



Lecture 4

From the Perceptron to the Support Vector Classifier

Gabriele Pergola and Amina Asif

Machine Learning Algorithms and Practice
CS429

October 2026

Part 1: The Perceptron and Its Limits

What should a learning algorithm penalise? From the 0–1 loss to the perceptron loss and algorithm, and the question it leaves open.

Part 1
Perceptron

Part 2
Maximal margin classifier

Part 3
Support vector classifier

Part 4
Regularisation

Next lecture
Support vector machine
(Lecture 5)

0-1 Loss in Practice

$$l(f(x_i), y_i) = \begin{cases} 0, & y_i f(x_i) > 0 \\ 1, & y_i f(x_i) \leq 0 \end{cases}$$

Suppose the current linear classifier is

$$f(\mathbf{x}) = 2x_1 - x_2 + 0.4 \quad \text{so } \mathbf{w} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \quad \mathbf{b} = 0.4$$

First example

$$\mathbf{x}_1 = \begin{bmatrix} 1.2 \\ 0.5 \end{bmatrix}, \quad y_1 = +1$$

The classifier score

$$\begin{aligned} f(x_1) &= \mathbf{w}^T \mathbf{x}_1 + b \\ &= 2(1.2) - 1(0.5) + 0.4 \\ &= 2.4 - 0.5 + 0.4 \\ &= \mathbf{2.3} \end{aligned}$$

$$\text{Then: } y_1 f(x_1) = (+1)(+2.3) =$$

$$\mathbf{+2.3} > 0$$

correct classification $\Rightarrow l = 0$

The same classifier applied to four labelled examples

$\mathbf{x}_i = (x_1, x_2)$	y_i	$f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i + b$	$m_i = y_i f(\mathbf{x}_i)$	Correct?	l
(1.2, 0.5)	+1	$2(1.2) - 1(0.5) + 0.4 = 2.3$	$(+1)(+2.3) = +2.3$	Yes	0
(0.2, 1.5)	+1	$2(0.2) - 1(1.5) + 0.4 = -0.7$	$(+1)(-0.7) = -0.7$	No	1
(0.4, 3.0)	-1	$2(0.4) - 1(3.0) + 0.4 = -1.8$	$(-1)(-1.8) = +1.8$	Yes	0
(0.5, 0.9)	-1	$2(0.5) - 1(0.9) + 0.4 = 0.5$	$(-1)(+0.5) = -0.5$	No	1

Why Can't We Just Minimise 0-1 Loss?

We now have exactly the objective we care about, so why not optimise it directly?

$$l_{0/1} = (1/N) \sum_i \mathbb{1}[y_i f(x_i) \leq 0]$$

All mistakes receive exactly the same loss.

Flat almost everywhere

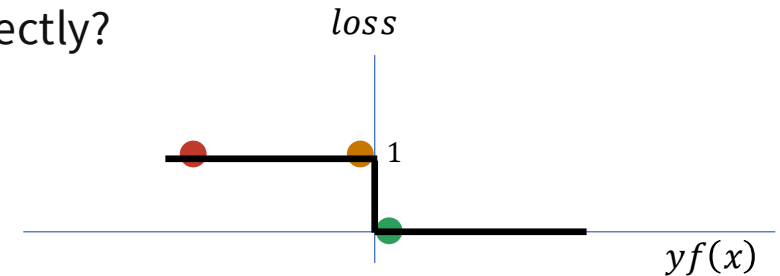
Small changes to w usually leave the loss unchanged.

Jump at the decision boundary

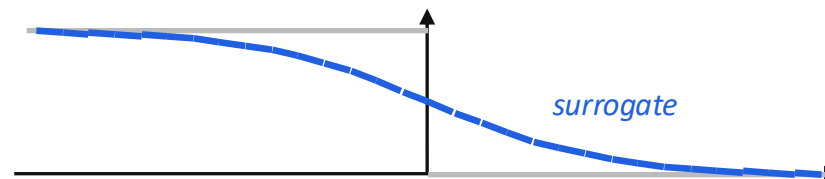
At $y_i f(x_i) = 0$ the loss jumps from 1 to 0, so it is **not differentiable** there.

No useful direction for gradient descent

Gradient descent does not know which small change reduces the loss.



- $yf(x) = -3$: badly wrong, $l = 1$
- $yf(x) = -0.1$: barely wrong, $l = 1$
- $yf(x) = +0.1$: barely correct, $l = 0$



Solution: use a **surrogate loss**.

It should give **larger penalties** to worse predictions, change **smoothly**, and be **easier** to optimise.

A Possible Surrogate: The Perceptron Criterion

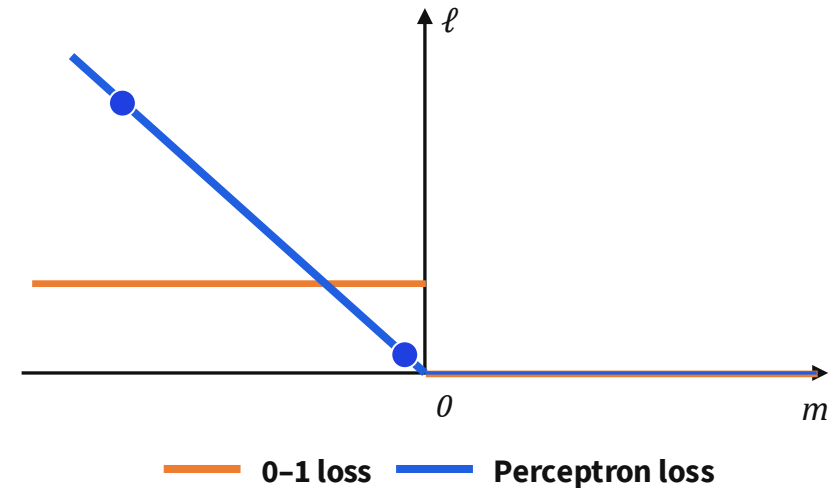
Main idea

Instead of giving every mistake the same loss of 1, penalise a misclassified example according to how far it is on the wrong side:

$$l_{\text{perc}}(y_i, f(x_i)) = \max(0, -y_i f(x_i))$$

$y_i f(x_i) > 0 \Rightarrow l_{\text{perc}} = 0$ correct side: no penalty
 $y_i f(x_i) < 0 \Rightarrow l_{\text{perc}} = -y_i f(x_i) > 0$ wrong side: larger mistakes receive larger penalties

Still: once $y_i f(x_i) > 0$, its loss is constantly zero. **Are all correct predictions equally good?**



Loss values for the four examples

$\mathbf{x}_i = (x_1, x_2)$	y_i	$f(x_i)$	$m_i = y_i f(x_i)$	Correct?	$\ell_{0/1}$	$\ell_{\text{perc}} = \max(0, -y_i f(x_i))$
(1.2, 0.5)	+1	+2.3	+2.3	Yes	0	$\max(0, -2.3) = 0$
(0.2, 1.5)	+1	-0.7	-0.7	No	1	$\max(0, +0.7) = 0.7$
(0.4, 3.0)	-1	-1.8	+1.8	Yes	0	$\max(0, -1.8) = 0$
(0.5, 0.9)	-1	+0.5	-0.5	No	1	$\max(0, +0.5) = 0.5$

The Perceptron Algorithm

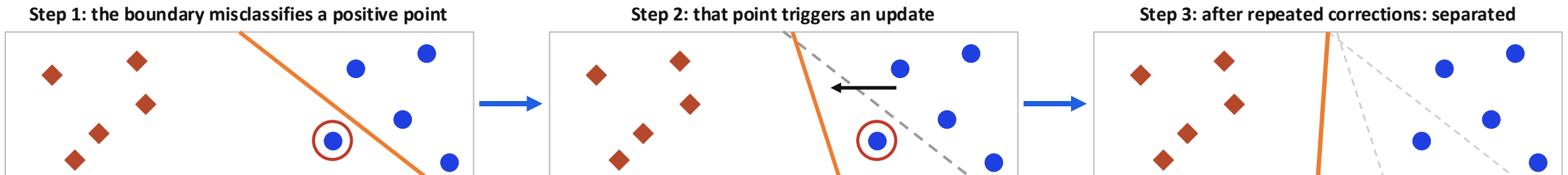
- Given:
 - Training examples: $\{(x_i, y_i) \mid i = 1 \dots N\}$, $y_i \in \{-1, +1\}$
 - Learning rate (step size): $\eta > 0$
- Initialise $w^{(0)} = 0$ and $b^{(0)} = 0$
- Until convergence ($k = 1 \dots K$ epochs, or no mistakes occur)
 - For $i = 1 \dots N$
 - Pick example x_i with label y_i
 - Compute $f(x_i) = w^{(t)\top} x_i + b^{(t)}$
 - If $y_i f(x_i) \leq 0$ (a mistake) then update w and b by gradient descent
$$w^{(t+1)} = w^{(t)} - \eta \nabla_w \ell_i(w^{(t)}) = w^{(t)} - \eta(-y_i x_i) = w^{(t)} + \eta y_i x_i$$
$$b^{(t+1)} = b^{(t)} - \eta \partial \ell_i / \partial b = b^{(t)} - \eta(-y_i) = b^{(t)} + \eta y_i$$
 - Check for convergence to stop (an epoch with no mistakes)

Why " ≤ 0 " rather than " < 0 "?

At the start $w = 0$ and $b = 0$, so every score is 0 and every perceptron loss is 0. Counting $m_i = 0$ as a mistake is what lets learning begin.

Indexing

t counts updates, not epochs: one epoch may contain several updates ($t \leftarrow t + 1$ after each).



The Perceptron Finds a Separator: but which one?

