(\mathbf{u}, \mathbf{v}) = (1 + \mathbf{u} \cdot \mathbf{v})^p$$

- is equivalent to expanding to a basis of all monomial combinations of the features in each vector up to degree *p* (with some scaling coefficients on the outputs)

Polynomial kernel example

- *d*=2, *p*=2

$$K(\mathbf{u}, \mathbf{v}) = (1 + u_1 v_1 + u_2 v_2)^2$$

Polynomial kernel example

- *d*=2, *p*=2

$$K(\mathbf{u}, \mathbf{v}) = (1 + u_1 v_1 + u_2 v_2)^2$$
$$= 1 + 2u_1 v_1 + 2u_2 v_2 + (u_1 v_1)^2 + (u_2 v_2)^2 + 2u_1 u_2 v_1 v_2$$

Polynomial kernel example

- *d*=2, *p*=2

$$
\begin{aligned}
K(\mathbf{u}, \mathbf{v}) &= (1 + u_1 v_1 + u_2 v_2)^2 \\
&= 1 + 2u_1 v_1 + 2u_2 v_2 + (u_1 v_1)^2 + (u_2 v_2)^2 + 2u_1 u_2 v_1 v_2 \\
&= [1, \sqrt{2}u_1, \sqrt{2}u_2, u_1^2, u_2^2, \sqrt{2}u_1 u_2] \cdot [1, \sqrt{2}v_1, \sqrt{2}v_2, v_1^2, v_2^2, \sqrt{2}v_1 v_2]
\end{aligned}
$$

Polynomial kernel example

- *d*=2, *p*=2

$$
\begin{aligned}
K(\mathbf{u}, \mathbf{v}) &= (1 + u_1 v_1 + u_2 v_2)^2 \\
&= 1 + 2u_1 v_1 + 2u_2 v_2 + (u_1 v_1)^2 + (u_2 v_2)^2 + 2u_1 u_2 v_1 v_2 \\
&= [1, \sqrt{2}u_1, \sqrt{2}u_2, u_1^2, u_2^2, \sqrt{2}u_1 u_2] \cdot [1, \sqrt{2}v_1, \sqrt{2}v_2, v_1^2, v_2^2, \sqrt{2}v_1 v_2]
\end{aligned}
$$

$$
\phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2]
$$

- Similar expansion holds for any d, p
- K is roughly O(d)
- But phi is roughly O(d^p)
- So computational saving could be pretty huge

Radial basis (aka Gaussian) kernel

- The kernel function

$$K(\mathbf{u}, \mathbf{v}) = e^{-\gamma \|\mathbf{u}-\mathbf{v}\|^2} = e^{-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}}$$

  → Two common parameterisations, where $\gamma = \frac{1}{2\sigma^2}$

- is equivalent to an infinite monomial expansion (with infinitely shrinking coefficients)
- Applying phi explicitly is impossible, but *K* is simple and compact

Radial basis (aka Gaussian) kernel

- Best understood as a similarity metric
  - → If **u** and **v** are close, K will be close to 1
  - → If **u** and **v** are very different, K will be close to 0
- Hyperparameter γ (or σ) governs the steepness of the drop off with distance

Can we use *any* similarity metric?

- A kernel is only valid if this relation actually holds

$$K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u}) \cdot \phi(\mathbf{v})$$

- *K* must actually define an inner product in some space
- For this to be true, the Gram matrix must be symmetric positive semidefinite

Gram (kernel) matrix

- For an ordinary dot product (aka linear kernel) and given our row-wise design matrix ordering, the Gram matrix is just $\mathbf{X}\mathbf{X}^\mathsf{T}$
- For the generalised inner product represented by the kernel, it's the matrix of all pairwise applications of K to the data:

$$\mathbf{K} = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_n) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_n, \mathbf{x}_1) & K(\mathbf{x}_n, \mathbf{x}_2) & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

Mercer's Condition

- *K* is a valid kernel if and only if **K** is symmetric positive semidefinite
- Specific application of more general Mercer's Theorem
- If this doesn't hold, QP problem to fit boundary will not converge
- Strictly, validity of *K* requires that **K** always be PSD for any data
- In practice fitting can converge if **K** is conditionally PSD
  - → ie, the **K** resulting from your dataset is PSD, even if others might not be

Non-numeric “inner products”

- As well as computational efficiency, kernels allow inner products to be defined for other kinds of data
  - → Text
  - → Sequences
  - → Graphs
- This effectively allows learning algorithms that support kernels to be used directly on the data without first embedding it in some numeric space
  - → An embedding is conceptually there, but the kernel bypasses the actual calculation

