Maximum Margin Classifier

A new twist to binary classification problem

Beginning: linear SVMs

- In the first part, we will consider a basic setup of SVMs, something called linear *hard margin SVM*. These definitions will not make much sense now, but we will come back to this later today
- The point is to keep in mind that SVMs are more powerful than they may initially appear
- Also for this part, we will assume that the data is linearly separable, i.e., the exists a hyperplane that perfectly separates the classes
- We will consider training using all data at once

SVM is a linear binary classifier

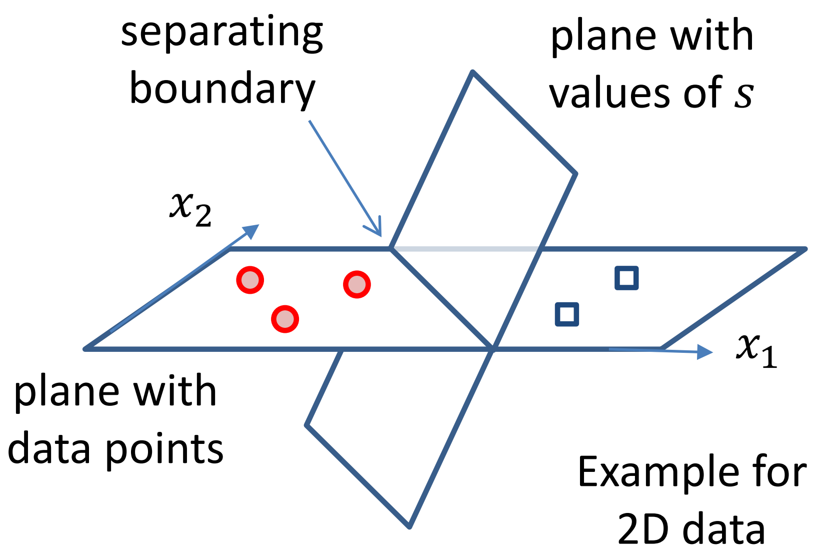
SVM is a binary classifier:

Predict class A if $s \geq 0$

Predict class B if $s < 0$

where $s = b + \sum_{i=1}^{m} x_i w_i$

SVM is a <u>linear classifier</u>: s is a linear function of inputs, and the separating boundary is linear

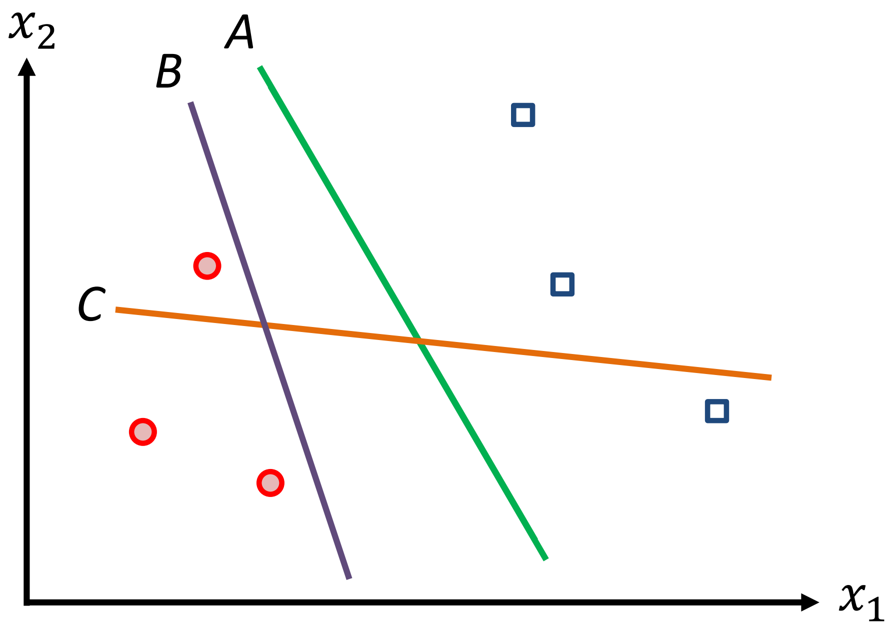


Example for 2D data

Choosing separation boundary

- An SVM is a linear binary classifier, so choosing parameters essentially means choosing how to draw a separation boundary (hyperplane)