Suppose the training data are linearly separable. There may be many decision boundaries for which the training error is zero.

$$f_A(x) = 0, \quad f_B(x) = 0, \quad f_C(x) = 0$$

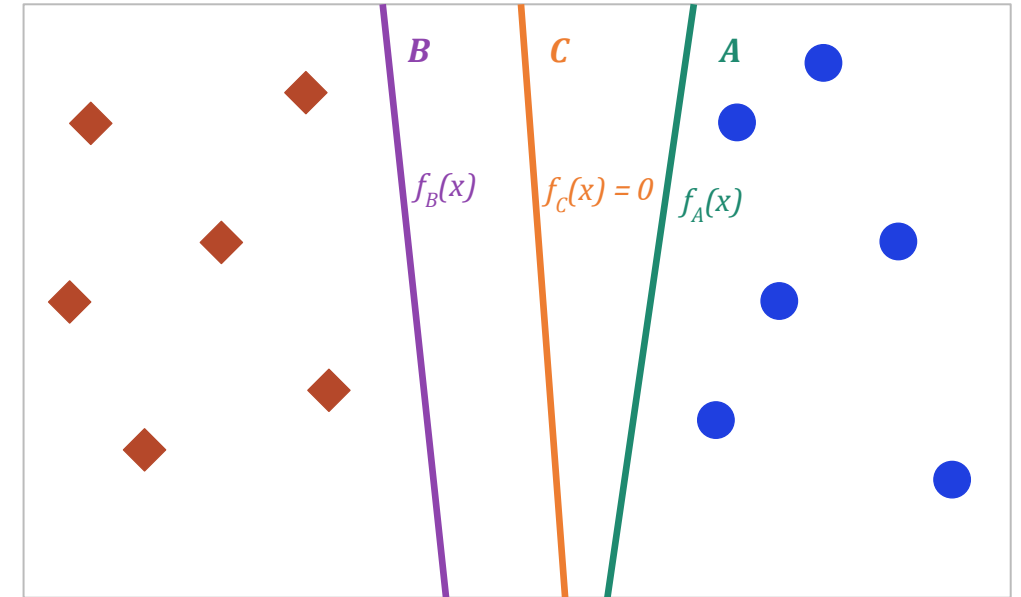
All three correctly classify every training example.

If they all make zero training errors, are they equally good?

- 0-1 error cannot distinguish between them.

Perceptron Convergence Theorem

If the training data are linearly separable, the perceptron [will find a separating hyperplane](#) after a finite number of updates.



The perceptron convergence theorem tells us:

a separating hyperplane will be found

but it does not tell us:

which separating hyperplane should be preferred

Can we prefer a separator that not only classifies the training data correctly, but also stays as far as possible from the training examples?

Margin and the maximal margin classifier

Part 2: The Maximal Margin Classifier

Linearly separable data. Measure the distance to the boundary, choose the widest separation, and turn it into a tractable problem.

Part 1

Perceptron

Part 2

**Maximal margin
classifier**

Part 3

Support vector classifier

Part 4

Regularisation

Next lecture

Support vector machine
(Lecture 5)

The **Functional Margin**: Combining Label and Score

Recall the linear score and binary labels:

$$f(x) = w^T x + b, \quad y_i \in \{-1, +1\}.$$

$$m_i = y_i f(x_i)$$

Interpretation

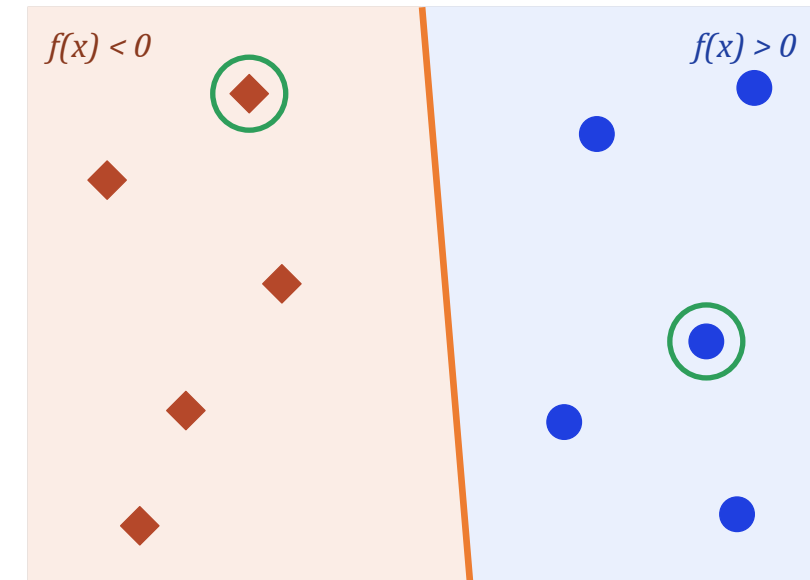
Functional margin	Interpretation
$m_i > 0$	Correctly classified
$m_i = 0$	On the decision boundary
$m_i < 0$	Incorrectly classified

At $m_i = 0$, the predicted class depends on the tie-breaking convention.

Why multiply by the label?

True label y_i	Score $f(x_i)$	Margin m_i
$+1$	$+2$	$+2$
-1	-2	$+2$

Both examples are correctly classified, despite having scores of opposite signs.

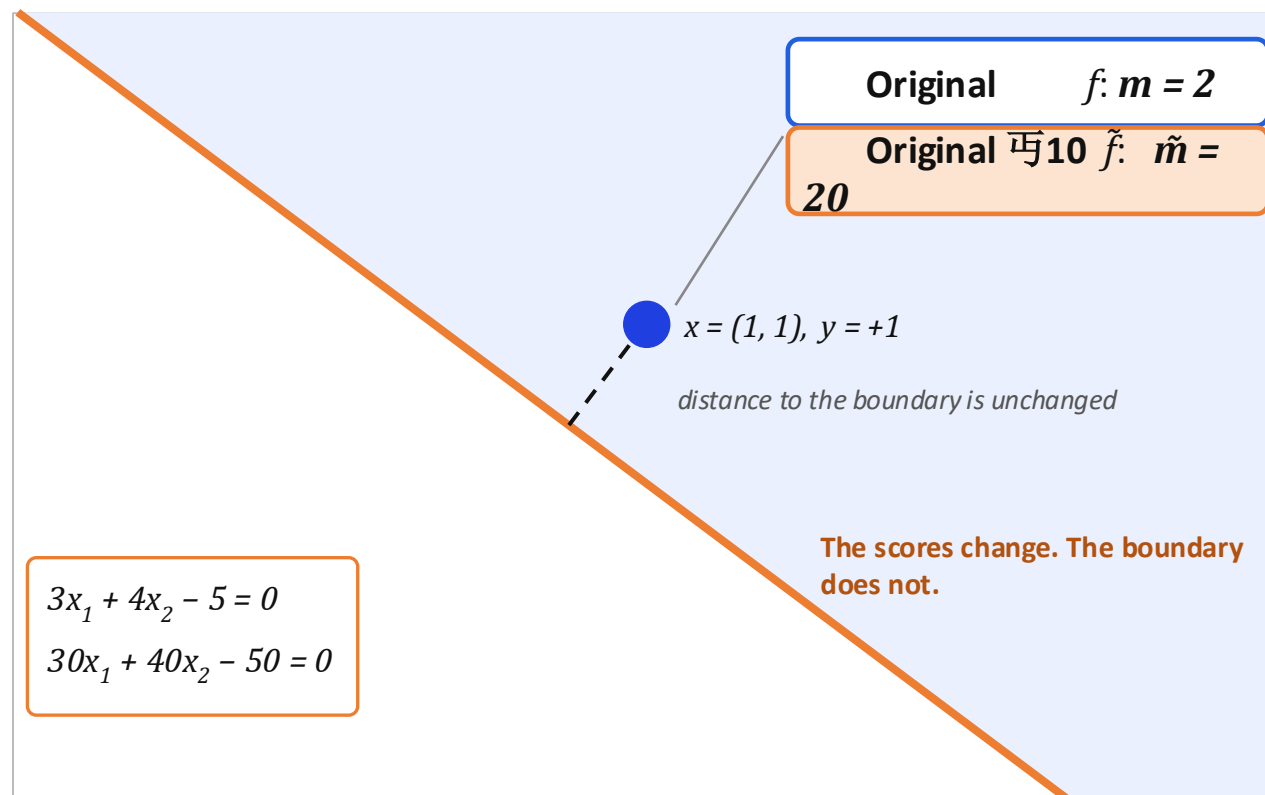


Circled: one correctly classified example from each class.

- The functional margin combines the true label with the model's score.
- Its sign indicates correctness. Its magnitude depends on the score scale.

Same Boundary, Different Scores

If a classifier gives a **larger score** to a correctly classified example, does that mean the example is **farther from the boundary**?



Original classifier

$$w = (3, 4), b = -5 \Rightarrow f(x) = 3x_1 + 4x_2 - 5$$

$$m = y f(x) = (+1)(3 + 4 - 5) = 2$$

Multiply both w and b by 10

$$\tilde{w} = (30, 40), \tilde{b} = -50 \Rightarrow \tilde{f}(x) = 30x_1 + 40x_2 - 50 = 10 f(x)$$

$$\tilde{m} = y \tilde{f}(x) = (+1)(30 + 40 - 50) = 20$$

In general, for any $c > 0$:

$$\tilde{w} = cw, \tilde{b} = cb \Rightarrow \tilde{f}(x) = c f(x) \Rightarrow$$

$$\tilde{m}_i = c m_i$$

So the margin of a correct example can be made arbitrarily large by increasing c ; the boundary and the predicted labels do not change.

Functional margin measures the signed score, not geometric distance. Its magnitude depends on how we scale the classifier.

If rescaling changes the functional margin but not the boundary, how do we measure the **actual distance** to the boundary?

The Scaling Problem

One classifier, many descriptions: why the size of the functional margin is meaningless.