Life beyond SVMs

- Kernels can be used anywhere an algorithm can be structured to depend only on inner products
- This is true of many of the learning algorithms we've looked at so far
  - → Ridge regression
  - → Logistic regression
  - → k-Nearest Neighbours
  - → Perceptron
- …and others we haven't seen yet like principal components analysis
- But SVMs are probably the "canonical" kernel method

4.6: Support Vector Machines

COMP0088 Introduction to Machine Learning · UCL Computer Science

You've already paid for this

- SVMs are the combination of soft margin maximisation and kernels
- So in a sense there's nothing left to say here!

- But let's review…

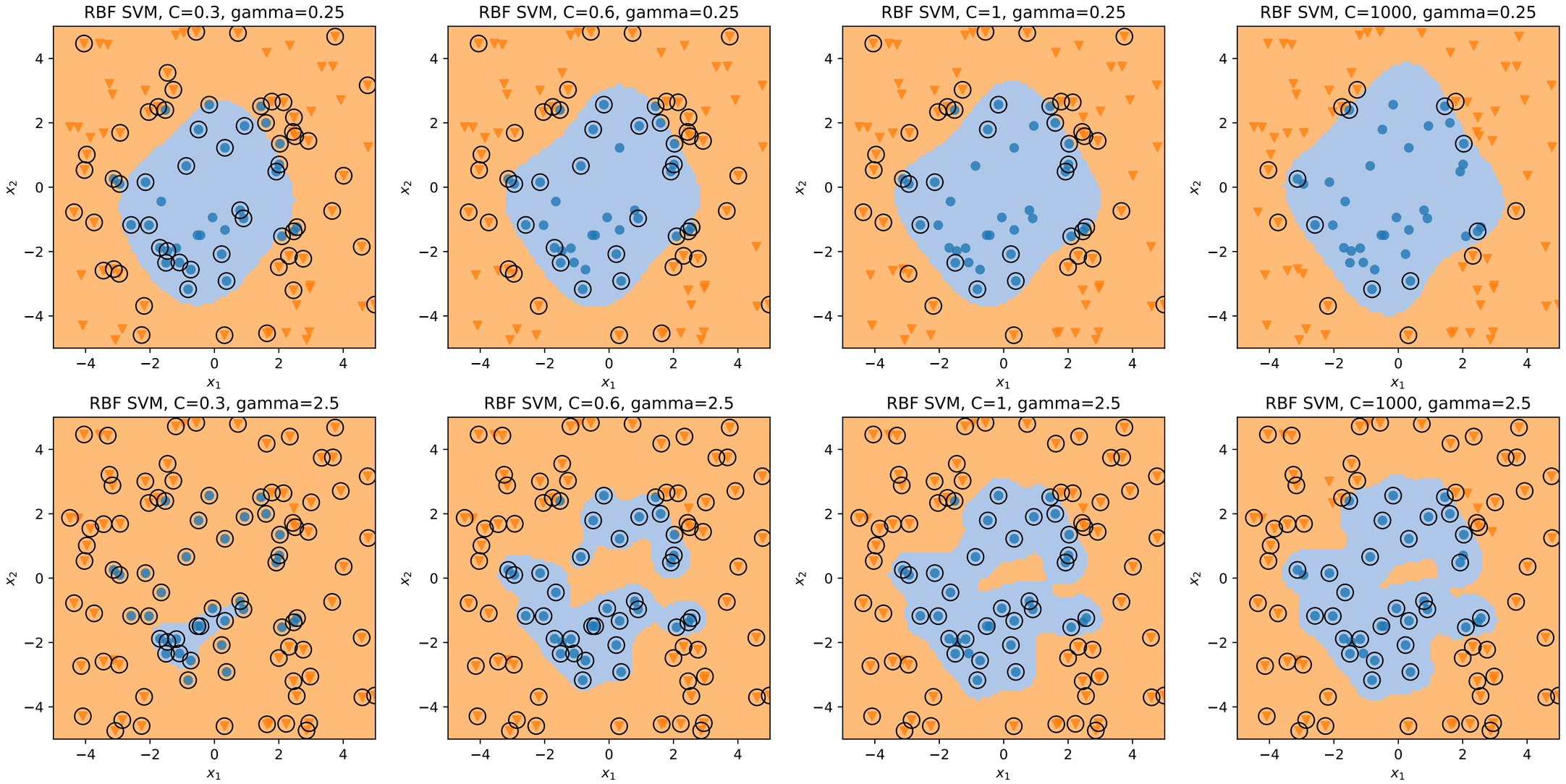
Background

- Mostly invented in 1960s in Soviet Union but remained obscure
- Published after Vapnik moved to US in the early 1990s
- Rose to prominence with notable successes, eg in computer vision
- Soon became the go-to supervised learning algorithm for mid-sized data
- Fashion subsequently moved on a little to Random Forests and then quite a lot more with the rise of Deep Learning
- But SVMs still very useful and applicable