- In 2D, the problem can be visualised as follows

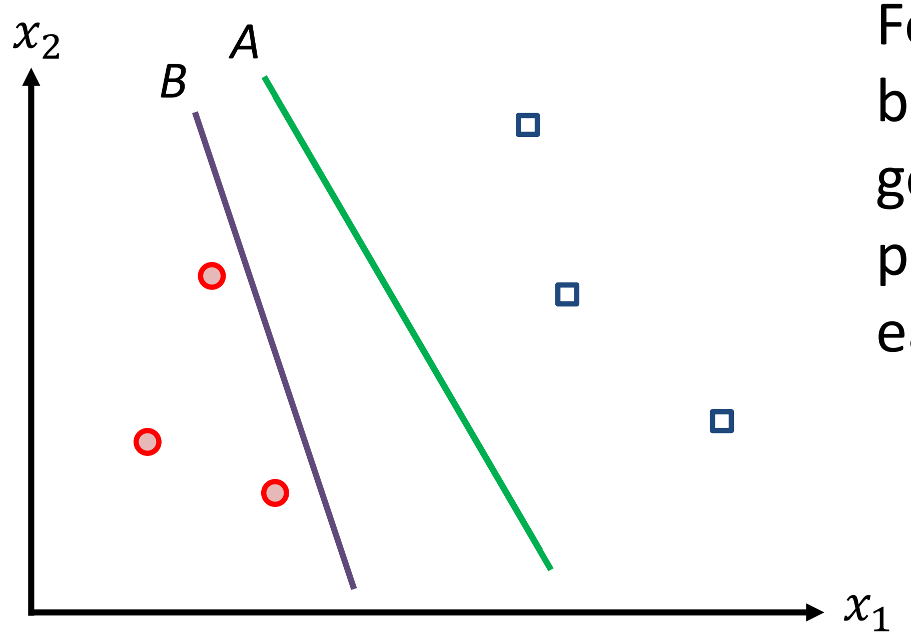


Which boundary should we use?

Line C is a clear “no”, but A and B both perfectly separate the classes

Which boundary should we use?

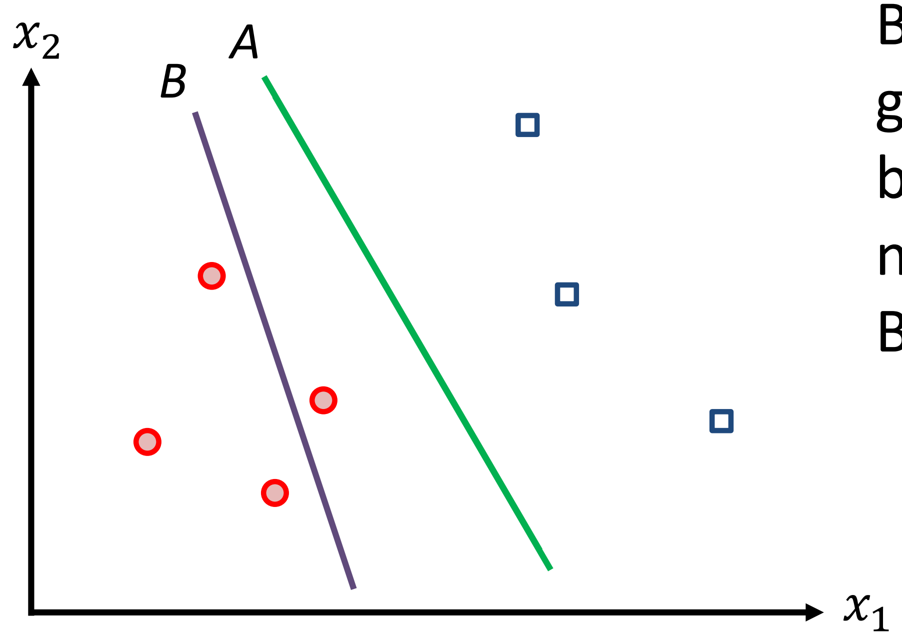
- Provided the dataset is linearly separable, the perceptron will find a boundary that separates classes perfectly. This can be any such boundary, e.g., A or B



For the perceptron, all such boundaries are equally good, because the perceptron loss is zero for each of them.

Which boundary should we use?

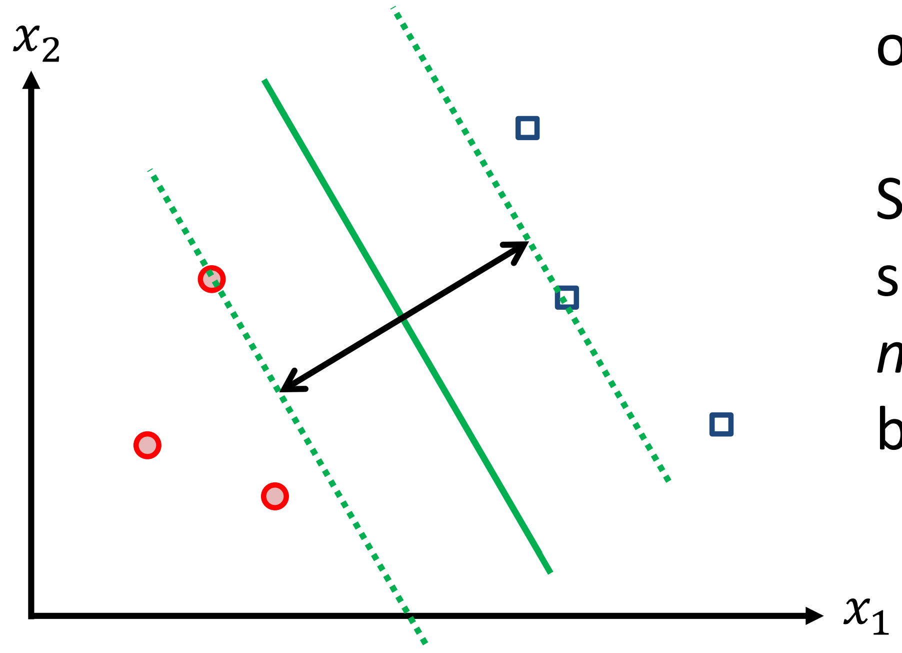
- Provided the dataset is linearly separable, the perceptron will find a boundary that separates classes perfectly. This can be any such boundary, e.g., A or B



But they don't look equally good to us. Line A seems to be more reliable. When new data point arrives, line B is likely to misclassify it