- For any $c > 0$, the parameters (w, b) and (cw, cb) describe exactly the same classifier: the same boundary and the same predictions. **Choosing c is a choice of description, not of classifier.**
- A quantity that changes under rescaling measures the parameters, not the classifier. The **functional margin** is such a quantity:

$$m_i(cw, cb) = c \cdot m_i(w, b)$$

- The **geometric margin** divides the scale out, so it measures the classifier itself:

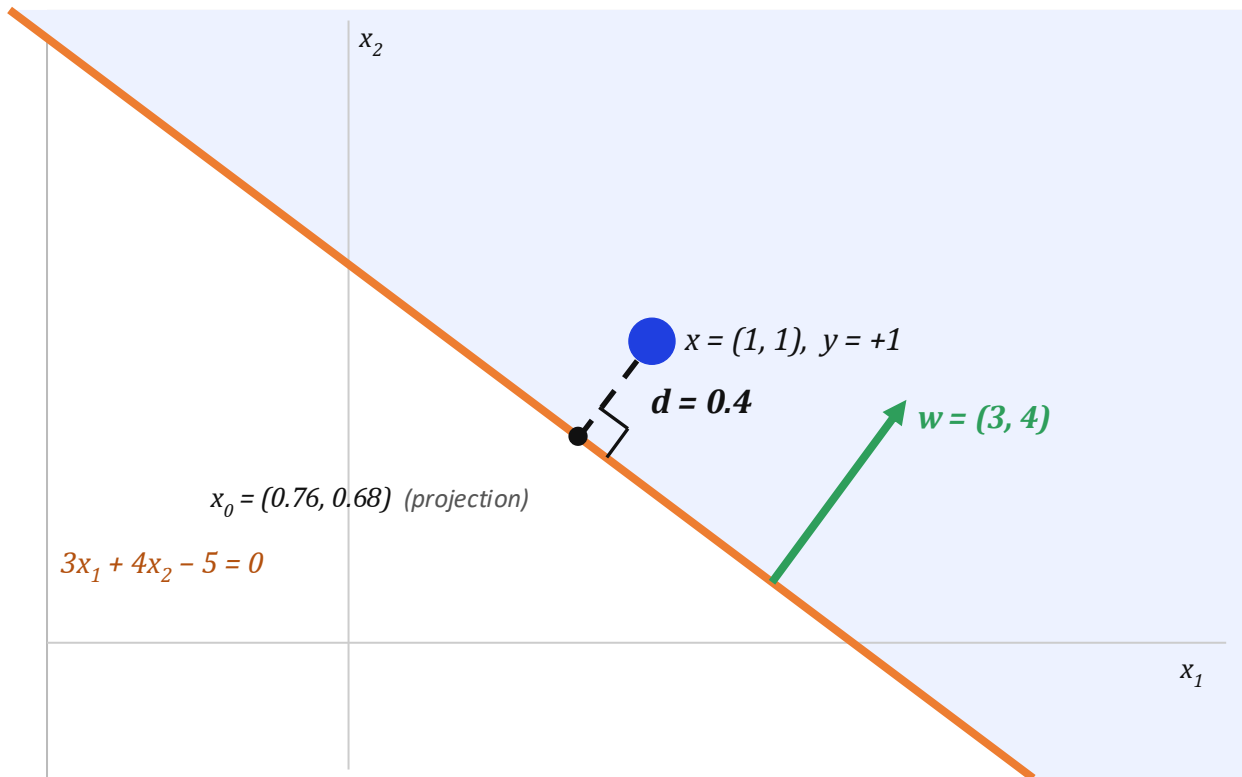
$$\gamma_i(cw, cb) = \frac{c m_i}{c \|w\|_2} = \gamma_i$$

- The two margins coincide when $\|w\|_2 = 1$: the geometric margin is simply the functional margin measured at one fixed scale.

If rescaling changes the functional margin but not the boundary,
how do we measure the **actual distance** to the boundary?

Geometric distance: measuring the actual separation

- The functional margin tells us whether an example is correctly classified, but its magnitude depends on the scaling of the classifier.
- We need a measure of **actual distance** from the decision boundary.



Step 1 — Calculate the score

$$f(x) = 3(1) + 4(1) - 5 = 2$$

The score is 2 above the boundary's score of zero. It is not a distance of 2.

Step 2 — How quickly does the score change?

$$w = (3, 4) \Rightarrow \|w\|_2 = \sqrt{3^2 + 4^2} = 5$$

Moving one unit perpendicular to the boundary, in the direction of w , increases the score by 5.

Step 3 — Convert the score to a distance

$$d = 2/5 = 0.4$$

$$d_i = |f(x_i)| / \|w\|_2 = |w^T x_i + b| / \|w\|_2$$

The absolute value makes the distance **nonnegative** on either side.

Distance = magnitude of score / score change per unit perpendicular distance

*Distance ignores which side of the boundary a point lies on. How can we make it **signed**, so that it is useful for **classification**?*

From Distance to Geometric Margin

- For classification we want positive values for correctly classified examples and negative values for incorrectly classified ones.
- Multiplying the score by the true label gives the signed counterpart of the distance.

Geometric margin

$$\gamma_i = \frac{y_i f(x_i)}{\|w\|_2} = \frac{y_i (w^\top x_i + b)}{\|w\|_2} = \frac{m_i}{\|w\|_2}$$

the functional margin, divided by the weight norm

Example: $\gamma = \frac{(+1)(2)}{5} = 0.4$

From the score to the geometric margin

$$\begin{array}{ccccc} f(x_i) = 2 & \xrightarrow{\times y_i} & m_i = 2 & \xrightarrow{\div \|w\|_2} & \gamma_i = 0.4 \\ \text{score} & & \text{functional margin} & & \text{geometric margin} \end{array}$$

Why this fixes the scaling problem

	Original	Original × 10
Functional margin m	2	20
Weight norm $\ w\ $	5	50
Geometric margin γ	0.4	0.4

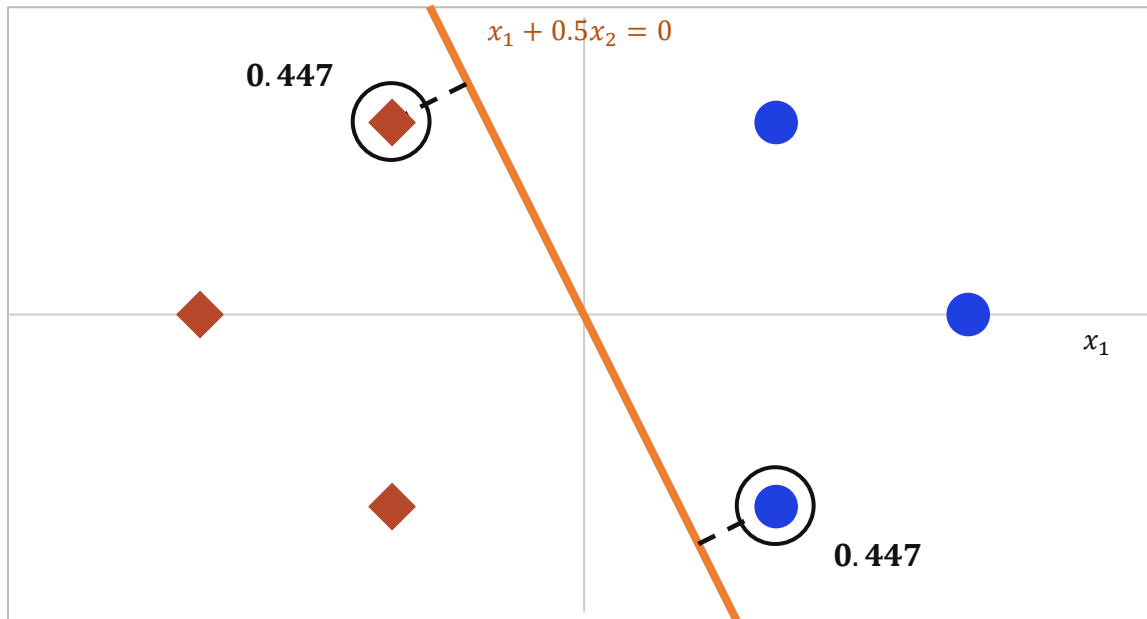
Dividing by the weight norm corrects for arbitrary positive rescaling of the parameters.

Now that we can measure the distance from each training example to the decision boundary, which boundary should we choose?

Maximum-Margin: Choosing the Widest Separation

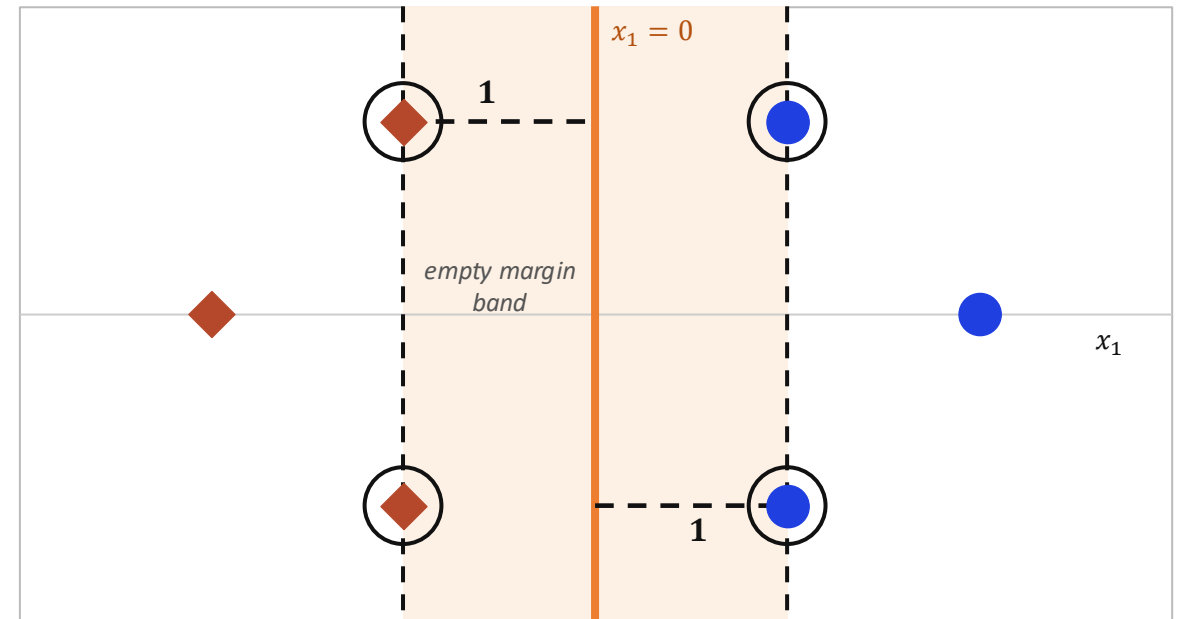
- Among the boundaries that correctly classify every training example, we choose the one that stays **farthest from its nearest examples**: maximum margin.

