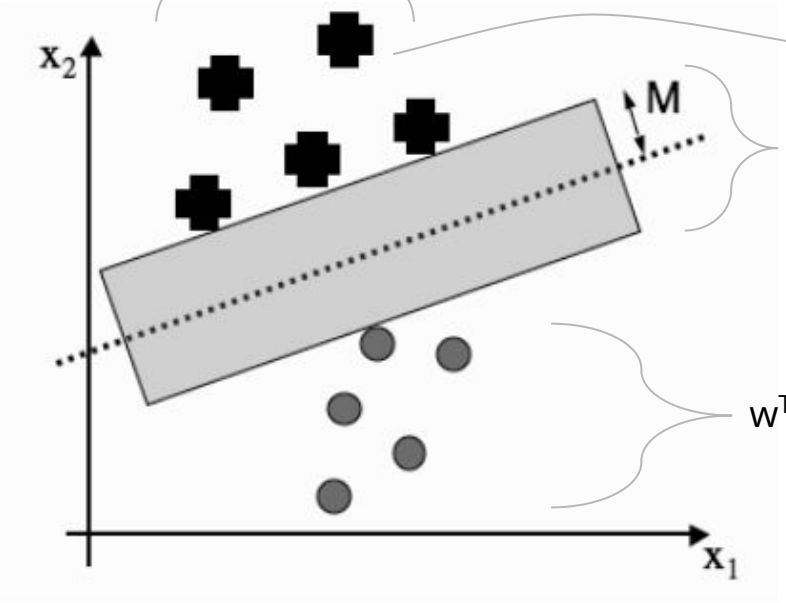
# Support Vector Machines

Connor Clarkson
Ben Lloyd-Roberts

# Problem Overview

SVM is used to model decision boundaries in data

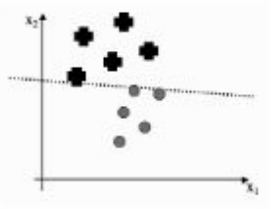
$w^T x + b \geq M$

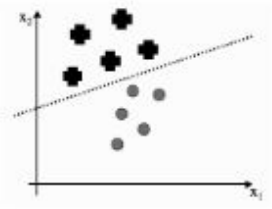


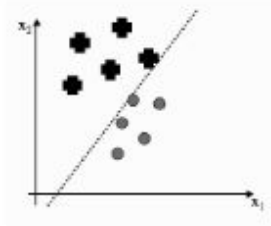

$x \in \mathbb{R}^D$

Make margin as large as possible

$w^T x + b \leq -M$

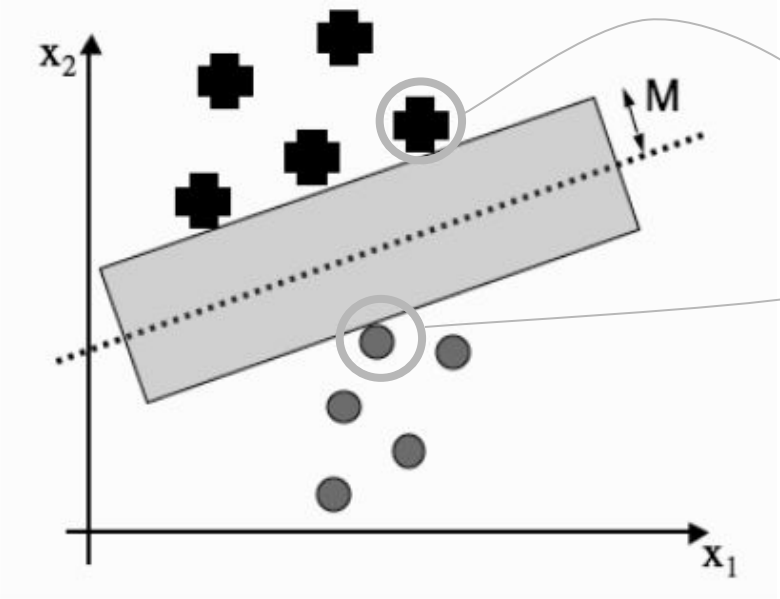






- Many ways to separate points
- Finding the best decision boundary is part of the optimization process

# The Margin and Support Vectors




Support Vectors

Hyperplane: $w^T x + b = 0$

: $w^T x + b \geq M$

: $w^T x + b \leq -M$

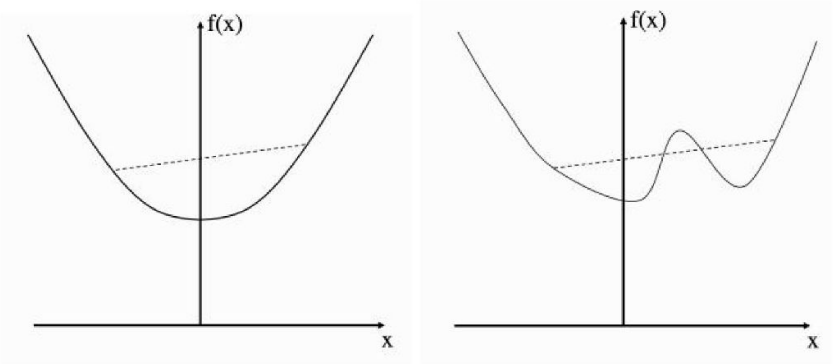
- Given our support vectors $x^+$ and $x^-$, we compute the separating hyperplane along the distance we travel from our support vectors
- We write the margin size M in terms of w
  - w is perpendicular to the boundary lines
- The width of the margin is $1/\|w\|$
- $1/\|w\|$ tells us that making M as large as possible is the same as making $w^T w$ as small as possible