Aiming for the safest boundary

- Intuitively, the most reliable boundary would be the one that is between the classes and as far away from both classes as possible



SVM objective captures this observation

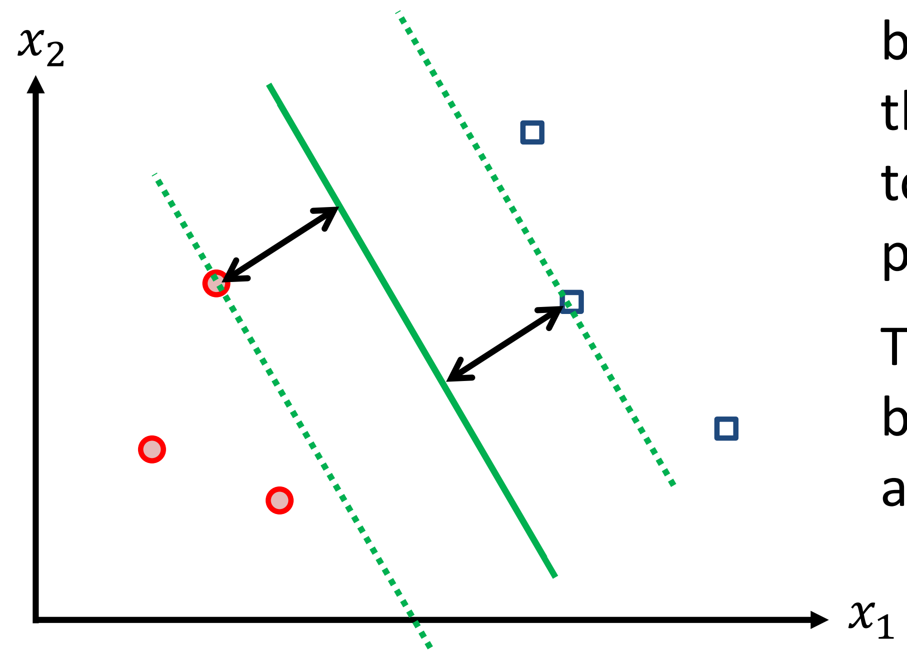
SVMs aim to find the separation boundary that *maximises the margin* between the classes

Maximum margin classifier

- An SVM is a linear binary classifier. During training, the SVM aims to find the separating boundary that maximises margin
- For this reason, SVMs are also called *maximum margin classifiers*
- The training data is fixed, so the margin is defined by the location and orientation of the separating boundary which, of course, are defined by SVM parameters
- Our next step is therefore to formalise our objective by expressing *margin width* as a function of parameters (and data)

Margin width

- While the margin can be thought as the space between two dashed lines, it is more convenient to define margin width as the distance between the separating boundary and the nearest data point(s)

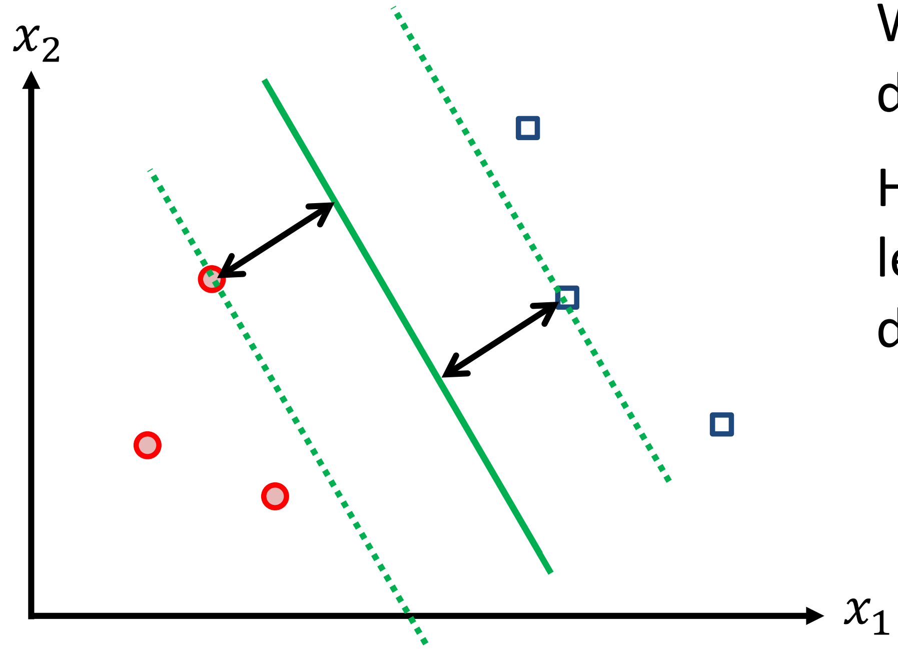


In the figure, the separating boundary is exactly "between the classes", so the distances to the nearest red and blue points are the same

The point(s) on margin boundaries from either side are called *support vectors*

Margin width

- While the margin can be thought as the space between two dashed lines, it is more convenient to define margin width as the distance between the separating boundary and the nearest data point(s)