A. Valid, but a narrow margin $f_A(x) = x_1 + 0.5x_2$



$$\gamma_{\min,A} = \frac{0.5}{\sqrt{1.25}} \approx 0.447$$

B. Maximum-margin separator $f_B(x) = x_1$



$$\gamma_{\min,B} = 1$$

How can we describe “**the distance to the nearest example**” mathematically?

The Maximum-Margin Principle

- The critical distinction is minimum, **not average**.

A boundary far from most examples but very close to one is still a poor choice.

Minimum geometric margin

$$\gamma_{\min}(w, b) = \min_{i=1, \dots, N} \frac{y_i(w^\top x_i + b)}{\|w\|_2}$$

The smallest value over all N training examples.

Find the orientation and position of the separating **boundary** that maximise its smallest geometric margin.

Maximum-margin principle

$$\max_{w, b} \min_i \frac{y_i(w^\top x_i + b)}{\|w\|_2}$$

A maximum-margin classifier chooses the separating boundary that maximises the distance to its closest training examples.

*We now know what to maximise: the minimum geometric margin.
How do we turn this into a convenient **optimisation problem**?*

From Maximum Margin to a Tractable Problem

- Scaling turns the max-min objective into a smooth constrained minimisation.

Step 1: the max-min objective is “awkward”

$$\max_{w,b} \min_i \frac{y_i(w^\top x_i + b)}{\|w\|_2}$$

A minimum inside a maximum: not smooth, and hard to optimise directly.

Step 2: we are free to choose the scale

Rescaling (w, b) by any $c > 0$ leaves the boundary unchanged. Choose the scale at which the closest example has functional margin exactly 1:

$$\min_i y_i(w^\top x_i + b) = 1$$

Step 3: the margin becomes $1/\|w\|_2$

$$\gamma_{\min} = \frac{1}{\|w\|_2}$$

So maximising the margin means minimising $\|w\|_2$, or equivalently $\frac{1}{2}\|w\|_2^2$, which is smooth and has gradient w .

Step 4: relax the equality

Maximal margin classifier (hard-margin SVM)

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} \|w\|_2^2 \\ \text{subject to} \quad & y_i(w^\top x_i + b) \geq 1, \quad i = 1, \dots, N \end{aligned}$$

Each one says that example i must lie on its correct side of the boundary with a functional margin of at least 1.

The Maximal Margin Classifier

- The closest training examples constrain the position and orientation of the optimal boundary. These are called **support vectors**.

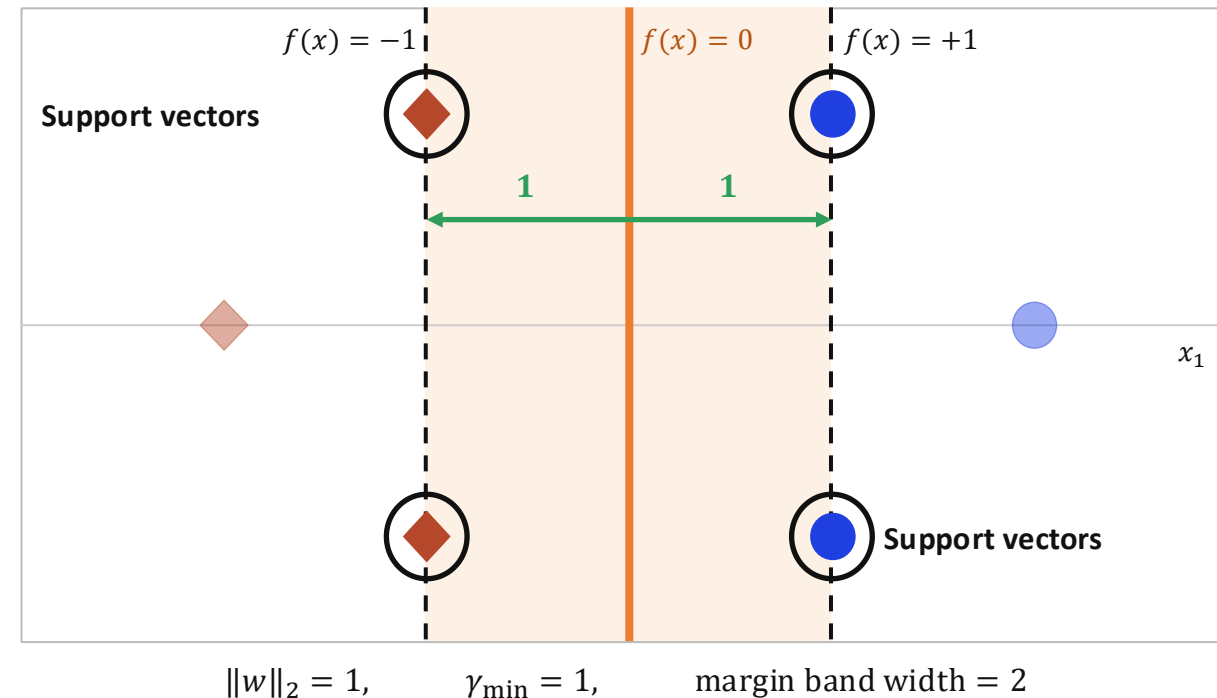
Maximal margin classifier

The separating hyperplane with the largest minimum geometric margin (also called the *hard-margin SVM*). Separable data only.

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} \|w\|_2^2 \\ \text{subject to} \quad & y_i(w^\top x_i + b) \geq 1 \end{aligned}$$

Minimisation objective: maximise the margin by minimising the weight norm under the constraints.

Constraints: all training examples are classified correctly with a functional margin of at least 1.



The maximal margin classifier chooses the separating boundary with the **largest minimum geometric margin**.

The Maximal Margin Classifier

Perceptron	Maximal Margin Classifier
Finds a separating hyperplane	Finds a maximum-margin separating hyperplane
Does not explicitly prefer wider margins	Explicitly maximises the minimum geometric margin

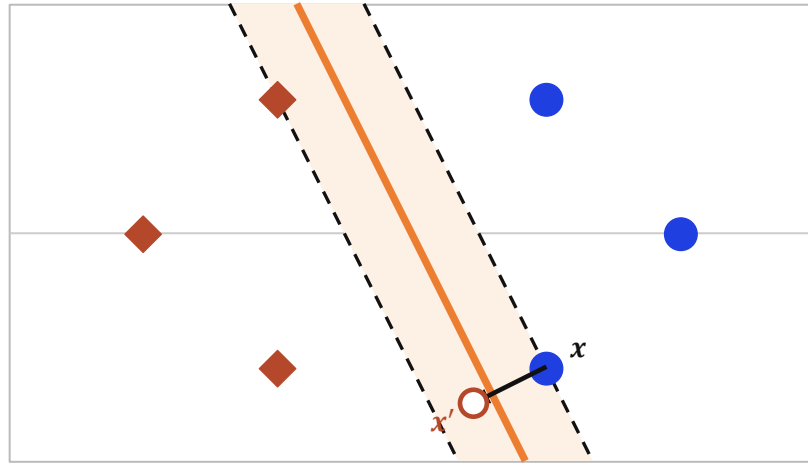
- Both methods can achieve **zero training error**, but the maximal margin classifier uses the **margin to prefer one separating boundary over another**.

*Both the Perceptron and the maximal margin classifier can achieve zero training error.
Why should we prefer the wider-margin solution?*

Why Prefer a Wider Margin? Robustness to Perturbations

A — Larger weights, narrower margin

$$f_A(x) = 2x_1 + x_2, \quad \|w_A\| = \sqrt{5} \approx 2.24$$

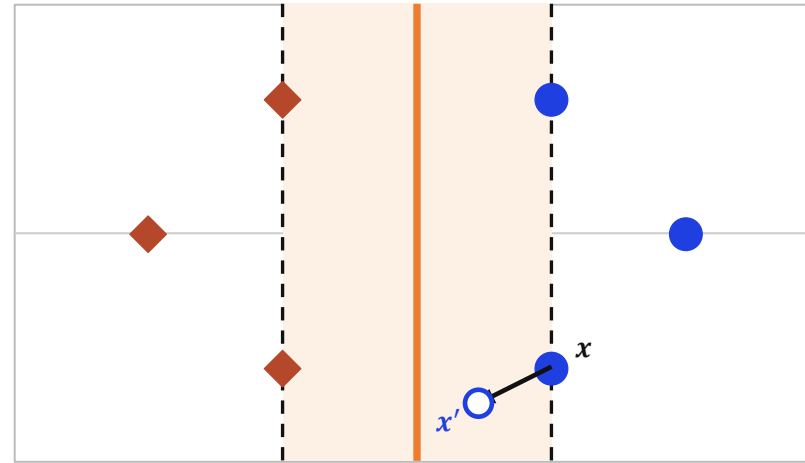


$$\gamma_{\min,A} = \frac{1}{\sqrt{5}} \approx \mathbf{0.447}$$

Higher score sensitivity: smaller geometric margin.