Linear vs non-linear

- Even though they use the same approach, linear and kernelised SVMs are significantly different in effect
- Linear SVM is just a linear model with slightly different ideas about what constitutes the right hyperplane
  - → It's clear what it means, the functional form is explicit, its over/underfitting behaviour may be a bit different according to data distribution but it's similarly interpretable
- RBF/poly/etc is much more black box and implicit
  - → Can fit almost arbitrarily complex boundary, but the meaningful criteria for doing so in very high or infinite dimensional spaces are largely opaque
  - → Interpretation is: search me, it just empirically fits

Decision boundary examples



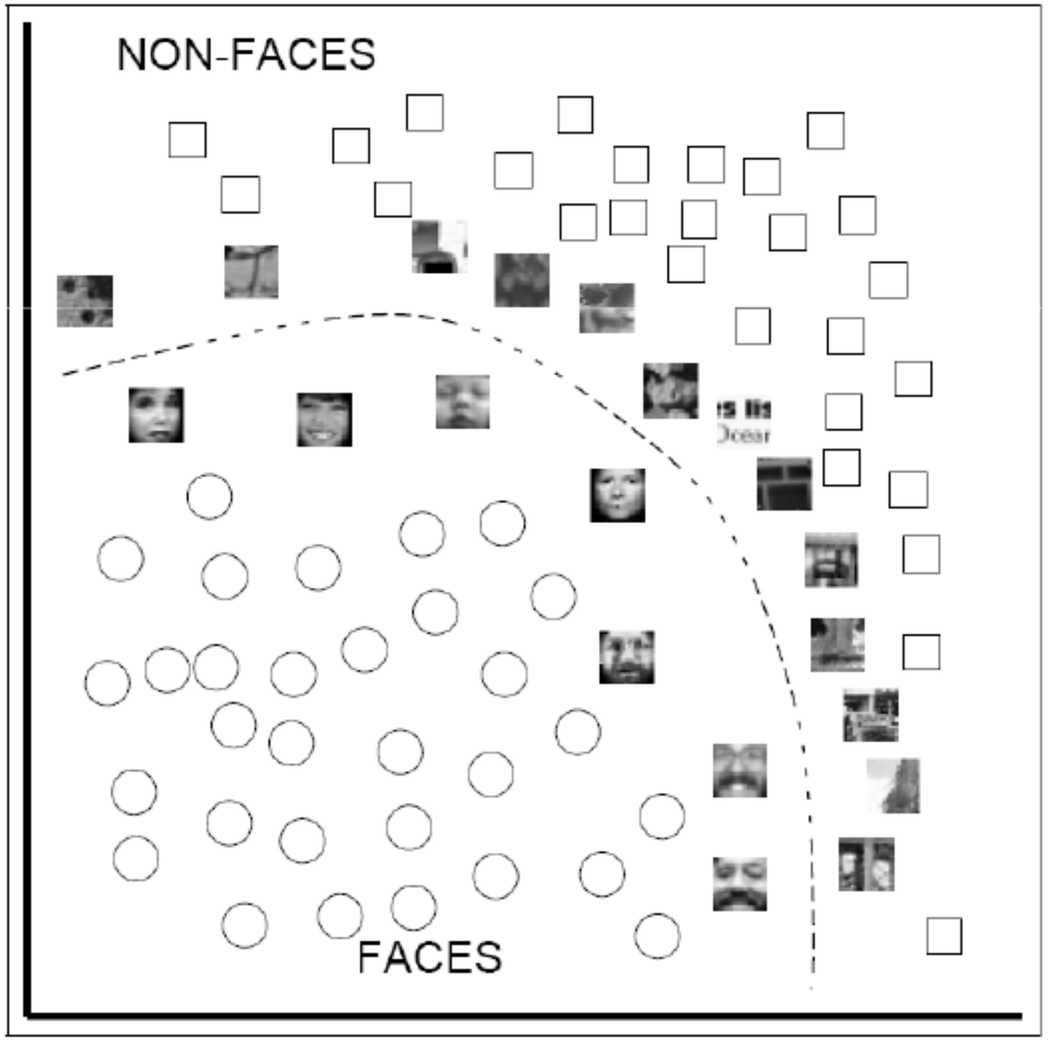
Handwritten digits

- MNIST: Modified National Institute of Standards and Technology — database of handwritten digits published by Yann LeCun in 1998 and very widely used as a computer vision testbed
- SVMs proved surprisingly successful on this task
  - → Strictly, the early SVM successes predate publication of MNIST, but were done using the same USPS source images that became MNIST
- The model has no CV knowledge or spatial structure, it just sort of works on the raw (or somewhat processed) data vectors
- This is an example of how kernel fitting liberates basis expansion from the realm of explicit domain knowledge into something purely data-driven

Other vision tasks

- SVMs had successes with other vision-related tasks in the 90s-10s
  - → Subsequently losing ground to CNNs
- Often not using the raw pixel vectors but some other explicit feature transformation, which might encode some prior CV knowledge
  - → wavelet coefficients
  - → histogram of gradients
- In such an application, feature extraction takes the role of basis specification