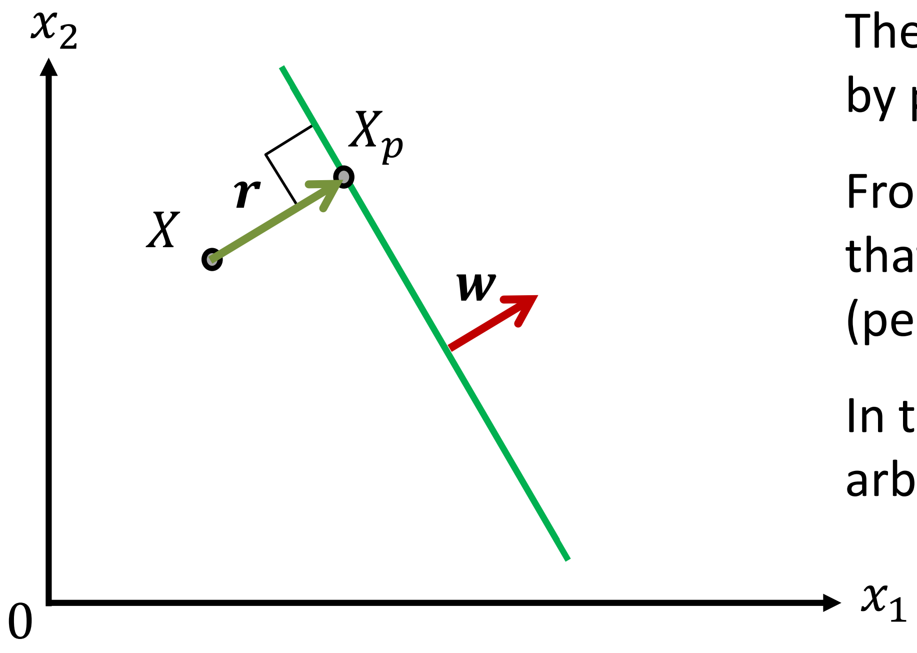


We want to maximise the distance to support vectors

However, before doing this, let's derive the expression for distance to an arbitrary point

Distance from point to hyperplane 1/3

- Consider an arbitrary point X (from either of the classes, and not necessarily the closest one to the boundary), and let X_p denote the projection of X onto the separating boundary
- Now, let $\boldsymbol{r}$ be a vector $\overline{XX_p}$. Note that $\boldsymbol{r}$ is perpendicular to the boundary, and also that $\|\boldsymbol{r}\|$ is the required distance



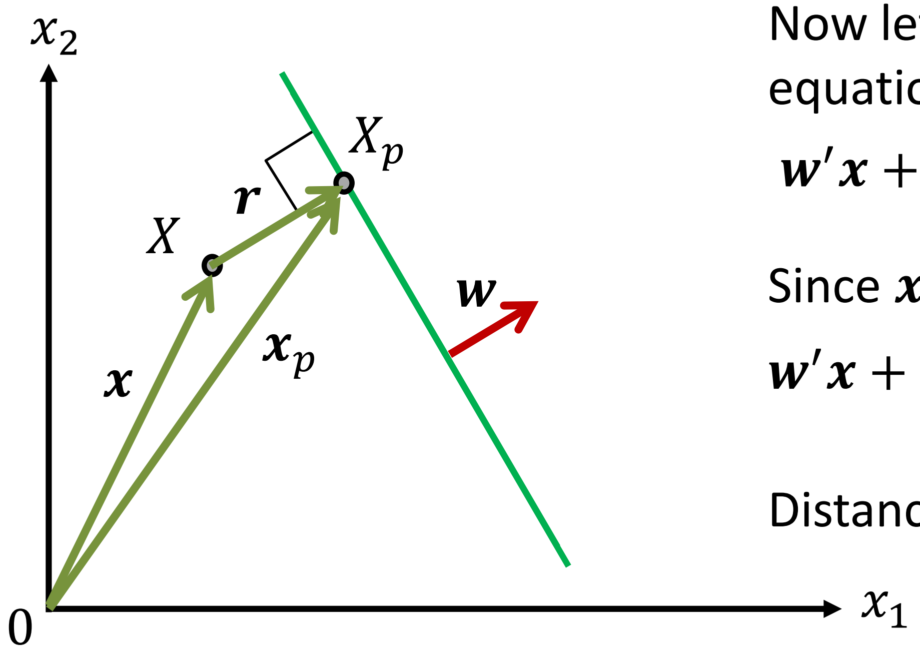
The separation boundary is defined by parameters $\boldsymbol{w}$ and b.

From our previous lecture, recall that $\boldsymbol{w}$ is a vector normal (perpendicular) to the boundary

In the figure, $\boldsymbol{w}$ is drawn from an arbitrary starting point

Distance from point to hyperplane 2/3

- Vectors $\boldsymbol{r}$ and $\boldsymbol{w}$ are parallel, but not necessarily of the same length. Thus $\boldsymbol{r} = \boldsymbol{w}\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{w}\|}$
- Next, points X and X_p can be viewed as vectors $\boldsymbol{x}$ and $\boldsymbol{x}_p$. From vector addition rule, we have that $\boldsymbol{x} + \boldsymbol{r} = \boldsymbol{x}_p$ or $\boldsymbol{x} + \boldsymbol{w}\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{w}\|} = \boldsymbol{x}_p$



Now let's multiply both sides of this equation by $\boldsymbol{w}$ and also add b:

$$\boldsymbol{w}'\boldsymbol{x} + b + \boldsymbol{w}'\boldsymbol{w}\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{w}\|} = \boldsymbol{w}'\boldsymbol{x}_p + b$$