B — Smaller weights, wider margin

$$f_B(x) = x_1, \quad \|w_B\| = 1$$

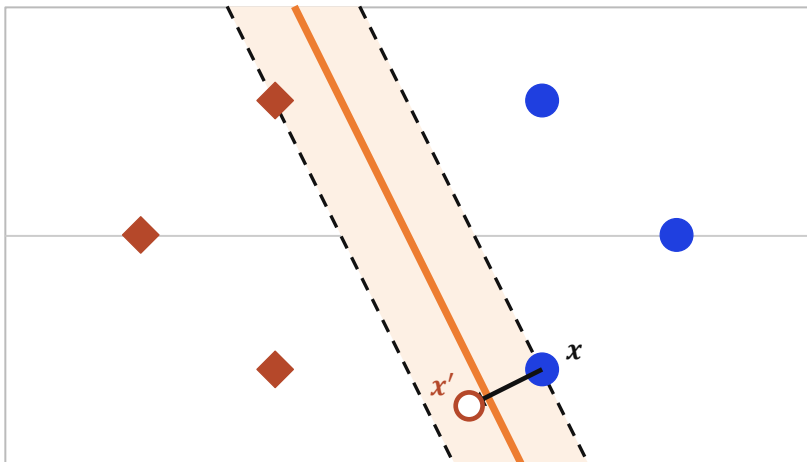


$$\gamma_{\min,B} = \mathbf{1}$$

Lower score sensitivity: larger geometric margin.

Understanding Regularization

- If $\mathbf{w}$ is too large (positive or negative)
 - Then a small change in $\mathbf{x}$ (e.g., due to noise) will cause a large change in the output $\mathbf{w}^T \mathbf{x} + b$
 - Can lead to errors
 - Controlling for this is called “Regularization”
- Achieved by minimizing:
$$R(f) = \mathbf{w}_1^2 + \mathbf{w}_2^2 + \dots + \mathbf{w}_d^2 = \mathbf{w}^T \mathbf{w} = \|\mathbf{w}\|^2$$



Small weights limit “the butterfly effect”

- Let’s quantify how sensitive the model is to a perturbation of its input
- $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$
- $f(\mathbf{x} + \delta \mathbf{x}) = \mathbf{w}^T (\mathbf{x} + \delta \mathbf{x}) + b = \mathbf{w}^T \mathbf{x} + b + \mathbf{w}^T \delta \mathbf{x} = f(\mathbf{x}) + \mathbf{w}^T \delta \mathbf{x}$
- $f(\mathbf{x} + \delta \mathbf{x}) - f(\mathbf{x}) = \mathbf{w}^T \delta \mathbf{x}$
- $\|f(\mathbf{x} + \delta \mathbf{x}) - f(\mathbf{x})\| = \|\mathbf{w}^T \delta \mathbf{x}\| \leq \|\mathbf{w}\| \|\delta \mathbf{x}\|$
(using Cauchy-Schwarz inequality $\|a^T b\| \leq \|a\| \|b\|$)
- Therefore, $\frac{\|f(\mathbf{x} + \delta \mathbf{x}) - f(\mathbf{x})\|}{\|\delta \mathbf{x}\|} \leq \|\mathbf{w}\|$

Change in model output per unit additive change in input is upper bounded by $\|\mathbf{w}\|$. Consequently, minimizing the norm of the weight vector (or its square) would lead to a regularization effect as it would limit the effect of any change in the input on the output.

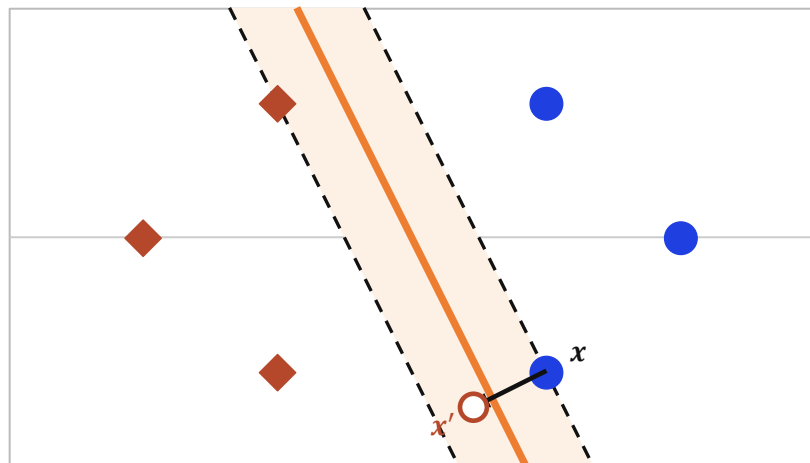
Vapnik showed that **minimizing “structural risk”** (combination of empirical error over N training examples and the norm of the weight vector) **leads to minimization of the upper bound on generalization loss/error (error over unseen examples).**

$$L_{gen}(\mathbf{w}) \leq L_{emp}(\mathbf{w}) + \mathcal{O}\left(\frac{\|\mathbf{w}\|}{\sqrt{N}}\right)$$

Why Prefer a Wider Margin? Robustness to Perturbations

A — Larger weights, narrower margin

$$f_A(x) = 2x_1 + x_2, \quad \|w_A\| = \sqrt{5} \approx 2.24$$

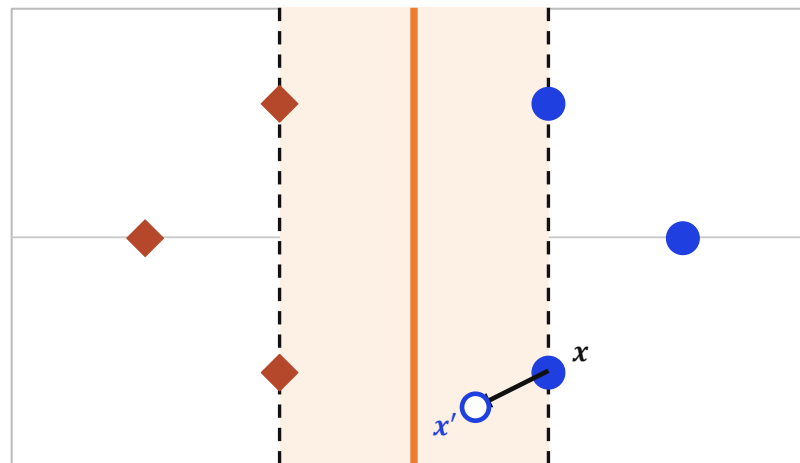


$$\gamma_{\min,A} = \frac{1}{\sqrt{5}} \approx 0.447$$

Higher score sensitivity: smaller geometric margin.

B — Smaller weights, wider margin

$$f_B(x) = x_1, \quad \|w_B\| = 1$$



$$\gamma_{\min,B} = 1$$

Lower score sensitivity: larger geometric margin.

So far we have assumed that a separating boundary exists. What if it does not?

Part 3: The Support Vector Classifier

Data that are not linearly separable; the boundary is still linear. Hinge loss, the role of C , optimisation and support vectors.

Part 1

Perceptron

Part 2

Maximal margin classifier

Part 3

Support vector classifier

Part 4

Regularisation

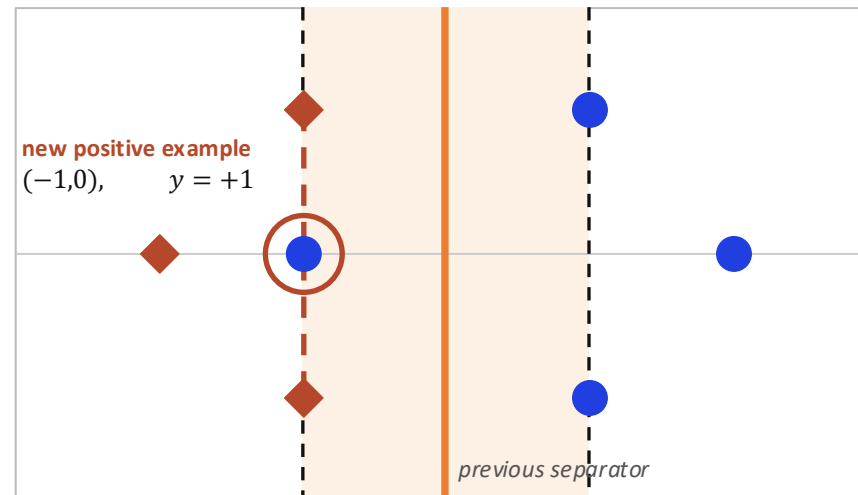
Next lecture

Support vector machine
(Lecture 5)

When Perfect Separation Is Impossible

- What if the training data cannot be separated by a straight line?
- I "always" can, with a transformation.. but at the cost of generalization!

One additional example



No straight line can correctly separate all the examples.

Imperfect separability is common

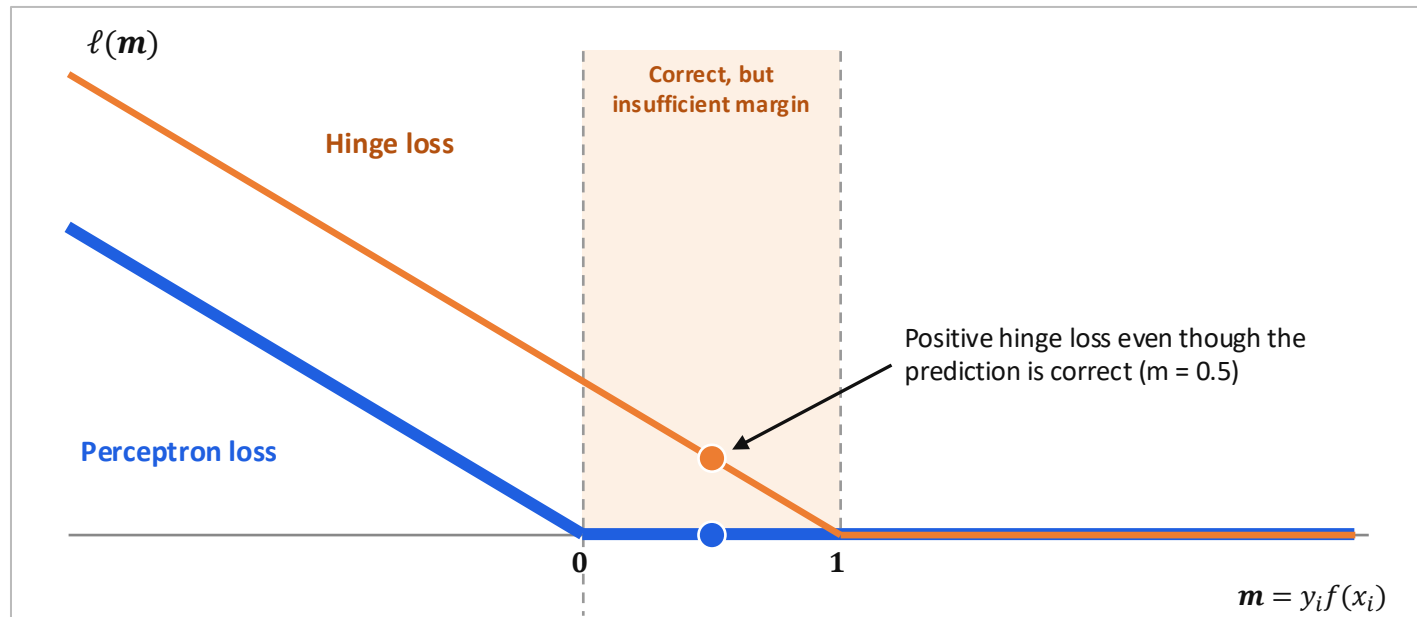
- Overlapping feature distributions
- Noisy measurements
- Potentially incorrect labels
- Outliers or unusual observations

- **Hard-margin** MMC requires every example to satisfy the margin constraints.