# Constrained Optimisation Problem (Linear Case)

What are we optimizing?

$$\text{minimise } \frac{1}{2}\mathbf{w}^T\mathbf{w} \text{ subject to } t_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \text{ for all } i = 1, \ldots n.$$

- Optimizing over gradient descent is slow and inefficient with these constraints
- Solution: use quadratic programming and take advantage of convex
- If we take any two points on the line and join them with a straight line then every point on the line will be above the curve



# Constrained Optimisation Problem (Cont.)

When we find the optimal solution, the Karush-Kuhn-Tucker (KKT) conditions will also be satisfied:

Lagrange multipliers

Condition 1: $$\lambda_i^*(1 - t_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*)) = 0$$

Condition 2: $$1 - t_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*) \leq 0$$

Condition 3: $$\lambda_i^* \geq 0$$

These conditions are only true for support vectors as those vectors are in the active set of constraints.

# Constrained Optimisation Problem (Cont.)

Now that we have defined out constraints we need to reformulate the function into a lagrangian function:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \lambda_i (1 - t_i(\mathbf{w}^T\mathbf{x}_i + b))$$

If we set the derivatives of the lagrangian function w.r.t w and b then we can find the saddle points (maxima):

$$\mathbf{w}^* = \underbrace{\sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i}_{\nabla_{\mathbf{w}}\mathcal{L}}, \underbrace{\sum_{i=1}^{n} \lambda_i t_i}_{\frac{\partial \mathcal{L}}{\partial b}}$$

# Constrained Optimisation Problem (Cont.)

$$\mathbf{w}^* = \underbrace{\sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i}_{\nabla_{\mathbf{w}} \mathcal{L}}, \underbrace{\sum_{i=1}^{n} \lambda_i t_i = 0}_{\frac{\partial \mathcal{L}}{\partial b}}$$

We can then substitute these expressions at the optimal value of w and b into our lagrangian function:

$$\mathcal{L}(\mathbf{w}^*, b^*, \boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n} \lambda_i t_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j t_i t_j \mathbf{x}_i^T \mathbf{x}_j$$

This equation is known as the dual problem. The aim being to maximise it w.r.t $\lambda_i$ variables.
The constraints are:

$$\lambda_i \geq 0 \qquad \sum_{i=1}^{n} \lambda_i t_i = 0.$$

# Constrained Optimisation Problem (Cont.)

So how do we predict?

- We are given the optimal w:

$$\mathbf{w}^* = \sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i, \sum_{i=1}^{n} \lambda_i t_i = 0$$

- We know that a support vector is $t_i(w^T x_i + b) = 1$. So we can substitute the w for our w*:

$$\left( t_j - \sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i^T \mathbf{x}_j \right)$$

- We average over the whole set of support vectors for stability:

$$b^* = \frac{1}{N_s} \sum_{\text{support vectors } j} \left( t_j - \sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i^T \mathbf{x}_j \right)$$

$$\mathbf{w}^{*T}\mathbf{z} + b^* = \left( \sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i \right)^T \mathbf{z} + b^*$$

# Kernelization

Consider basis comprising:

$$x_1, x_2, \ldots, x_d \quad \leftarrow \text{Scalar inputs}$$

$$x_1^2, x_2^2, \ldots, x_d^2 \quad \leftarrow \text{Squared inputs}$$

$$x_1 x_2, x_1 x_3, \ldots, x_{d-1} x_d. \quad \leftarrow \text{Pairwise product}$$

# Kernelization

Consider basis comprising:

$$x_1, x_2, \ldots, x_d \quad \leftarrow \text{Scalar inputs}$$

$$x_1^2, x_2^2, \ldots, x_d^2 \quad \leftarrow \text{Squared inputs}$$

$$x_1 x_2, x_1 x_3, \ldots, x_{d-1} x_d. \quad \leftarrow \text{Pairwise product}$$

Then for the case *d = 3*

$$\Phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, x_1^2, x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3).$$

# Kernelization

Consider basis comprising:

$$\begin{cases} x_1, x_2, \ldots, x_d & \leftarrow \text{Scalar inputs} \\ x_1^2, x_2^2, \ldots, x_d^2 & \leftarrow \text{Squared inputs} \\ x_1 x_2, x_1 x_3, \ldots, x_{d-1} x_d. & \leftarrow \text{Pairwise product} \end{cases}$$

$$\mathbf{w}^T \mathbf{x} + b = \left( \sum_{i=1}^{n} \lambda_i t_i \phi(\mathbf{x}_i) \right)^T \phi(\mathbf{z}) + b.$$

This becomes expensive!