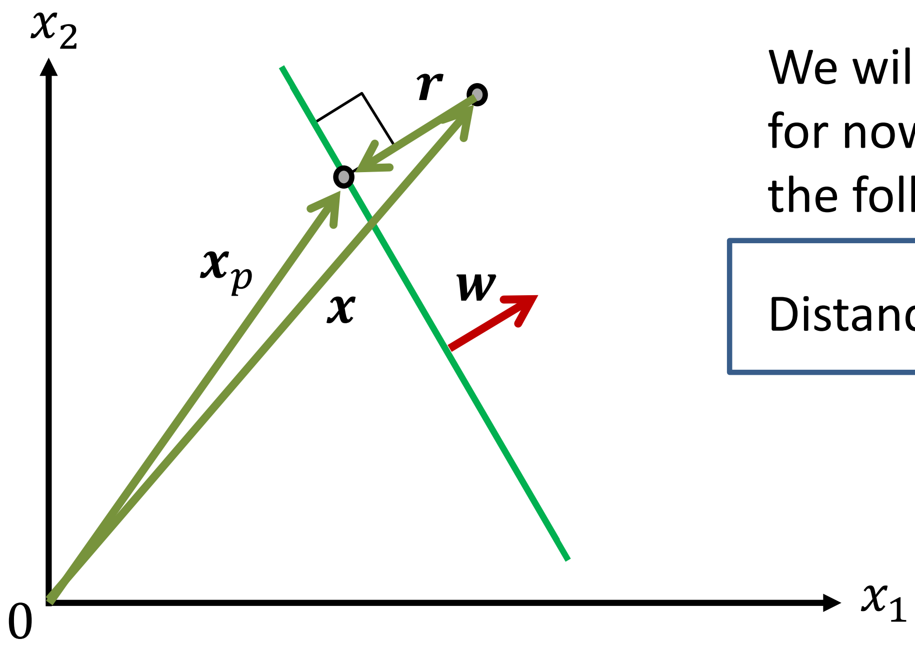
Since $\boldsymbol{x}_p$ lies on the boundary, we have

$$\boldsymbol{w}'\boldsymbol{x} + b + \|\boldsymbol{w}\|^2\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{w}\|} = 0$$

Distance is $\|\boldsymbol{r}\| = -\frac{\boldsymbol{w}'\boldsymbol{x}+b}{\|\boldsymbol{w}\|}$

Distance from point to hyperplane 3/3

- However, if we took our point from the other side of the boundary, vectors $\boldsymbol{r}$ and $\boldsymbol{w}$ would be anti-parallel, giving us $\boldsymbol{r} = -\boldsymbol{w}\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{w}\|}$
- In this case, distance is $\|\boldsymbol{r}\| = \frac{\boldsymbol{w}'\boldsymbol{x}+b}{\|\boldsymbol{w}\|}$



We will return to this fact shortly, and for now we combine the two cases in the following result:

$$\boxed{\text{Distance is } \|\boldsymbol{r}\| = \pm\frac{\boldsymbol{w}'\boldsymbol{x}+b}{\|\boldsymbol{w}\|}}$$

Encoding the side using labels

- Training data is a collection $\{\boldsymbol{x}_i, y_i\}$, $i = 1, \dots, n$, where each $\boldsymbol{x}_i$ is an m-dimensional instance and y_i is the corresponding binary label encoded as -1 or 1
- Given a perfect separation boundary, y_i encode the side of the boundary each $\boldsymbol{x}_i$ is on
- Thus the distance from the i-th point to a perfect boundary can be encoded as $\|\boldsymbol{r}_i\| = \frac{y_i\left(\boldsymbol{w}'\boldsymbol{x}_i + b\right)}{\|\boldsymbol{w}\|}$

Maximum margin objective

- The distance from the i-th point to a perfect boundary can be encoded as $\|\boldsymbol{r}_i\| = \frac{y_i(\boldsymbol{w}'\boldsymbol{x}_i+b)}{\|\boldsymbol{w}\|}$
- The margin width is the distance to the closest point
- Thus SVMs aim to maximise $\left(\min_{i=1,\ldots,n} \frac{y_i(\boldsymbol{w}'\boldsymbol{x}_i+b)}{\|\boldsymbol{w}\|}\right)$ as a function of $\boldsymbol{w}$ and b

Do you see any problems with this objective?

art: OpenClipartVectors at pixabay.com (CC0)

Remember that $\|\boldsymbol{w}\| = \sqrt{w_1^2 + \cdots + w_m^2}$

Constraining the objective

- SVMs aim to maximise $\left(\min_{i=1,\dots,n} \frac{y_i(\boldsymbol{w}'\boldsymbol{x}_i+b)}{\|\boldsymbol{w}\|}\right)$
- Introduce (arbitrary) extra requirement $\frac{y_{i^*}(\boldsymbol{w}'\boldsymbol{x}_{i^*}+b)}{\|\boldsymbol{w}\|} = \frac{1}{\|\boldsymbol{w}\|}$
  * Here i^* denotes the distance to the closest point
- We now have that SVMs aim to find