Could we allow some **violations** instead of demanding perfect separation?

From Perceptron Loss to Hinge Loss

- From correct classification to classification with a margin.



From one loss to the other

$$\ell_P(m) = \max(0, -m)$$

$$\ell_H(m) = \max(0, 1 - m)$$

The Perceptron requires the score to reach zero. Hinge loss shifts the zero-loss threshold to a functional margin of 1.

$$\ell_H(m) = \ell_P(m - 1)$$

m	Perceptron	Hinge
-1	1	2
+0.5	0	0.5
+1.5	0	0

- Hinge loss measures violations of the required functional margin. Like the functional margin, it depends on the scaling of w and b ; it is not a geometric distance.

The Perceptron penalises negative functional margins.

Hinge loss penalises every example whose functional margin is below 1, even when its predicted class is correct.

*Can we simply **minimise** the total hinge loss on its own?*

Hinge Loss Alone Is Not Enough

The scale shortcut: why the support vector classifier needs a penalty on the weights.

Intuition

- **Hinge loss asks every example for a functional margin of at least 1.** But the functional margin grows when we simply scale up w and b , and scaling does not move the boundary.
- So minimising hinge loss on its own can “**satisfy**” the margin requirement by **inflating** the parameters, rather than by placing the boundary well. Any separator, however close to the data, can be made to look perfect.
- **A penalty on $\|w\|$ removes this shortcut.** We have just seen it before!
Inflating the parameters now has a price, so the only cheap way to reach $m_i \geq 1$ is to keep the examples genuinely far from the boundary.

The weight penalty removes the freedom to rescale: margins must be earned by geometry, not by inflating the parameters.

Another reason to regularise the weights!

Support Vector Classifier: The Complete Objective

GOAL 1: MAXIMISE SEPARATION

$\frac{1}{2} \|w\|_2^2$ Prefer smaller weight norms and wider margins, under the canonical margin normalisation.

+

GOAL 2: PENALISE VIOLATIONS

$\ell_H(m_i) = \max(0, 1 - m_i)$ Allow violations of the desired functional margin, at a cost.



Support Vector Classifier (soft-margin)



$$\min_{w,b} \quad \frac{1}{2} \|w\|_2^2 + C * \sum_{i=1 \dots N} \max(0, 1 - y_i(w^\top x_i + b))$$

Weight penalty

Penalty for margin violations (a sum of hinge losses)

First term: $\frac{1}{2} \|w\|_2^2$ discourages large weights

- Under canonical margin constraints, smaller weight norms correspond to **wider geometric margins**.
- Controlling $\|w\|$ also controls the **sensitivity** of the score to input perturbations.
- It is the squared L2 regulariser.

Second term: $C \sum \ell_H(m_i)$ penalises shortfalls

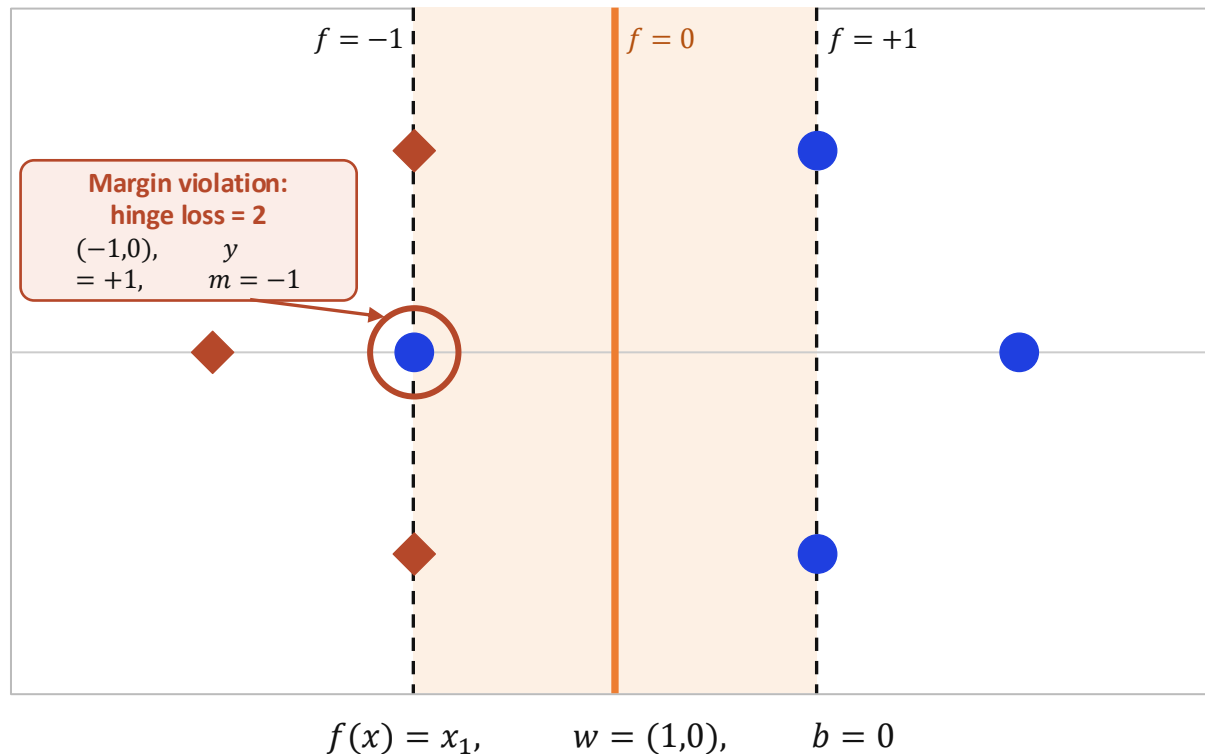
- $m_i \geq 1$: zero loss.
- $0 < m_i < 1$: correct, but a positive penalty.
- $m_i < 0$: misclassified, a larger penalty.

$C > 0$ sets how **heavily violations are penalised** relative to the weight norm.

- Rather than treating violations as forbidden, we assign them a **cost**.
- *A.k.a. Linear SVM, or Soft-margin SVM*

From Hard to Soft Margins: One Violation

- The same configuration that was infeasible under hard-margin constraints is now a valid candidate.



Evaluating the objective for this classifier

- $\frac{1}{2} \|w\|_2^2 = \frac{1}{2}$
- Six examples have zero hinge loss; the conflicting positive example has loss 2
- $\sum_i \ell_H(m_i) = 2$

$$J(w, b) = 0.5 + 2C$$

- At $C = 1$ this boundary happens to be optimal.
- The objective at this candidate, not the minimum for every C . For some C another boundary does better.

*But this introduces a new decision: **how expensive should those violations be?***

Part 4: Regularisation and Generalisation

The support vector classifier as regularised risk minimisation, why wider margins generalise, and how to choose the strength of the penalty.

Part 1

Perceptron

Part 2

Maximal margin classifier

Part 3

Support vector classifier

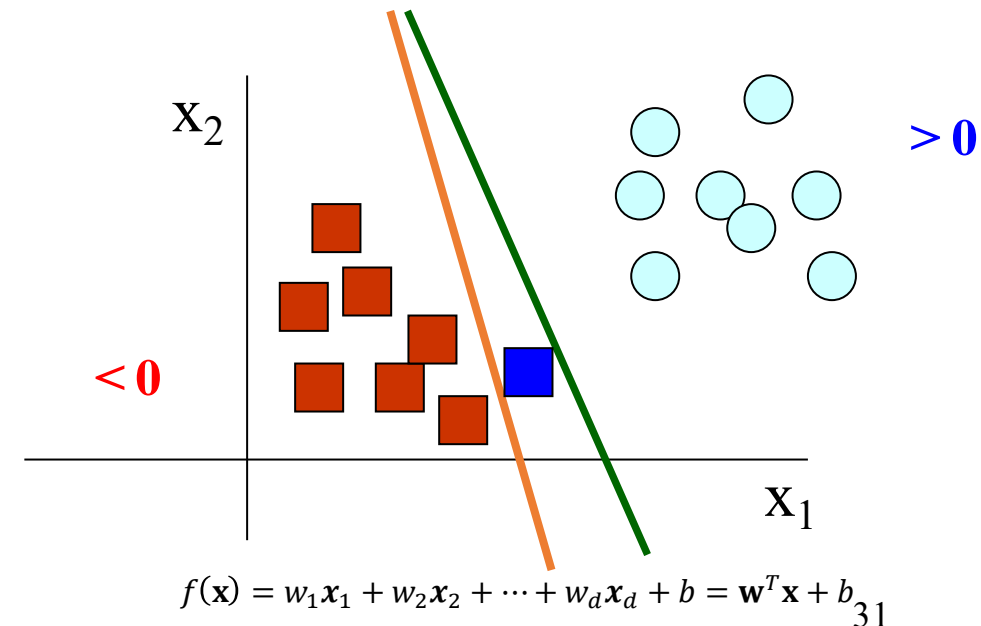
Part 4 Regularisation

Next lecture

Support vector machine
(Lecture 5)