Faces



People


B
www.corbis.com


H
www.corbis.com

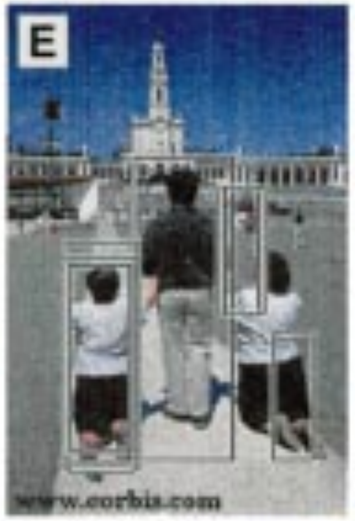

I


J


K


L


M

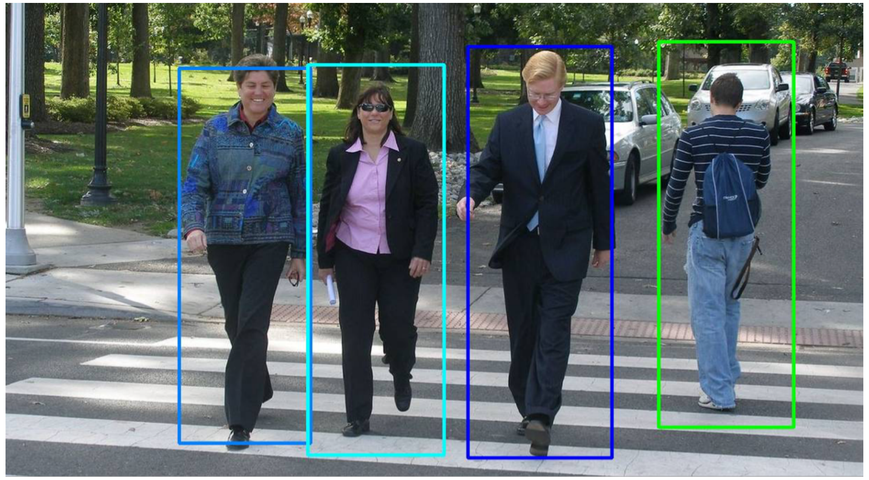

E
www.corbis.com

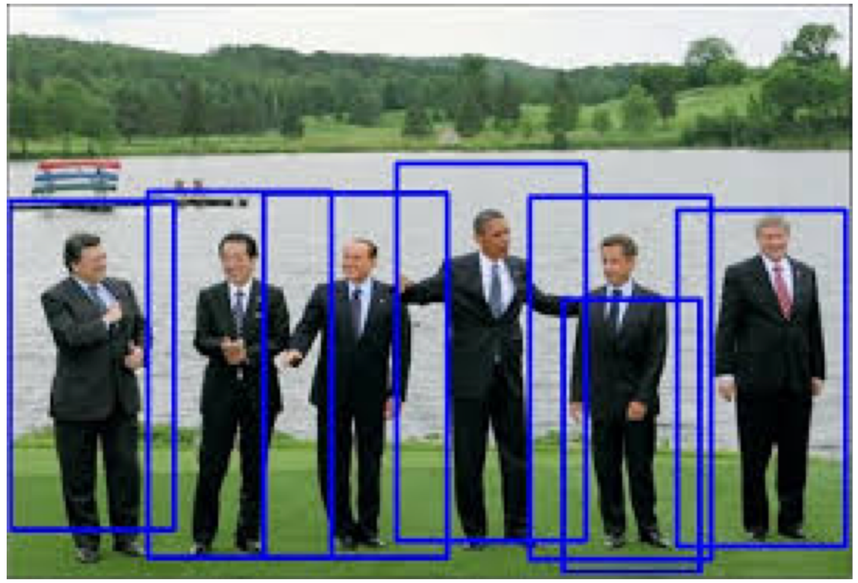

F
www.corbis.com

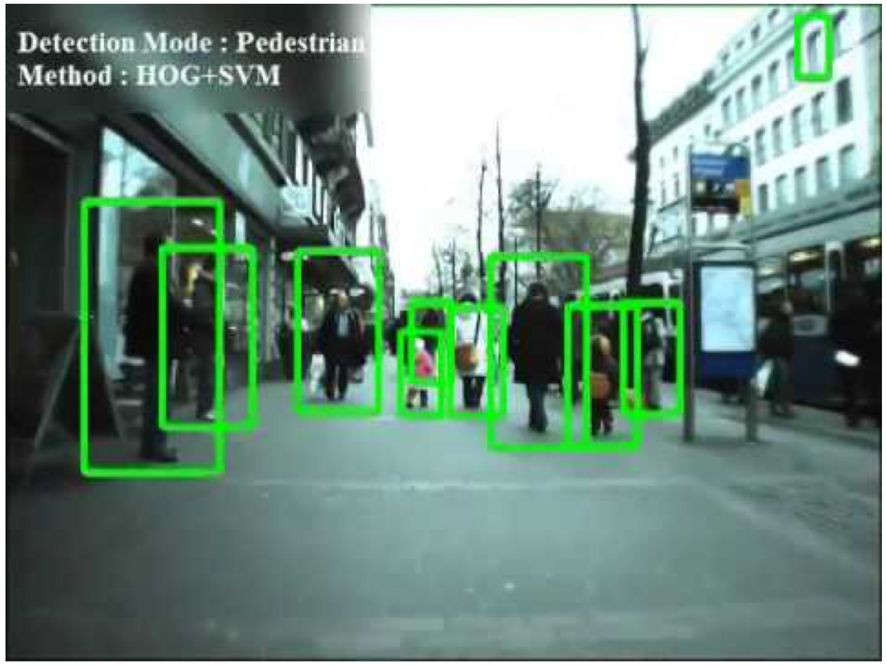

N
www.corbis.com

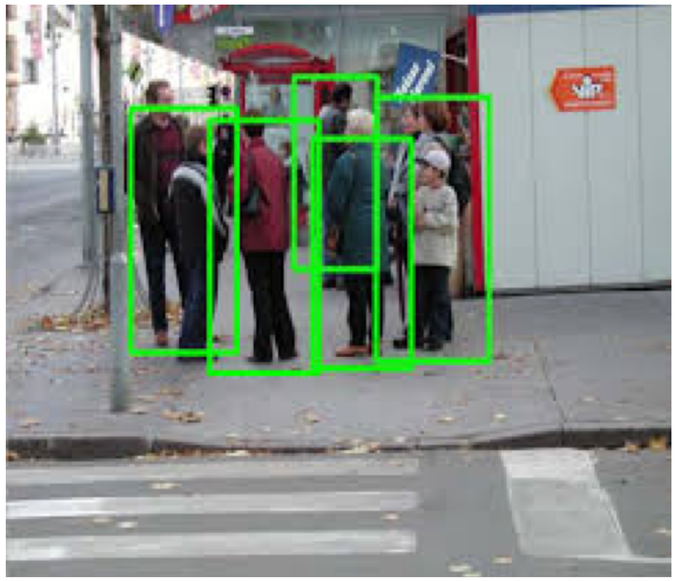

O


P






Detection Mode : Pedestrian
Method : HOG+SVM



Usage notes

- High dimensional data is more likely (though not guaranteed) to be approximately separable anyway, so projection into even higher or infinite dimensional basis tends not to be as helpful
- Focus on support vectors means that possibly undue weight is given to edge cases, so beware of overfitting
- High degree polynomial and RBF kernels are both able to fit almost arbitrarily complex boundaries, so beware of overfitting
- Fitting is highly sensitive to cost hyperparam C and also kernel hyperparams such as gamma/sigma or the polynomial degree
- May be slow for large datasets