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \|\boldsymbol{w}\|$$

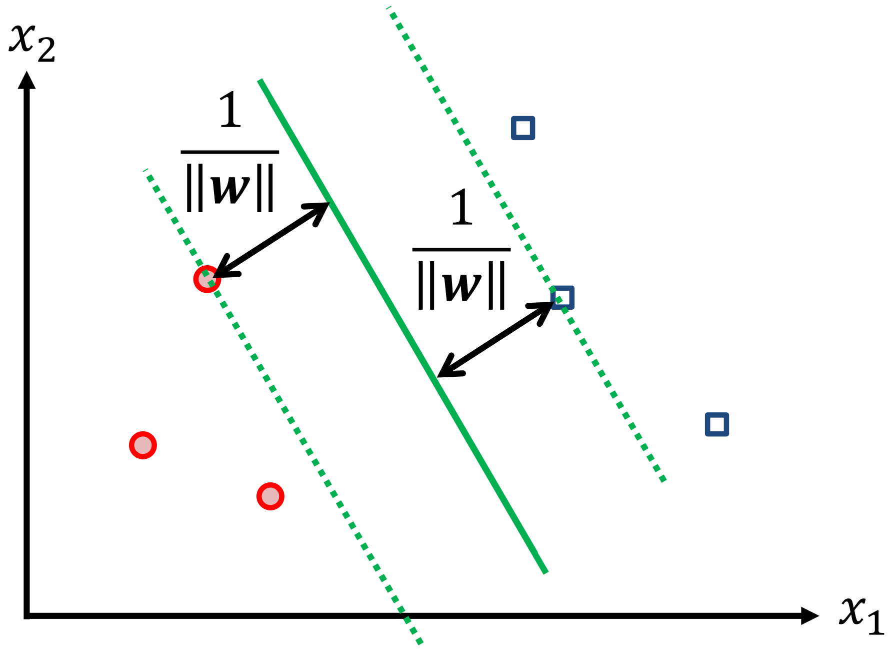
$$\text{s.t. } y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \geq 1 \text{ for } i = 1, \dots, n$$

Hard margin SVM objective

We now have a major result: SVMs aim to find

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \|\boldsymbol{w}\|$$

$$\text{s.t. } y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \geq 1 \text{ for } i = 1, \dots, n$$



x_2
x_1
$\frac{1}{\|\boldsymbol{w}\|}$
$\frac{1}{\|\boldsymbol{w}\|}$

Note 1: parameter b is optimised indirectly by influencing constraints

Note 2: all points are enforced to be on or outside the margin

Therefore, this version of SVM is called *hard-margin SVM*

Lagrangian for hard margin SVM

- Hard margin SVM objective is a constrained optimisation problem:

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$\text{s.t. } y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - 1 \geq 0 \text{ for } i = 1, \dots, n$$

- We approach this problem using the method of Lagrange multipliers/KKT conditions

- To this end, we first define the Lagrangian/KKT objective

$$L_{KKT}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \underbrace{\boxed{\frac{1}{2}\|\boldsymbol{w}\|^2}}_{\text{primal objective}} - \sum_{i=1}^{n} \lambda_i \underbrace{\boxed{\left(y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - 1\right)}}_{\text{constraints}}$$

KKT conditions for hard margin SVM

- Our Lagrangian/KKT objective is

$$L_{KKT}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \lambda_i (y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - 1)$$

- The corresponding KKT conditions are:
- $y_i((\boldsymbol{w}^*)'\boldsymbol{x}_i + b^*) - 1 \geq 0$ for $i = 1, \dots, n$ — this just repeats constraints, trivial
- $\lambda_i^* \geq 0$ for $i = 1, \dots, n$ — require non-negative multipliers in order for the theorem to work
- $\lambda_i^*(y_i((\boldsymbol{w}^*)'\boldsymbol{x}_i + b^*) - 1) = 0$ — "complementary slackness", we'll come back to that
- $\nabla_{\boldsymbol{w},b} L_{KKT}(\boldsymbol{w}^*, b^*, \boldsymbol{\lambda}^*) = 0$ — zero gradient, somewhat similar to unconstrained optimisation

Why use KKT conditions

- Proposition:
- If $\boldsymbol{w}^*$ and b^* is a solution of the primal hard margin SVM problem, then there exists $\boldsymbol{\lambda}^*$, such that together $\boldsymbol{w}^*$, b^* and $\boldsymbol{\lambda}^*$ satisfy the KKT conditions
- If some $\boldsymbol{w}^*$, b^* and $\boldsymbol{\lambda}^*$ satisfy KKT conditions then $\boldsymbol{w}^*$ and b^* is a solution of the primal problem
- Proof is outside the scope of this subject
  - Verify that SVM primal problem satisfies certain regularity conditions

Gradient of Lagrangian

- Our Lagrangian/KKT objective is

$$L_{KKT}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \lambda_i (y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - 1)$$

- The following conditions are necessary:

$$\frac{\partial L_{KKT}}{\partial b} = \sum_{i=1}^{n} \lambda_i y_i = 0$$

$$\frac{\partial L_{KKT}}{\partial w_j} = w_j - \sum_{i=1}^{n} \lambda_i y_i (\boldsymbol{x}_i)_j = 0$$

- Substitute the conditions into Lagrangian to obtain

$$L_{KKT}(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j$$

Re-parameterisation

- Let $\boldsymbol{w}^*$ and b^* be a solution of the primal problem. From the last KKT condition (zero gradient) we have

$$L_{KKT}(\boldsymbol{w}^*, b^*, \boldsymbol{\lambda}) = L_{KKT}(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j$$