Support Vector Machines (SVM)

- Support Vector Machines are linear classifiers that produce the optimal separating boundary (hyper-plane)
- Find $\mathbf{w}$ and b in a way so as to:
 - Minimize misclassification error over training data (**Empirical Risk Minimization**)
 - Maximize the margin (**Structural Risk Minimization**)
 - Or equivalently, maximize regularization
 - Or equivalently, minimize the individual absolute weights



SVM Optimization

$$\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{C}{N} \sum_{i=1}^N \max\{0, 1 - y_i f(\mathbf{x}_i)\}$$

Alternatively, we can use a weighting hyperparameter with the regularization term

$$\min_{\mathbf{w}} P(\mathbf{w}) = \frac{\lambda}{2} \mathbf{w}^T \mathbf{w} + \frac{1}{N} \sum_{i=1}^N \max\{0, 1 - y_i f(\mathbf{x}_i; \mathbf{w})\}$$

Gradient Descent Update Rule: $\mathbf{w}_k \leftarrow \mathbf{w}_{k-1} - \alpha \nabla P(\mathbf{w}_{k-1})$

Gradient calculation: $\nabla P = \lambda \mathbf{w} + \frac{\partial}{\partial \mathbf{w}} \max\{0, 1 - y(\mathbf{w}^T \mathbf{x})\}$

$\frac{\partial}{\partial \mathbf{w}} \max\{0, 1 - y(\mathbf{w}^T \mathbf{x})\} = \begin{cases} 0 & yf(\mathbf{x}; \mathbf{w}) > 1 \\ -y\mathbf{x} & \text{else} \end{cases} = \mathbf{1}(yf(\mathbf{x}) < 1)(-y\mathbf{x})$

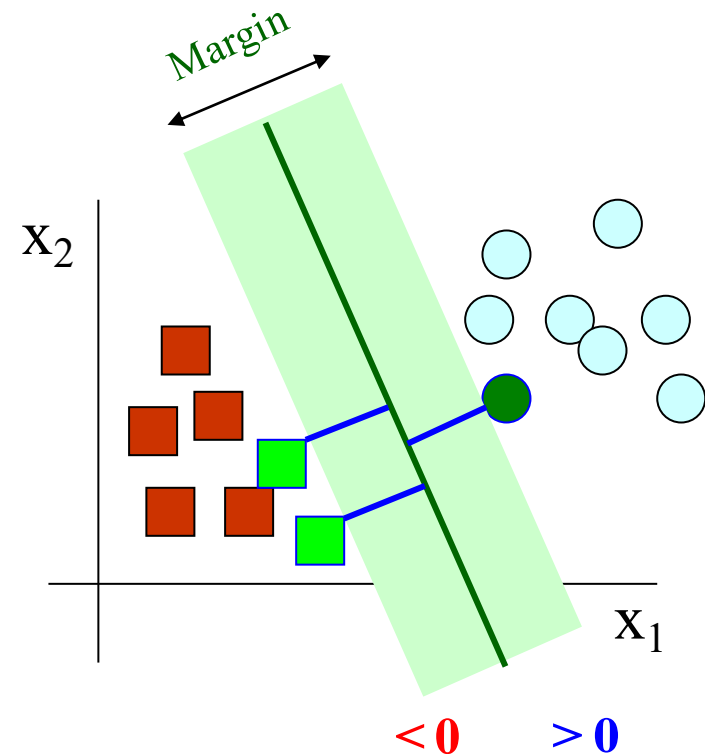
$\frac{\partial}{\partial \mathbf{w}} \left(\frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \right) = \lambda \mathbf{w}$

$$\nabla P = \lambda \mathbf{w} + \mathbf{1}(yf(\mathbf{x}) < 1)(-y\mathbf{x}) = \lambda \mathbf{w} - \mathbf{1}(yf(\mathbf{x}) < 1)(y\mathbf{x})$$

Final Weight Update Rule: $\mathbf{w}_k \leftarrow \mathbf{w}_{k-1} - \alpha \lambda \mathbf{w}_{k-1} + \alpha \mathbf{1}(yf(\mathbf{x}; \mathbf{w}_{k-1}) < 1)(y\mathbf{x})$

Support Vector Machines

- Support Vector Machines, in their basic form, are linear classifiers that give maximum margin or regularization
- Principles of Operation
 - Minimize the number of training errors
 - Achieved by minimizing hinge loss
 - Maximize margin
 - Allows noise tolerance
 - Allows Regularization
 - Perform Nonlinear Classification
 - Achieved through feature transformations/kernels
- The points that determine the margin are called **Support Vectors**



Coding in Python

```
import numpy as np
import matplotlib.pyplot as plt
import itertools

class RegularizedPerceptron:
    def __init__(self, Lambda = 0.0, margin = 0.0, alpha = 0.1, epochs = 1000):
        self.alpha = alpha
        self.epochs = epochs
        self.W = np.array([0])
        self.bias = np.random.randn()
        self.Lambda = Lambda #not used in perceptron
        self.Margin = margin #0.0 in Perceptron
    def fit(self, Xtr, Ytr):
        d = Xtr.shape[1]
        self.W = np.random.randn(d)
        for e in range(self.epochs):
            finished = True
            for i, x in enumerate(Xtr):
                if self.score(np.atleast_2d(x)) * Ytr[i] < self.Margin:
                    self.W += self.alpha * Ytr[i] * x
                    self.bias += self.alpha * Ytr[i]

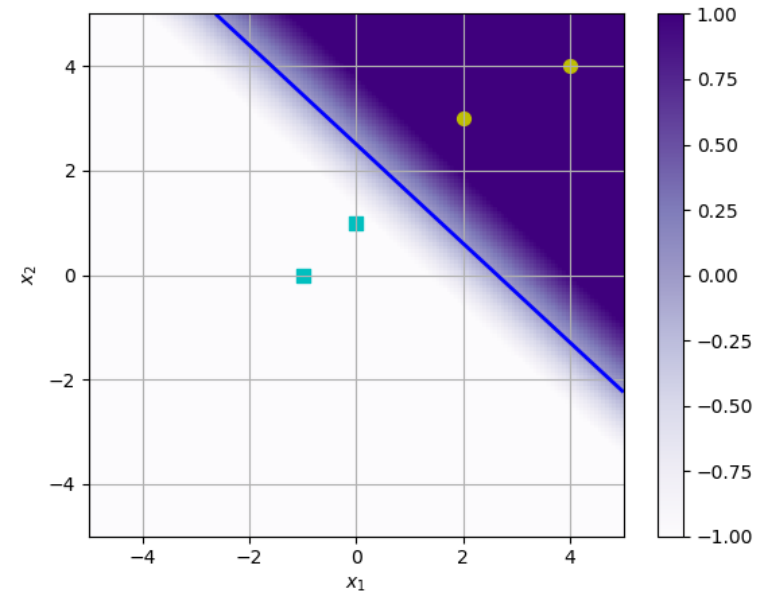
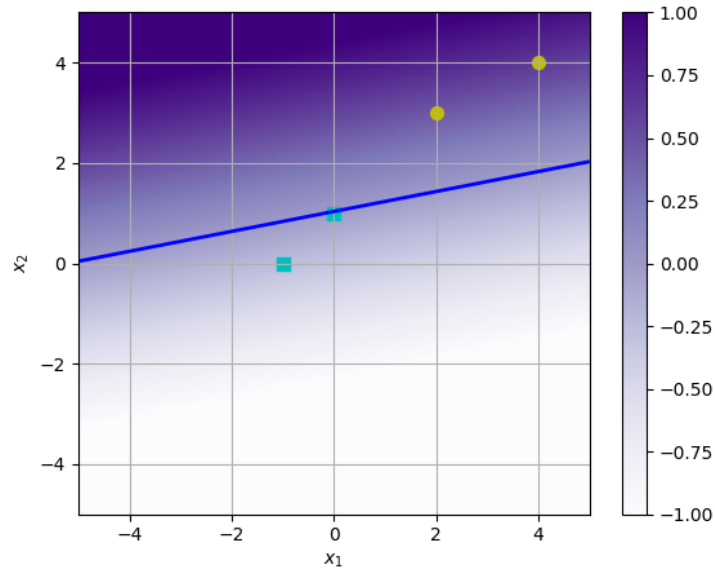
            self.W = self.W - self.alpha * self.Lambda * self.W    #Regularization update

    def score(self, x):
        return np.dot(x, self.W) + self.bias

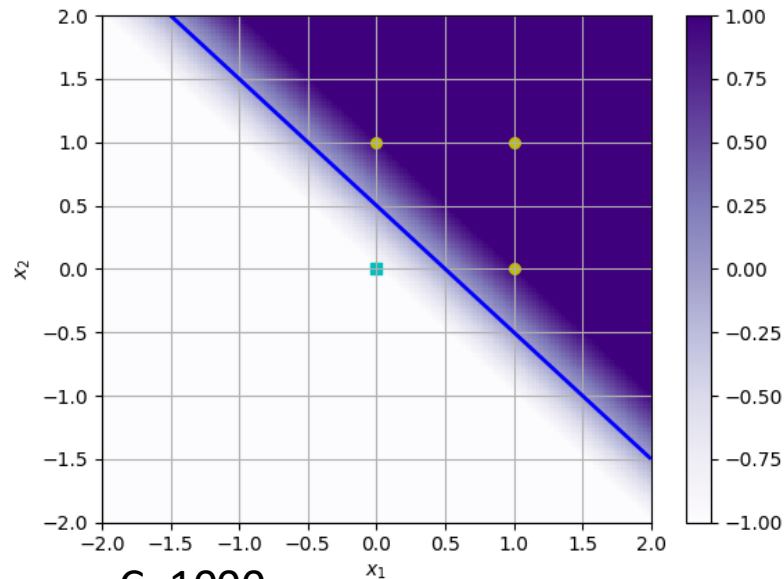
    def predict(self, x):
        return np.sign(self.score(x))
```

```
if __name__ == '__main__':
    from plotit import plotit
    Xtr = np.array([[ -1, 0], [ 0, 1], [ 4, 4], [ 2, 3]])
    ytr = np.array([ -1, -1, +1, +1])
    clf = RegularizedPerceptron(Lambda = 0.1, margin = 1.0)
    clf.fit(Xtr, ytr)
    z = clf.score(Xtr)
    print("Prediction Scores:", z)
    y = clf.predict(Xtr)
    print("Prediction Labels:", y)
    plotit(Xtr, ytr, clf=clf.score, conts=[0], extent = [-5, +5, -5, +5])
```