Then for the case *d = 3*

$$\Phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, x_1^2, x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3).$$

# Kernelization

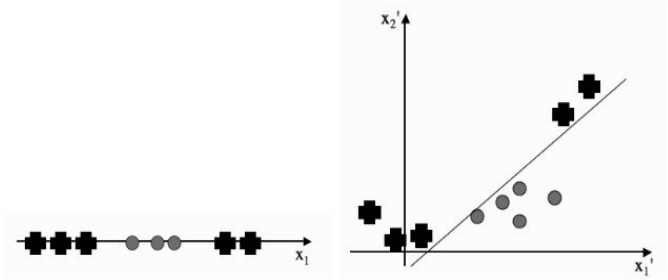
Suppose: $\Phi(\mathbf{x})^T\Phi(\mathbf{y}) = 1 + 2\sum_{i=1}^{d} x_i y_i + \sum_{i=1}^{d} x_i^2 y_i^2 + 2 \sum_{i,j=1;i<j}^{d} x_i x_j y_i y_j$

Simplified by factorising to $(1+\mathbf{x}^T\mathbf{y})^2$ reduces complexity from O($d^2$) to O(d)

Linear kernel holds for polynomials of degree *s*.

Knowledge of $\Phi(\cdot)$ no longer necessary

Any **symmetric positive definite** function can be expressed in terms of the inner product of some $\Phi(\cdot)$




Using $x_1^2$ as well as $x_1$ allows these two classes to be separated.

# Choosing Kernels

**Kernel**
Any symmetric, positive definite function $\mathrm{K}(\mathrm{x},\mathrm{y})$

- Symmetry allows for transposition
- Forces positivity on arbitrary functions
- Convolve kernels to make another kernel

**Common Basis Functions**

- Polynomials

$$\mathbf{K}(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}^T\mathbf{y})^s$$

- Sigmoid

$$\mathbf{K}(\mathbf{x},\mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y}-\delta)$$

- Radial basis function (RBF)

$$\mathbf{K}(\mathbf{x},\mathbf{y}) = \exp\left(-(\mathbf{x}-\mathbf{y})^2/2\sigma^2\right)$$

# The SVM Algorithm - Initialisation

- Select kernel given the basis
  - Computation of $\mathbf{K} = \mathbf{X}\mathbf{X}^T$
  - For d=1, return $\mathbf{K}$ for polynomial of degree *d* return $\frac{1}{\sigma}\mathbf{K}^d$
  - For RBF Kernel, compute $\mathbf{K} = \exp(-(\mathbf{x} - \mathbf{x}')^2/2\sigma^2)$
- Set kernel parameters
- Compute distance matrix between data points

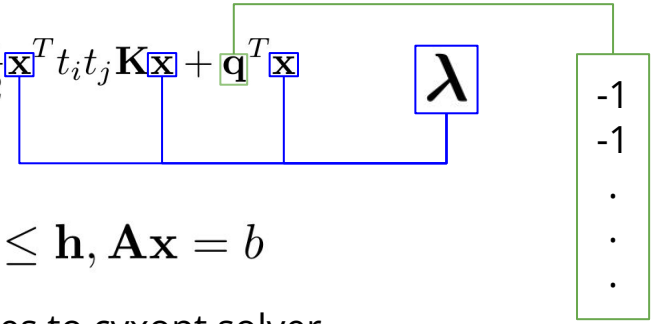
# The SVM Algorithm - Training

- Assemble constraint set as matrices to solve:

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T t_i t_j \mathbf{K}\mathbf{x} + \mathbf{q}^T\mathbf{x}$$

subject to

$$\mathbf{Gx} \leq \mathbf{h}, \mathbf{Ax} = b$$




- Pass matrices to cvxopt solver
- Identify Support Vectors and discard remaining training data.
- Compute $b^*$

# The SVM Algorithm - Training

- Assemble constraint set as matrices to solve:

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T t_i t_j \mathbf{K}\mathbf{x} + \mathbf{q}^T\mathbf{x} \longrightarrow \max_{\boldsymbol{\lambda}} = \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\boldsymbol{\lambda}^T \boldsymbol{\lambda}\mathbf{t}\mathbf{t}^T \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j)\boldsymbol{\lambda},$$

subject to

$$\mathbf{G}\mathbf{x} \leq \mathbf{h}, \mathbf{A}\mathbf{x} = b \longrightarrow 0 \leq \lambda_i \leq C, \sum_{i=1}^{n} \lambda_i t_i = 0.$$

- Pass matrices to cvxopt solver
- Identify Support Vectors and discard remaining training data.
- Compute $b^*$

# The SVM Algorithm - Classification

Given  a new observation **z:**

- Compute inner product of test data and support vectors
- Classify as :

$$\sum_{i=1}^{n} \lambda_i t_i \mathbf{K}(\mathbf{x}_i, \mathbf{z}) + b^*$$

Returning the label (hard classification) or value (soft classification) of $y$.

# Multi-Class Classification

N-Class classification problem needs reformulating

SVM only works when comparing two classes.

Solution: Train one SVM to classify one class against all others.

N-classes → N SVMs

Determine strongest prediction by position of the basis vector input **within** the class region