- Parameters $\boldsymbol{\lambda}$ are still unknown

- In order for $\boldsymbol{w}^*$, b^* and $\boldsymbol{\lambda}^*$ to satisfy all KKT conditions, $\boldsymbol{\lambda}^*$ must maximise $L_{KKT}(\boldsymbol{\lambda})$
  - Proof is outside the scope of the subject

Lagrangian dual for hard margin SVM

- Given the above considerations, in order to solve the primal problem, we pose a new optimisation problem, called *Lagrangian dual problem*

$$\underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j$$

$$\text{s.t. } \lambda_i \geq 0 \text{ and } \sum_{i=1}^{n} \lambda_i y_i = 0$$

- This is a so-called *quadratic optimisation problem*, a standard problem that can be solved using off-the-shelf software

Hard margin SVM

- Training: finding $\boldsymbol{\lambda}$ that solve

$$\operatorname*{argmax}_{\boldsymbol{\lambda}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j$$

s.t. $\lambda_i \geq 0$ and $\sum_{i=1}^{n} \lambda_i y_i = 0$

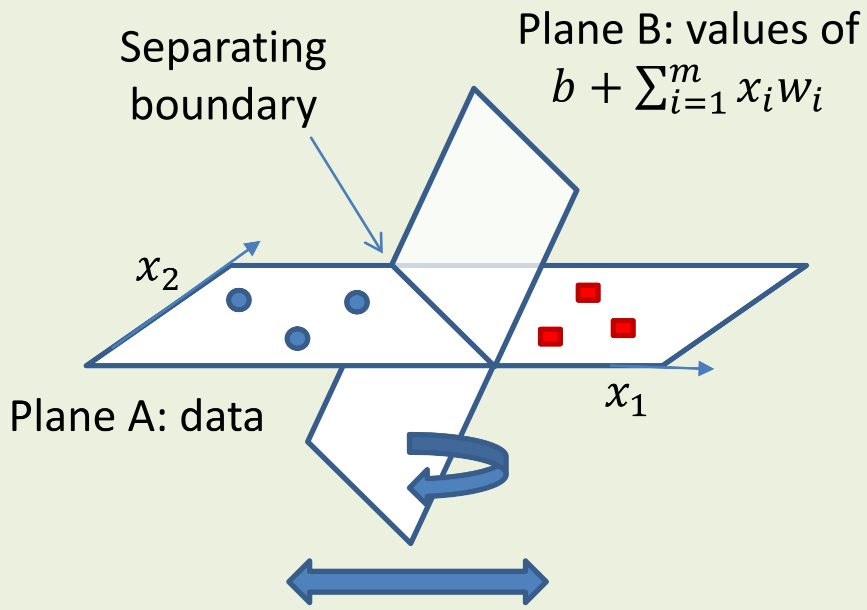
- Making predictions: classify new instance $\boldsymbol{x}$ based on the sign of

$$s = b^* + \sum_{i=1}^{n} \lambda_i^* y_i \boldsymbol{x}_i' \boldsymbol{x}$$

- Here b^* can be found by noting that for arbitrary training example j we must have $y_j\left(b^* + \sum_{i=1}^{n} \lambda_i^* y_i \boldsymbol{x}_i' \boldsymbol{x}_j\right) = 1$

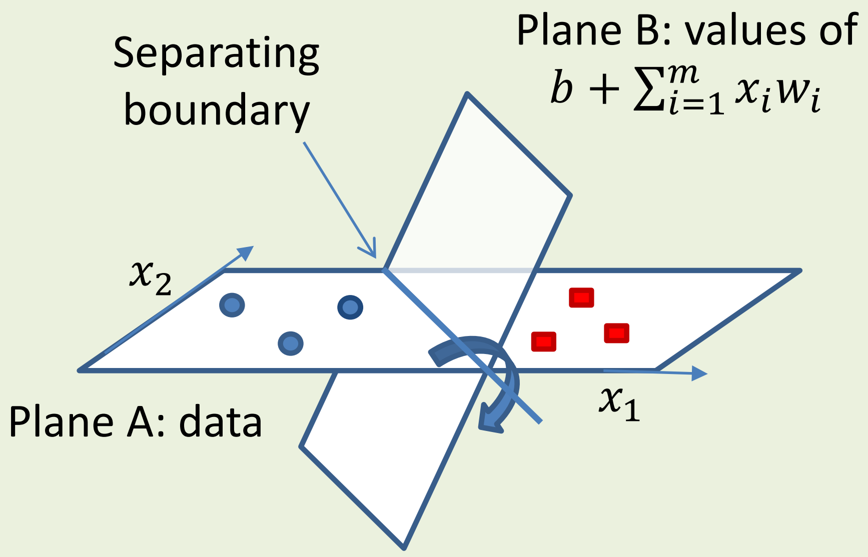
Geometry of SVM training 1/2

- Training a linear SVM essentially means moving/rotating plane B so that separating boundary changes
- This is achieved by changing w_i and b



Geometry of SVM training 2/2

- However, we can also rotate Plane B along the separating boundary
- In this case, the boundary does not change
    - Same classifier!
- The additional requirement fixes the angle between planes A and B to a particular constant