Difference between Perceptron and SVM

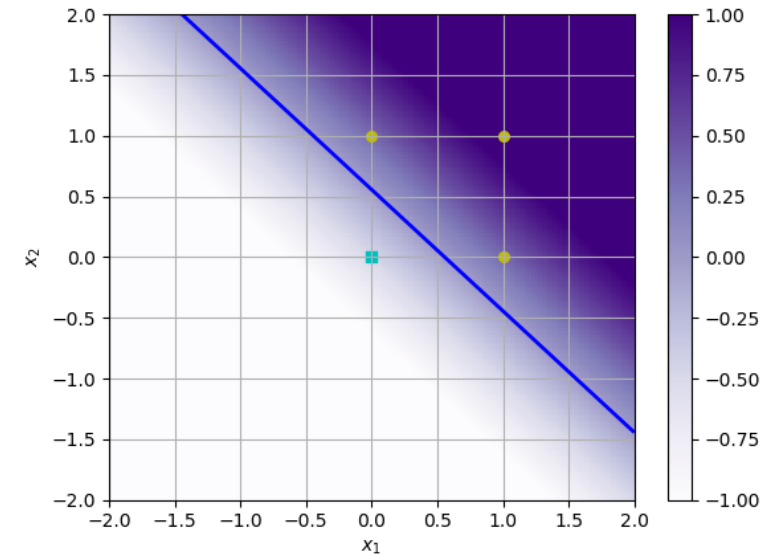


SVM in Sklearn



$C=1000$

$w=[2,2], b=-1$



$C=1$

$w = [0.84 \ 0.84], b=-0.465$

```
import numpy as np
from sklearn.svm import LinearSVC as Classifier
```

```
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([-1,1,1, 1])
clf = Classifier(class_weight='balanced',C=100)
clf.fit(X, y)
f = clf.predict(X)
print('Coefficients before adding additional feature:', clf.coef_,clf.intercept_)
print('Predictions before adding additional feature:',f)
plotit(X,y,clf = clf.decision_function,conts=[0],extent=[-2,+2,-2,+2])
```

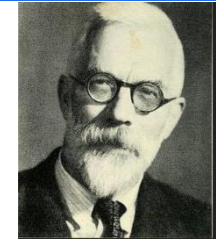
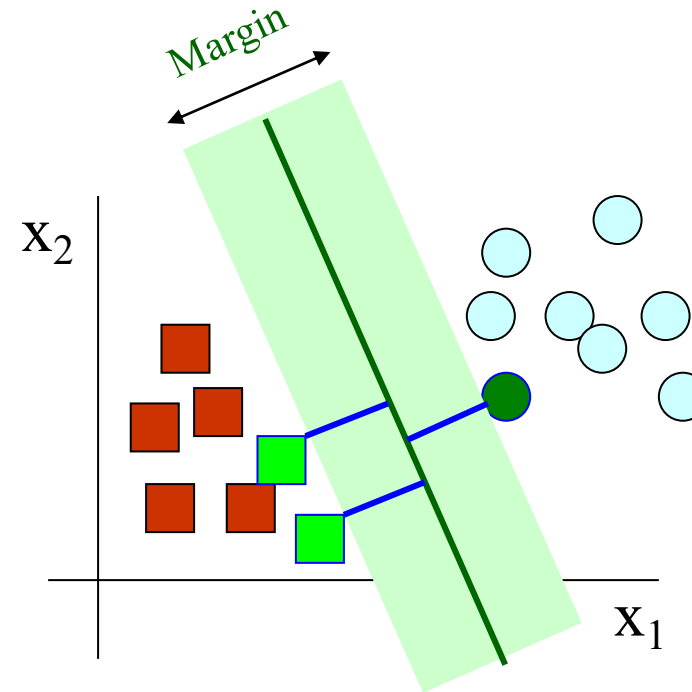
Wanna Play?

- Use the Java Applet at:
- <https://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- Set “-t 0 -c 100”

SVMs up till now

$$\min_{w,b} \frac{1}{2} w^T w + \frac{C}{N} \sum_{i=1}^N \max\{0, 1 - y_i f(x_i)\}$$

- Vapnik and Chervonenkis:
 - Theoretical foundations for SVMs
 - Structural Risk Minimization
- Corinna Cortes
 - Soft SVM (1995)
- Bernard Scholkopf (1997)
 - Representer Theorem
 - Kernels allow
 - nonlinear boundaries
 - representation of non-vectorial data



R. A. Fisher
1890-1962



Rosenblatt
1928-1971



V. Vapnik
1936 -



Chervonenkis
1938 - 2014



C. Cortes
1961 -



Scholkopf
1968 -

Regularisation in Practice: Choosing C

Intuition

C is a dial between trusting the data and trusting the preference for simplicity.

- **C too large:** every violation is expensive, so the boundary bends to fit noisy or unusual examples. Narrow band, low bias, high variance.
- **C too small:** violations are cheap, so the boundary may ignore real structure in the data. Wide band, high bias, low variance.

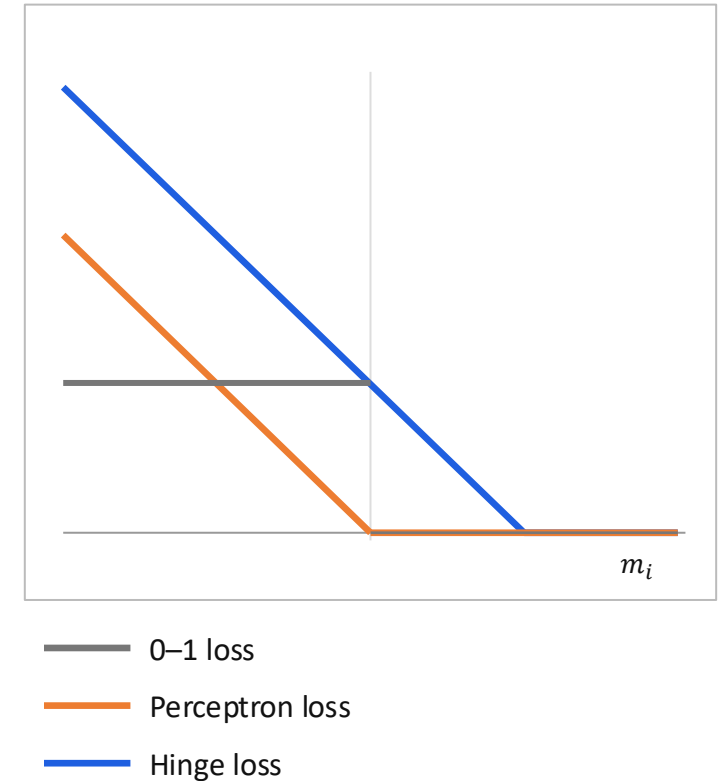
The right value depends on how noisy and how overlapping the classes are. It cannot be derived in advance: it is chosen from data.

In practice

- **Choose C by validation:** k-fold cross-validation over a logarithmic grid, for example $C \in \{10^{-3}, 10^{-2}, \dots, 10^3\}$.
- **Standardise the features first:** $\|w\|$ depends on the units of the features, so unscaled features are penalised unequally.

Summary: From Margins to Loss Functions

Concept	Expression	What it tells us
Functional margin	$m_i = y_i f(x_i)$	Signed score: its sign shows correctness
Geometric margin	$\frac{m_i}{\ w\ _2}$	Signed distance to the boundary; unchanged by rescaling w and b
0-1 loss	$1[m_i \leq 0]$	Counts mistakes; flat, so no useful gradient
Perceptron loss	$\max(0, -m_i)$	Penalises mistakes only; zero once correct
(SVM) Hinge loss	$\max(0, 1 - m_i)$	Also penalises correct points with $m < 1$, inside the margin



- One quantity, m_i , and the losses differ only in how they treat it.
- **Perceptron:** a correct example costs nothing, so any separator is acceptable.
- **SVM:** a correct example is not enough; we also want it “outside” the margin.

Summary: From the Perceptron to Support Vector Machines

- Three margin-based classifiers, each relaxing one assumption of the one before.

	Perceptron	Maximal margin classifier	Support vector classifier (Linear SVM)	Support vector machine (Lecture 5)
Data	Linearly separable (otherwise it never converges)	Linearly separable only	Any data	Any data
Boundary	Linear	Linear	Linear	Nonlinear in the inputs (linear in a feature space)
Objective	$\max(0, -m_i)$: mistakes only	$\min \frac{1}{2}\ w\ ^2$ subject to $m_i \geq 1$	$\frac{1}{2}\ w\ ^2 + C \sum \max(0, 1 - m_i)$	Same objective, with a kernel
What it adds	Finds a separator, but not a particular one	Chooses the widest separation	Tolerates violations, at a cost set by C	Changes the representation

Nested special cases:

- The maximal margin classifier is the support vector classifier with $C \rightarrow \infty$ (on separable data);
- The support vector classifier is the support vector machine with a linear kernel.