Regularised training error as objective

- Recall ridge regression objective

$$\text{minimise } \left(\boxed{\sum_{i=1}^{n}(y_i - \boldsymbol{w}'\boldsymbol{x}_i)^2} + \boxed{\lambda\|\boldsymbol{w}\|^2}\right)$$

- Hard margin SVM objective

data-dependent training error

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \boxed{\|\boldsymbol{w}\|}$$

data-independent regularisation term

$$\text{s.t. } \boxed{y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \geq 1 \text{ for } i = 1, \dots, n}$$

- The constraints can be interpreted as loss

$$l_\infty = \begin{cases} 0 & 1 - y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \leq 0 \\ \infty & 1 - y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) > 0 \end{cases}$$

Hard margin SVM loss

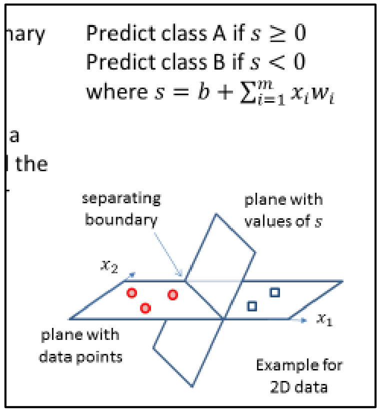
- The constraints can be interpreted as loss

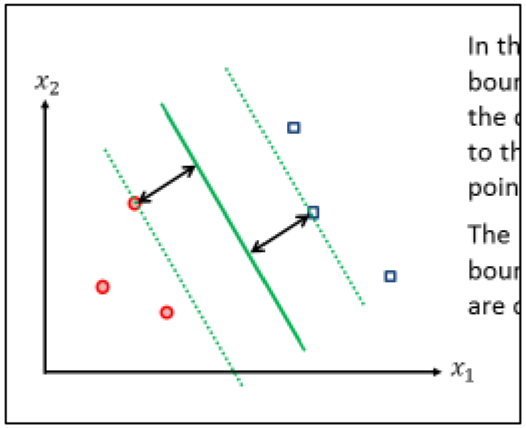
$$l_\infty = \begin{cases} 0 & 1 - y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \leq 0 \\ \infty & 1 - y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) > 0 \end{cases}$$

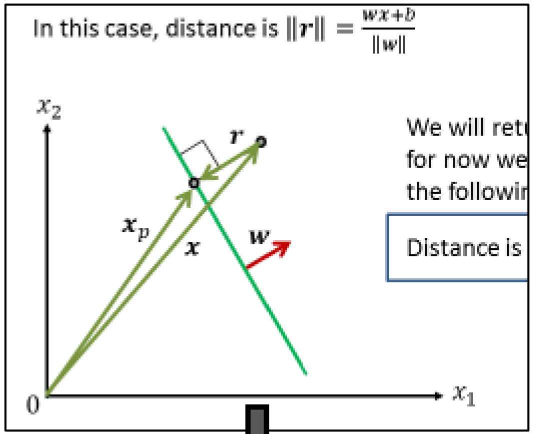
- In other words, for each point:
  - If it's on the right side of the boundary and at least $\frac{1}{\|\boldsymbol{w}\|}$ units away from the boundary, we're OK, the loss is 0
  - If the point is on the wrong side, or too close to the boundary, we immediately give infinite loss thus prohibiting such a solution altogether

Solving the dual problem

- The SVM Lagrangian dual problem is a *quadratic optimisation problem*. Using standard algorithms this problem can be solved in in $O(n^3)$
- This is still inefficient for large data. Several specialised solutions have been proposed
- These solutions mostly involve decomposing the training data and breaking down the problem into a number of smaller optimisation problems that can be solved quickly
- The original SVM training algorithm called *chunking* exploits the fact that many of λs will be zero
- *Sequential minimal optimisation* (SMO) is another algorithm which can be viewed as an extreme case of chunking. SMO is an iterative procedure that analytically optimises randomly chosen pairs of λs in each iteration







Hard margin SVM objective is a constrained optimisation problem:

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{w}\|^2$$

s.t. $y_i(\boldsymbol{wx}_i + b) - 1 \geq 0$ for $i = 1, \dots, n$

- The corresponding KKT conditions a
- $y_i(\boldsymbol{wx}_i + b) - 1 \geq 0$ for $i = 1, \dots, n$
- $\lambda_i \geq 0$ for $i = 1, \dots, n$
- $\lambda_i(y_i(\boldsymbol{wx}_i + b) - 1) = 0$
- $\nabla_{\boldsymbol{w},b} L_{KKT}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = 0$ zero gradie to uncons

Hard margin SVM Lagrangian dual problem is

$$\underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i\lambda_j y_i y_j \boldsymbol{x}_i\boldsymbol{x}_j$$

s.t. $\lambda_i \geq 0$ and $\sum_{i=1}^{n} \lambda_i y_i = 0$

- Training: finding $\boldsymbol{\lambda}$ that solve

$$\underset{\boldsymbol{\lambda}}{\operatorname{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i\lambda_j y_i y_j \boldsymbol{x}_i\boldsymbol{x}_j$$

s.t. $\lambda_i \geq 0$ and $\sum_{i=1}^{n} \lambda_i y_i = 0$

- Making predictions: classify new instance $\boldsymbol{x}$ bas sign of

$$s = b + \sum_{i=1}^{n} \lambda_i y_i \boldsymbol{x}_i \boldsymbol{x}$$

graphics: stux at pixabay.com (CC0)